Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

Solving Quantum Many-Body Problem with Feed-Forward Neural Networks

Sheng-Hsuan Lin

Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

Solving Quantum Many-Body Problem with Feed-Forward Neural Networks

Author:	Sheng-Hsuan Lin
1st examiner:	Univ.-Prof. Dr. Lode Pollet
2nd examiner:	Univ.-Prof. Dr. Frank Pollmann
Submission Date:	April 1st, 2018

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

April 1st, 2018 Sheng-Hsuan Lin

Acknowledgments

The work in this thesis could not be done without the support and help from many people. I would especially like to thank

- Alexander Wolf for motivating me to start thinking about the connection between machine learning and computational methods in condense matter physics, which this thesis only cover a small part of it.
- Fabian Heidrich-Meisner for providing research opportunity and guidance so I could keep in touch with physics in my Master study.
- Lode Pollet for granting the freedom to explore and providing support when needed.
- Frank Pollmann for useful discussion and ideas that could not be covered in this thesis due to time limitation.
- Jonas Greitemann for comparing results and teaching me not only physics but programming.
- Lei Wang for pointing out the connection to reinforcement learning.
- Jinguo Liu for useful discussion in Marburg.
- Ivan Glasser, Giuseppe Carleo and Zi Cai for sharing detail of their works.
- Yu Wang, Yuanze Chen, and Yi-An Lai for useful discussions about machine learning.
- Cordula Weber for helps with administrative task.
- my family for all the support throughout the years.

"The problems of continuous mathematics are the problems that science and engineering are built upon; without numerical methods, science and engineering as practiced today would come quickly to a halt. They are also the problems that preoccupied most mathematicians from the time of Newton to the twentieth century. As much as any pure mathematicians, numerical analysts are the heirs to the great tradition of Euler, Lagrange, Gauss and the rest. If Euler were alive today, he wouldn't be proving existence theorems."

-Lloyd N. Trefethen

Abstract

Motivated by the success of describing quantum states with restricted Boltzmann machine, we consider general feed-forward neural networks as variational wavefunctions. We show that with variational Monte Carlo method or with supervised learning, neural networks could be trained to represent quantum states. We demonstrated the ability of neural network representing quantum states on frustrated problems including 1d and 2d J_1-J_2 model. The difficulties of optimization large number of parameters are addressed. Considering the connection between natural gradient method and stochastic reconfiguration method, we point out possible ways to extend the result to deep networks with large number of parameters.

Contents

Part I.

Introduction and Background Theory

1. Introduction

One well-known challenge in both quantum many-body and machine learning problem is the underlying exponentially large dimensionality. On the Physics side, the large dimensionality lies in both the description of the problem, i.e. the Hamiltonian, and the solutions, i.e. the eigenstates of the Hamiltonian. Although we call the eigenstates the solutions, in fact, the quantities we are interested in are the expectation values of the observable. If one could have the eigenstate or wavefunction, one could calculate the expectation values explicitly. But even if one does not know the wavefunction, if one could directly calculate the expectation values, one still solve the problem. This includes a broad range of methods like, Quantum Monte Carlo (QMC), which are not the interest of this thesis. In the following, we review different methods involving explicit computation of the wavefunction.

The difficulty comes in two-fold: What is an efficient representation and how to find the representation through the optimization efficiently. Physicists have been trying to come up with different efficient expression of the wavefunction, which by polynomial many parameters could be able to capture the essence of the quantum state. A whole range of techniques including, Matrix Product States (MPS), Projected Entangled Pair States (PEPS), Entangled Plaquette States (EPS)/Correlator Product States (CPS), String Bond States (SBS), etc, were developed, which all fall under the category of tensor network methods.

In one dimension, the great success of MPS comes not only from the fact the majority of physically interesting states are area-law entangled, which leads to efficient representation, but also because there exist efficient algorithms, e.g. DMRG, in finding the targeted wavefunction. Optimization of DMRG like methods can be carried out efficiently by sweeping and updating the core of the tensor. This can not be generalized in higher dimension.

In higher dimension, sometimes even though we know the efficient representation of the state, we could not efficiently find the targeted state. In other words, we could identify a relevant subspace. However, the computational cost for optimization is too expensive and need to rely on approximation. Even after the optimization and identifying the state, the computation of expectation value involves contraction that is $\#P$ hard [75].

Variational Monte Carlo (VMC) is an optimization method in finding the ground state with a given parametrization of the wavefunction (see [22] for a general review). VMC is a less efficient method but also sign-problem free. Back in the old days, people usually optimized over some relatively restricted wavefunctions with less than ten parameters. As a result, even though VMC itself is unbiased, the choice of how wavefunction is parametrized could be highly biased. One could perform VMC with MPS, which then is biased in terms of entanglement [74, 76, 13, 80]. By increasing number of variational parameters, the result will be less biased.

Machine learning problems, on the other hand, are also about extracting relevant information from exponentially large dimensionality. Recently, Giuseppe et al. propose the idea of using restricted Boltzmann machine (RBM) as variational trial wavefunction [9], which brought up discussion over the connection between RBM and Tensor Network [12, 23, 16, 45]. These study focused on the physical properties of RBM, where the properties of general neural network is yet studied. RBM stems from the hope to mimic the memory scheme of people, which learns the underlying input probability distribution. It is used for mainly for unsupervised learning [42, 73]. RBM could be viewed as an undirected probabilistic graphical model (PGM) or a stochastic neural network. In this thesis, we consider general feed-forward neural network as quantum state (NNQS) and study their properties.

As like RBM, feed-forward neural networks with one hidden layer is shown to be an universal approximators [43, 15]. As like MPS without truncation spans the full Hilbert space and MPS with finite bond dimension covers the low entanglement corner of the Hilbert space, the question is whether networks with small enough hidden units can represent the wavefunction we are interested in. In addition, deep neural networks are able to represent certain class of functions with polynomial many parameters while it takes a shallow networks exponentially many hidden units to do so [59]. It might be the case that the wavefunctions we are interested in could only be represented by special kind of architecture. Aside from numerical testing with different architectures, an analysis with the maximum entanglement that a neural network could encode might be useful. Such analysis might also help the Machine Learning community to develop a way to systematically design new neural network.

The structure of the thesis is as follows. We review the variational Monte Carlo method in chapter 2, including the problem setup in 2.1, technical details for variational Monte Carlo method in 2.2 and the trial wavefunctions in 2.3. In chapter 3, we describe the background for supervised learning in 3.1 and give linear models as example in 3.2 and motivates the neural networks as functions with composite non-linear transformations in 3.3. The connection of variational Monte Carlo and reinforcement learning is discussed in 3.4. With the connection, a promising approximated second-order method for neural network known as K-FAC is introduced in 3.5. Even though not the main concern of this thesis, interesting connection for methods in condensed matter physics and machine learning is discussed in 3.6. Previous works related to this thesis are reviewed in chapter 4. Results in chapter 5 include the explanation of the NNQS implementation in 5.1, result for VMC with NNQS in 5.2 and result for supervised learning with NNQS in 5.3. We end with discussion of NNQS and possible future works in 5.4.

Part II.

Body

2. Method I: Variational Monte Carlo

2.1. Problem Description

We consider the discrete quantum many-body problem on lattice. The problems are defined by the Hamiltonian of the system, of which the size grows exponentially with the systems size. One fundamental problem in condensed matter physics is about solving these large scale eigenvalue problems

$$\hat{H} |\psi\rangle = E |\psi\rangle .$$

Consider a lattice of N sites, $\boldsymbol{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_N\}$ is the computational basis representing the spin configuration. Suppose on each site, there is d local degree of freedom. The total Hilbert space is of dimension d^N. For a system with spin-$1/2$ particles, we would have $\sigma_i = 0, 1$ representing $|\uparrow\rangle, |\downarrow\rangle$, and $d = 2S + 1 = 2$. A pure quantum state is a vector in this Hilbert space and can be expressed in the basis $|\sigma_1, \sigma_2, \ldots, \sigma_N\rangle$.

$$|\psi\rangle = \sum_{\sigma} |\sigma\rangle \langle\sigma|\psi\rangle = \sum_{\sigma_1,\ldots,\sigma_N} \psi_{\sigma_1,\ldots,\sigma_N} |\sigma_1, \ldots, \sigma_N\rangle \tag{2.1}$$

where in general $\psi_{\sigma_1,\ldots,\sigma_N} \in \mathbb{C}^{d^N}$. A restriction to $\mathbb{R}$ is possible if the Hamiltonian is not complex-valued. In quantum physics, there are many situations where we are interested in the ground state, which is the eigenvector with the lowest eigenvalue of some Hamiltonian $\boldsymbol{H}$ describing the physical system. Once the state is known, various physical quantities could be computed from the state. However, the exponential growth of the dimension with respect to the system size N render both the exact representation and solving the eigenvalue problem impossible.

One way to avoid this problem is to compute the physical quantities without knowing the state, e.g. QMC. Another way to avoid this problem is to solve this problem approximately with efficient representation of the state. To be concrete, the exact wavefunction $|\psi\rangle$ is approximated by a function Ψ, which used less parameters, i.e. polynomial in system size. We denote the parameters of the function as $\mathbf{w}$ and the function Ψ maps the given configuration to the corresponding coefficient in the vector, that is

$$\Psi : \sigma \in \{0, \ldots, d-1\}^N \to \Psi(\sigma; \mathbf{w}) = \psi_{\sigma_1,\ldots,\sigma_N} \in \mathbb{C} \tag{2.2}$$

throughout this thesis, we abused the notation between ψ and Ψ and simply write the wavefunction as,

$$|\psi\rangle = \sum_{\boldsymbol{\sigma}} |\boldsymbol{\sigma}\rangle \langle\boldsymbol{\sigma}|\psi\rangle = \sum_{\boldsymbol{\sigma}} \psi(\boldsymbol{\sigma}; \mathbf{w}) |\boldsymbol{\sigma}\rangle = \sum_{\boldsymbol{\sigma}} \psi_{\sigma_1,\ldots,\sigma_N} |\boldsymbol{\sigma}\rangle \tag{2.3}$$

Typically the work flow would be the following: With a large matrix $\hat{H}$ describing system, one first choose a specific kind of parameterization of the wavefunction. Then, the eigenvalue problem is then cast as an optimization problem of finding the optimal parameters $\mathbf{w}_{\text{opt}}$ that minimized the energy.

$$\mathbf{w}_{\text{opt}} = \arg\min_{\mathbf{w}} \frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle} \tag{2.4}$$

Based on prior knowledge about the state and different physical arguments, one could assume a physical plausible trial wavefunction with fewer parameters. Even though, efficient representations are biased, they usually give good description of the system. It could also be thought that we are less biased with increasing number of parameters.

We will describe different ways to parametrized the wavefunction in section 2.3. There are different techniques developed to solve this optimization problem for specific forms of wavefunction. For example, DMRG for MPS. Usually these specific kind of algorithm is faster. In the following section 2.2, we describe the variational Monte Carlo method, which is a general algorithm that could solve the optimization problem for generic wavefunction.

2.2. Variational Monte Carlo algorithm

VMC is performed for two purpose. (1.) To calculate the expectation value of observable, i.e. $\langle\psi|\hat{O}|\psi\rangle$ / $\langle\psi|\psi\rangle$, given a wavefunction ψ. (2.) To find the optimal parameters for some objective function, e.g. energy expectation value, in some restricted subspace. In other word, given $|\psi\rangle = |\psi(\mathbf{w})\rangle$, to find $\mathbf{w}_{opt}$ such that

$$\mathbf{w}_{\text{opt}} = \arg\min_{\mathbf{w}} \frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle}. \tag{2.5}$$

2.2.1. Expectation value of observable

We first show that the expectation value of any quantum mechanical observable, $\hat{O} = \hat{O}^\dagger$, can be calculated on a set of configurations obtained from Markov chain Monte Carlo (MCMC) sampling from the wavefunction.

$$\frac{\langle\psi|\hat{O}|\psi\rangle}{\langle\psi|\psi\rangle} = \sum_{\alpha}\sum_{\beta}\frac{\langle\psi|\alpha\rangle\langle\alpha|\hat{O}|\beta\rangle\langle\beta|\psi\rangle}{\langle\psi|\psi\rangle} \tag{2.6}$$

$$= \sum_{\alpha}\frac{|\langle\alpha|\psi\rangle|^2}{\langle\psi|\psi\rangle}\sum_{\beta}\langle\alpha|\hat{O}|\beta\rangle\frac{\langle\beta|\psi\rangle}{\langle\alpha|\psi\rangle} \tag{2.7}$$

$$= \sum_{\alpha}P(\alpha)O_{loc}(\alpha) \doteq \langle O_{\text{loc}}\rangle \tag{2.8}$$

$$\approx \frac{1}{N_{MC}}\sum_{i=1}^{N_{MC}}O_{loc}(\alpha_i) \tag{2.9}$$

where α, β are complete basis and the local value of the observable at configuration $\boldsymbol{\sigma}$,

$$O_{\text{loc}}(\boldsymbol{\sigma}) = \sum_{\boldsymbol{\sigma}'}\langle\boldsymbol{\sigma}|\hat{O}|\boldsymbol{\sigma}'\rangle\frac{\Psi(\boldsymbol{\sigma}')}{\Psi(\boldsymbol{\sigma})} \tag{2.10}$$

Assume there are at most k non-zero elements per row, the lattice size N, and the cost for evaluating the wavefunction is given by C_{eval}. One could immediately see that the cost to evaluate local value is of order $\mathcal{O}(kC_{eval})$. C_{eval} depends on the form of wavefunction and would usually be $\mathcal{O}(N^2)$ or $\mathcal{O}(N^3)$. k depends on the operator chosen. Even though the dimension of the operator grows exponentially with respect to system size d^N, physical observable is usually sparse. Therefore, we could still evaluate local value exactly. This should apply for all operators that are sparse, i.e. small support. We will call the local value of the Hamiltonian simply by the local energy.

The approximation only comes in at the last step, where we approximate the exact expectation value of the local value of observable by Markov chain sampling, as shown in 2.2.3.

2.2.2. Minimization of expectation value of observable

To solve the optimization problem, eq.(2.4), the simplest approach would be applying gradient descent. We will show that we can formulate the evaluation of gradient in the same structure as before. Therefore, the computation of exact gradient is replaced by stochastic sampling. We end up with stochastic gradient descent algorithm. [1]

Before diving into the formula for gradient, it is useful to first define some operators.

[1] The optimization problem is similar to that in policy-based Reinforcement Learning. See 3.4.

We define the log derivative with respect to the k-th parameter in the function,

$$\Delta_k(\boldsymbol{\sigma}) \doteq \frac{1}{\Psi(\boldsymbol{\sigma};\mathbf{w})}\partial_{w_k}\Psi(\boldsymbol{\sigma};\mathbf{w}) \tag{2.11}$$

$$\hat{\Delta}_k \doteq \sum_{\boldsymbol{\sigma}} |\boldsymbol{\sigma}\rangle \left[\frac{1}{\Psi(\boldsymbol{\sigma};\mathbf{w})}\partial_{w_k}\Psi(\boldsymbol{\sigma};\mathbf{w})\right] \langle\boldsymbol{\sigma}| \tag{2.12}$$

$$\hat{\Delta}_k |\psi\rangle = \sum_{\alpha} |\alpha\rangle \left[\frac{1}{\langle\alpha|\psi\rangle}\partial_{w_k}\langle\alpha|\psi\rangle\right] \langle\alpha|\psi\rangle \tag{2.13}$$

$$= \sum_{\alpha} |\alpha\rangle \, \partial_{w_k}\langle\alpha|\psi\rangle \tag{2.14}$$

$$= \partial_{w_k}|\psi\rangle \tag{2.15}$$

Note that if the parameter is a complex number, one should be careful about the complex conjugate.

$$\hat{\Delta}_k^* \doteq \sum_{\boldsymbol{\sigma}} |\boldsymbol{\sigma}\rangle \left[\frac{1}{\Psi(\boldsymbol{\sigma};\mathbf{w})^*}\partial_{w_k^*}\Psi(\boldsymbol{\sigma};\mathbf{w})^*\right] \langle\boldsymbol{\sigma}| \tag{2.16}$$

$$\langle\psi|\,\hat{\Delta}_k^* \doteq \sum_{\boldsymbol{\sigma}} \partial_{w_k^*}\Psi(\boldsymbol{\sigma};\mathbf{w})^* \langle\boldsymbol{\sigma}| = \sum_{\boldsymbol{\sigma}} \partial_{w_k^*}\langle\psi|\boldsymbol{\sigma}\rangle \langle\boldsymbol{\sigma}| \tag{2.17}$$

In the following, we consider two scenarios. One is an general complex-valued wavefunction parameterized by complex variables and the other with real variables. These two scenarios are of no difference in terms of physics, but are important to distinguish in numerical implementation.

If we consider complex parametrization, that means $w_i \in \mathbb{C}$ and $\psi : \mathbf{w} \in \mathbb{C}^{N_w} \to \psi(\mathbf{w}) \in \mathbb{C}$. To minimize a real-valued objective function, f, with respect to complex parameters w, the steepest ascent direction is the direction of the derivative of the complex-conjugated variables. And with the Wirtinger derivative, we have

$$w \to w - \partial_{w^*} f$$
$$\frac{\partial f}{\partial w^*} = \frac{1}{2}\left(\frac{\partial f}{\partial w_R} + i\frac{\partial f}{\partial w_I}\right)$$

$$\partial_{w_k^*}\frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle} = \frac{\left(\partial_{w_k^*}\langle\psi|\,\right)\hat{H}|\psi\rangle + 0}{\langle\psi|\psi\rangle} - \frac{\left[\left(\partial_{w_k^*}\langle\psi|\,\right)|\psi\rangle + 0\right]\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle^2} \tag{2.18}$$

$$= \langle\Delta_k^* E_{\text{loc}}\rangle - \langle\Delta_k^*\rangle\langle E_{\text{loc}}\rangle \tag{2.19}$$

where we use the property that $\partial_{w_k^*}|\psi(\mathbf{w})\rangle = 0$. We can now define the gradient vector,

$$F_k = \langle\Delta_k^* E_{\text{loc}}\rangle - \langle\Delta_k^*\rangle\langle E_{\text{loc}}\rangle \tag{2.20}$$

$$= \langle\Delta_k^*(E_{\text{loc}} - \langle E_{\text{loc}}\rangle)\rangle \tag{2.21}$$

Now, we could see if we consider a complex-valued wavefunction parametrized only by real parameters, with the property from Wirtinger derivative, we immediately have,

$$F_k = 2\mathrm{Re}\Big[\langle \Delta_k^* E_{loc}\rangle - \langle \Delta_k^*\rangle \langle E_{loc}\rangle\Big] \tag{2.22}$$

The same formula could also be derived by considering the complex-valued wavefunction with real parametrization, i.e. $w_i \in \mathbb{R}$, $\psi : \mathbf{w} \in \mathbb{R}^{N_w} \to \psi(\mathbf{w}) \in \mathbb{C}$,

$$\begin{aligned}
\partial_{w_k} \frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle} &= \frac{\langle\partial_{w_k}\psi|\hat{H}|\psi\rangle + \langle\psi|\hat{H}|\partial_{w_k}\psi\rangle}{\langle\psi|\psi\rangle} - \frac{\left[\langle\partial_{w_k}\psi|\psi\rangle + \langle\psi|\partial_{w_k}\psi\rangle\right]\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle^2} \\
&= \Big[\sum_\alpha P(\alpha)\frac{\langle\partial_{w_k}\psi|\alpha\rangle}{\langle\psi|\alpha\rangle}E_{loc}(\alpha) + \text{c.c.}\Big] - \Big[\sum_\alpha P(\alpha)\frac{\langle\partial_{w_k}\psi|\alpha\rangle}{\langle\psi|\alpha\rangle}\langle E_{loc}\rangle + \text{c.c.}\Big] \\
&= \sum_\alpha P(\alpha)\Big[\Delta_k^*(\alpha)E_{loc}(\alpha) - \Delta_k^*(\alpha)\langle E_{loc}\rangle + \text{c.c.}\Big] \\
&= 2\mathrm{Re}\Big[\langle\Delta_k^* E_{loc}\rangle - \langle\Delta_k^*\rangle\langle E_{loc}\rangle\Big]
\end{aligned}$$

Now we have the simple gradient descent algorithm for Variational Monte Carlo. See algorithm 1.

Result: The ground state wavefunction

Randomly initialize parameters $\mathbf{w}^{(0)}$;
Choose stepsize δt;
while *not converge* **do**
  Compute the gradient vector $\mathbf{f}$;
  Update parameters with $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \delta t\mathbf{f}$
end

Algorithm 1: Vanilla gradient descent algorithm

We could in general divide stochastic gradient descent algorithms into two categories. The first category is first-order methods. This includes methods which have strategy for adaptive stepsize [74] or accumulating gradient history to lower the statistical error, but are based on the information from first-order derivative of the objective function.

The second category is second-order methods. There are two different type of second-order methods. The Newton method involves the inverse of the Hessian matrix of the objective function [94]. Approximation and stabilization could be added to ensure faster and stable convergence [85, 62, 89, 95, 88]. The stochastic reconfiguration method (SR), a.k.a. natural gradient, involves the inverse of the metric on the projective Hilbert space [84].

For most of the optimization in this thesis, we choose to use the SR method proposed by Sorella [84]. We first give the outline of the algorithm and give the derivation in the next section 2.2.4. The SR overlap matrix S, a.k.a. empirical Fubini-Study metric, is defined as

$$S_{kk'} \doteq \langle \Delta_k^* \Delta_{k'} \rangle - \langle \Delta_k^* \rangle \langle \Delta_{k'} \rangle \tag{2.23}$$

The algorithm is defined as in algorithm 2

Result: The ground state wavefunction

Randomly initialize parameters $\mathbf{w}^{(0)}$;
Choose stepsize δt;
while *not converge* **do**
    Compute the gradient vector $\mathbf{f}$ and SR matrix S;
    Update parameters with $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \delta t S^{-1}\mathbf{f}$
end

Algorithm 2: Stochastic Reconfiguration algorithm

2.2.3. Statistics of Variational Monte Carlo

The essential part of all VMC algorithm is to compute the expectation value with respect to the probability $P(\alpha) \propto |\psi(\alpha)|^2$. Note that the probability is defined by the wavefunction, which in general would not be normalized. Due to the large dimension, the normalization constant is intractable and so is to compute the expectation value exactly. The way to overcome this is by stochastic sampling.

By stochastic sampling, we want to have a set of independent configurations $\{\boldsymbol{\sigma}_1, \cdots, \boldsymbol{\sigma}_{N_{MC}}\}$ where the frequency of the configuration is proportional to the probability. Then the expectation value of certain observable is given by,

$$\mathbb{E}_{\boldsymbol{\sigma}\sim P}\Big[O_{loc}(\boldsymbol{\sigma})\Big] \approx \frac{1}{N_{MC}} \sum_{i=1}^{N_{MC}} O_{loc}(\boldsymbol{\sigma}_i)$$

However, except for some specific case [21], direct sampling from wavefunction is impossible. We therefore resort to Markov Chain Monte Carlo (MCMC) sampling. Here, we give a breif review of Metropolis algorithm [56] with local update used in this thesis.

There are several remarks on the properties of Markov chain sampling.

- Consecutive configurations sampled are correlated. Especially when local update is applied to ensure a higher acceptance ratio. This is a problem because we will underestimate the statistical error we made and the optimization might be unstable without enough sampling.

Result: Expectation value of observable or independent sampled configurations from the wavefunction

Randomly initialize configuration $\boldsymbol{\sigma}$;
Warm up the system by repeating step 1 and 2 long enough;
for $i = 1$ **to** N_{MC} **do**
  (1) Propose a new configuration $\boldsymbol{\sigma}'$, which is based on a local update from the current configuration, e.g. nearest neighbor swap, with probability $T(\boldsymbol{\sigma} \to \boldsymbol{\sigma}')$.;
  (2) Accept the new configuration $\boldsymbol{\sigma}'$ with probability $\min(1, \frac{T(\boldsymbol{\sigma}\to\boldsymbol{\sigma}')|\psi(\boldsymbol{\sigma}')|^2|}{T(\boldsymbol{\sigma}'\to\boldsymbol{\sigma})|\psi(\boldsymbol{\sigma})|^2|})$;
  (3) Compute the local values of the observables or store the configuration. ;
end

Algorithm 3: Metropolis-Hastings algorithm for VMC

We know that for independent and identically distributed random variables X_i, the variance of a sum of random variables is equal to the sum of the variances, i.e.

$$\mathrm{Var}\Big[\sum_i X_i\Big] = \sum_i \mathrm{Var}\Big[X_i\Big]$$

If the sampling is uncorrelated, we have

$$\mathrm{Var}\left[\frac{1}{N_{MC}}\sum_{i=1}^{N_{MC}} O_{loc}(\boldsymbol{\sigma}_i)\right] = \frac{1}{N_{MC}^2}\mathrm{Var}\Big[\sum_{i=1}^{N_{MC}} O_{loc}(\boldsymbol{\sigma}_i)\Big] \tag{2.24}$$

$$= \frac{1}{N_{MC}^2}\sum_{i=1}^{N_{MC}} \mathrm{Var}\Big[O_{loc}(\boldsymbol{\sigma}_i)\Big] = \frac{\mathrm{Var}\Big[O_{loc}(\boldsymbol{\sigma})\Big]}{N_{MC}} \tag{2.25}$$

The equality in second line is based on the i.i.d. property. Naively applying this formula leads to underestimate of the variance.

- A standard way to solve this problem is by binning analysis. We can reorder the sum such that

$$\frac{1}{N_{MC}}\sum_{i=1}^{N_{MC}} O_{loc}(\boldsymbol{\sigma}_i) = \frac{1}{N_{bin}}\sum_{i=1}^{N_{bin}} O_{loc,i}^{bin}, \qquad O_{loc}^{bin} = \frac{1}{n_b}\sum_{i=1}^{n_b} O_{loc}(\boldsymbol{\sigma}_i) \tag{2.26}$$

The binned local values of the observable $O_{loc,i}^{bin}$ would be uncorrelated if the size of the bins is large enough. Typically, one can plot the variance with respect to the size of the bins. The variance would saturate if the size of the bin is larger than the autocorrelation length, i.e. the maximum length between correlated configuration in

the Markov chain. In practice, one would calculate the observables on the fly while keeping the moving averages and variances.

An alternative way is to only consider samples from Markov chain that are uncorrelated. In principle, we could figure out the autocorrelation length by binning analysis, then we consider the samples that is separated by each other longer than that. In practice, one could simply sample once after N update, where N is the size of the system. The argument is that after N updates, the configuration could be completely different from the previous one. We consider this approach in this thesis.

- Warm-up stage is expensive. However it is necessary to discard a portion of Markov chain sampling, because at the beginning of the Markov chain, the probability of the configuration is not yet the underlying distribution of the wavefunction. One way to get around this is to store the configuration at the end of MC sampling. If the gradient descent only change the wavefunction by a small amount, the configuration stored from previous run should be still in equilibrium. Therefore, we may skip the warm-up process. However, small stepsize in gradient descent does not guarantee small amount of change in wavefunction. We will see how to safely update the wavefunction with SR method in the next section.

- Unlike most Monte Carlo methods, VMC is sign-problem free and gives access to the wavefunction. The error is given by std(E) $= \sqrt{\frac{(2\tau+1)Var(E)}{N_{MC}}}$. Also since variational energy is an upper bound of the ground state energy. We could always give a precise upper bound of the ground state energy.

2.2.4. Stochastic Reconfiguration / Natural Gradient Method

Stochastic gradient descent (SGD) is indisputably simplest but influential algorithm for nonlinear optimization problem. Unlike the success of SGD in neural network learning, SGD for learning quantum amplitude is inefficient if not possible. One major problem of SGD algorithm is that it does not taking into account the geometry of the parameter space.[2] The parameter space is usually curved. That means small change in some parameters may have drastically different output. One should take this into account while moving in the parameter space to find optimal parameters. The use of natural gradient guarantees smooth learning in the parameter space.

In this section, we first discuss the natural gradient [2], i.e. steepest descent method on a Riemannian manifold. Two examples are shown. The first example of natural gradient is the Fisher information metric in stochastic modeling. In the second example, we show SR method can be formulated as natural gradient descent method with Fubini-Study metric for quantum system. In the next section, we derive the SR methods from the physical motivation of imaginary time evolution. This then give the SR a physical meaning.

[2] A way of putting this would be SGD tries to summing a contravaient vector with a covarient vector. To make the formula make sense, we have already assume an Euclidean geometry.

An optimization problem is defined by the parameter space $\mathcal{M} = \{\mathbf{w} \in \mathbb{R}^n\}$ and an objective function $\mathcal{L}(\mathbf{w})$ on it. An implicit assumption is usually assumed here. That is $\mathcal{M}$ is a Euclidean space with which $\mathbf{w}$ is the orthonormal coordinate. The distance is given by,

$$||d\mathbf{w}||^2 = \sum_i (dw^i)^2 \tag{2.27}$$

However, we should consider the general case, where the distance is given by,

$$||d\mathbf{w}||^2 = \sum_{ij} g_{ij}(\mathbf{w}) dw^i dw^j \tag{2.28}$$

where g_{ij} is the Riemannian metric and could be a function of $\mathbf{w}$. The Euclidean distance with orthonormal coordinate is a special case with $g_{ij} = \delta_{ij}$.

The steepest descent direction in a Riemannian space is given by,

$$\tilde{\nabla}\mathcal{L}(\mathbf{w}) = G^{-1}(\mathbf{w})\nabla\mathcal{L}(\mathbf{w}) \tag{2.29}$$

where $G^{-1} = (g^{ij})$ is the inverse matrix of $G = (g_{ij})$. The statement above comes from the definition of steepest descent direction [3]. The steepest descent direction $d\mathbf{w}$ of function $\mathcal{L}(\mathbf{w})$ at $\mathbf{w}$ is defined as the vector that minimize $\mathcal{L}(\mathbf{w} + d\mathbf{w})$ where the vector $d\mathbf{w}$ has a fixed length, i.e. $||d\mathbf{w}||^2 = \epsilon$ and ϵ is a sufficient small constant.

This is a constrained optimization problem, that is

$$\arg\min_{d\mathbf{w}} \mathcal{L}(\mathbf{w} + d\mathbf{w}) \approx \mathcal{L}(\mathbf{w}) + d\mathbf{w} \cdot \nabla\mathcal{L}(\mathbf{w}) \tag{2.30}$$

$$\text{subject to} \qquad ||d\mathbf{w}||^2 = \epsilon \tag{2.31}$$

The problem is solved with Lagrangian method,

$$\frac{\partial}{\partial dw_i}\Big[\mathcal{L}(\mathbf{w}) + d\mathbf{w} \cdot \nabla\mathcal{L}(\mathbf{w}) + \lambda \sum_{jk} dw_j g_{jk} dw_k\Big] = 0 \tag{2.32}$$

$$\nabla\mathcal{L}(\mathbf{w})_i - 2\lambda \sum_j g_{ij} dw_j = 0 \tag{2.33}$$

$$d\mathbf{w} = \frac{1}{2\lambda} G^{-1} \nabla\mathcal{L}(\mathbf{w}) \tag{2.34}$$

where λ is determined by the constraint of the length of vector. In principle, $d\mathbf{w}$ gives the steepest descent direction in the limit of $\epsilon \to 0$. The vector $\tilde{\nabla}\mathcal{L}(\mathbf{w}) = G^{-1}(\mathbf{w})\nabla\mathcal{L}(\mathbf{w})$ is called the natural gradient of $\mathcal{L}$ in the Riemannian space. In the special case of Euclidean and orthonormal cooridnate, the gradient and natural gradient are the same. However, in general, we have the natural gradient descent algorithm,

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta\tilde{\nabla}\mathcal{L}(\mathbf{w}^{(t)}) \tag{2.35}$$

[3] The phrase direction here is abused in a way that the vector is not necessary normalized.

For those who works with proper covariant and contravariant tensor indices. The result is trivial and could be stated as,

$$\partial^i \mathcal{L} = g^{ij} \partial_j \mathcal{L} \tag{2.36}$$

$$w^i \leftarrow w^i - \eta \partial^i \mathcal{L} \tag{2.37}$$

In the following, we show the usage of natural gradient with two different metric in the parameter space.

Example 1: Fisher information metric Consider a machine learning problem in finding an optimal probability distribution P with respect to some loss function $L = L(P)$. We further assume that the "model" $P = \mathrm{P}(x; \mathbf{w})$ is the probability density function of continuous variable[4] x parametrized by $\mathbf{w}$. Gradient descent methods could then be used to minimize the loss function $L(\mathbf{w})$.

A common measure of the distance between two probability distributions is the KL divergence. We would see that by expanding the KL divergence to the second order, we obtain the Fisher information metric. As a result, if one want to apply natural gradient method in stochastic model learning, one need to compute the Fisher information metric. This is the most common form of natural gradient one usually encounter in the context of machine learning.

If we have two probability distributions $p(\mathbf{w})$ and $p(\mathbf{w}')$, where $\mathbf{w}' = \mathbf{w} + d\mathbf{w}$. Therefore, these two probability distributions are very similar. The distance between these two probability distributions is given by the KL divergence.

$$\begin{aligned}
\mathrm{d}(\mathbf{w}, \mathbf{w}') &= \mathrm{KL}(P(x;\mathbf{w}) || P(x;\mathbf{w}')) \\
&= \int P(x;\mathbf{w}) \log \frac{P(x;\mathbf{w})}{P(x;\mathbf{w}')} dx \\
&= \int P(x;\mathbf{w}) \log P(x;\mathbf{w}) dx - \int P(x;\mathbf{w}) \log P(x;\mathbf{w}) dx \\
&\quad - \sum dw_i \int P(x;\mathbf{w}) \partial_{w_i} \log P(x;\mathbf{w}) dx \\
&\quad - \frac{1}{2} \sum dw_i dw_j \int P(x;\mathbf{w}) \partial_{w_i} \partial_{w_j} \log P(x;\mathbf{w}) dx + \Delta(dw^3) \\
&= -\frac{1}{2} \sum dw_i dw_j \int P(x;\mathbf{w}) \partial_{w_i} \partial_{w_j} \log P(x;\mathbf{w}) dx \\
&= \frac{1}{2} \sum dw_i dw_j \int P(x;\mathbf{w}) \Big[\partial_{w_i} \log P(x;\mathbf{w}) \partial_{w_j} \log P(x;\mathbf{w}) \Big] dx
\end{aligned}$$

[4] All the argument below can be generalized to the discrete case.

where we used the properties,

$$\sum dw_i \partial_{w_i} \int P(x;\mathbf{w})dx = 0$$
$$\sum dw_i dw_j \partial_{w_i}\partial_{w_j} \int P(x;\mathbf{w})dx = 0$$

If one consider the symmetrized KL divergence $\mathrm{J}(p,p') = \mathrm{KL}(p||p') + \mathrm{KL}(p'||p)$, we would then have the Fisher information metric,

$$g_{ij}(\mathbf{w}) = \int P(x;\mathbf{w})\Big[\partial_{w_i}\log P(x;\mathbf{w})\partial_{w_j}\log P(x;\mathbf{w})\Big]dx \tag{2.38}$$
$$= \mathbb{E}_{x\sim P}\Big[\partial_{w_i}\log P(x;\mathbf{w})\partial_{w_j}\log P(x;\mathbf{w})\Big] \tag{2.39}$$

Example 2: Fubini-Study metric In terms of quantum mechanics, a common metric in projective Hilbert space is the Fubini-Study metric. Consider two wavefunctions $\psi(\mathbf{w})$ and $\psi(\mathbf{w}')$, where $\mathbf{w}' = \mathbf{w} + d\mathbf{w}$. We can now define the distance between parameters $\mathbf{w}$ and $\mathbf{w}'$ as the angle between these two wavefunctions.

$$\mathrm{d}(\mathbf{w},\mathbf{w}') = \arccos\sqrt{\frac{\langle\psi_{\mathbf{w}}|\psi_{\mathbf{w}'}\rangle\langle\psi_{\mathbf{w}'}|\psi_{\mathbf{w}}\rangle}{\langle\psi_{\mathbf{w}}|\psi_{\mathbf{w}}\rangle\langle\psi_{\mathbf{w}'}|\psi_{\mathbf{w}'}\rangle}} \tag{2.40}$$

gives the quantum angle between two unnormalized wavefunctions.

By substituting the properties,

$$|\psi_{\mathbf{w}'}\rangle = |\psi_{\mathbf{w}}\rangle + \sum_i dw_i \frac{\partial}{\partial w_i}|\psi_{\mathbf{w}}\rangle$$
$$= |\psi_{\mathbf{w}}\rangle + \sum_i dw_i \hat{\Delta}_i |\psi_{\mathbf{w}}\rangle$$

and Taylor expand the denominator, square root, and arccos to the second order. In the end we would have,

$$\mathrm{d}(\mathbf{w},\mathbf{w}')^2 \tag{2.41}$$
$$= \arccos^2\left(\sqrt{\frac{1+\sum_i dw_i\langle\Delta_i\rangle + \sum_i dw_i^*\langle\Delta_i^*\rangle + \sum_{ij} dw_i^* dw_j\langle\Delta_i^*\rangle\langle\Delta_j\rangle}{1+\sum_i dw_i\langle\Delta_i\rangle + \sum_i dw_i^*\langle\Delta_i^*\rangle + \sum_{ij} dw_i^* dw_j\langle\Delta_i^*\Delta_j\rangle}}\right) \tag{2.42}$$
$$\approx \arccos^2\left(\Big[1 - \sum_{ij} dw_i^* dw_j\big(\langle\Delta_i^*\Delta_j\rangle - \langle\Delta_i^*\rangle\langle\Delta_j\rangle\big)\Big]^{1/2}\right) \tag{2.43}$$
$$\approx \sum_{ij} dw_i^* dw_j\big[\langle\Delta_i^*\Delta_j\rangle - \langle\Delta_i^*\rangle\langle\Delta_j\rangle\big] \tag{2.44}$$

Note that we still have the same result if we expend the $|\psi_{\mathbf{w}'}\rangle$ to second order in dw. The Fubini-Study metric is defined as,

$$S_{ij} = \left[\langle \Delta_i^* \Delta_j \rangle - \langle \Delta_i^* \rangle \langle \Delta_j \rangle \right] \tag{2.45}$$

$$= \mathbb{E}_{\alpha \sim P=|\psi|^2} \Big[\partial_{w_i^*} \log \psi^*(\alpha; \mathbf{w}^*) \partial_{w_j} \log \psi(\alpha; \mathbf{w}) \Big] \tag{2.46}$$

$$- \mathbb{E}\Big[\partial_{w_i^*} \log \psi^*(\alpha; \mathbf{w}^*) \Big] \mathbb{E}\Big[\partial_{w_j} \log \psi(\alpha; \mathbf{w}) \Big] \tag{2.47}$$

From algorithm 2, SR method could be interpreted as natural gradient descent with Fubini-Study metric.

Fisher information metric and Fubini-Study metric are of size, $N_w \times N_w$. For large-scale optimization problem, the expectation value could not be evaluated exactly. Therefore, one usually compute the empirical Fisher Information metric or Fubini-Study metric by stochastic sampling. The natural gradient requires the inverse of the metric, which is computational expensive. One could solve the linear equation approximately or exactly with iterative solver. See section 2.2.7. One could also make further approximation based on the structure of the function. For example, assuming statistically independence between group of variables can lead to block structure in the metric. See section 3.5.

Fubini-Study metric is Hermitian. The symmetric real part defines a Riemannian metric and the antisymmetric imaginary part defines a symplectic form on the projective Hilbert space. Fubini-Study metric is also known as the Quantum Geometric Tensor (QGT). Since it defines a measurement of quantum distance, it is considered as a tool to probe quantum phase transition related to the concept of quantum fidelity. [6, 102].

Consider a wavefunction described by $\psi(x; \mathbf{w}) = e^{r(x;\mathbf{w})+i\theta(x;\mathbf{w})}$. The probability for certain state is defined as $P(x; \mathbf{w}) = |\psi(x; \mathbf{w})|^2$. Fubini-Study metric has the form [20],

$$S_{ij} = \frac{1}{4}\mathbb{E}_{x\sim P}\Big[\partial_{w_i^*} \log P(x; \mathbf{w}) \partial_{w_j} \log P(x; \mathbf{w}) \Big] + \mathbb{E}\Big[\partial_{w_i^*} \theta \partial_{w_j} \theta \Big] - \mathbb{E}\Big[\partial_{w_i^*} \theta \Big] \mathbb{E}\Big[\partial_{w_j} \theta \Big] \tag{2.48}$$

$$- i\mathbb{E}\Big[\partial_{w_i^*} \log P(x; \mathbf{w}) \partial_{w_j} \theta - \partial_{w_i^*} \theta \partial_{w_j} \log P(x; \mathbf{w}) \Big] \tag{2.49}$$

In the case when the phase is independent, it is equivalent to four times the classical Fisher Information metric.

2.2.5. Stochastic Reconfiguration / Time Evolution

Stochastic reconfiguration has a physical interpretation. It is a (stochastic) method for solving the imaginary time evolution of the wavefunction in the variational subspace, which is essentially the time-dependent variational principle (TDVP). In this section, we first show the imaginary time evolution as method for solving ground state. We then formulate the imaginary time evolution in the subspace spanned by variational parameters.

Imaginary time evolution is related to the power method in solving extreme eigenvalue problem. The idea of power method is as follows. Assume a positive semi-definite matrix

M has one dominating eigenvalue, λ_e. By iteratively applying matrix vector multiplication $M^{N_{iter}}v$, we can "project" out the unnecessary part of the vector v, that is all the coefficient in eigenbasis except the one with dominating eigenvalue will be suppressed exponentially $(\lambda_i/\lambda_e)^{N_{iter}}$, $\forall i \neq e$. We end up with the eigenvector with extreme eigenvalue.

Unlike power method, where the projection is carried though discrete steps, imaginary time evolution is the continuous relaxation with differential equation, where the steady state solution is the ground state.

$$\frac{d\left|\psi\right\rangle}{d\tau} = -\hat{H}\left|\psi\right\rangle \tag{2.50}$$

$$\left|\psi(\tau)\right\rangle = e^{-\hat{H}\tau}\left|\psi(0)\right\rangle = \lim_{N\to\infty}(1-\frac{\tau}{N}\hat{H})^N\left|\psi(0)\right\rangle$$

That is all the coefficients in the eigenbasis except the ground state are again suppressed exponentially, when we keep the vector normalized.

$$\left|\psi(\tau)\right\rangle = e^{-E_0\tau}\Big[c_0\left|E_0\right\rangle + e^{-(E_1-E_0)\tau}c_1\left|E_1\right\rangle + \dots\Big]$$

Power method is similar to the result of solving the ODE with discretization in time, but not the same [5]. We could not directly work with both two formulation since the physical dimension is exponentially large. However, this is more than a tautology of the problem. We will see by formulating in continuous setting, we can derive the same SR equation in the parameter space which is different from the ODE in the physical space.

We now derive the stochastic reconfiguration formula by solving the linear equation from imaginary time evolution in the subspace of the variational parameters. First, we discretize the imaginary time evolution of state by a small time step $d\tau$, which gives

$$e^{-d\tau\hat{H}}\left|\psi\right\rangle = (1-d\tau\hat{H})\left|\psi\right\rangle + \mathcal{O}(d\tau^2). \tag{2.51}$$

Note that the state is not normalized any more. To formulate the equation in parameter space, we want to find the parameters $\mathbf{w}'$ close to $\mathbf{w}$ that represents the imaginary time evolved wavefunction.

$$dw_0\left|\psi\right\rangle + \sum_{j\neq 0} dw_j \frac{\partial}{\partial w_j}\left|\psi\right\rangle \approx \left|\psi\right\rangle - d\tau\hat{H}\left|\psi\right\rangle \tag{2.52}$$

We find the coefficients dw_j with the basis $\{\left|\psi\right\rangle, \frac{\partial}{\partial w_j}\left|\psi\right\rangle\}$ to represent the wavefunction. In other word, we solve the above equation in the subspace $\{\left|\psi\right\rangle, \frac{\partial}{\partial w_j}\left|\psi\right\rangle\}$. This leads to equations,

$$dw_0\left\langle\psi|\psi\right\rangle + \sum_{j\neq 0} dw_j\left\langle\psi|\partial w_j\psi\right\rangle = \left\langle\psi|\psi\right\rangle - d\tau\left\langle\psi|\hat{H}|\psi\right\rangle \tag{2.53}$$

$$dw_0\left\langle\partial_{w_i}\psi|\psi\right\rangle + \sum_{j\neq 0}\left\langle\partial_{w_i}\psi|\partial w_j\psi\right\rangle dw_j = \left\langle\partial_{w_i}\psi|\psi\right\rangle - \left\langle\partial_{w_i}\psi|\hat{H}|\psi\right\rangle \tag{2.54}$$

[5] One does not need to shift the spectrum to guarantee the positiveness of the matrix. We can make time step τ very small, while N_{iter} could only be integer.

Substitute the dw_0 in the second equations by expression from first equation. We have,

$$\sum_j \Big[\langle \Delta_i^* \rangle \langle \Delta_j \rangle - \langle \Delta_i^* \Delta_j \rangle \Big] \frac{dw_j}{d\tau} = \Big[\langle \Delta_i^* H \rangle - \langle \Delta_i^* \rangle \langle H \rangle \Big] \tag{2.55}$$

We again obtain the formula of stochastic reconfiguration. In addition, we can easily obtain the formula for real time evolution in t by replacing τ with it in the equation.

$$\dot{\mathbf{w}} = -S^{-1}\mathbf{f} \qquad \text{Imaginary time evolution} \tag{2.56}$$

$$\dot{\mathbf{w}} = -iS^{-1}\mathbf{f} \qquad \text{Real time evolution} \tag{2.57}$$

This real time evolution formulation, also known as t-VMC, has been applied to boson [8, 7], spin system [10], fermionic system [46].

The derivation above is essentially the time-dependent variational principle (TDVP) principle. We brieftly review TDVP below which based on normalized wavefunction.

$$\tilde{\psi}(\tau) = \frac{\psi(\tau)}{\sqrt{\langle \psi(\tau) | \psi(\tau) \rangle}}$$

Substituting the normalized wavefunction inside the equation (2.50), we have

$$\frac{d}{d\tau}\tilde{\psi}(\tau) = -(\hat{H} - \langle \hat{H} \rangle)\tilde{\psi}(\tau).$$

In terms of the variational wavefunction with parameters $\mathbf{w}$, this become

$$\sum_i \frac{dw_i}{d\tau}\frac{\partial}{\partial w_i}\tilde{\psi}(\tau) \approx -(\hat{H} - \langle \hat{H} \rangle)\tilde{\psi}(\tau). \tag{2.58}$$

Minimizing the difference measured in L_2 norm,

$$\min_{\dot{w}_i} \Big\| \sum_i \frac{dw_i}{d\tau}\frac{\partial}{\partial w_i}\tilde{\psi}(\tau) + (\hat{H} - \langle \hat{H} \rangle)\tilde{\psi}(\tau) \Big\| \tag{2.59}$$

gives the best approximate imaginary time evolution in the variational subspace,

$$\sum_j \langle \partial_{w_i^*}\tilde{\psi}(\tau) | \partial_{w_j}\tilde{\psi}(\tau) \rangle \, \dot{w}_j = -\langle \partial_{w_i^*}\tilde{\psi}(\tau) | \hat{H} - \langle \hat{H} \rangle | \tilde{\psi}(\tau) \rangle \, . \tag{2.60}$$

Equation (2.60) is known as the TDVP equation, which we usually see in other context, e.g. TDVP-MPS. It recover the form of SR for unnormalized wavefunction. We could say SR is the stochastic method for TDVP.

2.2.6. Damping/Regularization and Stepsize

In practice, it is usually consider a good practice to add a damping to the empirical metric. That is we work with $S + \lambda I$ instead of S, if we are not considering solving the exact time evolution but only finding the ground state. It is known as the Tikhonov regularization or Tikhonov damping [53].

Damping (Regularization) in Newton's method The damping term has various motivations and interpretations. One motivation comes from the instability of Newton's method, which solves the original problem of finding the optimal update with second-order approximation, i.e.:

$$M(d\mathbf{w}) = E(\mathbf{w}_0) + \nabla E(\mathbf{w}_0)^T d\mathbf{w} + \frac{1}{2} d\mathbf{w}^T \mathcal{H} d\mathbf{w}. \tag{2.61}$$

The $\mathcal{H}$ is the Hessian of the objective function E. This method with the corresponding optimal update $d\mathbf{w}_{opt} = \mathcal{H}^{-1}\nabla E(\mathbf{w}_0)$ is known as the Newton's method. Note that there is no stepsize in the formula.

The update proposed is usually unstable. One reason might be the Hessian matrix is ill-conditioned. This usually means some very large and poor update in the direction corresponds to small eigenvalues. Naively scaling the update $d\mathbf{w}_{opt} * \gamma$ with $\gamma < 1$ will lead to small update for all directions. It is considered better to add a damping term λI to improve the condition number by increasing the singular values, which corresponds to remove the large and poor update.

The failure could be understood as that the optimal update is beyond the region where quadratic approximation could applies. We should modify the approximated problem to restrict the update proposal. There are two ways to do this: (1) Constrained optimization method known as the Trust region method. (2) Unconstrained optimization with penalty for large stepsize. Both methods are closely related.

Trust region method considers a radius r and the region $R = \{x : \|x\| \leq r\}$, which we believe the second-order approximation is reliable, and the constrained optimization is given as,

$$\arg\min_{d\mathbf{w}: d\mathbf{w} \in R} M(d\mathbf{w}). \tag{2.62}$$

The penalty-based unconstrained optimization is the damping method. It is defined as

$$\arg\min_{d\mathbf{w}} M(d\mathbf{w}) + \frac{\lambda}{2}\|d\mathbf{w}\|^2 \tag{2.63}$$

and gives exactly the damping term if Euclidean distance is considered.

In both methods, optimal choice for r and λ might change throughout the optimization process. One recommended adjustment heuristic is the Levenberg-Marquardt algorithm.

The third reason damping term appears is because of regularization. If we add an l_2 regularization term, i.e. $\|\mathbf{w}\|^2$ to the objective function $E(\mathbf{w})$, we will have $\mathcal{H} + \lambda I$ instead

of $\mathcal{H}$ in the second-order approximation. In this case, the difference is that the ∇E is also changed.

By observing that large damping bring the update back to gradient descent direction, we could also interpret damping as mixing the gradient descent and natural gradient descent.

Damping (Regularization) in SR Natural gradient defined the optimal direction for infinitesimally small step. However, in practice, the natural gradient optimization considers update with finite stepsize like,

$$\mathbf{w}^{(k+1)} \leftarrow \mathbf{w}^{(k)} - \gamma_k \tilde{\nabla} E(\mathbf{w}^k). \tag{2.64}$$

The design of scheduling of stepsize is usually based on heuristic. We discuss it in next part. Aside from the difficulty of choosing suitable stepsize, we consider the natural gradient as the optimal update from the quadratic form,

$$M(d\mathbf{w}) = E(\mathbf{w}_0) + \nabla E(\mathbf{w}_0)^T d\mathbf{w} + \frac{1}{2\gamma} d\mathbf{w}^T S d\mathbf{w}. \tag{2.65}$$

We could see this as a different second-order approximation to $E(\mathbf{w}_0 + d\mathbf{w})$. Indeed, in many machine learning setting, the Hessian and the metric are often related or share the same form [65]. The optimal update with finite stepsize fail whenever $M(d\mathbf{w})$ fails to be a good local approximation to $E(\mathbf{w}_0 + d\mathbf{w})$. By making this connection, all the arguments for Newton's method could apply here.

The damping term is for improving condition number, which has another important effect of improving the convergence speed in CG iterative solver. It is also understood as penalty term to restrict the updates, the l_2 regularization and the mixing of gradient and natural gradient descent.

Stepsize with SR It is often tricky to design a good scheduling for stepsize with natural gradient. Adding a damping term might also affect the optimal scheduling. Non-adaptive scheduling like $\gamma_k = c/k$ for some constant c might work. In [26], they suggest a non-adaptive scheduling as $\gamma_k = c/\sqrt{k}$ with large damping in the beginning of VMC optimization for neural network quantum state. They also point out that it also works with fixed stepsize while having a small damping ranging from 10^{-4} to 10^{-8} for several runs. More complicated scheduling is described in [63].

We suggest to adopt the scheduling of stepsize with trust region argument, which suggest to measure the step in Fubini-Study norm and determine the stepsize by

$$\min(\delta_0, \sqrt{\frac{\delta}{d\theta^T g d\theta}}).$$

δ_0 and δ are hyperparameters to tune and could be kept constant or combined with annealing scheduling.

2.2.7. Computational Complexity

Expectation value of observable: For sparse operators, the expectation value could be compute at the cost of $\mathcal{O}(N_{MC}NC_{eval})$, where N is the system size, N_{MC} the number of independent Monte Carlo samples and C_{eval} the cost of evaluation of the amplitude.

We first run the MCMC to obtain N_{MC} independent MC samples. This would cost about $\mathcal{O}(N_{MC}NC_{eval})$, where we assume the auto-correlation length is of order N. After having N_{MC} Monte Carlo samples, we could compute the expectation of the sparse operator with the cost $\mathcal{O}(N_{MC}NC_{eval})$, where the N now comes from the non-zero elements per row of the operator.

For the neural network quantum state we consider in this thesis, the C_{eval} is of $\mathcal{O}(N^2)$ from the matrix-vector multiplication. This gives the overall complexity of $\mathcal{O}(N_{MC}N^3)$, which is cubic to the system size. Note that this is independent of the dimension of the problems. As a result, VMC has advantages for computing higher dimensional problem.

First-order methods: With the independent MC samples given, we can computer the derivative with the cost $\mathcal{O}(N_{MC}C_{grad})$. For the neural network quantum state, the cost for computing the derivative C_{grad} is about the same as evaluation of amplitude C_{eval}.

For first order method, the computational cost is dominated by the generation of independent MC sample and the computation of the local energy.

Second-order methods: For large number of parameters, the cost for second-order methods is bound by the cost of solving the system of linear equations.

The metric S is of the size $N_w \times N_w$. Explicit inversion would cost $\approx \mathcal{O}(N_w^3)$. Iterative solver with matrix explicit formed would still cost $\approx \mathcal{O}(N_w^2)$. This limit the number of variational parameters to be atmost $\mathcal{O}(10^4)$. To exploit the expression with larger number of parameters, different approaches were developed in both physics and machine learning community [6]. In the following, we review different approaches based on generic or specific variational functions.

General methods for all variational functions:

- **Iterative solver**: Eric Neuscamman et al. [61] consider the Conjugate Gradient (CG) method. Giuseppe Carleo et al. [9] consider the MINRES-QLP mthod.

  The idea is based on that we are always approximating the metric by sampling. Therefore, the metric matrix would be low rank. One should not form the matrix explicitly, but should store the unaggregated derivative as M, which is then of size $N_{MC} \times N_w$. The linear equations could be solved with iterative solvers and the matrix vector multiplication defined as $M^T(Mf)$. The cost is reduced to only $\approx \mathcal{O}(N_w N_{MC})$. The cost of storage and computation is linear with respect to N_w.

[6] For review article see [65]. However, it does not include methods recent developed. We give a brief review of a new method named K-FAC [54] in section 3.4 and 3.5.

This is the main approach adopted in this thesis.

- **Diagonal approximation of the metric** Adam, RMSprop, Adagrad. No better than well-tuned SGD, SGD with momentum.
- **HF, Krylov Subspace method**

Methods designed for specific variational functions:

- **TDVP-MPS** The TDVP algorithm in the context of matrix product state (MPS) is a deterministic algorithm that solve the TDVP equation by exploiting the gauge freedom in matrix product states [33, 34]. It not only could scale up to $> 10^6$ number of parameters, but it solve it exactly. However, the application to system of dimension $d \geq 2$ is limited because of the exponential growing of bond dimension in MPS.
- **K-FAC, TONGA, KFRA** These are method developed for neural network to approximate the empirical Fisher information metric. K-FAC: Comparing to iterative solver, e.g. CG, it works with the approximate metric and inverse metric by tensor decomposition. The most crucial advantage of this method is it can keep a moving average of the metric. It reuse the information from sampling in previous updates.

  One disadvantage of iterative solver is that with too many sampling, the cost $\approx \mathcal{O}(N_w N_{MC})$ would be too expensive even for truncated iteration. But without enough sampling, the approximation to the metric is too crude such that we might not get the benefit of using second-order methods.

  It is clear that by keeping the history of the metric, we can work with smaller number of sampling per update, which is crucial to scale up to large number of parameters.

We give a comparison for the difference in VMC, supervised learning and reinforcement learning in section 3.4

2.2.8. Fidelity and Entanglement

We showed that VMC could compute sparse Hermitian operator. In this section, we show that we can also compute the fidelity between two wavefunctions and the Renyi-2 entropy by using the replica-trick. The formulas are similar that we need to sample from independent distribution, which could be easily implemented with two independent MC sampling.

Fidelity (overlap) between two pure states, ψ_1, ψ_2, is defined as,

$$\mathcal{F}(|\psi_1\rangle, |\psi_2\rangle) = \sqrt{\frac{\langle\psi_1|\psi_2\rangle \langle\psi_2|\psi_1\rangle}{\langle\psi_1|\psi_1\rangle \langle\psi_2|\psi_2\rangle}} \tag{2.66}$$

Fidelity is a quantity we can check for the closeness of two wavefunctions we obtained. It also act as a reasonable objective in supervised learning.

We can compute the inner products between two wavefunctions inside the square root by sampling from these two wavefunctions at the same time.

$$\frac{\langle\psi_1|\psi_2\rangle\langle\psi_2|\psi_1\rangle}{\langle\psi_1|\psi_1\rangle\langle\psi_2|\psi_2\rangle} = \Big[\sum_{\boldsymbol{\sigma}} \frac{|\langle\boldsymbol{\sigma}|\psi_1\rangle|^2}{\langle\psi_1|\psi_1\rangle}\frac{\langle\boldsymbol{\sigma}|\psi_2\rangle}{\langle\boldsymbol{\sigma}|\psi_1\rangle}\Big]\Big[\sum_{\boldsymbol{\sigma}'} \frac{|\langle\boldsymbol{\sigma}'|\psi_2\rangle|^2}{\langle\psi_2|\psi_2\rangle}\frac{\langle\boldsymbol{\sigma}'|\psi_1\rangle}{\langle\boldsymbol{\sigma}'|\psi_2\rangle}\Big] \tag{2.67}$$

$$= \sum_{\boldsymbol{\sigma}}\sum_{\boldsymbol{\sigma}'} P_1(\boldsymbol{\sigma})P_2(\boldsymbol{\sigma}')\frac{\langle\boldsymbol{\sigma}'|\psi_1\rangle}{\langle\boldsymbol{\sigma}|\psi_1\rangle}\frac{\langle\boldsymbol{\sigma}|\psi_2\rangle}{\langle\boldsymbol{\sigma}'|\psi_2\rangle} \tag{2.68}$$

where $\boldsymbol{\sigma}, \boldsymbol{\sigma}'$ are sample from P_1, P_2.

Renyi-2 entropy. The Renyi entropy is defined as,

$$S_\alpha(|\psi\rangle) = \frac{1}{1-\alpha}\ln \mathrm{Tr}\rho_A^\alpha \tag{2.69}$$

, where ρ_A is the reduced density matrix of subsystem A from tracing out subsystem B of a bipartite system. The Renyi entropy recover the von Neumann entropy in the limit $\alpha \to 1$.

We will show that we could actually calculation the Renyi-2 entropy by introducing a SWAP gate [37, 98].

$$S_2 = -\ln \mathrm{Tr}(\rho_A^2) = -\ln\left(\langle \mathrm{SWAP}\rangle\right) \tag{2.70}$$

Consider a system to be divide into two regions A and B with complete basis $|\alpha\rangle$ and $|\beta\rangle$. The wavefunction in these basis is given by $\psi = \sum_{\alpha,\beta}|\alpha\rangle|\beta\rangle\langle\alpha,\beta|\psi\rangle$. The Swap operator act on the tensor product of two wavefunctions and is defined as

$$\mathrm{SWAP}_A\Big(\sum_{\alpha_1,\beta_1}|\alpha_1\rangle|\beta_1\rangle\langle\alpha_1,\beta_1|\psi\rangle\Big)\otimes\Big(\sum_{\alpha_2,\beta_2}|\alpha_2\rangle|\beta_2\rangle\langle\alpha_2,\beta_2|\phi\rangle\Big) \tag{2.71}$$

$$= \sum_{\alpha_1,\beta_1}|\alpha_2\rangle|\beta_1\rangle\langle\alpha_1,\beta_1|\psi\rangle\sum_{\alpha_2,\beta_2}|\alpha_1\rangle|\beta_2\rangle\langle\alpha_2,\beta_2|\phi\rangle \tag{2.72}$$

Now we have,

$$\frac{\langle\psi\otimes\psi|\mathrm{SWAP}_A|\psi\otimes\psi\rangle}{\langle\psi|\psi\rangle\langle\psi|\psi\rangle} = \sum_{\alpha_1,\beta_1,\alpha_2,\beta_2} C^*_{\alpha_2,\beta_1}C^*_{\alpha_1,\beta_2}C_{\alpha_1,\beta_1}C_{\alpha_2,\beta_2} \tag{2.73}$$

$$= \sum_{\alpha_1,\alpha_2,}(\rho_A)_{\alpha_1,\alpha_2}(\rho_A)_{\alpha_2,\alpha_1} \tag{2.74}$$

$$= \mathrm{Tr}(\rho_A^2) \tag{2.75}$$

where the $C_{\alpha,\beta}$ is the normalized probability amplitude. To calculate this quantities numerically, we have to compute,

$$\frac{\langle\psi\otimes\psi|\text{SWAP}_A|\psi\otimes\psi\rangle}{\langle\psi|\psi\rangle\langle\psi|\psi\rangle} = \sum_{\alpha_1,\beta_1,\alpha_2,\beta_2} \frac{\langle\psi|\alpha_2,\beta_1\rangle\langle\psi|\alpha_1,\beta_2\rangle\langle\alpha_1,\beta_1|\psi\rangle\langle\alpha_2,\beta_2|\psi\rangle}{\langle\psi|\psi\rangle\langle\psi|\psi\rangle} \tag{2.76}$$

$$= \Big[\sum_{\alpha_1,\beta_1} \frac{|\langle\alpha_1,\beta_1|\psi\rangle|^2}{\langle\psi|\psi\rangle}\frac{\langle\psi|\alpha_2,\beta_1\rangle}{\langle\psi|\alpha_1,\beta_1\rangle}\Big]\Big[\sum_{\alpha_2,\beta_2} \frac{|\langle\alpha_2,\beta_2|\psi\rangle|^2}{\langle\psi|\psi\rangle}\frac{\langle\psi|\alpha_1,\beta_2\rangle}{\langle\psi|\alpha_2,\beta_2\rangle}\Big] \tag{2.77}$$

$$= \sum_{\boldsymbol{\sigma}}\sum_{\boldsymbol{\sigma}'} P(\boldsymbol{\sigma})P(\boldsymbol{\sigma}')\frac{\langle\tilde{\boldsymbol{\sigma}}|\psi\rangle}{\langle\boldsymbol{\sigma}|\psi\rangle}\frac{\langle\tilde{\boldsymbol{\sigma}}'|\psi\rangle}{\langle\boldsymbol{\sigma}'|\psi\rangle} \tag{2.78}$$

where $\boldsymbol{\sigma} = (\alpha_1,\beta_1), \boldsymbol{\sigma}' = (\alpha_2,\beta_2), \tilde{\boldsymbol{\sigma}} = (\alpha_2,\beta_1), \tilde{\boldsymbol{\sigma}}' = (\alpha_1,\beta_2)$. The above calculation is known as the replica-trick.

2.3. Trial wavefunctions

As from equation (2.2), the function form, which comes from our prior knowledge, encodes the underlying physical properties of the state.

In this section, we review the wavefunctions that are used most in spin system and are related to Restricted Boltzmann Machine (RBM). They are EPS/CPS, SBS [26] and MPS [12]. We then introduce the RBM as wavefunctions. In the end, we give the formulation for neural network wavefunction as a generalization of RBM wavefunction.

2.3.1. Matrix Product States (MPS)

MPS is defined as,

$$\psi(\boldsymbol{\sigma}) = \text{Tr}\Big(\prod_{i=1}^{N} A_i^{\sigma_i}\Big), \tag{2.79}$$

where each $A_i^{\sigma_i}$ at site i with spin σ_i is independent matrix. The largest rank of the matrices is denoted as D, the bond dimension of the MPS. The tensor $A_i^{\sigma_i}$ itself is referred to as the core tensor.

Computational Complexity: Memory : $\mathcal{O}(NdD^2)$. Evaluating amplitude : OBC $\mathcal{O}(ND^2)$ PBC $\mathcal{O}(ND^3)$

2.3.2. Entangled Plaquette States (EPS), Correlator Product State (CPS)

EPS/CPS is defined as

$$\psi_w(\boldsymbol{\sigma}) = \prod_{p=1}^{P} C_p^{\boldsymbol{\sigma}_p}. \tag{2.80}$$

The wavefunction is constructed by the product of amplitude on P plaquette, where each plaquette is a subset of the lattice $\boldsymbol{\sigma}_p \subset \boldsymbol{\sigma}$, and $C_p^{\boldsymbol{\sigma}_p}$ is a general vector in the Hilbert space corresponding to the plaquette p. As a result, the size of plaquette is usually chosen to be small since the coefficient in this subspace again grows exponentially [25, 57, 11]. A typical example is the Jastrow wavefunction for spin system, which could be viewed as an two site correlator product state,

$$\psi_w(\boldsymbol{\sigma}) = \prod_{i<j} f_{ij}(\sigma_i, \sigma_j).$$

Another format which is useful in some context, is to write the correlator as diagonal operator in computational basis,

$$\langle \sigma_i, \sigma_j, \dots | \hat{C}^{\sigma_i,\sigma_j,\cdots} | \sigma_{i'}, \sigma_{j'}, \dots \rangle = \delta_{\sigma_i,\sigma_{i'}} \delta_{\sigma_j,\sigma_{j'}} \dots C^{\sigma_i,\sigma_j,\cdots} \tag{2.81}$$

The EPS/CPS is defined with a reference state $|\Phi\rangle$,

$$|\Psi\rangle = \prod_{p=1}^{P} \hat{C}_p^{\boldsymbol{\sigma}_p} |\Phi\rangle \tag{2.82}$$

A common choice of reference state is the combination of all product states $|\Phi\rangle = \sum_{\boldsymbol{\sigma}} |\boldsymbol{\sigma}\rangle$, one recover the original definition, i.e. equation (2.80). The operator representation is useful in fermionic system, where one could take the free fermion as the reference state. In fact, it is equivalent to Jastrow-Slater wavefunctions. The Jastrow factor is the same as the correlator except that it is parameterized in exponential form. The Jastrow-Slater wavefunction for fermion system is obtained by applying the Jastrow factor on the reference state, which is a Slater determinant of some orbitals.

$$|\Psi\rangle = \hat{J} |\Phi_D\rangle \tag{2.83}$$

$$\hat{J} = \exp(c + \sum_i c_i \hat{n}_i + \sum_{i,j} c_{i,j} \hat{n}_i \hat{n}_j + \cdots) \tag{2.84}$$

$$|\Phi_D\rangle = \det[\phi_1 \phi_2 \cdots \phi_N] \tag{2.85}$$

The term EPS and CPS is used interchangeably. In many context, EPS is based on a short-range plaqquete, and hence the term Entangled Plaquette States. CPS is with long-range correlators but it could also be with local correlators or plaquettes. Therefore, we simple refer them as EPS/CPS.

Physical argument:

- EPS/CPS without overlapping plaquettes describe the many-body states in a mean-field fashion, while correlation is included once the plaquettes overlapped.
- EPS/CPS associate variational degrees of freedom directly with correlations between groups of sites.

- EPS/CPS are complete in the limit when the size of plaquette/correlator become the system size.
- EPS/CPS contains Laughlin wavefunction, Toric code, RVB wavefunction.
- EPS/CPS with local correlators satisfy area law. EPS/CPS with long-range correlators may give volume law entanglement.

Computational Complexity: The state can be obtain stochastically and non-stochastically.

Example: We give two example of EPS/CPS expressing topological order state.

Ground State of 2d Toric Code Model

$$C_{\square}^{\sigma_i,\sigma_j,\sigma_k,\sigma_l} = \begin{cases} 1, & \text{if } \sigma_i+\sigma_j+\sigma_k+\sigma_l = 0 \mod 2 \\ 0, & \text{if } \sigma_i+\sigma_j+\sigma_k+\sigma_l = 1 \mod 2 \end{cases} \tag{2.86}$$

For generalization in higher dimension, see [38].

Laughlin wavefunction is proposed to describe fractional quantum Hall effect. At filling fraction $1/m$, the wavefunction is given as,

$$\psi(z_1,\ldots,z_N) = \Big[\prod_{\lambda<\mu}^{N}(z_\lambda - z_\mu)^m\Big]\prod_i e^{-\alpha|z_i|^2} \tag{2.87}$$

If we restricted the wavefunction on a lattice of L sites and associate each z_i the occupation number σ_i, then we would have

$$|\psi\rangle = \sum_{\boldsymbol{\sigma}}\prod_i C_1^{\sigma_i}\prod_{i<j} C_2^{\sigma_i,\sigma_j}\hat{P}_N\,|\sigma_1,\ldots,\sigma_L\rangle \tag{2.88}$$

where

$$C_1 = \begin{pmatrix} 1 \\ e^{-\alpha|z_i|^2} \end{pmatrix} \tag{2.89}$$

$$C_2 = \begin{pmatrix} 1 & 1 \\ 1 & (z_i - z_j)^m \end{pmatrix} \tag{2.90}$$

Mapping between MPS and EPS/CPS We give a brief comment on the connection between MPS, PEPS and EPS/CPS. Following [69], we could view MPS and PEPS as EPS with auxiliary sites, and also we can rewrite EPS as TNS with copy gate.

2.3.3. String-Bond States (SBS)

String-Bond States are usually defined in $d \geq 2$ systems.

$$\psi_w(\boldsymbol{\sigma}) = \prod_m \psi^m_{MPS}(\boldsymbol{\sigma}_m) \tag{2.91}$$

$$= \prod_m \mathrm{Tr}\Big(\prod_{i=1}^{N_m} A^{\sigma_i}_{m,i}\Big) \tag{2.92}$$

To avoid the exponential growth of dimension with the size of plaquette and to retain the flexibility of the wavefunction, one can combine the idea of EPS/CPS, and MPS. By parametrizing the strings $\boldsymbol{\sigma}_m \subset \boldsymbol{\sigma}$ that cover the lattice with MPS and construct the wavefunction by the product of the amplitude of each string [76, 78].

String-bond states are complete. The completeness is in the sense that one could have a long snail-like string covering the lattice, and because MPS are complete with large enough bond dimension, SBS are therefore complete.

In general, the core tensors in MPS are not commutable. The order of how the string goes through the lattice matters. It is argued in the paper[78], that the strings should reflect the geometry of the system in a way that strings first goes through sites that are closely coupled by the Hamiltonian.

Physical argument: One can view SBS as subset of EPS/CPS, where the plaquettes still cover the physical lattice but are parametrized in MPS form. Another way to view this is by considering the MPS in EPS/CPS form, which introduces the local auxiliary sites. SBS are then an factorized form of PEPS, where the plaquettes are furthere decomposed as smaller plaquettes. Therefore, SBS form a subset of PEPS. One should note that, like in EPS/CPS, it is in principle possible to construct SBS with volume law entanglement. If the string goes through the lattice in some random patterns, it is possible that bipartite will cut through bond linking to every other sites in the other part. For detail, see arguments in [16]
Computational complexity: For evaluating each string $\psi_w(\boldsymbol{\sigma})$, the computational cost is $\mathcal{O}(MND^2)$ ($\mathcal{O}(MND^3)$ for PBC) , where M is the number of string and D the bond dimension. This is favorably cheap comparing to PEPS. Note that comparing to snake DMRG, the bond dimension D is much smaller.

2.3.4. Restricted Boltzmann Machine Quantum State (RBMQS)

A Restricted Boltzmann Machine is defined as,

$$\Psi_{RBM}(\boldsymbol{\sigma};\mathcal{W}) = \sum_{\{h_i\}} e^{\sum_j a_j\sigma_j + \sum_i b_i h_i + \sum_{ij} W_{ij} h_i \sigma_j} \tag{2.93}$$

$$= e^{\sum_j a_j\sigma_j} \times \Big[\sum_{\{h_i\}} e^{\sum_i h_i(b_i + \sum_j W_{ij}\sigma_j)}\Big] \tag{2.94}$$

$$= e^{\sum_j a_j\sigma_j} \times \prod_{i=1}^{N} F_i(\boldsymbol{\sigma}) \tag{2.95}$$

where

$$F_i(\boldsymbol{\sigma}) = \begin{cases} 2\cosh[b_i + \sum_j W_{ij}\sigma_j], & \text{if } h_i = \pm 1 \\ 1 + \exp[b_i + \sum_j W_{ij}\sigma_j], & \text{if } h_i = 0, 1 \end{cases} \tag{2.96}$$

Just like any other machine learning algorithm, RBM is used to learn the probability distribution, which means the function value is real and always equal or larger than zero. Giuseppe Carleo et al. [9] proposed to represent probability amplitude by RBM by generalizing the parameters $\mathcal{W} = \{\boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b}\}$ to complex numbers.

In machine learning community, the convention is usually with $h_i, \sigma_j \in \{0, 1\}$, while in the original work of RBMQS, it is chosen that $h_i, \sigma_j \in \{-1, 1\}$. This ensures the spin-flip symmetry. By setting the one-body terms to zero $a_j = 0$, we have an invariant wavefunction with respect to total spin-flip. However, in this thesis, we choose the $\{0, 1\}$ convention.

Restricted Boltzmann Machine as Entangled Plaquettte States/Correlator Product States: The wavefunction is composed of two part, the one-body term and the plaquette term.

$$\Psi_{RBM}(\boldsymbol{\sigma};\mathcal{W}) = \prod_j e^{a_j\sigma_j} \prod_{i=1}^{N} F_i(\boldsymbol{\sigma}) \tag{2.97}$$

$$= \prod_j C_1^{\sigma_j} \prod_i C_{k_i}^{\boldsymbol{\sigma}_i} \tag{2.98}$$

The order of the i-th plaquette is denoted by k_i, which is determined by the non-zero connection W_{ij}. So, short-range RBMQS are EPS/CPS. Strictly speaking, RBMQS are not exactly EPS/CPS if one defined EPS/CPS as product of exact vectors in subspaces. Each of the F_i does not parametrize the whole subspace. Short-range RBMQS is only a subset of EPS/CPS. In fact, the way it parametrizes the subspaces is equivalent to MPS in SBS, [26].

Restricted Boltzmann Machine as String Bond State: Aside from the one-body term, the rest of the wavefunction is given by [7],

$$\Psi_{RBM}(\boldsymbol{\sigma};\mathcal{W}) \sim \prod_{i=1}^{N} F_i(\boldsymbol{\sigma}) \tag{2.99}$$

$$\propto \prod_i (\exp[-b_i - \sum_j W_{ij}\sigma_j] + \exp[b_i + \sum_j W_{ij}\sigma_j]) \tag{2.100}$$

$$\propto \prod_i \mathrm{Tr} \begin{pmatrix} e^{-b_i - \sum_j W_{ij}\sigma_j} & 0 \\ 0 & e^{b_i + \sum_j W_{ij}\sigma_j} \end{pmatrix} \tag{2.101}$$

$$\propto \prod_i \mathrm{Tr}\Big(\prod_{j\in I} A_{i,j}^{\sigma_j} \Big) \tag{2.102}$$

where

$$A_{i,j}^{\sigma_j} = \begin{pmatrix} e^{-b_i/N - W_{ij}\sigma_j} & 0 \\ 0 & e^{b_i/N + W_{ij}\sigma_j} \end{pmatrix} \tag{2.103}$$

and I being the set of indices in i-th plaquette.

There are several differences between traditional SBS and RBM. Note the for RBM the core tensor in the MPS parametrizing the plaquette has bond dimension 2, and only diagonal elements in the matrices. We denote the two free parameters as $e^{b_i/N} = \beta$, $e^{W_{ij}} = \omega$. This gives

$$A_{i,j}^{\sigma_j=-1} = \begin{pmatrix} \beta^{-1}\omega & 0 \\ 0 & \beta\omega^{-1} \end{pmatrix}, \qquad A_{i,j}^{\sigma_j=1} = \begin{pmatrix} \beta^{-1}\omega^{-1} & 0 \\ 0 & \beta\omega \end{pmatrix}. \tag{2.104}$$

Following the same derivation for $\{0,1\}$ convention, we have instead,

$$A_{i,j}^{\sigma_j=0} = \begin{pmatrix} 1 & 0 \\ 0 & \beta \end{pmatrix}, \qquad A_{i,j}^{\sigma_j=1} = \begin{pmatrix} 1 & 0 \\ 0 & \beta\omega \end{pmatrix} \tag{2.105}$$

Again, this again shows a different parametrization for diagonal matrix product state. For each core tensor, the degree of freedom is only 2. Therefore, RBM is a subset of SBS with special parametrization.

The diagonal core tensor implies there is no fixed ordering for the MPS on each string. Strings in traditional SBS are shorter and do not cover the full lattice. In [26], Glasser et al. name these SBS covering the full lattice with only diagonal element in MPS as non-local dSBS. For SBS covering the full lattice with general MPS, it is named as non-local SBS. They further generalized the wavefunction to bond dimension $D = 3$, with matrices,

$$A_{i,j}^{\sigma_j} = \begin{pmatrix} a_{i,j}^{\sigma_j} & 0 & 0 \\ 0 & b_{i,j}^{\sigma_j} & 0 \\ 0 & 0 & c_{i,j}^{\sigma_j} \end{pmatrix} \tag{2.106}$$

[7] Here the $\{-1,1\}$ convention is used. Derivation applies to $\{0,1\}$.

This leads to wave function as,

$$\Psi_{\text{non-local dSBS}}(\boldsymbol{\sigma};\mathcal{W}) = \prod_i \Big(\prod_j a_{i,j}^{\sigma_j} + \prod_j b_{i,j}^{\sigma_j} + \prod_j c_{i,j}^{\sigma_j} \Big) \tag{2.107}$$

Their numerical results for Laughlin state suggest that EPS is preferred to short-range RBM. (Easier to optimized in terms of numerical stability) Fully-connected RBM give good results. For chiral spin liquid, they show that RBM work better than local SBS, non-local dSBS, EPS and Jastrow wavefunction with and without projected reference states.

Restricted Boltzmann Machine as Feed-Forward Neural Network: In the following, we show that RBM can be casted as a feed-forward neural network with restricted structure. Consider the case with $h_i, \sigma_j \in \{0, 1\}$, omitting the one-body terms, we rewrite the product over F_i as,

$$\Psi_{RBM}(\boldsymbol{\sigma};\mathcal{W}) \sim \prod_{i=1}^{N} F_i \tag{2.108}$$

$$= \exp\Big[\sum_i \log(1 + e^{b_i + \sum_j W_{ij}\sigma_j}) \Big] \tag{2.109}$$

$$= \exp\Big[\sum_i f(b_i + \sum_j W_{ij}\sigma_j) \Big] \tag{2.110}$$

The function $f(x) = \log(1 + e^x)$ is the so-called softplus activation, which could be consider as the continuous version of rectified linear unit. See figure 3.2. The argument inside the softplus function is an affine maping from the spin configuration. Therefore, we can view the RBM as a one-hidden layer feed-forward neural network with softplus activation, sum pooling, exponential function at output unit and an additional bias term that we leave out in the above discussion.

By viewing RBM as feed-forward neural network, we have several advantages. (1) We can generalized the wavefunction by adding additional hidden layers. The complexity of the wavefunction can be controlled by not only the width but the depth of the network. (2) With one-hot encoding, we can generalize wavefunction to larger physical dimension, e.g. spin-1 system. This is different from the naive generalization of RBM or the generalization in dSBS.

In the following section, we show how to generalize RBMQS. For the motivation and detail of neural networks see 3.3.

2.3.5. Neural Network Quantum State (NNQS)

From equation (2.2), the function is defined on the finite set. Since neural networks take real numbers as input, an naive approach would be taking the configuration $\boldsymbol{\sigma} \in \{0, \ldots, d-1\}^N$, i.e. indices for tensor, as input. However, one should note that there is no concept of

"magnitude" for categorical data. A dog is not larger than a cat by one and so is the spin-up particle to spin-down particle. They are simply different objects. By encoding the d-level system with magnitude, $0, \ldots, d-1$, one introduces bias. An unbiased way to encode the categorical input is the one-hot representation.

$$X : \boldsymbol{\sigma} \in \{0, \ldots, d-1\}^N \longrightarrow X(\boldsymbol{\sigma}) \in \mathbb{R}^{N \times d} \tag{2.111}$$

$$X(\boldsymbol{\sigma})_{ij} = \begin{cases} 1, & \text{if } \sigma_i = j \\ 0, & \text{else} \end{cases} \tag{2.112}$$

One-hot encoding is sparse. This may be problematic if d is large, e.g. for word embedding in Natural Language Processing. For spin model and fermionic system in physics, d is typically not too large. However, for bosonic system, cutoff should be introduced.

Feed-forwad neural networks are neural networks with no loop connections. It could contain fully-connected layers and convolution layers. Fully-connected layers are layers with weight matrices that determine the affine transformation.

A general k-layers fully-connected feed-forward neural network quantum state can be described as,

$$\Psi_{NN}(\sigma; \mathcal{W}) = g^{(out)}(b^{(out)} + W^{(out)} g^{(k)}(b^{(k)} + W^{(k)} g^{(k-1)}(\ldots g^{(1)}(b^{(1)} + W^{(1)} \mathrm{Vec}(X(\boldsymbol{\sigma})))))) \tag{2.113}$$

It is easier to read from the layer-wise expansion (3.10).

We denote the vectorization of the one-hot representation of spin configuration $X(\sigma)$ as $\mathrm{Vec}(X(\boldsymbol{\sigma}))$. The variational parameters are related to the affine transformation in the intermediate layers $\mathcal{W} = \{b^{(i)}, W^{(i)}\}$. In general, the nonlinear activation function could be different for different layers. In practice, we choose the same activation throughout the network, but with the last activation function chosen as exponential function.

For complex weight, we set the output dimension of the last weight matrix $W^{(out)}$ as one, and interpret the output after the exponential function as the probability amplitude of the wavefunction. For real-valued weight, we set the output dimension of the last weight matrix $W^{(out)}$ as two and interpret them as the log magnitude and the phase.

2.3.6. Convolutional Neural Network Quantum State (CNNQS)

Like fully-connected neural network, the convolution layer is define by the convolution kernel and convolution operation. By replacing the affine transformation with the convolution operation,

$$z = b + W \times x \longrightarrow z = b + W * x$$

we now have the convolutional neural network quantum state.

$$\Psi_{CNN}(\sigma; \mathcal{W}) = g^{(out)}(b^{(out)} + W^{(out)} * g^{(k)}(b^{(k)} + W^{(k)} * g^{(k-1)}(\ldots g^{(1)}(b^{(1)} + W^{(1)} * X(\boldsymbol{\sigma}))))) \tag{2.114}$$

Again, it might be easier to read from the expansion (3.14).

Note that we no longer need to vectorize the one-hot encoding. The physical dimension d is now regarded as different channels in the input. For example, a two dimensional spin-1 chain will have the same form as input as an image with RGB channels. We will see an example of translational-invariant RBM as CNN in the next section.

In general, a convolutional neural network could include fully-connected layers, pooling layers and even shortcut connection. Convolutional neural networks without fully-connected layers are called fully convolutional network (FCN). The detail of the convolutional neural network and the derivative with respect to the weights in neural networks are discussed in section 3.3.

Computational complexity: For evaluating the amplitude, we need to compute roughly $k+1$ matrix-vector multiplication. This gives the computational cost and storage as $\mathcal{O}(kN^2)$, where k is the depth of network and N the system size. The cost for derivative and log derivative are also about the same.

2.3.7. Symmetry and Invariance

In general, for building wavefunction respecting certain symmetries, we should write down a function that pick up a phase under certain transformation. However, this restriction is sometimes too strong. For the simplicity of the designing the network and the feasible optimization, we only require the wavefunction to be invariant after certain operation. This is a weaker condition comparing to exploit the full symmetry.

Given a unsymmetrized function $f(x)$ and a symmetry transformation $g : x \to g(x)$, one naive way to symmetrized the function is by

$$f^s(x) = \prod_{i=0}^{N_g-1} f(g^{(i)}(x)), \quad \text{or} \quad f^s(x) = \sum_{i=0}^{N_g-1} f(g^{(i)}(x)), \tag{2.115}$$

where $g^{(i)} = \underbrace{g \circ \cdots \circ g}_{\text{i times}}$ and $g^{(N_g)} = \mathbf{1}$ with g keep applying till the transformation gives the identity. The above formula contains two crucial steps. First, one need to create a set of equivariant input $\{x, g(x), g^{(2)}(x), \ldots, g^{(N_g-1)}(x)\}$ The set is invariant under symmetry transform, but each individual element is not invariant under symmetry transform.

$$\{x, g(x), g^{(2)}(x), \ldots, g^{(N_g-1)}(x)\} \to \{g(x), g^{(2)}(x), \ldots, g^{(N_g-1)}(x), x\} \tag{2.116}$$

Then, we could see one construct a symmetrized function by applying a invariant operation over the set.

The method above symmetrizes the function regardless the structure of the function. It is simple but comes with disadvantages like redundancy in parametrization of the function and greater expense in computation cost. [17] A better way of encoding symmetry would be to construct the function with equivariant operations in intermediate stages and an invariant operation in the end.

2.3.8. Translational Invariant Restricted Boltzmann Machine

One might already see that to symmetrize RBM wavefunction, we only need to force the F_i and the bias term to be equivariant, since we have products in the end.

In other words, we want

$$\{F_i(\boldsymbol{v})\} = \{F_i(g(\boldsymbol{v}))\} \tag{2.117}$$

We show in the following a way to satisfy the requirement above. We first set the number of hidden units as $N_h = N_g \times N_c, N_c \in \mathbb{N}$. Then we relabel F_i as F_i^c where $i \in [0, \dots, N_g - 1], c \in [0, \dots, N_c - 1]$. Now if we have

$$\mathcal{G} F_i^c(\boldsymbol{v}) = F_i^c(g(\boldsymbol{v})) \tag{2.118}$$

where $\mathcal{G}$ is the corresponding transform for function F_i^c, and map it to another element in the set $\{F_0^c, \dots, F_{N_g-1}^c\}$. We would have N_c equivariant sets of functions F. Here we choose $\mathcal{G}$ such that $\mathcal{G} F_i^c(\boldsymbol{v}) = F_{(i+1)}^c(\boldsymbol{v})$.

Example: Periodic Boundary Condition If a N sites system has a periodic boundary condition, and the Hamiltonian is l-site translation invariant, i.e. $[H, g_l] = 0$ and $g_l^{N/l} = 1$. Therefore, the eigenstates can be labeled by a quantum number $n \in [1, \dots, N/l]$, which corresponds to a phase $g_l \psi = \exp(2\pi n l/N)\psi$ and that the wavefunction is invariant after N/nl transformation.

We consider relaxing the constraint of the wavefunction, such that the function does not necessarily pick up a phase, but is invariant after N/nl transformation.

Recall,

$$F_i(\boldsymbol{\sigma}) = \begin{cases} 2\cosh[b_i + \sum_j W_{ij}\sigma_j], & \text{if } h_i = \pm 1 \\ 1 + \exp[b_i + \sum_j W_{ij}\sigma_j], & \text{if } h_i = 0, 1 \end{cases} \tag{2.119}$$

we want to find a constraint on W_{ij}, b_i so equation (2.118) is satisfied.

By requiring,

$$b_{i+1} + \sum_j W_{i+1,j}\sigma_j = b_i + \sum_j W_{ij}\sigma_{j+l} \tag{2.120}$$

One solution is given,

$$b_i = b_{i+1} \tag{2.121}$$

$$W_{i,j} = W_{(i+1),(j+l)} \tag{2.122}$$

$$i \in [0, \dots, N_g - 1], \quad N_g = N/nl \tag{2.123}$$

As we will see later, this restrict the W to be a convolution operation with stride l.

The symmetrized wavefunction gives,

$$\prod_{c=1}^{N_c}\prod_{i=1}^{N_g} F_i^c = \exp\Big[\sum_c \sum_i \log(1+e^{b_i^c+\sum_j W_{ij}^c \sigma_j})\Big] \tag{2.124}$$

$$= \exp\Big[\sum_c \sum_i f(b^c + W_i^c * \sigma)\Big] \tag{2.125}$$

We can view the symmetrized wavefunction in two different aspects.

From the aspect of machine learning, there are some things that need to notice about the convolution in the formula above. It is a stride-l circular convolution. That means one need to pad the input periodically. This is quite different from how convolution is usually performed. In fact, zero-padding does not provide exact translational equivariant. In most of the scenario in deep learning, e.g. Computer Vision, translational equivariance is not the actual concern. However, in terms of physics, the system size is usually small and the error is not negligible.

We now see, with the analogy to convolutional neural network, one can easily generalize the formula to physical system in higher dimensions or higher spin numbers. In addition, it provides another way to systematically increasing the complexity of the wavefunction by going "deeper".

3. Method II: Machine Learning

3.1. Supervised Learning

In short, supervised learning is an optimization problem of finding the best approximated function mapping $h(x; \mathbf{w})$ between the given the training data with input x and output label y. By showing the machine enough example of the (x, y) pairs, the goal is that the machine could learn to find the relation between x and y not by remembering the data. We give a brief summary about the statistical framework of supervised learning theory below.

Input Space $\mathcal{X}$: with elements $\mathbf{x} \in \mathcal{X}$. Input $\mathbf{x}$ is usually a vector, e.g. $\mathbb{R}^d$. More general input, e.g. tensor, graph, could also be possible.

Output Space $\mathcal{Y}$: with elements $y \in \mathcal{Y}$. Output y may be continuous or discrete. This leads to two different types of learning problems. The **regression** problems deal with continuous output space $\mathcal{Y}$, e.g. $y \in \mathbb{R}$. The **classification** problems deal with discrete output $\mathcal{Y}$, e.g. $y \in \{0, 1\}$.

Input Data / Training Data of the learner is the set $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)},), ..., (\mathbf{x}^{(n)}, y^{(n)})\}$ where $\mathbf{x} \in \mathcal{X}$, $y \in \mathcal{Y}$, $\mathcal{D} \in (\mathcal{X} \times \mathcal{Y})^n$

The Hypotheses Set $\mathcal{H}$ could be a finite or infinite set of functions with element $h \in \mathcal{H} : \mathcal{X} \longmapsto \mathcal{Y}$. The goal of supervised learning is to find the optimal hypotheses, which should correctly predict y by $h(x)$ given x, after seeing the training data. In other words, supervised learning is a map $\mathcal{D} \to \mathcal{Y}^{\mathcal{X}}$. In most case, we consider the hypothesis set parametrized by parameters $\mathbf{w}$.

Probabilistic assumption Assume there is some unknown probability measure P over $\mathcal{X} \times \mathcal{Y}$, resulting in identically and independently distributed random variables (x_i, y_i). Note this is a weaker assumption than assuming there is a deterministic mapping between $(x, y) = (x, f(x))$. The expectations with respect to P is denoted by $\mathbb{E}$.

Loss function and Risk function To formulate an optimization problem, we define the loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, which measures how far $h(x)$ is from the true label y. For a regression problem, the simple choice of loss function is the square loss $L(y, h(x)) = |y - h(x)|^2$. The risk, a.k.a. cost function, is the average error of the full distribution, defined by,

$$R(h) = \int_{\mathcal{X} \times \mathcal{Y}} L(y, h(x)) dP(x, y). \tag{3.1}$$

The optimization problem is to find the hypothesis that gives the lowest error, i.e. risk function. However, the underlying probability distribution is unknown. The only estimate we could have for the probability distribution is from the training data in our hands.

Therefore, the empirical risk (in-sample error, training error) is defined as,

$$\hat{R}(h) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, h(x_i)). \tag{3.2}$$

The training (learning) ends with the learner (algorithm) returning the optimal hypothesis that minimizes the empirical risk.

Empirical Risk Minimization (ERM). The whole framework of supervised learning is based on minimization of the empirical risk, which is the "surrogate" of the true risk function. The price one has to pay is the discrepancy between the empirical risk and the true risk, i.e. $R(h) - \hat{R}(h)$. This is known as the generalization error. A good learning happens when we have low empirical risk and also low generalization error. Simply remembering or over-fitting the training data would lead to low empirical risk but could not work well on new data given, thus not being able to generalize.

One of the main focus of learning theory is the mathematical study about how to determine an accurate bound for the generalization error for the hypothesis set given [58, 79, 100]. Here we give an example of typical theorem in Probably Approximately Correct (PAC) learning framework without proof. For an binary classification problem on a hypothesis set $\mathcal{F} \subseteq \{-1, 1\}^{\mathcal{X}}$. For some small $\delta, \epsilon \in (0, 1]$, and with $0/1$ loss, then $\forall h \in \mathcal{F}$, we have $|\hat{R}(h) - R(h)| \leq \epsilon$ holds over repeated sampling with probability at least $1 - \delta$, if the sampling size $|\mathcal{D}| \geq \nu_{\mathcal{F}}$.

$$\nu_{\mathcal{F}}(\epsilon, \delta) = \Theta\Big(\frac{VCdim(\mathcal{F}) + \ln(\frac{1}{\delta})}{\epsilon^2}\Big)$$

$VCdim$ is the Vapnik-Chervonenkis dimension that measures the capacity of the function class, which is roughly the degree of freedom of the function considered.

By the theorem, the number of samples we need to have, in order to guarantee the generalization, grows with the complexity of the model, the smaller generalization bound, and the probability we stay in the bound. It tells us qualitatively how many sampling we need have in order to have an accuracy we need. It could also tell us how accurate the model can be given the size of training data. This formulation gives a mathematical explanation for phenomena, like over-fitting.

Similar theorems exist in multi-label or continuous output space. It could explain why machine learning could work well with a low empirical risk. However, in general, the number of samples require for certain bound is usually way higher than the number of samples needed in practice. This is one aspect of the common argument, "we do not understand machine learning (deep learning)".

Validation error and test error. In practice, we usually keep separate all the data we have into training data $\mathcal{D}$, validation data $\mathcal{D}_V$ and test data $\mathcal{D}_T$. The validation data and test data could be viewed also coming from the underlying unknown probability distribu-

tion P. We define the validation error $\hat{R}_V$ and the test error $\hat{R}_T$ as

$$\hat{R}_V(h) \doteq \frac{1}{|\mathcal{D}_V|} \sum_{(x,y)\in\mathcal{D}_V} L(y, h(x)) \tag{3.3}$$

$$\hat{R}_T(h) \doteq \frac{1}{|\mathcal{D}_T|} \sum_{(x,y)\in\mathcal{D}_T} L(y, h(x)). \tag{3.4}$$

The test data serves as a mean to estimate the true risk since it is not seen in the training process and is statistically independent from the training data. In principle, we can also give a generalization bound from test error, but in practice we simply rely on and report the test error.

The validation data is kept for an intermediate stage of learning. Usually, there is some free parameters in the learning process that would affect the final outcome, which we call them the hyperparameters. To tune this hyperparameters, we need some "test data", which is now the validation data. In order to keep the test data statistically independent from the learning process, we separate the data into three parts.

Regularization. Based on the trade-off between model complexity and generalization error, we could argue there should be an optimal model complexity for the chosen problem such that it gives the lowest true risk. One solution would be to consider a sequence of hypotheses sets $\mathcal{H}_1 \subset \mathcal{H}_2 \subset ...$ with increasing complexity. After minimizing risk function with models of different complexity, we could check the validation error and determine the optimal model complexity.

One simpler solution for this is to add a complexity penalty term to the risk function. Then, we minimize instead the regularized empiricial risk

$$\hat{R}(h) + \frac{\lambda}{2}\|A\mathbf{w}\|^2 \tag{3.5}$$

where A is usually the identity. The free parameter λ should be determined by validation error.

3.2. Linear Model

Below, we formulate the problem of linear regression in the language of supervised learning.

Linear Regression: Linear regression is a regression problem with input vector $\mathbf{x}^T \in \mathbb{R}^d$ and continuous output $y \in \mathbb{R}$.

The input data is denoted as $\mathbf{X} \in \mathbb{R}^{N\times d}$, where each row is a data vector, and labels $\boldsymbol{y} \in \mathbb{R}^N$. Assuming a linear relation between the label and the input, $h(\mathbf{x}; \mathbf{w}) = \mathbf{x}^T\mathbf{w}$, this gives us a linear model with parameters $\mathbf{w}$. [1]

[1]Note that we omit the bias term in the model. Instead of $\mathbf{w}^T\mathbf{x} + b$, we could always append one in the input data. The bias term is absorbed into the weights by changing $\mathbf{x}^T \to [1, \mathbf{x}^T]$.

The loss function we are considering is the the L_2 loss,

$$L(w) = (y - \mathbf{x}^T\mathbf{w})^2$$

The empirical risk is

$$\hat{R}(h(\mathbf{w})) = \frac{1}{N}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$$

The learning is understood as the process of finding the optimal $\mathbf{w}$. In this case, we have a closed form solution,

$$\begin{aligned} \nabla_{\mathbf{w}}\hat{R}(\mathbf{w}) &= 2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\boldsymbol{y} = 0 \\ \mathbf{w}_{opt} &= \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\boldsymbol{y} \end{aligned} \tag{3.6}$$

After "learning", we can make prediction (inference) $\hat{y}_{new}$ on new data $\mathbf{x}_{new}$ based on the learned model $h(\mathbf{x}; \mathbf{w}_{opt})$,

$$\hat{y}_{new} = \mathbf{x}_{new}^T\mathbf{w}_{opt} \tag{3.7}$$

Linear Regression for Classification The classification problem is defined by the label being binary $y \in \{-1, 1\}$. We could solve the classification problem with linear regression by embedding the binary label in continuous space.

We can still consider the L_2 loss, and all the formula in linear regression above still follows. The only thing that is different is in the prediction. We should interpret the continuous label predicted as a binary label.

$$\hat{y}_{new} = \mathbf{x}_{new}^T\mathbf{w}_{opt} \longrightarrow \text{sgn}(\hat{y}_{new})$$

This is of course not the best way to solve a classification problem. But we will see that this simple method gives us some intuition about decision boundary and the effect of nonlinear transform.

The decision boundary is the hyperplane defined by

$$\mathbf{x}^T\mathbf{w}_{opt} = 0 \tag{3.8}$$

By the definition, data is divide into two sets by the hyperplane and data is labeled to be positive or negative by the side it is in. For a binary classification problem, we said the input data is linear separable, if there exist at least one hyperplane with all the data classified correctly.

Non-linear transform Now we consider the case where we have a 2-dimension input data $\{1, x_1, x_2\}$. We see that the decision boundary is a line in $x_1 - x_2$ plane.

$$x_1w_1 + x_2w_2 + w_3 = 0$$

We show an example of data that is not linear separable in figure 3.1. The data is not linear separable in the basis x_1, x_2. Since applying linear model is equivalent to draw a

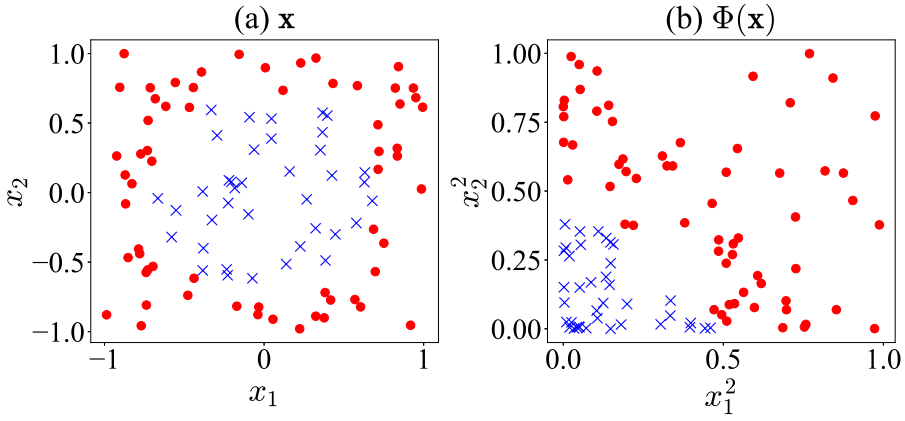

Figure 3.1.: Same data is plotted in the basis of $\{x_1, x_2\}$ in (a) and $\{x_1^2, x_2^2\}$ in (b). It is easily to see data is linear separable in (b).

straight line for decision boundary, many data will be classified wrongly and the model is expected to perform poorly.

By inspection, we know that to have a good result we need to draw a circular decision boundary. We could effectively do this with linear model with a nonlinear transform $\Phi(\mathbf{x}) = [x_1^2, x_2^2]$ on the original data $\mathbf{x} = [x_1, x_2]$. We can then separate the data with the boundary,

$$w_1 x_1^2 + w_2 x_2^2 + w_3 = 0$$

which is a line in $\Phi(\mathbf{x})$ space and a circle in $\mathbf{x}$ space. We can even consider a second order mapping, $\Phi(\mathbf{x}) = [x_1, x_2, x_1^2, x_1 x_2, x_2^2]$. By this mapping, we can have decision boundary with conic sections.

In general, we can consider any mapping,

$$\Phi : \mathbf{x} \in \mathbb{R}^d \to \Phi(\mathbf{x}) \in \mathbb{R}^D \tag{3.9}$$

and define the linear model in the transformed space $h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \Phi(\mathbf{x})$. The model defined in high-dimensional space is usually more powerful comparing to a linear model in the original input space. However, with a more complicated model, one might need to pay the the price of over-fitting [2].

Support Vector Machine (SVM) could be considered as linear model in high-dimensional space (sometimes even infinite dimensional) with clever regularization to prevent over-fitting. Neural Network consider the nonlinear transformation defined by composite functions, i.e.:

$$\Phi(\mathbf{x}) = \phi_n(\phi_{(n-1)}(\cdots(\phi_1(\mathbf{x}; \mathbf{w}_1)\cdots); \mathbf{w}_{n-1}); \mathbf{w}_n)$$

[2] One well-known example of nonlinear transform and over-fitting is the data fitting with polynomial regression, where the nonlinear transform is simply $x \to \{1, x, x^2, \dots\}$.

Moreover, since we parametrize the functions in the composition, we will learn the optimal non-linear transform within the composite function class.

3.3. Neural Network

In this section, we review the definition of feedforward neural networks and give the definition of the activation functions. We then review the backpropagation method and the optimization methods. In the end, we talk about how to randomly initialize the weights of the network. For more detail, see [30].

Same as discussed in 2.3.5 and 2.3.6, a general k-layers fully-connected feed-forward neural network with input vector $\mathbf{x}$ is defined as,

$$f_{NN}(\mathbf{x};\mathcal{W}) = g^{(out)}(W^{(out)}h^{(k)} + b^{(k)}) \tag{3.10}$$
$$h^{(k)} = g^{(k)}(W^{(k)}h^{(k-1)} + b^{(k-1)}) \tag{3.11}$$
$$\vdots \tag{3.12}$$
$$h^{(1)} = g^{(1)}(W^{(1)}\mathbf{x} + b^{(1)}) \tag{3.13}$$

where we expand the equation (2.113).

In convolution neural network, we replace the matrix products by convolution operations and we have

$$f_{CNN}(\mathbf{x};\mathcal{W}) = g^{(out)}(W^{(out)} * h^{(k)} + b^{(k)}) \tag{3.14}$$
$$h^{(k)} = g^{(k)}(W^{(k)} * h^{(k-1)} + b^{(k-1)}) \tag{3.15}$$
$$\vdots \tag{3.16}$$
$$h^{(1)} = g^{(1)}(W^{(1)} * \mathbf{x} + b^{(1)}) \tag{3.17}$$

which is the expansion from (2.114).

Activation function Training deep neural networks is made possible partially because of the development of new activation functions. Now people move away from using the hyperbolic-tangent or the sigmoid function $f(x) = \frac{1}{1+e^{-x}}$ to using rectified linear unit (ReLU) [60]. Alternative activation functions are still research topic. We show the definition of some promising but not well-accepted activation functions including exponential linear units (ELU) [14], scaled exponential linear units (SELU) [49], Swish [68].

Definitions of some activation functions are given below and shown in figure 3.2.

- $\text{ReLU}(x) = \max(0, x)$

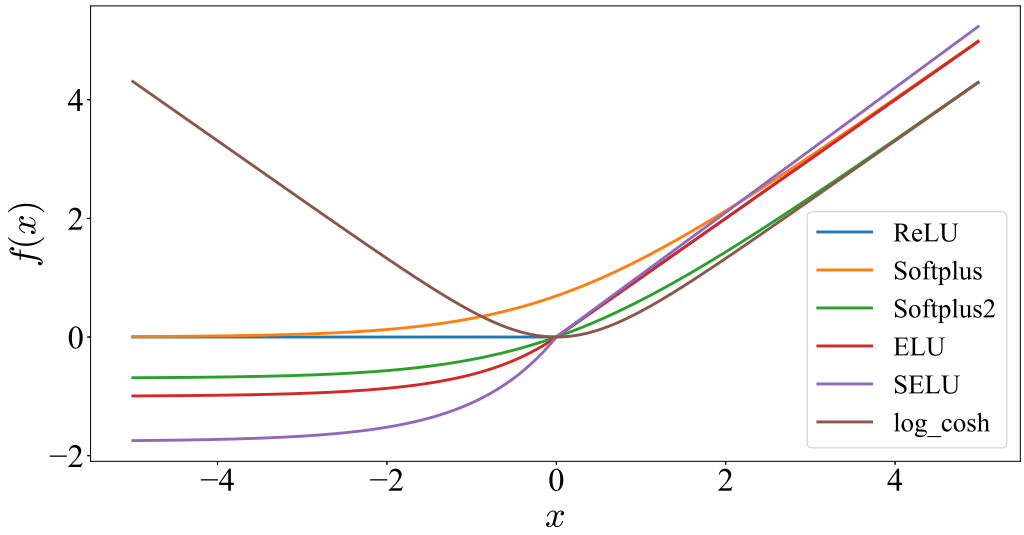

Figure 3.2.: ReLU, Softplus, ELU converge for large x (the red line). For ELU, we have $\alpha = 1$. Parameters in SELU $\alpha \approx 1.673, \lambda \approx 1.0507$ are chosen such that we have a fix point at mean zero and unit variance for the signals [49].

- $\text{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1), & \text{otherwise} \end{cases}$.
- $\text{SELU}(x) = \lambda \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1), & \text{otherwise} \end{cases}$
- $\text{Softplus}(x) = \ln(1 + e^x)$
- $\text{Softplus2}(x) = \ln(\frac{1+e^x}{2})$

Complex-valued Neural Network By complex-valued neural networks or complex neural networks, we mean neural networks with complex-valued parameters. The output, however, could be real or complex. Some recent works on complex-valued neural networks could be found in [90, 32]. The idea is simple. One could replace all the real-valued weights in the network with complex-valued weights. We will give the formulation and the advantages below. The nontrivial part is the design of activation function. For that, we will review different generalization of the ReLU activation function to complex domain.

We can think of the complex vector $\mathbf{x} = \mathbf{x}_R + i\mathbf{x}_I$ as two real vectors, and the complex weight matrix $\mathbf{W} = \mathbf{W}_R + \mathbf{W}_I$. The multiplication or convolution operation is given by,

$$\mathbf{W} * \mathbf{x} = (\mathbf{W}_R * \mathbf{x}_R - \mathbf{W}_I * \mathbf{x}_I) + i(\mathbf{W}_R * \mathbf{x}_I + \mathbf{W}_I * \mathbf{x}_R). \tag{3.18}$$

We see that if we represent $\mathbf{x}$ by concatenating the real and imaginary part of the vector into a larger vector. We have,

$$\begin{bmatrix} \mathcal{R}(\mathbf{W} * \mathbf{x}) \\ \mathcal{I}(\mathbf{W} * \mathbf{x}) \end{bmatrix} = \begin{bmatrix} \mathbf{W}_R & -\mathbf{W}_I \\ \mathbf{W}_I & \mathbf{W}_R \end{bmatrix} \begin{bmatrix} \mathbf{x}_R \\ \mathbf{x}_I \end{bmatrix} \tag{3.19}$$

The complexity of the weights could be seen as a structured the real-valued weight as above, which reduce the number of parameters by a factor 2. In practice, we consider the formulation above of two real-valued weights, since modern deep learning libraries do not support convolution operation with complex type.

Complex activation function To generalize ReLU to complex domain, it is natural to require the generalization satisfy the condition that,

$$\forall x \in \mathbb{R} \ \text{Complex ReLU}(x) = \text{ReLU}(x)$$

Guberman [32] proposed to use,

$$\text{zReLU}(z) = \begin{cases} z, & \text{if } \theta_z \in [0, \pi/2] \\ 0, & \text{otherwise} \end{cases} \tag{3.20}$$

Chiheb et al. [90] proposed to apply the activation separately on both the real and the imaginary part, i.e.

$$\text{CReLU}(z) = \text{ReLU}(\mathcal{R}(z)) + i\text{ReLU}(\mathcal{I}(z)) \tag{3.21}$$

Here we also give another different generalization sReLU motivated from the softplus activation. Since softplus activation is considered as the smooth approximation of ReLU, we first consider the softplus activation in complex domain. We show the visualization of softplus activation in complex domain $\text{softplus}(z) = \log(1 + e^z)$ in figure 3.3, which suggest the following definition.

$$\text{sReLU}(z) = \begin{cases} z, & \text{if } \theta_z \in [-\pi/2, \pi/2] \\ 0, & \text{otherwise} \end{cases} \tag{3.22}$$

3.3.1. Optimization of Neural Network

Back-propagation of Fully-Connected Neural Network In the neural network, the back-propagation is the technique of carrying out the chain rule throughout the layers of the neural network to obtain the gradient of each weight matrix. In modern deep learning library, such process can be generated through an computation graph, and techniques in dynamic programming are involved so computation can be replaced by storing the intermediate result through out the step.

First we need to make the convention clear. Consider a L layer neural network. The 0'th layer is the input layer and the L'th layer the output layer. The index of the layer is denoted by $l \in \{0, 1, \ldots, L\}$. N_l is the number of neurons in l'th layer. By w^l_{jk} and b^l_j, we denote the component in the weight matrix W^l and the bias vector b^l connecting layer $l-1$ to layer l.

The output of the hidden layer is, $x^l = g(W^l x^{l-1} + b^l)$. We define the intermediate unit before the activation function is applied as, $z^l = W^l x^{l-1} + b^l$. The partial derivative is calculated with respect to some objective function f over the output, $\delta^l_j = \frac{\partial f}{\partial z^l_j}$. In the case of VMC, f is an logarithm function $f(x) = \log x$.

The intermediate derivative in the output layer and the subsequent layer is given by,

$$\delta^m_j = \sum_k \frac{\partial f}{\partial x^m_k} \frac{\partial x^m_k}{\partial z^m_j} = g'(z^m_j) \frac{\partial f}{\partial x^m_j} = g'(z^m_j) \tag{3.23}$$

$$\delta^l_j = \sum_k \delta^{l+1}_k \frac{\partial z^{l+1}_k}{\partial z^l_j} = \sum_k \delta^{l+1}_k w^{l+1}_{kj} g'(z^l_j) \tag{3.24}$$

where the summation runs over all the neurons in the layer considered. The derivative with respect to the weight matrix and the bias:

$$\frac{\partial f}{\partial b^l_j} = \sum_k \frac{\partial f}{\partial z^l_k} \frac{\partial z^l_k}{\partial b^l_j} = \delta^l_j \tag{3.25}$$

$$\frac{\partial f}{\partial w^l_{jk}} = \sum_i \frac{\partial f}{\partial z^l_i} \frac{\partial z^l_i}{\partial w^l_{jk}} = \delta^l_j x^{l-1}_k \tag{3.26}$$

Back-propagation of Convolutional Neural Network Here we consider the case in one dimension [3],

$$z^l_i = W^l * x^{l-1}_i + b^l_i = \sum_{a=0}^{|w|-1} w^l_a x^{l-1}_{i+a} + b^k_i \tag{3.27}$$

[3] Machine Learning community tend to use the word convolution while they actually mean cross-correlation. There might be a sign difference in the definition comparing to other material.

Then again we introduce the δ derivative with respect to intermediate element.

$$\delta_i^l = \frac{\partial f}{\partial z_i^l} = \sum_{i'} \sum_a \frac{\partial f}{\partial z_{i'}^{l+1}} \frac{\partial w_a^{l+1} \sigma(z_{i'+a}^l)}{\partial z_i^l} \tag{3.28}$$

$$= \sum_{i'} \delta_{i'}^{l+1} w_{i-i'}^{l+1} \sigma'(z_i^l) \tag{3.29}$$

$$= \sum_{j=0}^{|w|-1} \delta_{i-j}^{l+1} w_j^{l+1} \sigma'(z_i^l) \tag{3.30}$$

$$\delta_i^l = \delta^{l+1} * \text{flip}(W_i^{l+1}) \sigma'(z_i^l) \tag{3.31}$$

The derivative with respect to the convolution kernel,

$$\frac{\partial f}{\partial w_a^l} = \sum_k \frac{\partial f}{\partial z_k^l} \frac{\partial z_k^l}{\partial w_a^l} = \sum_{i=0}^{|\delta^l|-1} \delta_i^l x_{i+a}^{l-1} = \delta^l * x_a^{l-1} \tag{3.32}$$

$$\frac{\partial f}{\partial b_j^l} = \sum_k \frac{\partial f}{\partial z_k^l} \frac{\partial z_k^l}{\partial b_j^l} = \delta_j^l \tag{3.33}$$

Stochastic gradient descent algorithm A good review on first-order stochastic gradient descent algorithms is given in [70]. We have already given an overview of the second-order methods in VMC section, for more detail see [53].

3.3.2. Initialization of Neural Network

Training neural network is like cooking. Everything seems easy when one follows the steps in the cookbook. However, things could easily go wrong for non-expert if instructions are not followed. It is the case, even till 2010, that people struggle with training deep neural network. There are several reasons for this: (1) the initialization, (2) the activation function, (3) the architecture. Deep learning is only possible with the development of Glorot initialization, He initialization, ReLU, batch normalization, residual network etc. We address the issue of initialization here.

Bad initialization could easily push the activation function toward the saturated region, where the gradient vanishes. As a result, people used to initialize the parameters by unsupervised pretraining[19] or greedy layer-wise training procedure [4]. Even if the activation function is chosen such that there should be no saturation, e,g, ReLU, the networks still have difficulties to converge [83]. Globat and Bengio [27] study the propagation of gradients in deep neural network and proposed a proper initialization scheme. He et al. [40] follow the same idea and study the initialization for ReLU activations. The idea of He initilization is reviewed and the discussion for initialization in complex-valued neural network is followed.

MSRA Initialization, a.k.a. He Initialization Here, we follow the derivation in [40] for 2-dim convolutional neural networks. It is straight forward to generalize the derivation to other dimensions. The basic idea is to keep the variance of the output invariant in every layer. For each layer, we have,

$$\begin{aligned}\mathbf{y}_l &= W_l\mathbf{x}_l + \mathbf{b}_l \\ \mathbf{x}_l &= f(\mathbf{y}_{l-1})\end{aligned}$$

In each layer, $\mathbf{x}$ is a vector of size $n = k^2c$, which comes from the reshaping a $k \times k$ block of c input channels. k is the filter size. W is of size $d \times n$. d is the number of output channels. The relationship should holds $d_{l-1} = c_l$. The f is the activation function chosen.

Several assumptions are made here. We assume the elements $[W_l]_{ij}$ are independent and identically distributed (i.i.d.) random variables denoted by w_l. In addition, w_l are symmetric distributed around zero and have zero mean. We assume the elements $[\mathbf{x}_l]_i$ are also i.i.d. random variables denoted by x_l, and are independent with $[W_l]_{ij}$. Then, we have,

$$\mathrm{Var}[y_l] = n_l\mathrm{Var}[w_l x_l] \tag{3.34}$$

$$\mathrm{Var}[w_l x_l] = \mathrm{Var}[w_l]\mathrm{Var}[x_l] + \mathrm{E}[w_l]^2\mathrm{Var}[x_l] + \mathrm{Var}[w_l]\mathrm{E}[x_l]^2 \tag{3.35}$$

From the assumption, we have $\mathrm{E}[w_l] = 0$. y_l is symmetric and has zero mean if $b_l = 0$. Therefore, $\mathrm{Var}[y_l] = \mathrm{E}[y_l^2]$. However, $\mathrm{E}[x_l] \neq 0$ if f is not symmetric around zero.

$$\mathrm{Var}[y_l] = n_l\mathrm{Var}[w_l]\mathrm{E}[x_l^2] \tag{3.36}$$

$$= n_l\mathrm{Var}[w_l](\alpha\mathrm{Var}[y_{l-1}]) \tag{3.37}$$

where α is a number depends on the activation function and should be determined from $\mathrm{E}[x_l^2] = \alpha\mathrm{E}[y_{l-1}^2]$. For, ReLU, $x_l = max(0, y_{l-1})$, we have $\mathrm{E}[x_l^2] = \frac{1}{2}\mathrm{E}[y_{l-1}^2]$. Now, we obtain the iterative relation for forward propagation of a general activation,

$$\mathrm{Var}[y_l] = \alpha n_l\mathrm{Var}[w_l]\mathrm{Var}[y_{l-1}] \tag{3.38}$$

For a L layers neural network, we have,

$$\mathrm{Var}[y_L] = \mathrm{Var}[y_1]\Big(\prod_{l=2}^{L}\alpha n_l\mathrm{Var}[w_l]\Big) \tag{3.39}$$

From the equation above, we see that a bad weight initialization would lead to exponential growing or decay of the magnitude of the output. As a result, it is suggested to initialize w_l by a zero-mean symmetric distribution, e.g. Gaussian, with standard deviation of $\sqrt{\frac{1}{\alpha n_l}}$.

In general, the value α depends on both the activation function and the distribution of w_l. While in the case of ReLU and PReLU, the assumptions above already determine

α values		
Activation function	α	Distribution
ReLU	$\frac{1}{2}$	All symmetric zero mean
PReLU	$\frac{1}{2}(1+a^2)$	All symmetric zero mean
Softplus	≈ 0.921	Gaussian unit std
Softplus2	≈ 0.284	Gaussian unit std
elu	≈ 0.645	Gaussian unit std

Table 3.1.: The comparison between different alpha values in network initialization.

the value of α, it is not the case for other activation functions. In the case of ReLU, He et al. suggested to use Gaussian distribution. In the following, we gives the values of α assuming w_l is from Gaussian distribution.

So far, we only give the iterative relation for forward propagation. In [40], the iterative relation in backward propagation is also studied with a similar relation

$$\hat{\alpha}\hat{n}_l \mathrm{Var}[w_l] = 1 \tag{3.40}$$

where $\hat{\alpha} = 1/2$ for ReLU, and $\hat{n} = k^2 d$. Setting up the weight with this equation would lead to a factor of c_2/d_L in forward propagation, which is acceptable number for typical neural networks. We will simply follow the criterion derived from forward propagation.

One remark is given here. Even with proper initialization, it is not guarantee the gradient can propagate nicely. A more radical approach is introduced recently, named self-normalizing neural network [49].

3.4. Policy Based Reinforcement Learning

The approach in Reinforcement Learning (RL) could be roughly divided into three categories: policy-based, value-based, model-based. They are not mutually exclusive. Here, we review some result of policy-based RL that is related to VMC.

Setting: In policy-based RL, one parametrizes the policy distribution as $\pi_\theta = \pi(\mathbf{a}|\mathbf{s},\theta)$, where $\mathbf{s}$ is the state and $\mathbf{a}$ the action, which would lead to different states. A reward is assigned for each state action pair $r_t(\mathbf{s}_t, \mathbf{a}_t)$. There are different policy objective functions, here we consider the average reward per time-step as objective function,

$$J(\theta) = \sum_{\mathbf{s}} d^{\pi_\theta}(\mathbf{s}) \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s};\theta) r(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\pi_\theta}[r] \tag{3.41}$$

where $d^{\pi_\theta(\mathbf{s})}(\mathbf{s})$ is the stationary distribution of the Markov chain generated by π_θ.

Policy-based RL is an optimization problem. Finding the maximum of the policy objective function gives us the policy that would generate the best overall rewards. The essence of

the problem is to find the parameters of a distribution to maximize/minimize an expectation. Similar problems show up in variational inference, fitting latent-variable models, where the problem is to optimize

$$\mathcal{L}(\theta) = \mathbb{E}_{p(z|\theta)}[f(z)]. \tag{3.42}$$

The optimization problem could be solved by stochastic gradient descent/ascent. One inserts the log derivative to recover the form of expectation value.

$$\nabla_\theta \mathcal{L}(\theta) = \mathbb{E}_{p(z|\theta)}[f(z)\nabla_\theta \log p(z|\theta)] \tag{3.43}$$

$$\hat{g}[f] = f(z)\nabla_\theta \log p(z|\theta), \qquad z \sim p(z|\theta) \tag{3.44}$$

The stochastic gradients $\hat{g}$ is obtain from sampling and is unbiased, i.e. $\mathbb{E}[\hat{g}] = \nabla_\theta \mathcal{L}(\theta)$. The method is known under different names, including score-function estimator [50], likelihood ratio methods, and **REINFORCE** [99] or **policy gradient** when applied to optimize the policy objective function,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[r \log \pi(\mathbf{a}|\mathbf{s};\theta)] \tag{3.45}$$

$$= \sum_{\mathbf{s}} d^{\pi_\theta}(\mathbf{s}) \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s};\theta)\nabla_\theta \log \pi(\mathbf{a}|\mathbf{s};\theta) r(\mathbf{s},\mathbf{a}). \tag{3.46}$$

With different policy objective functions, the policy gradient still has the same form, with r replaced by long-term reward. One disadvantage of using policy gradient is that the gradient estimator usually has high variance.

Control variates In RL, one usually choose the state-value function $V^\pi(s) = \mathbb{E}_\pi[r|s_1 = s]$ as a baseline. Observe that

$$\mathbb{E}_{p(z|\theta)}[V^\pi(s)\nabla_\theta \log p(\mathbf{s}|\mathbf{a})] = \sum_{\mathbf{s}} d^{\pi_\theta}(\mathbf{s}) V^\pi(s) \nabla_\theta \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s};\theta) = 0$$

suggest that we can improve the variance of the policy gradient by considering,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[(r - V^{\pi_\theta}) \log \pi(\mathbf{a}|\mathbf{s};\theta)]. \tag{3.47}$$

The $(r - V^{\pi_\theta})$ is the advantage function which suggest update direction toward policy better than average [4].

Other methods were developed to create low-variance, unbiased gradient estimator such as the reparameterization trick [99, 48], combining the value function by actor-critic algorithm, or even advantage actor-critic estimator (A2C) [87].

[4] In general, the method consider to reduce the variance of the stochastic estimator by subtracting a control variate $c(z)$ and adding back the known mean $\mathbb{E}_{p(z)}[c(z)]$.

$$\hat{g}_{new}(z) = \hat{g}(z) - c(z) + \mathbb{E}_{p(z)}[c(z)]. \tag{3.48}$$

The variance would be reduced if $c(z)$ is positively correlated with $\hat{g}(z)$.

Another direction is to consider second-order methods discussed before with policy gradient, including natural policy gradient [47], trust region policy optimization (TRPO) [77], actor critic using Kronecker-factored trust region (ACKTR) [101].

Deep reinforcement learning is simply the same as the discussion above, but with policy function represented by neural networks. While with the well-known success in mastering the game of go by self-play [81, 82], deep reinforcement learning is also known for problems like sample-inefficient, not easily generalizing to different task, local minima, numerical instability.

Comparison with VMC and supervised learning Recall in VMC, we also have (2.8) of the same form. The gradient estimate in VMC (2.20) or (2.22) is similar to the REINFORCE method with baseline in policy-based RL.

In fact, VMC is very similar to reinforcement learning. However, there is some small differences. In VMC, we work with quantum amplitude instead of probability. We do not parametrize the policy function, but the full quantum amplitude. The transition between state occur as a result of MCMC sampling. Also the wavefunction is often not normalized. The baseline term appears from the derivative of the normalization constant. This similarity suggests that it is possible that one could adapt methods in RL to VMC to improve VMC method.

On the other hand, VMC/RL is quite different from supervised learning. In supervised learning, we usually assume that we have "fixed" data coming from i.i.d. sampling of an unknown distribution. In VMC/RL, however, we do not have a "fixed" data set. The data is generated by MCMC sampling of the current solution. After updating the parameters of the function, we always need to sample again.

The scenario for tasks in deep learning and VMC are quite different. In deep learning, one often work with models up to millions parameters. The data is given and stored in the computer. Because of the memory constraint, one works with the data in mini batch, which usually with the size of ten to one hundred.

In VMC, the number of parameters strongly depends on the wavefunction chosen. However, this usually only range from tens to thousands. One major difference is new data must be "generated" in each step by MCMC. Though the model is small, one need to estimate the energy correctly in order to have the gradient correct. For stability of the optimization, there is no "mini-batch" gradient descent. In each iteration the size of data sampled could be tens of thousands, depending on the size of system studied. See table 3.2 [5].

Because of these difference up to several order of magnitudes, the training strategy should be different. For typical deep learning task, even though second-order methods and natural gradient method converge faster than gradient descent methods in terms of number of iterations, it is slower in terms of wall clock time. The main reason is that the calculation of second-order matrix is expensive. As a result, the dominant methods in

[5] dataset like ImageNet competition[71]

deep learning community are still gradient descent based methods. The situation is different for VMC calculation, where the bottleneck of computation lies in the generation of data through MCMC process or the calculation of local energy. The cost of second-order methods are at most of the same. Therefore, using second-order method or natural gradient method is more efficient even in terms of wall clock time.

Comparison of Computation Complexity			
Number/Size of	VMC	(Supervised) Deep Learning	Reinforcement Learning
Mini-batch Data : N_d # of parameters N_w # of iteration N_c	$\mathcal{O}(1000)$ $\leq \mathcal{O}(10^6)$ $\mathcal{O}(10000)$	$\mathcal{O}(10)$ to $\mathcal{O}(100)$ $\geq \mathcal{O}(10^6)$ n/a	$\mathcal{O}(10)$ to $\mathcal{O}(1000)$ $\leq \mathcal{O}(10^6)$ n/a
Preferred Methods :	Second-order methods	First-order methods	Second-order methods

Table 3.2.: The brief comparison between VMC, deep supervised learning and deep reinforcement learning.

3.5. Kronecker-Factored Approximate Curvature

James Martens developed the Kronecker-Factored Approximation for the Fisher information matrix in his PhD study [53]. This is applied to simple fully-conneted network [54], convolutional network [31], and is scalable for distributed computing [3]. In supervised learning, it is shown by Matt Johnson and Daniel Duckworth from Google Brain that such method can be applied to ResNet-50 on SVHN dataset. K-FAC has strong scaling behavior in terms of data presented and requires 2.3x to 7.8x fewer steps comparing to SGD algorithm. It has successfully been applied to Reinforcement Learning combining with the Advantage-Actor-Critic (A2C) method known as ACKTR is currently one of the state of the art methods.

Kronecker-Factored Approximate Curvature (K-FAC) methods are based on two approximations. (1.) Approximating Fisher information matrix by a blocked matrix. Each block corresponds to different layers. (2.) Assuming the statistic independence between the activations and the backpropagated derivatives. With these two assumption, we could derive the simplest version of K-FAC on fully-connectected neural networks.

Consider the weight matrix $W \in \mathbb{R}^{C_{out} \times C_{in}}$ in the l^{th} layer, where C_{out} and C_{in} are the number of output/input neurons of the layer. The output distribution of the network is $p(y|x)$ and the log-likelihood $L = \log p(y|x)$. The log derivative of the weight matrix is given by $\nabla_W L = (\nabla_s L)a^T$, where $a \in \mathbb{R}^{C_{in}}$ is the input activation vector, s is the pre-activation vector for the next layer, and $s = Wa$. With the first assumption, we consider the block of Fisher information matrix corresponding to layer l in the neural network as

F_l.

$$F_l = \mathbb{E}[\text{vec}\{\nabla_W L\}\text{vec}\{\nabla_W L\}^T] = \mathbb{E}[aa^T \otimes \nabla_s L(\nabla_s L)^T] \tag{3.49}$$

$$\approx \mathbb{E}[aa^T] \otimes \mathbb{E}[\nabla_s L(\nabla_s L)^T] \doteq A \otimes S \doteq \hat{F}_l \tag{3.50}$$

With the second assumption, we obtain the Kronecker-factored approximated fisher block $\hat{F}_l$. Immediately, we see with this approximation, one can compute the natural gradient for weight matrix W efficiently

$$\hat{F}_l^{-1}\text{vec}\{\nabla_W J\} = \text{vec}\{A^{-1}\nabla_W J S^{-1}\} \tag{3.51}$$

where J is the objective function, and the identity

$$(A \otimes S)^{-1}\text{vec}\{x\} = A^{-1} \otimes S^{-1}\text{vec}\{x\} = A^{-1}xS^{-1}$$

is used.

There are several advantages for computing approximated natural gradient with K-FAC. The first advantage is that the cost, as shown in the equation above, is about the computation on matrices with the same size as W. This makes natural gradient as cheap as gradient descent methods even in deep networks. Secondly, Hessian-Free algorithm or natural gradient with iterative solver could not accumulate data from previous updates. As a result, small batch size would lead to poor update. With K-FAC, one could work with small batch size and with stable update by keeping moving average of the approximation to the Fisher matrix block.

Application to Reinforcement Learning In RL, sampling could be expensive. As a result, advanced optimization methods which try to extract more information from the a single batch usually outperform simple gradient descent algorithm, e.g. TRPO. Combining with actor-critic method and with an update control preventing large updates to policy, K-FAC is applied to RL successfully [101].

In RL, the underlying distribution is defined by the policy or the value function, which changes with the update of parameters. Nevertheless, it is observed in practice that keeping moving average of the metric still works well for RL problems.

3.6. Connection to Methods in Condensed Matter Physics

EPS/CPS as Graphical Model

EPS/CPS have the same form of factor graph in Graphical Model. Both factor the high dimension tensor as products of smaller tensors. While for quantum states the value of the tensor is in general complex, the value for graphical model could only be positive number. Earlier work have tried to generalize the graphical model for wavefunction and translate the algorithm for graphical model to calculating physical quantities. There are

two different generalization done with this analogy of quantum amplitude from classical probability. Matthew and David develop quantum Graphical Models and quantum (loopy) belief propagation by working at the level of density matrix and introducing non-commutable algebra structure. Hasting also came up with the same idea. [51, 36] Tucci et al. has developed another approach to quantum Graphical Model and related algorithm based on simply replacing the probabilities with complex valued amplitudes [91, 92, 93].

Even though it may be obvious that one could always write the EPS/CPS in tensor network language and vice versa, ignoring the resulting complicated structure. In [69], a nice correspondence between graphical model, i.e. EPS/CPS and tensor network states is formulated using hypergraph. With this formulation, it is clear that tensor network states are EPS/CPS with auxiliary sites. The formulation of EPS/CPS as correlator operator on some references states has also interesting correspondence in graphic models. It is equivalent to saying the graph (EPS/CPS) describe the likelihood function and the references states is the prior. An uniform prior is the equal weighted superposition of product states.

Application of Tensor Network in Machine Learning

Recently, it is shown that Tensor Network could be applied to Machine Learning. For example, MPS [86], MERA for supervised learning [52], MPS for word embedding [66] and language model [67], and MPS/MPO for Neural Network compression [64, 24].

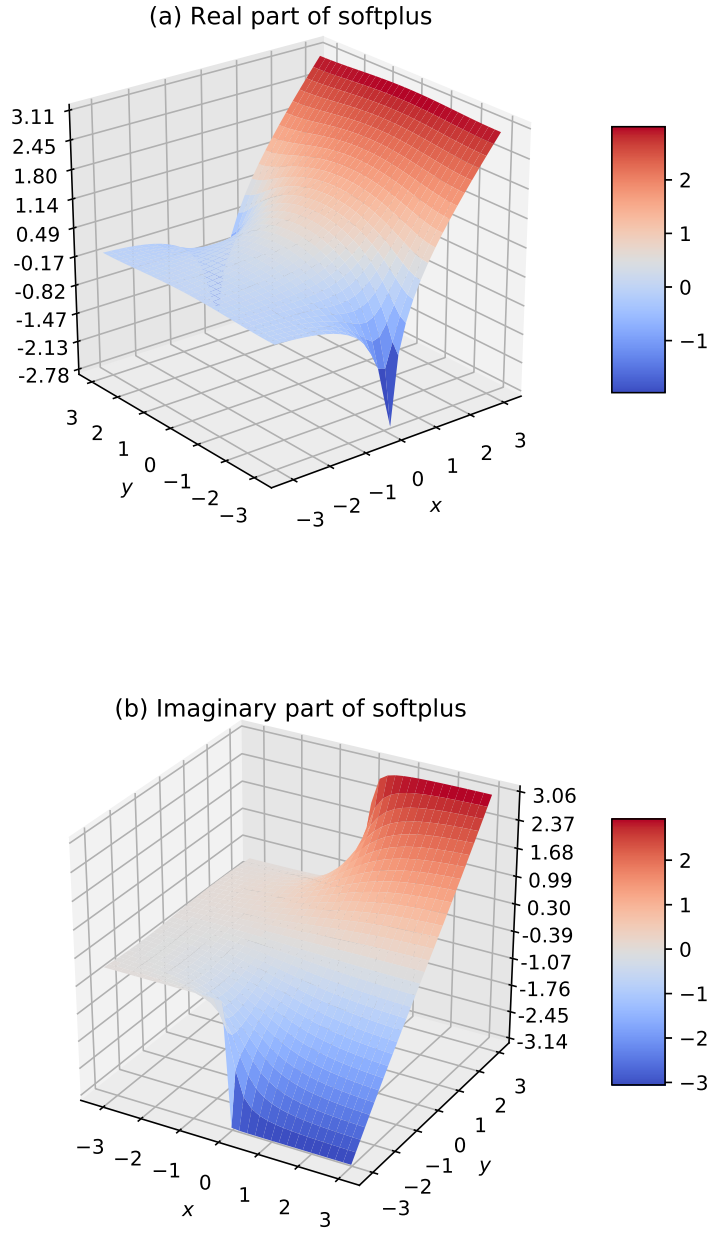

Figure 3.3.: We show the surface plot of softplus activation in complex domain by separating the real (a) and imaginary part (b) of the activation.

4. Literature Review

As a new emerging method to solve quantum many-body problem, there is only a few related work. While most of the work done so far is with RBMQS, NNQS is only considered by Zi Cai and Jinguo Liu [5] and Hiroki Saito [72]. A short remark on their work is given. We start with an overview of some other NNQS-related numerical methods and the J_1-J_2 model we tested on.

J_1-J_2 model

We consider previous results for J_1-J_2 model as baseline for comparison. For the convenience of comparison, we consider 8×8 lattice with $J_2/J_1 = 0.5$, where numbers for DMRG [28], finite PEPS [96], VMC with Lanczos [44] are available. These finite size results are consistent and difference in variational energy per site is smaller than 10^-3. However, they suggest different nature of the ground state for $J_2/J_1 \approx 0.5$. VMC result [44] suggests a gapless $\mathcal{Z}_2$ spin liquid between $0.45 \leq J_2/J_1 \leq 0.6$. DMRG [28] predicts a plaquette VBS phase between $0.5 < J_2/J_1 < 0.61$. In [96], they found a paramagnetic ground state with exponentially decaying spin-spin correlation in between $0.572 < J_2/J_1 \leq 0.6$.

Depends on the numerical methods, different scenarios are reported for J_1-J_2 model. The nature of the underlying state seems to be more difficult to understand with the advance of competitive method with almost the same variational energy. For recent work in DMRG and iPEPS, see [97] and [35]. Though the predictions of the nature for underlying state are different, the consistent variational energies are still a valid baseline for testing new numerical method for frustrated quantum spin systems.

SBS

Since RBMQS is a subset of SBS, it is reasonable to compare the previous result with SBS. In [76, 78], SBS is applied to two dimension and three dimension frustrated systems. In [76], frustrated XX model on 2D lattice with open boundary conditions (OBC) and periodic boundary conditions (PBC) are tested, where for the OBC cases the result is better than PEPS and for the PBC cases there is no other comparable methods. The result for 2D Ising model with transverse field with PBC is shown with a relative error of the order 10^{-3} using bond dimension $D = 2$. In [78], 2D J1-J2 model with lattice size up to 10×10, 3D Ising model with transverse field on $8 \times 8 \times 8$ PBC lattice, and 3D frustrated XX model in a transverse field are considered. For both OBC and PBC, the 2D J1-J2 model results comparing to PEPS data give relative error range around 2×10^{-2} to 8×10^{-2}. This implies these states are still away from ground state. In our result, we see that RBMQS and NNQS could reach a better results for PBC comparing with SBS.

RBMQS

Since the original work for RBMQS on Heisenberg model [9], theoretical works [12, 23,

16, 45] are done to show the connection of RBM and Tensor Network and suggest the powerful representation of deep Boltzmann Machine (DBM). However, there is no efficient training method for DBM. On the other hand, RBMQS has been generalized for fermionic systems with the name F-RBM and applied to Hubbard model [63]. The F-RBM is a projector formed with two RBM over some reference state. The F-RBM with pair-product wavefuncion is shown to be competitive wavefunction for studying strongly correlated fermionic system.

Because of the non-local geometry of the RBMQS, it is shown that RBMQS could approximate a chiral spin liquid with better accuracy comparing to EPS and SBS [26].

NNQS

The main goal of this thesis is to apply NNQS to high dimensional $d \geq 2$ frustrated problem. NNQS could also be applied to problem without sign-problem. Hiroki Saito performed VMC with fully-connected neural network for Bose-Hubbard model [72]. Their results are in good agreement with exact diagonalization and the Gutzwiller approximation.

For problem with sign problem, Zi Cai and Jinguo Liu [5] consider supervised learning with fully-connected neural network for J_1-J_2 in one dimension for up to three layers. They also performed VMC optimization with 1-hidden layer fully-connected neural network. There are several drawbacks in their approach.

In supervised learning task, they separate quantum amplitude into sign and magnitude. Two *independent* networks are trained separately. While the prediction of quantum amplitude is essentially a regression problem, the separation of one regression problem into a regression problem for magnitude and a classification problem for sign has several problems.

Firstly, we can not put back two problems to one single regression problem. That is after training separately and has a good enough result, one can not continue the training with two networks jointly by supervised learning or VMC optimization. This is because for the classification network (sign-ANN) to work, a step-function is applied. Gradient descent could not be applied. If one considers to apply sigmoid/cosine function instead, the output of the regression network for magnitude is then affected by the multiplication of sigmoid/cosine function. Weights in the regression network for magnitude would need to learn again and will end up in a different network. This means we can not start with some known physical state from other methods, perform supervised learning to have a good starting point, and then do VMC optimization in the final stage. In the our setup, supervised learning over the network could be perform directly.

Secondly, it is observed in practice that having one neural network to learn multiple tasks could usually have better performance. By one neural network, we mean a neural network consisted of base network and multiple heads [82, 39].

In addition, forcing the network to only learn the "sign" limits the generalization to model with complex Hamiltonian. By modifying the classification network to learn the phase, the problem becomes a regression problem. In the end, we see we better try to learn the phase and magnitude as regression problems together.

Part III.

Results and Discussion

5. Results and Discussion

In this thesis, we focus on spin model on 1-dimension ring and a 2-dimension torus. The Hamiltonian for anti-ferromagnetic Heisenberg and J1-J2 model are defined as,

$$\mathcal{H}_{\text{AFH}} = J \sum_{\langle ij \rangle} [\sigma_i^x \sigma_j^x + \sigma_i^y \sigma_j^y + \sigma_i^z \sigma_j^z] \tag{5.1}$$

$$\mathcal{H}_{J_1\text{-}J_2} = J_1 \sum_{\langle ij \rangle} [\sigma_i^x \sigma_j^x + \sigma_i^y \sigma_j^y + \sigma_i^z \sigma_j^z] \tag{5.2}$$

$$+ J_2 \sum_{\langle\langle ij \rangle\rangle} [\sigma_i^x \sigma_j^x + \sigma_i^y \sigma_j^y + \sigma_i^z \sigma_j^z] \tag{5.3}$$

where $\langle \cdot \rangle, \langle\langle \cdot \rangle\rangle$ denote the nearest neighbor and next nearest neighbor on the lattice with periodic boundary condition.

5.1. Implementation

For the study in this thesis, VMC and supervised learning code for general NNQS was developed. The guide to the code is given in the Appendix A.1. The implementation is based on Python with libraries including NumPy, SciPy, and TensorFlow [1]. While NumPy and SciPy are standard numerical libraries in python, TensorFlow is an open source numerical library with reverse-mode automatic differentiation. It was originally developed by researchers and engineers working on the Google Brain Team for neural networks. The analytic formula or symbolic differentiation for derivative quickly become intractable for neural networks and numerical differentiation is not accurate and not efficient. Modern deep learning libraries, e.g. TensorFlow, PyTorch, CNTK, Theano, are all designed with automatic differentiation. These libraries usually have highly optimized tensor computation implementation on CPU and GPU. For using TensorFlow, one must first define the complete computation graph [1] and during run-time the data would "flow" forward and backward in the graph. The overhead with python is usually negligible since we keep almost all the operations in TensorFlow.

Given the enormous amount of effort deep learning community devoted in developing TensorFlow, there are still some cases where the functionality needed for physics is not built in. Here we briefly state the difficulties.

[1] We define the network in Class tf_network.

- To construct the S matrix in SR method, we have to approximate the expectation value of the outer product of the gradient. That means we must have unaggregated gradient (batch gradient), which is an ill-defined concept in terms of automatic differentiation. One could only compute the derivative per configuration at a time, which is slow. There are different ways to speed this up [29]. In the implementation, we exploit the parallelization in "while_loop" function [2].

- Many function does not support complex type. One has to manually split and operate on the real and the imaginary part. The derivative of a complex-valued function parametrized by real variables in TensorFlow, would be automatically casted back to real. This is done because it is expected that gradient is taken from a real-valued function. However, for the calculation of $\langle \Delta^* \rangle$ of a complex-valued wavefunction, one then need to split the real and the imaginary part. Basically, special care need to be taken with the complex derivative used.

- To implement translational invariant wavefunction, it involves the circular convolution. Such operation is not supported since images are not considered to be periodic, and convolution is used with valid padding or zero padding. One could implement periodic padding with convolution which gives the result of circular convolution [3]. Another alternative way of doing this is to use fast Fourier transform (FFT) [55], since the circular convolution is the matrix multiplication in Fourier space. However, it is observed that for the usually small filter size we considered the performance is worse than periodic padding.

5.1.1. Wavefunction evaluation with level 3 parallelization

In NNQS, the evaluation of wavefunction is based on forward pass of the neural network. Depending on the size of network, the input could take batch size up to hundreds or thousands. This suggest a parallelization in amplitude evaluation, which is different from the usual parallelization scheme for VMC calculation.

For most of the VMC wavefunction, usually only one configuration could be evaluated at a time. Parallelization is made by having multiple walkers on different processors. Communication is needed to (1.) collect the local energy and log derivative (2.) compute the final gradient and update parameters for all walkers. If the number of parameters is not large, the bottleneck would be in the calculation from the walkers. Weak scaling is expected, while strong scaling should hold till the computation bottleneck is not in the walkers.

Now, we can keep the parallelization between different processors, but exploit the parallelization in a lower level, e.g. level-3 BLAS operation, by evaluating the amplitudes in batch. We exploit the speed up with coarse-grain parallelism. While the evaluation of

[2] See build_unaggregated_gradient method in class tf_network.
[3] This is done in "network\tf_wrapper".

single configuration involves matrix-vector multiplication, of which the computation to memory access ratio is about $\mathcal{O}(1)$, the evaluation over batch of configurations involves matrix-matrix multiplication, of which the computation to memory access ratio is about $\mathcal{O}(N)$ [18]. This can easily speed up memory bound computation. Other advantages include parallelization with blocks operation and avoiding function call overhead in python.

5.1.2. GPU Speed up

With the implementation based on TensorFlow, one can easily switch using GPU for computation simply by running the code on machine with GPU without changing the code. A significant speed up is observed for deep NNQS. For MCMC sampling and VMC without SR method, a factor of 10 speedup is observed for standard deep neural network architecture like ResNet10 [41], which is a neural network with 20 layers, and 746434 number of parameters in total. However, we could not obtain meaningful VMC result simply without SR method.

5.2. VMC Result

The optimization in VMC is quite tricky. We first describe the network architecture design and the argument behind using exponential in the output unit in 5.2.1 and we discuss some detail for obtaining stable VMC result in 5.2.2. The final result of VMC with NNQS is given in 5.2.3.

5.2.1. Distribution of the coefficients of eigenvectors

There are several important questions one should ask before performing any machine learning techniques. What are (the distribution of) input data and output data? Is my method the correct way to describe it? We always need to change our models according to the data given. We might even need to change the loss function in the case when the data is imbalanced.

What is the typical distribution of the coefficients of the eigenstate, especially the lowest eigenstate, look like? To answer this question, we diagonalize the Hamiltonian exactly on a 4×4 lattice with periodic boundary condition and plot the distribution of the coefficients for Heisenberg model and J_1-J_2 model with $J_2/J_1 = 0.5$.

Without considering the symmetries, the distributions of all the coefficients are of size $\dim(\mathcal{H}) = 65536$ and are shown in figure 5.1 and 5.2. Both distributions have peaks center around zero, where most of the coefficients lies in. After zooming in, we could see there are a few coefficients of order $\mathcal{O}(10^{-1})$. For example, there are actually two largest coefficients corresponding to the Neel state. It is difficult to learn this given that it appears only 2 times out of 65536. However, these "outlier" in the distributions accounts for the actual physics.

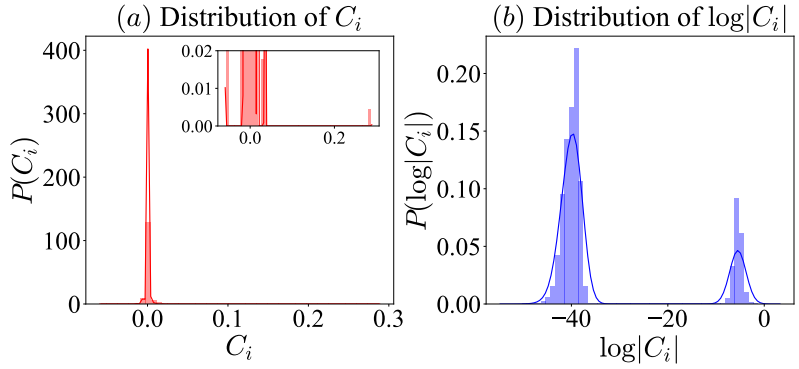

Figure 5.1.: The distribution of (a) the coefficients and (b) the logarithm of the absolute value of the coefficients. Here, we consider the ground state of Heisenberg model.

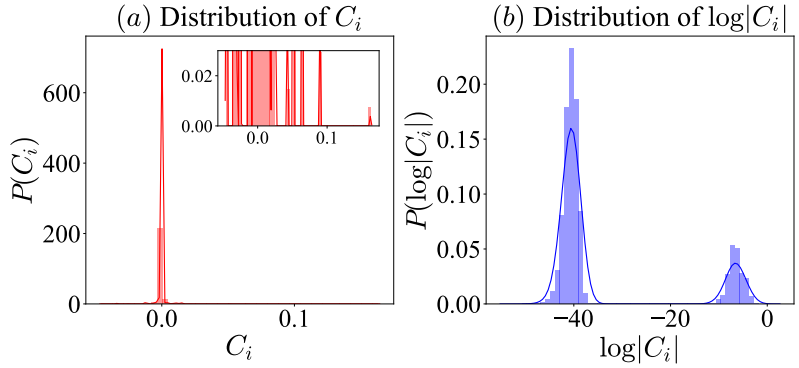

Figure 5.2.: The distribution of (a) the coefficients and (b) the logarithm of the absolute value of the coefficients. Here, we consider the ground state of J_1-J_2 model with $J_2/J_1 = 0.5$.

One solution to this problem is to take the logarithm. If we do not consider the sign of the coefficients, we see that the logarithm of the coefficients are distributed around two positions. The peak around -40 are the numerical zeros, which are about a factor of machine precision in double precision smaller than the larger coefficients. The peak closer to zero gives the physical amplitudes. This seems to be a possible distribution that neural network could predict. As we will see in the experiments, it is better to predict the magnitude of the amplitude in log basis.

For the J_1-J_2 model, the Hamiltonian is a block matrix, where the ground state is at total spin $\sum_i \sigma_i^z = 0$ sector. The total spin zero sector is of dimension 12870. We plot the distribution of the coefficients in this subspace in figure 5.3 and 5.4.

We see that the numerical zeros disappear after restricting to the subspace. That is to say, the coefficients are all of finite magnitude, hence not sparse, in the restricted subspace.

The distribution without logarithm transform is almost the same as before. The histogram suggest that it should be easier to work in the log space since the distribution is closer to normal distribution.

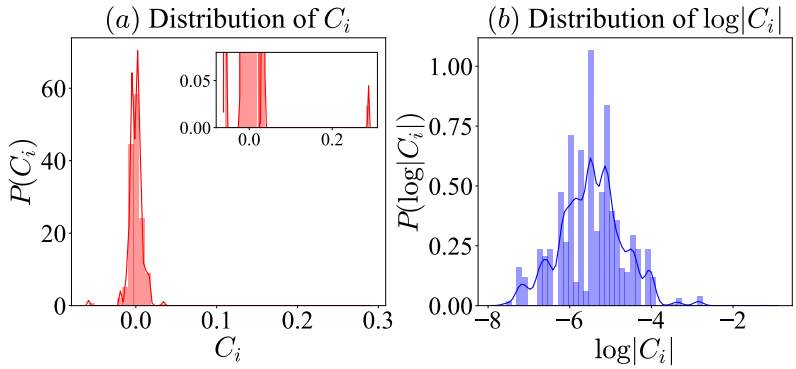

Figure 5.3.: The distribution of (a) the coefficients and (b) the logarithm of the absolute value of the coefficients. Here, we consider the ground state of Heisenberg model.

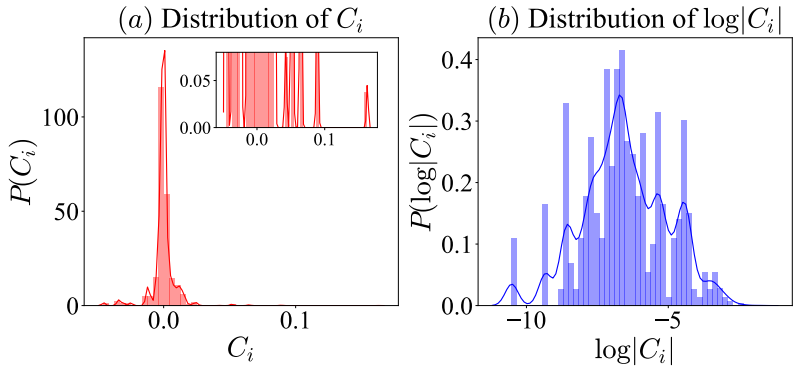

Figure 5.4.: The distribution of (a) the coefficients and (b) the logarithm of the absolute value of the coefficients. Here, we consider the ground state of J_1-J_2 model with $J_2/J_1 = 0.5$.

Numerical experiment We test the effect of log transform on the amplitude by comparing two 1-hidden layer neural networks. The architecture of the network is shown in figure 5.5 The ground state for testing is the Heisenberg model on 4×4 lattice with periodic boundary condition.

The network has 16 input units and 64 hidden units with tanh as activation. For a linear network, we have one output unit for real number. To work in the logarithm space, we have two output units, one predicting the magnitude, r, and the other predicting the phase,

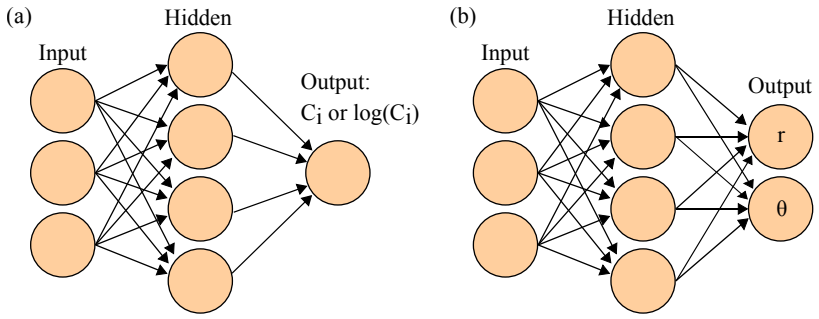

Figure 5.5.: (a) The architecture of real-valued 1-hidden layer neural networks with linear output. The output is taken as C_i, the coefficient, which is real in this setup. (b) The architecture of real-valued 1-hidden layer neural networks with outputs as magnitude r and phase θ. The coefficient is given by $C_i = \text{Re}\,[e^{r+i\theta}]$. Note that for complex-valued neural networks predicting magnitude and phase, the network structure would look like (a), where the output complex number is interpreted as the complex number $\log(C_i)$.

θ. The output is then, Re $[e^{r+i\theta}]$. The number of parameters are 1153, and 1218 respectively. Network is trained with stochastic reconfiguration, constant damping factor 10^{-4}, and gradient descent update rule with stepsize 3×10^{-3}. Best results are reported from 10 different initialization runs. The result is given in figure 5.6

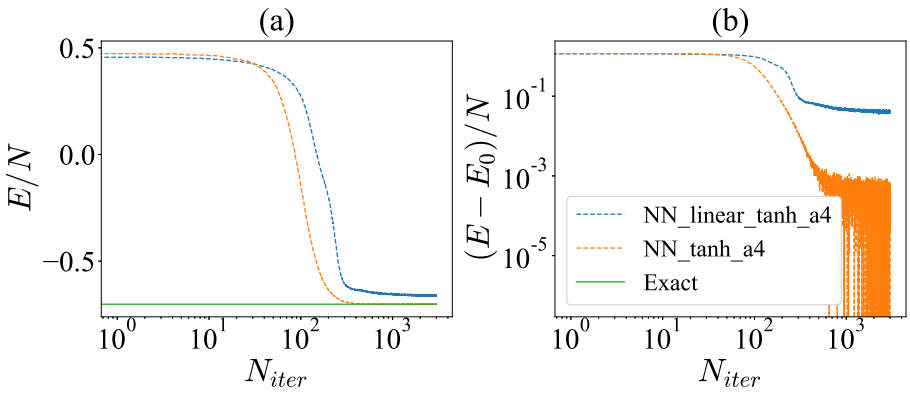

Figure 5.6.: Convergence plot: The energy per site is plotted against the number of iteration. The network with exponential output unit could converge to the ground state, while the linear output network fails.

Lesson learned here is that one should restricted to subspace if possible. take log.

5.2.2. Optimization methods

We discuss some general tips on how to train the network to convergence.

1. **Special care is taken to ensure numerical stability of exponential output**. In most of the neural networks we considered, including RBM, a global sum pooling is applied following an exponential operation in the final stage. The magnitude r is an extensive quantity. It grows with the system size and also the width of the network. For single precision floating point number, overflow occurs for $\exp(x)$ with roughly $x > 89$, which is not a big number consider a system size of 10×10.

   We consider three different ways to deal with the problem: **(1) Clipping the magnitude, (2) subtracting an stabilizer, (3) working with log magnitude**. (3) seems to be most natural solution in terms of numerical properties and exactness, while (1) is stabler in some cases.

   To clip the magnitude, one setup a range according to the precision one works with, e.g. $[-70, 70]$ and clip the magnitude once the magnitude is outside the range. This avoid the overflow or underflow, but make the gradient of configurations with clipping vanish. After some coefficient touch the ceiling, the other should start getting gradient for shrinking magnitude. The advantage of this approach is that for any configuration input, the output would be a valid number. One can easily restart the Monte Carlo sampling without running into problem.

   The fact that the magnitude is extensive is similar to the energy of the configuration in classical Monte Carlo calculation. For the computation of VMC, only the difference between two configuration matters. As a result, we can store the largest magnitude for each VMC update, and subtract the magnitude by the largest magnitude we have in the previous run. This will make almost all coefficient become smaller than one and is than stable. We do not need to set an upper bound. However, problems may occur when one naively restart the Monte Carlo sampling from random initialization. Since the largest magnitude from previous run might be large, and the random initialized configuration might have very small magnitude, underflow would occur and Metropolis update would fail. Special care need to be taken in latter stage of optimization if one want to restart the Monte Carlo sampling.

   The second methods fail because we are still working with the exponential unit before the division between two configurations. The natural way to avoid this problem is to work with exponential of the difference instead, i.e.:

$$\frac{\langle\alpha|\psi\rangle}{\langle\beta|\psi\rangle} \longrightarrow \exp\left[\log(\langle\alpha|\psi\rangle) - \log(\langle\beta|\psi\rangle)\right].$$

   By this, we could have a stable algorithm in the Metropolis update. Note that even though complex-valued output is natural in this setting, we could still work with real-valued output with the phase being π or $-\pi$.

2. **Good initialization**. With a good initialization scheme, for random configurations the distribution of the amplitudes should be roughly randomly distributed around origin. In terms of real amplitudes, this means we will have positive and negative values. This is crucial for optimization. It is usually enough by following the standard neural network initialization scheme.

3. **Stochastic reconfiguration is necessary for convergence to good local minimum.** Results from numerical experiment suggests that none of the first-order methods converges to good local minimum. With SR method, wavefunction usually converges to a different and better local minimum. See figures 5.7, 5.8 and discussion of setup of the experiment below.

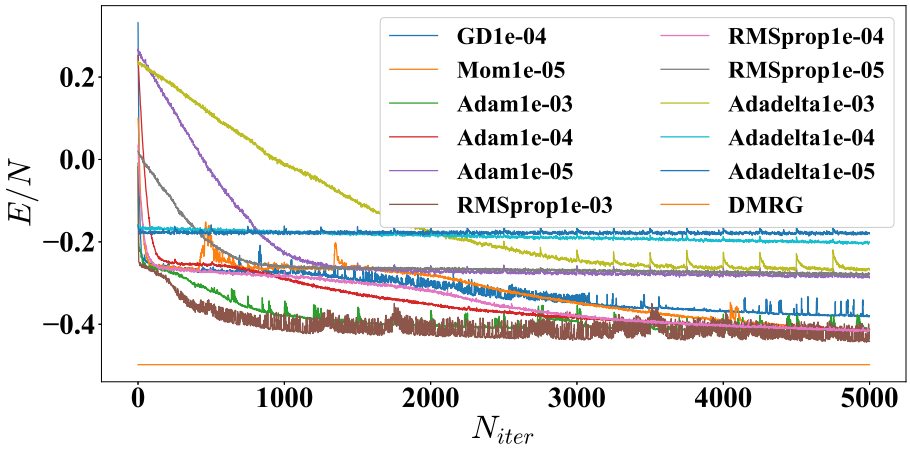

Figure 5.7.: Convergence plot: The energy per site is plotted against the number of iteration. The label shows the name of the method and the stepsize. For example, RMSprop1e-03 is the RMSprop method with stepsize 10^{-3}.

4. **Restart Monte Carlo Sampling**. It is observed that variational energy in VMC may jump to a higher value, when one restart the Monte Carlo walkers after a long VMC optimization. This happens for frustrated problem in larger system size. One will observe the new walkers in configuration with large ferromagnetic domain. A possible explanation is that since RBMQS or NNQS are flexible and extremely large output is allowed, for certain favorable configuration the magnitude would grow large very fast. This effectively creates disconnected peaks in the wavefunction, which results in configuration space not being ergodic anymore because of the exponential barrier of the deep valley. New walkers however might land on some disconnected part in the configuration space, which is left out in previous VMC optimization. This would then usually lead to a jump in the variational energy.

   To prevent such behaviour, it is recommend to introduce new random walkers after certain period of time. The cost for warming up the walker is acceptable if this is

not done in every update. In this thesis, we re-initialize all walkers after hundreds of updates.

5. **Trust Region method** For deeper networks, it is observed that it is helpful to adopt the stepsize selection based on argument from trust region [101, 3]. While large updates in wavefunctions would often ruin VMC iteration, conservative and small updates slow down the convergence. By stepsize selection, we can speed up the small update that occur at early stage of deep network optimization. For detail of the stepsize selection method, see 2.2.5.

6. **Multiple runs**. Even with all the correct setup, different random initialization usually lead to different final result. By different final result, we mean that the difference in variational energy per site is greater than roughly 2%. For a simple problem, e.g. 4×4 Heisenberg model, more than 70% of the runs could converge to the ground state. But for harder problem, it might happen that less than 30% converge to the best energy one could obtain. So for all the result reported, we have all at least run the same optimization 10 times.

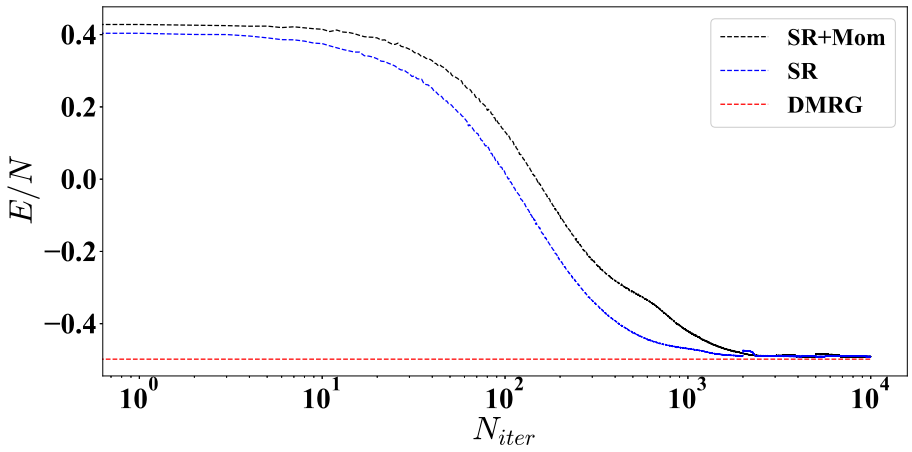

Figure 5.8.: Convergence plot: The energy per site is plotted against the number of iteration. The blue line is optimized with the SR method with SGD-like update and a stepsize ranging from 10^{-3} to 10^{-4}. The black line is optimized with the SR method with Momentum-like update and a stepsize ranging from 10^{-4} to 10^{-5}.

In the following, we demonstrate the VMC optimization for J_1-J_2 model on 8×8 lattice with $J_2/J_1 = 0.5$. While the problem itself is hard, we consider a relatively simple wavefunction, the 2-site translational invariant RBM (tRBM). The complexity of the wavefunction is controlled by the number of channels α. Here, we consider $\alpha = 8$, which has

1048 variational parameters. For both first-order methods and SR method, we consider 5000 Monte Carlo samples per update. Best results are reported from 10 runs.

For first-order methods, we consider the vanilla stochastic gradient descent (GD), stochastic gradient descent with Momentum method (Mom), Adam, RMSprop and Adadelta with various stepsizes. The result is shown in figure 5.7. We see that with first-order method, we could not reach comparable result with DMRG method. This is not because the wavefunction chosen is incapable of representing the state. We see in the optimization with SR method that this is indeed a problem of optimization. We are not able to reach a good local minimum with first-order methods even with only 1048 variational parameters.

The plot includes 5000 updates. For more updates with same stepsize or smaller stepsize, the results remain the same. Notice that for first-order method, the results look promising in the beginning stage of the optimization. This is consistent with the suggestion in [26], which recommend to use a large regularization/damping for the beginning stage of the optimization.

In contrast, the stochastic reconfiguration method shows better optimization results. In figure 5.8, we see that for both SGD-like update or Momentum-like update, we could have comparable result with DMRG with 1.5% difference in variational energy.

5.2.3. VMC with NNQS

Now, we give the main result of this thesis. We consider VMC with three different NNQS on 8×8 lattice for J_1-J_2 model with $J_2/J_1 = 0.5$ and PBC. The first network is 1-hidden layer neural network with real weights (NN). The activation function is tanh and with two real value output representing phase θ and magnitude r. The wavefunction is then $\mathrm{Re}[e^{r+i\theta}]$.

The second network is 2-site translational invariant RBM (tRBM) with filter size the same as system size, i.e. 8×8. We consider complex weight and softplus2 activation function. The third network is two-hidden layer fully convolution network (FCN2), which repeats the circular convolution in first layer of tRBM in its second layer. The filter size for both layers of FCN2 are 4×4, but with stride $= 1$ in the first layer, and stride $= 2$ in the second layer. The number of channel is doubled in the second layer.

After VMC optimization, the best variational energy per site by tRBM is $E_{tRBM}/N = -0.4924$ and FCN2 $E_{FCN2}/N = -0.4929$. This is about 1% higher than variational energy obtained by other methods. This result is however better than SBS result reported in [78]. The accuracy becomes higher with the increase of number of parameters but we observe the gain decreases because the hardness of optimization. In addition, the accuracy becomes higher with the increasing depth of the network. However, our result is limited to two hidden layer networks, since the optimization for three hidden layer networks is too high.

We could observe patterns in the weights of the NNQS. For the visualization plot of the weights in NNQS see Appendix B.

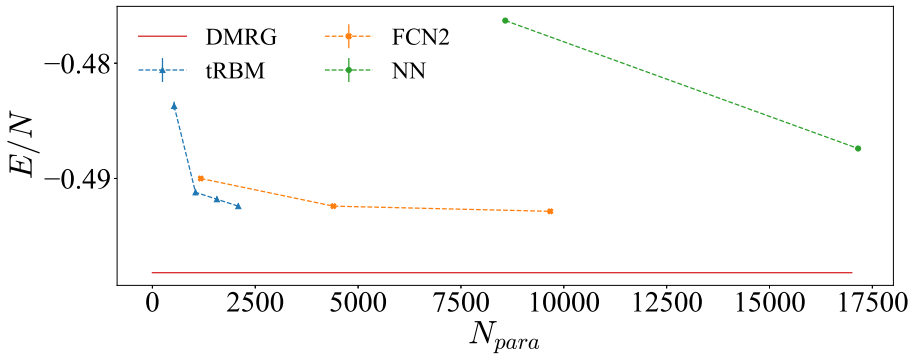

Figure 5.9.: Variational energy per site versus number of variational parameters. The baseline is the DMRG result in [28].

5.3. Supervised Learning Result

We consider the supervised learning for quantum amplitude as a regression problem. The input is the spin configuration in computation basis $\boldsymbol{\sigma}$ [4]. The output is the quantum amplitude $\psi(\boldsymbol{\sigma}) = \langle \boldsymbol{\sigma}|\psi\rangle \in \mathbb{R}$, since we consider real Hamiltonian.

There is some subtlety in defining the cost (risk) function. Our goal is simple that we want to maximize the fidelity $\mathcal{F}$ or its square.

$$\mathcal{F}(|\psi_1\rangle, |\psi_2\rangle) = \sqrt{\frac{\langle\psi_1|\psi_2\rangle \langle\psi_2|\psi_1\rangle}{\langle\psi_1|\psi_1\rangle \langle\psi_2|\psi_2\rangle}}$$

However, for unnormalized wavefunctions, such quantities is global and does not have meaning for specific configuration. This is made explicit by looking at a target wavefunction $|\psi_1\rangle$ and a real variational wavefunction $|\psi_2\rangle$,

$$\mathcal{F}(|\psi_1\rangle, |\psi_2\rangle) = \frac{\langle\psi_1|\psi_2\rangle}{\sqrt{\langle\psi_1|\psi_1\rangle \langle\psi_2|\psi_2\rangle}} = \sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma}) \frac{\langle\boldsymbol{\sigma}|\psi_2\rangle}{\langle\boldsymbol{\sigma}|\psi_1\rangle} \sqrt{\frac{\langle\psi_1|\psi_1\rangle}{\langle\psi_2|\psi_2\rangle}} \tag{5.4}$$

Recall the definition (3.1), the loss function could be $\frac{\langle\boldsymbol{\sigma}|\psi_2\rangle}{\langle\boldsymbol{\sigma}|\psi_1\rangle}$, if the variational wavefunction and the target wavefunction are both normalized. However, the variational wavefunction is usually not normalized. As a result, no corresponding loss function for fidelity could be defined, even assuming the target wavefunction is normalized. This is because $\langle\psi_2|\psi_2\rangle$ is a global quantities defined on all configurations, which is intractable.

If we could not formulate the loss function, that means we can not perform stochastic gradient descent for optimization and thus we can not work with large problem that we

[4] Note that is equivalent to only taking half of the one-hot encoding. This is because of the redundancy in one hot encoding for categorical data with two classes, i.e. spin-1/2.

are interested in. Though we could not work with fidelity directly, we can define some other objective functions such that by minimizing those objective functions, the fidelity would be maximized. We point out three different approaches.

One simplest approach that would be simply consider the L_2 loss, $|\langle \boldsymbol{\sigma}|\psi_1\rangle - \langle \boldsymbol{\sigma}|\psi_2\rangle|^2$. If the data comes from a uniform sampling, this is the same as minimizing the distance in Hilbert space. The problem with this approach is that most coefficients are small, as we see in figure 5.4. Directly applying this method will typically result in a machine learned to output zero regardless of the input. To address the issue of imbalanced data, in [5], they consider training with sampling from the target wavefunction and modifying the output label to $\sqrt{\mathbb{D}}\langle \boldsymbol{\sigma}|\psi\rangle$, where $\mathbb{D}$ denote the Hilbert space dimension.

The second approach is to work with cosine similarity defined on a batch of data. That is,

$$-\mathcal{F}_{batch}(|\psi_1\rangle, |\psi_2\rangle) = -\frac{\sum_{\boldsymbol{\sigma}\in batch}\langle\psi_1|\boldsymbol{\sigma}\rangle\langle\boldsymbol{\sigma}|\psi_2\rangle}{\sqrt{\sum_{\boldsymbol{\sigma}\in batch}\langle\psi_1|\boldsymbol{\sigma}\rangle\langle\boldsymbol{\sigma}|\psi_1\rangle}\sqrt{\sum_{\boldsymbol{\sigma}\in batch}\langle\psi_2|\boldsymbol{\sigma}\rangle\langle\boldsymbol{\sigma}|\psi_2\rangle}} \tag{5.5}$$

By minimizing this cost, we try to align some coefficients of the vectors at a time. It is observed that we could learn the full quantum amplitude with this cost function. We observe no difference in training with data sampled from uniform distribution or the wavefunction distribution.

5.3.1. From exact diagonalization

We demonstrate that supervised learning with batch cosine similarity as cost function for 1d J_1-J_2 model on 16 sites with periodic boundary condition and $J_2/J_1 = 0.2$. The ground state is obtained from exact diagonalization for full Hilbert space without restricting to symmetry subspace. We consider batch size of 500 and learning rate of 10^{-3}. The first example is with 1-hidden layer neural network with softplus2 activation and exponential output unit with Momentum method. The second example is tRBM with Adam. There is no special reason for the choice and the hyper-parameters are not tuned. The result is shown in figure 5.10.

The dimension of the Hilbert space is $\dim(\mathcal{H}) = 65536$. In this case, we can compute the exact (total) fidelity and it is also marked in the figure. The networks converge around 50000 iterations, which is about ~ 380 epochs. The large number in epochs needed might become a problem for training large problems.

It should be stressed that all hyper-parameters are not tuned in this experiment. There is no specific reason for choosing Momentum method and Adam. With tuning the learning rate, it is possible to have a faster convergence. The training is quite robust. The only problem observed is the increasing of magnitude in the wavefunctions when the stepsize is too large, which results in overflow. Regularization on the weight might help in this case but is not tested.

We further demonstrate as in the inset of figure 5.10 that it is easy to combine VMC with supervised learning result. Note that this is different from [5]. With the combination

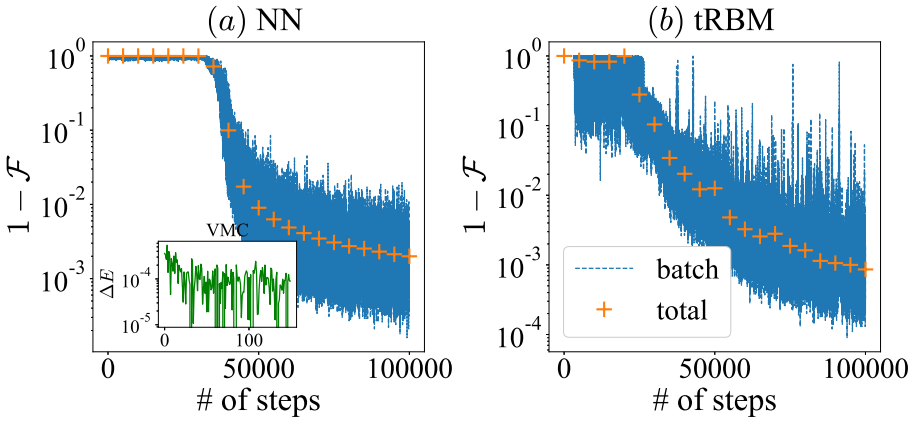

Figure 5.10.: Supervised learning with (a) NN (b) tRBM. We minimize the negative batch fidelity and effectively minimize the negative (total) fidelity between the target wavefunction and variational wavefunction. In (a) inset, we show the error in variational energy with VMC optimization after supervised learning.

of VMC and supervised learning, it is possible to use supervised learning to train the wavefunctions with previous results to a good initial point in the parameter space, and VMC to fine tune and reach even better result.

One could use direct sampling from the explicit eigenvector given to generate batch of training data for supervised learning. For larger problem, where the exact eigenvector could not be formed, one could use MCMC sampling from the wavefunction. We describe an alternative approach based on MCMC sampling.

5.3.2. From Monte Carlo sampling

The third approach actually considers directly the fidelity as cost function. However, the problem is formulated by considering sampling from the target wavefunction and the variational wavefunction. Consider (2.67), we see we can actually define a loss function

$$L = -\frac{\langle \boldsymbol{\sigma}|\psi_1\rangle}{\langle \boldsymbol{\sigma}'|\psi_1\rangle}\frac{\langle \boldsymbol{\sigma}'|\psi_2\rangle}{\langle \boldsymbol{\sigma}|\psi_2\rangle}, \tag{5.6}$$

where $\boldsymbol{\sigma} \sim P_1(\boldsymbol{\sigma}), \boldsymbol{\sigma}' \sim P_2(\boldsymbol{\sigma}')$.

By minimize this loss function with sampling from the target wavefunction and variational wavefunction, we direct maximize the fidelity.

5.4. Discussion and Future work

In this thesis, we review the methods in machine learning and physics that are related to VMC. We provide implementation based on TensorFlow for supervised learning and VMC

with NNQS. The implementation supports both CPU/GPU backends. The result of NNQS with VMC on 2d J_1-J_2 model suggests that it is not yet but a promising method for high dimensional frustrated problem.

Feed-forward neural network, especially convolution neural network, with various different activation could reach an reasonable result with VMC. It is robust to the structure chosen. It suggests that it is very likely that general neural networks cover the space of physical states.

However, the optimization is hard. We show example that with 1-hidden layer neural network, with first-order method, one can not find the good local minimum in the variational space. We suggest two different ways to circumvent the difficulty that is supervised learning and K-FAC.

We formulate the supervised learning with batch fidelity as cost function. It is shown that one could learn the quantum amplitude directly without splitting the sign and magnitude. This makes it possible to carry out VMC optimization after supervised learning. With supervised learning as a good initialization scheme for VMC optimization, it is possible that one could tackle hard problem with increasing number of parameters.

The increasing number of parameters with increasing depth could also be solved by introducing approximated SR method. K-FAC is a promising approximation for Fisher information matrix for neural networks. Since for ground state calculation in VMC we do not need the exact overlapping matrix, it is possible to combine K-FAC and VMC with NNQS for approximated SR method.

Due to the time constraints, many interested topics remain unexplored and left for future work. This includes the two methods above for extending VMC to deeper networks, studying the effect of activation functions, depth of the networks and regularization.

Appendix

A. Neural Network Quantum State (NNQS)

A.1. Guide

To run the project, first clone the repository from github.

git clone https://github.com/ShHsLin/Neural-Network-Quantum-State.git

Requirements: python 2.7 (may not support some operation), python >=3, TensorFlow >=1.4, Numpy, Scipy

Commands: To see the help list as in A.2, and run the VMC of NNQS, one can run

```
# Show the help list
python train_NQS.py --help
# Run VMC for 2d AFH on 4x4 lattice, 2000 MC sampling per update
# for 100 steps
python train_NQS.py --l 4 --dim 2 --num_sample 2000 --num_iter 100
```

Source Code A.1.

More details are given in the README.

git clone https://github.com/ShHsLin/Neural-Network-Quantum-State

```
usage: train_NQS.py [-h] [--l L] [--net WHICH_NET] [--lr LR]
                    [--num_iter NUM_ITER] [--num_sample NUM_SAMPLE]
                    [--batch_size BATCH_SIZE] [--alpha ALPHA] [--opt OPT]
                    [--H H] [--dim DIM] [--J2 J2] [--SR SR] [--reg REG]
                    [--path PATH] [--act ACT] [--SP SP]
                    [--using_complex USING_COMPLEX]

Variational Monte Carlo with NNQS

optional arguments:
  -h, --help            show this help message and exit
  --l L                 system size. Default: 10
  --net WHICH_NET       Name of the Neural Network. Default: sRBM
  --lr LR               learning rate. Default: 1e-3
  --num_iter NUM_ITER   number of iteration for optimization. It is suggested
                        that the multiplication of learning rate and number of
                        iteration be around one, i.e. lr * num_iter = 1.
                        Default: 1500
  --num_sample NUM_SAMPLE
                        Number of sampling in Monte Carlo process. Default:
                        5000
  --batch_size BATCH_SIZE
                        Batch size in Network pretraining. Default: 500
  --alpha ALPHA         controll parameter for model complexity. Default: 4
  --opt OPT             optimizer for the neural network. Default: Momentum
                        method
  --H H                 target Hamiltonian for optimization. Default: AFH
  --dim DIM             Dimension of the system. 1d: chain, 2d: square
                        lattice.Input should be integer. Default: 1
  --J2 J2               The J2/J1 value in J1J2 model Default: 1.
  --SR SR               Using Stochastic Reconfiguration (SR) method or not.
                        Giving True(1) or False(0). Default: 1.
  --reg REG             Scaling factor for fixed scale weight decay for
                        regularization, s.t. gradient += scale * W. Default:
                        0.
  --path PATH           path to the directory where wavefunction and E_log are
                        saved. Default: ''
  --act ACT             nonlinear activation function in the networkDefault:
                        softplus2
  --SP SP               True(1): single precision, False(0): double precision
                        Default: True(1)
```

Source Code A.2.: The help list from "python train_NQS.py –help"

B. Visualization of Weights

We show the visualization of tRBM and FCN2 wavefunction for J_1-J_2 model on 8×8 lattice with $J_2/J_1 = 0.5$. The setup for the VMC optimization is as in 5.2.2. We could observe clear patterns in tRBM wavefunction. However, for a two-layer convolution network FCN2, the pattern is relatively obscure.

B.1. tRBM

The tRBM wavefunction with complexity $\alpha = 16$ has one convolutional layers and the filter size 8×8. The convolutional layer is with filter of dimension $[8, 8, 1, 16]$, which is a 8×8 size filter mapping from 1 channel to 16 channels. Ignoring the biases term, we visualize the real and the imaginary part of these 16 different 8×8 filters in figure B.1 and B.2.

The tRBM wavefunction also has an one-body bias term, which is equivalent to a convolution filter of the size of unit cell. The visualization of the bias filter are given in figure B.3 and B.4.

B.2. FCN2

The FCN2 wavefunction with complexity $\alpha = 12$ has two convolutional layers and the filter size 4×4. The first layer is with filter of dimension $[4, 4, 1, 12]$, which is a 4×4 size filter mapping from 1 channel to 12 channels. Ignoring the biases term, we visualize the real and the imaginary part of these 12 different 4×4 filters in figure B.5 and B.6.

The second convolutional layer is with filter of dimension $[4, 4, 12, 24]$. We also show the visualization of 288 filters in figure B.7 and B.8.

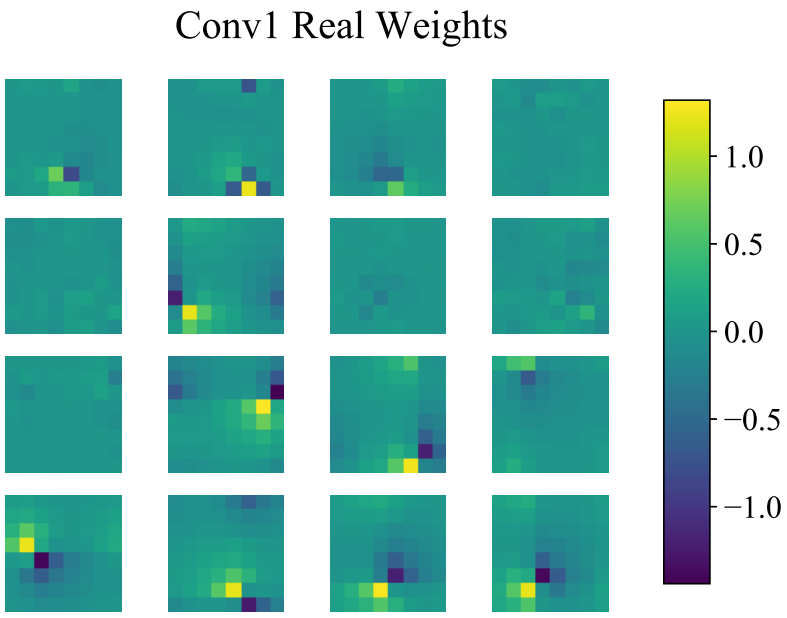

Figure B.1.: The visualization of the real part of the convolution kernel.

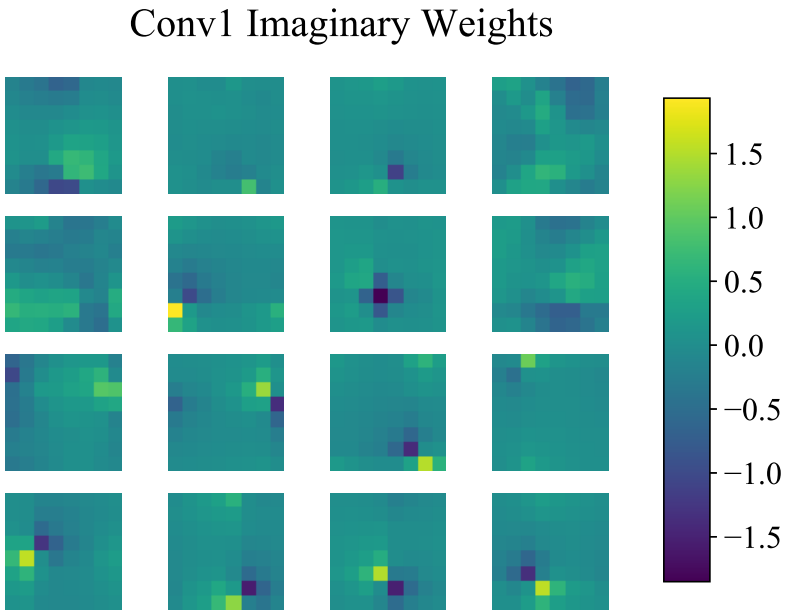

Figure B.2.: The visualization of the imaginary part of the convolution kernel.

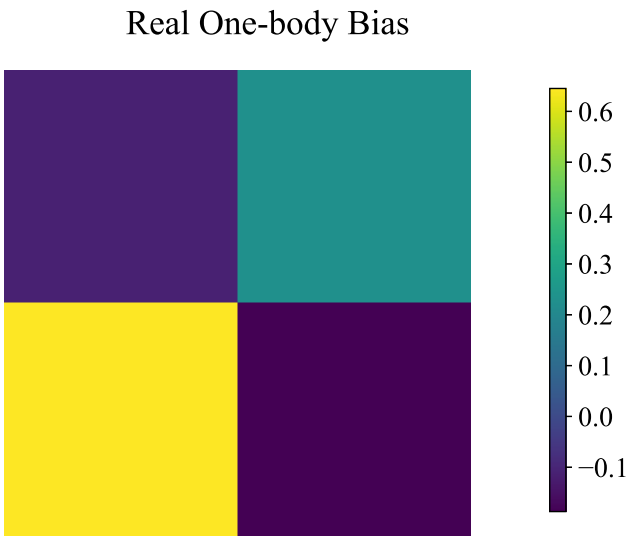

Figure B.3.: The visualization of the real part of the one-body bias term as convolution kernel.

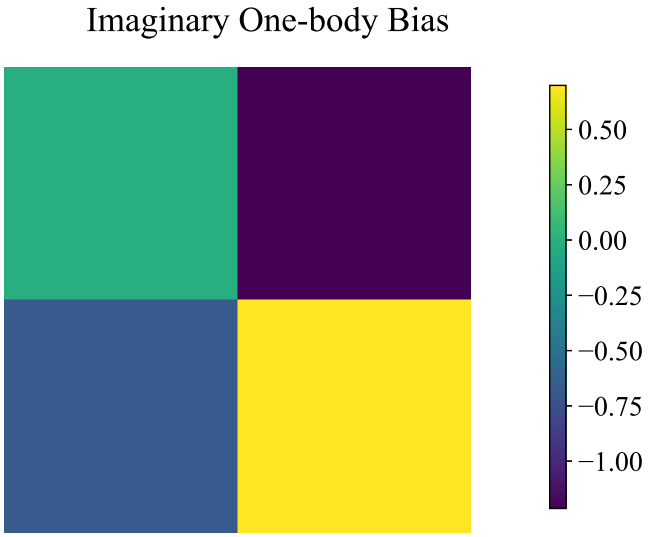

Figure B.4.: The visualization of the imaginary part of the one-body bias term as convolution kernel.

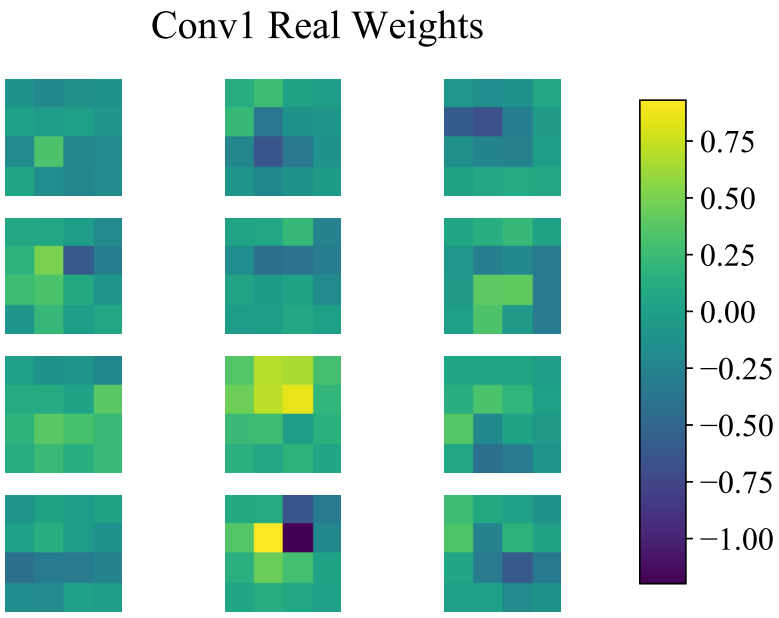

Figure B.5.: The visualization of the real part of the first convolution kernel.

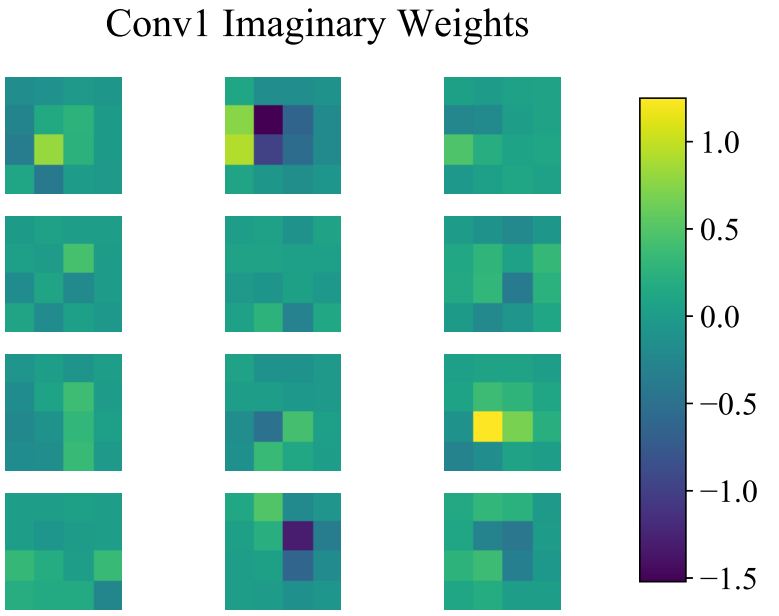

Figure B.6.: The visualization of the imaginary part of the first convolution kernel.

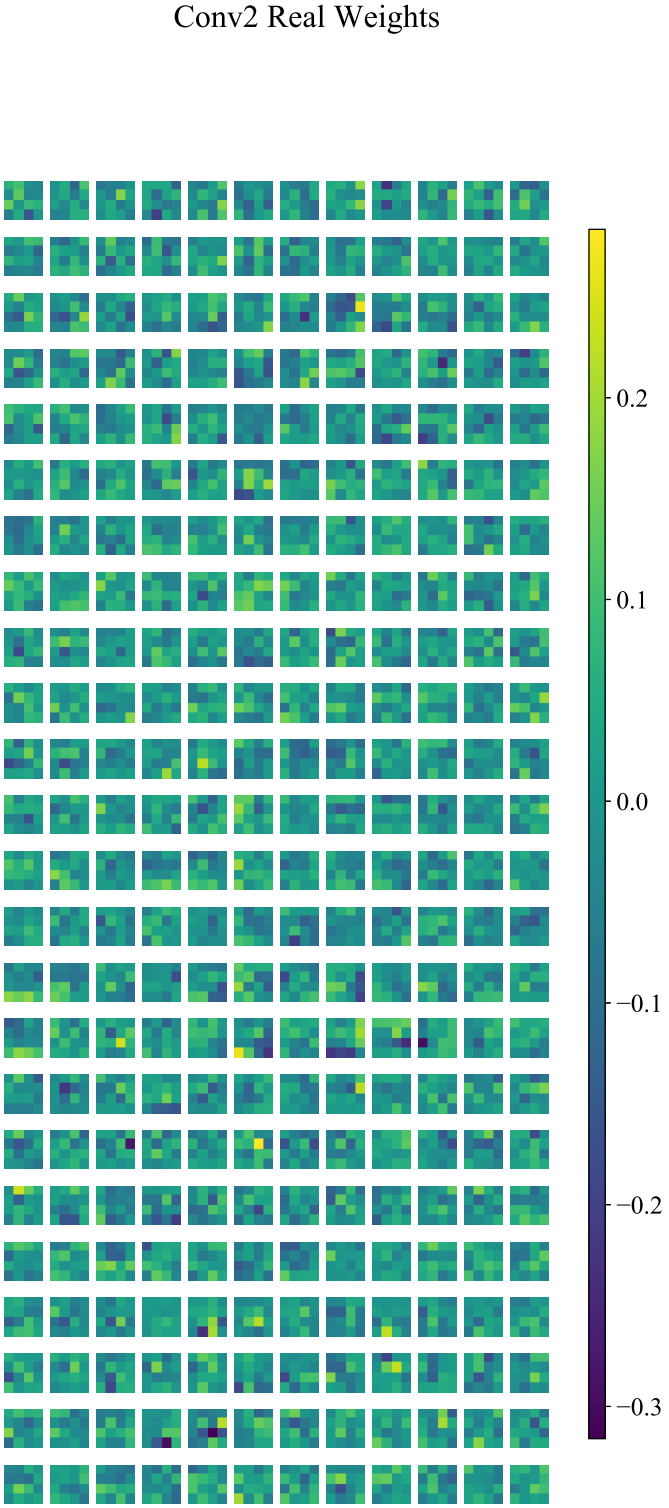

Figure B.7.: The visualization of the real part of the second convolution kernel.

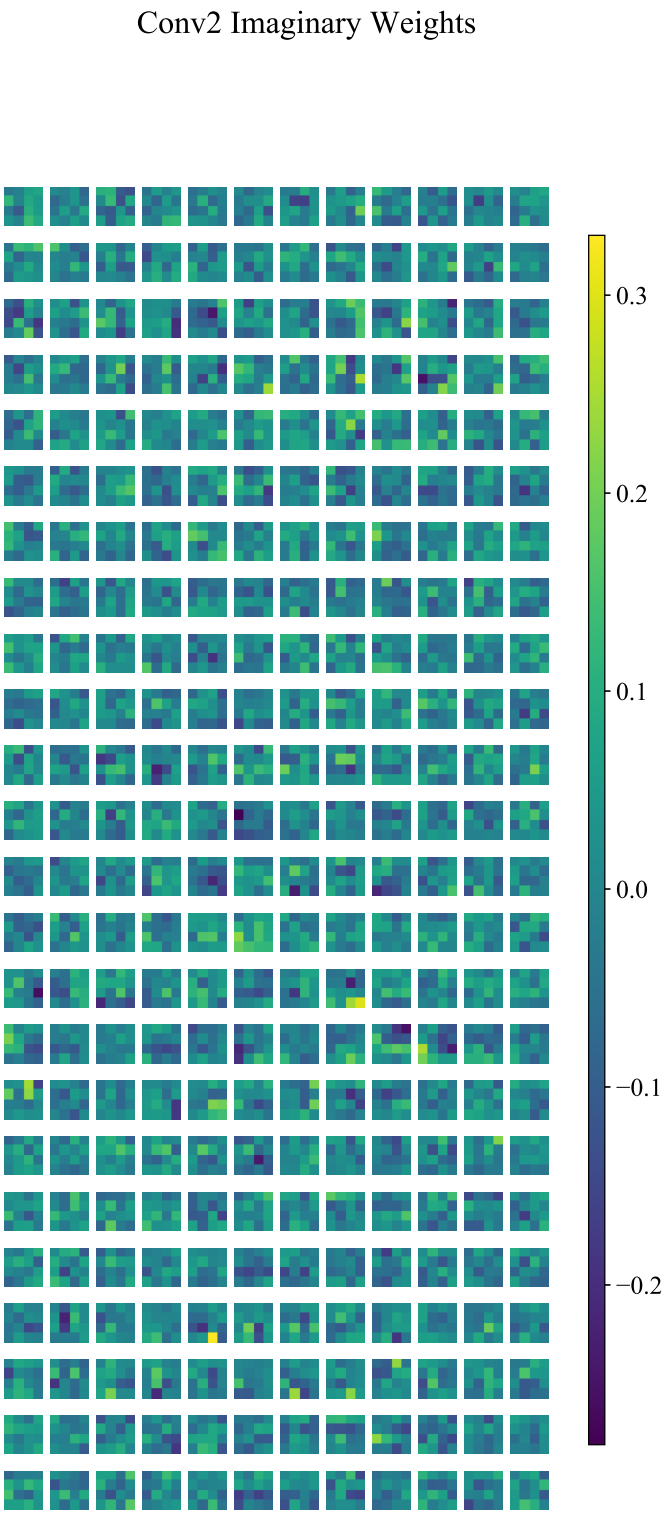

Figure B.8.: The visualization of the imaginary part of the second convolution kernel.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

[3] Jimmy Ba, Roger Grosse, and James Martens. Distributed second-order optimization using kronecker-factored approximations. In *International Conference on Learning Representations*, 2017.

[4] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[5] Zi Cai and Jinguo Liu. Approximating quantum many-body wave functions using artificial neural networks. *Phys. Rev. B*, 97:035116, Jan 2018.

[6] Lorenzo Campos Venuti and Paolo Zanardi. Quantum critical scaling of the geometric tensors. *Phys. Rev. Lett.*, 99:095701, Aug 2007.

[7] Giuseppe Carleo, Federico Becca, Laurent Sanchez-Palencia, Sandro Sorella, and Michele Fabrizio. Light-cone effect and supersonic correlations in one- and two-dimensional bosonic superfluids. *Phys. Rev. A*, 89:031602, Mar 2014.

[8] Giuseppe Carleo, Federico Becca, Marco Schiró, and Michele Fabrizio. Localization and glassy dynamics of many-body quantum systems. *Scientific reports*, 2:243, 2012.

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

[10] Lorenzo Cevolani, Giuseppe Carleo, and Laurent Sanchez-Palencia. Protected quasilocality in quantum systems with long-range interactions. *Phys. Rev. A*, 92:041603, Oct 2015.

[11] Hitesh J. Changlani, Jesse M. Kinder, C. J. Umrigar, and Garnet Kin-Lic Chan. Approximating strongly correlated wave functions with correlator product states. *Phys. Rev. B*, 80:245116, Dec 2009.

[12] Jing Chen, Song Cheng, Haidong Xie, Lei Wang, and Tao Xiang. Equivalence of restricted boltzmann machines and tensor network states. *Phys. Rev. B*, 97:085104, Feb 2018.

[13] Chung-Pin Chou, Frank Pollmann, and Ting-Kuo Lee. Matrix-product-based projected wave functions ansatz for quantum many-body ground states. *Phys. Rev. B*, 86:041105, Jul 2012.

[14] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[15] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[16] Dong-Ling Deng, Xiaopeng Li, and S. Das Sarma. Quantum entanglement in neural network states. *Phys. Rev. X*, 7:021021, May 2017.

[17] Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. *arXiv preprint arXiv:1602.02660*, 2016.

[18] Jack J Dongarra and Danny C Sorensen. Linear algebra on high performance computers. *Applied Mathematics and Computation*, 20(1-2):57–88, 1986.

[19] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Artificial Intelligence and Statistics*, pages 153–160, 2009.

[20] Paolo Facchi, Ravi Kulkarni, V.I. Man'ko, Giuseppe Marmo, E.C.G. Sudarshan, and Franco Ventriglia. Classical and quantum fisher information in the geometrical formulation of quantum mechanics. *Physics Letters A*, 374(48):4801 – 4803, 2010.

[21] Andrew J. Ferris and Guifre Vidal. Perfect sampling with unitary tensor networks. *Phys. Rev. B*, 85:165146, Apr 2012.

[22] W. M. C. Foulkes, L. Mitas, R. J. Needs, and G. Rajagopal. Quantum monte carlo simulations of solids. *Rev. Mod. Phys.*, 73:33–83, Jan 2001.

[23] Xun Gao and Lu-Ming Duan. Efficient representation of quantum many-body states with deep neural networks. *Nature communications*, 8(1):662, 2017.

[24] Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

[25] A. Gendiar and T. Nishino. Latent heat calculation of the three-dimensional $q = 3, 4$, and 5 potts models by the tensor product variational approach. *Phys. Rev. E*, 65:046702, Apr 2002.

[26] Ivan Glasser, Nicola Pancotti, Moritz August, Ivan D. Rodriguez, and J. Ignacio Cirac. Neural-network quantum states, string-bond states, and chiral topological states. *Phys. Rev. X*, 8:011006, Jan 2018.

[27] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[28] Shou-Shu Gong, Wei Zhu, D. N. Sheng, Olexei I. Motrunich, and Matthew P. A. Fisher. Plaquette ordered phase and quantum phase diagram in the spin-$\frac{1}{2}$ J_1-J_2 square heisenberg model. *Phys. Rev. Lett.*, 113:027201, Jul 2014.

[29] Ian Goodfellow. Efficient per-example gradient computations. *arXiv preprint arXiv:1510.01799*, 2015.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[31] Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582, 2016.

[32] Nitzan Guberman. On complex valued convolutional neural networks. *arXiv preprint arXiv:1602.09046*, 2016.

[33] Jutho Haegeman, J. Ignacio Cirac, Tobias J. Osborne, Iztok Pižorn, Henri Verschelde, and Frank Verstraete. Time-dependent variational principle for quantum lattices. *Phys. Rev. Lett.*, 107:070601, Aug 2011.

[34] Jutho Haegeman, Christian Lubich, Ivan Oseledets, Bart Vandereycken, and Frank Verstraete. Unifying time evolution and optimization with matrix product states. *Phys. Rev. B*, 94:165116, Oct 2016.

[35] R Haghshenas and DN Sheng. The $u(1)$-symmetric ipeps study of the spin-$1/2$ square $j\_\{1\}$-$j\_\{2\}$ heisenberg model. *arXiv preprint arXiv:1711.07584*, 2017.

[36] M. B. Hastings. Quantum belief propagation: An algorithm for thermal quantum systems. *Phys. Rev. B*, 76:201102, Nov 2007.

[37] Matthew B. Hastings, Iván González, Ann B. Kallin, and Roger G. Melko. Measuring renyi entanglement entropy in quantum monte carlo simulations. *Phys. Rev. Lett.*, 104:157201, Apr 2010.

[38] Huan He, Yunqin Zheng, B Andrei Bernevig, and Nicolas Regnault. Entanglement entropy from tensor network states for stabilizer codes. *arXiv preprint arXiv:1710.04220*, 2017.

[39] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[42] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[43] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[44] Wen-Jun Hu, Federico Becca, Alberto Parola, and Sandro Sorella. Direct evidence for a gapless Z_2 spin liquid by frustrating néel antiferromagnetism. *Phys. Rev. B*, 88:060402, Aug 2013.

[45] Yichen Huang and Joel E Moore. Neural network representation of tensor network and chiral states. *arXiv preprint arXiv:1701.06246*, 2017.

[46] Kota Ido, Takahiro Ohgoe, and Masatoshi Imada. Time-dependent many-variable variational monte carlo method for nonequilibrium strongly correlated electron systems. *Phys. Rev. B*, 92:245106, Dec 2015.

[47] Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538, 2002.

[48] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[49] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pages 972–981, 2017.

[50] Jack PC Kleijnen and Reuven Y Rubinstein. Optimization and sensitivity analysis of computer simulation models by the score function method. *European Journal of Operational Research*, 88(3):413–427, 1996.

[51] M.S. Leifer and D. Poulin. Quantum graphical models and belief propagation. *Annals of Physics*, 323(8):1899 – 1946, 2008.

[52] Ding Liu, Shi-Ju Ran, Peter Wittek, Cheng Peng, Raul Blázquez García, Gang Su, and Maciej Lewenstein. Machine learning by two-dimensional hierarchical tensor networks: A quantum information theoretic perspective on deep architectures. *arXiv preprint arXiv:1710.04833*, 2017.

[53] James Martens. *Second-order optimization for neural networks*. PhD thesis, University of Toronto (Canada), 2016.

[54] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[55] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

[56] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[57] Fabio Mezzacapo, Norbert Schuch, Massimo Boninsegni, and J Ignacio Cirac. Ground-state properties of quantum many-body systems: entangled-plaquette states and variational monte carlo. *New Journal of Physics*, 11(8):083026, 2009.

[58] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[59] Guido F Montúfar. Universal approximation depth and errors of narrow belief networks with discrete units. *Neural computation*, 26(7):1386–1407, 2014.

[60] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[61] Eric Neuscamman, C. J. Umrigar, and Garnet Kin-Lic Chan. Optimizing large parameter sets in variational quantum monte carlo. *Phys. Rev. B*, 85:045103, Jan 2012.

[62] M. P. Nightingale and Vilen Melik-Alaverdian. Optimization of ground- and excited-state wave functions and van der waals clusters. *Phys. Rev. Lett.*, 87:043401, Jul 2001.

[63] Yusuke Nomura, Andrew S. Darmawan, Youhei Yamaji, and Masatoshi Imada. Restricted boltzmann machine learning for solving strongly correlated quantum systems. *Phys. Rev. B*, 96:205152, Nov 2017.

[64] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 442–450. Curran Associates, Inc., 2015.

[65] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

[66] Vasily Pestun, John Terilla, and Yiannis Vlassopoulos. Language as a matrix product state. *arXiv preprint arXiv:1711.01416*, 2017.

[67] Vasily Pestun and Yiannis Vlassopoulos. Tensor network language model. *arXiv preprint arXiv:1710.10248*, 2017.

[68] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941*, 2017.

[69] Elina Robeva and Anna Seigal. Duality of graphical models and tensor networks. *arXiv preprint arXiv:1710.01437*, 2017.

[70] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[71] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[72] Hiroki Saito. Solving the bose–hubbard model with machine learning. *Journal of the Physical Society of Japan*, 86(9):093001, 2017.

[73] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM, 2007.

[74] A. W. Sandvik and G. Vidal. Variational quantum monte carlo simulations with tensor-network states. *Phys. Rev. Lett.*, 99:220602, Nov 2007.

[75] Norbert Schuch, Michael M. Wolf, Frank Verstraete, and J. Ignacio Cirac. Computational complexity of projected entangled pair states. *Phys. Rev. Lett.*, 98:140506, Apr 2007.

[76] Norbert Schuch, Michael M. Wolf, Frank Verstraete, and J. Ignacio Cirac. Simulation of quantum many-body systems with strings of operators and monte carlo tensor contractions. *Phys. Rev. Lett.*, 100:040501, Jan 2008.

[77] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[78] Alessandro Sfondrini, Javier Cerrillo, Norbert Schuch, and J. Ignacio Cirac. Simulating two- and three-dimensional frustrated quantum systems with string-bond states. *Phys. Rev. B*, 81:214426, Jun 2010.

[79] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[80] Olga Sikora, Hsueh-Wen Chang, Chung-Pin Chou, Frank Pollmann, and Ying-Jer Kao. Variational monte carlo simulations using tensor-product projected states. *Phys. Rev. B*, 91:165113, Apr 2015.

[81] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[82] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[83] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[84] Sandro Sorella. Generalized lanczos algorithm for variational quantum monte carlo. *Phys. Rev. B*, 64:024512, Jun 2001.

[85] Sandro Sorella. Wave function optimization in the variational monte carlo method. *Phys. Rev. B*, 71:241103, Jun 2005.

[86] E Miles Stoudenmire and David J Schwab. Supervised learning with quantum-inspired tensor networks. *arXiv preprint arXiv:1605.05775*, 2016.

[87] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[88] Julien Toulouse and CJ Umrigar. Full optimization of jastrow–slater wave functions with application to the first-row atoms and homonuclear diatomic molecules. *The Journal of chemical physics*, 128(17):174101, 2008.

[89] Julien Toulouse and Cyrus J Umrigar. Optimization of quantum monte carlo wave functions by energy minimization. *The Journal of chemical physics*, 126(8):084102, 2007.

[90] Chiheb Trabelsi, Olexa Bilaniuk, Dmitriy Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J Pal. Deep complex networks. *arXiv preprint arXiv:1705.09792*, 2017.

[91] Robert R Tucci. Quantum bayesian nets. *International Journal of Modern Physics B*, 9(03):295–337, 1995.

[92] Robert R Tucci. How to compile a quantum bayesian net. *arXiv preprint quant-ph/9805016*, 1998.

[93] Robert R Tucci. Factorization of quantum density matrices according to bayesian and markov networks. *arXiv preprint quant-ph/0701201*, 2007.

[94] C. J. Umrigar and Claudia Filippi. Energy and variance optimization of many-body wave functions. *Phys. Rev. Lett.*, 94:150201, Apr 2005.

[95] C. J. Umrigar, Julien Toulouse, Claudia Filippi, S. Sorella, and R. G. Hennig. Alleviation of the fermion-sign problem by optimization of many-body wave functions. *Phys. Rev. Lett.*, 98:110201, Mar 2007.

[96] Ling Wang, Zheng-Cheng Gu, Frank Verstraete, and Xiao-Gang Wen. Tensor-product state approach to spin-$\frac{1}{2}$ square J_1-J_2 antiferromagnetic heisenberg model: Evidence for deconfined quantum criticality. *Phys. Rev. B*, 94:075143, Aug 2016.

[97] Ling Wang and Anders W Sandvik. Critical level crossings in the square-lattice spin-$1/2j\_1 - j\_2$ heisenberg antiferromagnet. *arXiv preprint arXiv:1702.08197*, 2017.

[98] Julia Wildeboer and N. E. Bonesteel. Spin correlations and topological entanglement entropy in a non-abelian spin-one spin liquid. *Phys. Rev. B*, 94:045125, Jul 2016.

[99] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.

[100] Michael M Wolf. Mathematical foundations of supervised learning. 2017.

[101] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5285–5294, 2017.

[102] Paolo Zanardi, Paolo Giorda, and Marco Cozzini. Information-theoretic differential geometry of quantum phase transitions. *Phys. Rev. Lett.*, 99:100603, Sep 2007.