DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Quantum dynamics simulations with GPU support

Iulia-Otilia Mustea

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Quantum dynamics simulations with GPU support

Quantendynamik-Simulationen mit GPU-Unterstützung

Author:	Iulia-Otilia Mustea
Supervisor:	Univ.-Prof. Dr. Christian Mendl
Advisor:	Irene López Gutiérrez, M.Sc.
Submission Date:	15.07.2021

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.07.2021 Iulia-Otilia Mustea

Acknowledgments

I would like to thank my supervisor, Prof.Dr. Christian Mendl, for giving me the opportunity of working on my Master's Thesis at the Quantum Computing Research Group. I am especially grateful to my advisor, Irene López Gutiérrez, for her guidance, feedback and support to navigate this project, which was based on her previous research work. I am thankful for having access to any valuable resources that were used in this project, such as the previous code and the GPU cluster.

In addition, I am very grateful for all the support of my close friends and colleagues during my studies. My deepest thanks are going to Radu Ciocoveanu for his continuous encouragement and advice. Finally, I must express my profound gratitude to my parents and grandparents, for their unfailing support, unconditional love and care.

Abstract

In the last years, the field of quantum many-body systems has seen a sharp development, specifically in simulating interacting systems. However, a crucial challenge for current computational methods is the efficient numerical simulation of nonequilibrium real-time evolution due to the high-dimensionality of quantum systems. The state-of-the-art shows that artificial neural network encodings of quantum many-body wave function efficiently describe the time dynamics.

In this thesis, an efficient machine learning-inspired approach has been employed to simulate the time dynamics. The neural-network quantum states are used to approximate the implicit midpoint rule method for solving the time-dependent Schrödinger equation. The proposed neural network architecture for the wave-function ansatz is the RBM (Restricted Boltzmann Machine), which has been shown to represent the ground state of various Hamiltonians with high accuracy. As a concrete example, this thesis studies the application of the transverse-field Ising model on a one-dimensional lattice of different sizes, which exhibits an accuracy comparable to the stochastic configuration method. To deal with the high-dimensionality of a transverse-field Ising model on a lattice with periodic boundary conditions, the use of the GPU is employed in delivering results for larger lattice sizes. Observations show that the use of the GPU is critical for achieving results in large systems.

Contents

1 Introduction

This chapter details the motivation behind the thesis and its impact, along with a review of the related work in quantum many-body systems simulations.

1.1 Motivation

Among the various fields of science, quantum mechanics can be regarded as one of the most fascinating and challenging topics. However, it is possibly the hardest to grasp in the classical world. The wave function Ψ is a fundamental object in quantum physics, which contains all the information of a quantum state, be it a single particle or a complex molecule [1]. In the last years, the field of quantum many-body systems has seen a sharp development, specifically in simulating interacting systems. However, a crucial challenge for current computational methods is the efficient numerical simulation of nonequilibrium real-time evolution due to the high-dimensionality of quantum systems. The motivation of the thesis is to find a strategy to reduce the exponential complexity of the many-body wave function Ψ , which in our case depends on the time-dependent Schrödinger equation.

The Schrödinger equation is a linear partial differential equation that governs the wave function of a quantum-mechanical system [2].

$$i\frac{\partial \psi}{\partial t} = H\psi. \tag{1.1}$$

A system in quantum physics is governed by a matrix H, called the Hamiltonian, which describes the energy configuration of the given system and determines the interaction between its particles. In Schrödinger's equation (1.1), H governs how a quantum state changes over time and the argument t denotes the time dependence of Ψ. In large many-body systems, the size of the Hamiltonian grows exponentially with the L number of spin configurations, as $H \in \mathbb{C}^{2^L \times 2^L}$. Therefore, the complexity of the system as a whole increases exponentially.
Specifically, in this thesis, the transverse-field Ising model was used. It features particles arranged on a regular lattice with nearest neighbor interactions determined by the alignment or anti-alignment of spin projections. The model has the following Hamiltonian:

$$H = -h\sum_j \sigma_j^x - J\sum_{\{i,j\}} \sigma_i^z \sigma_j^z, \tag{1.2}$$

where J is a parameter that represents the spin-spin interaction, h is a parameter that represents the external field, i and j are label lattice sites. The notation $\{i, j\}$ indicates that sites i and j are nearest neighbors.

1.2 Related Work

Due to investigating one- and two-dimensional lattice quantum many-body systems at large sizes, there is a need for continuous search of numerical techniques to provide accurate verification of quantum simulations. Therefore, multiple approaches have been developed to solve this issue.

In the last decades, tensor networks (class of many-body wavefunctions of complexity fixed by a parameter known as bond dimension [3]) have been developed and applied to classically simulate quantum many-body systems, representing the large wave function with a set of local tensors connected via auxiliary indices. Lately, tensor networks have been adapted for supervised learning [4], taking advantage of the variational studies in quantum mechanics and developments in machine learning. According to [1], tensor networks are a compression approach for solving the quantum many-body problem, which relies on the efficient representation of the wave function. Also, there are numerical approaches to solving the quantum many-body problem, which sample a finite number of physically relevant configurations of the wave function [1]. This includes stochastic approaches like the quantum Monte Carlo (QMC) and its improved version, stochastic reconfiguration, which rely on probabilistic frameworks typically demanding a positive-semidefinite wave function [1].

However, both the stochastic reconfiguration and tensor networks fail in some cases, mostly due to the sign problem in quantum Monte Carlo and to the inefficiency of current tensor networks in high-dimensional systems [1]. As a result, despite the success of these methods, a large number of issues exist.

Artificial neural networks and their expansion in the latest years proved useful to help solve numerous problems in science, such as robotics, bioinformatics, medical image analysis, etc. Specifically, the success of neural networks has been proven in high dimensional problems, such as finding the eigenvalue and eigenfunction of the Fokker-Planck operator [5] and the prediction of three neutronic parameters for the Ringhals-1 BWR unit [6]. Therefore, due to the expansion of artificial neural networks, proven results in high dimensional problems, and the need for dimensionality reduction in quantum systems, neural networks were investigated and proven to solve the quantum many-body problem [1] [7] [8].

Neural networks have an advantage in handling large entanglement [7] in comparison with tensor network methods, which are limited to relatively short time intervals. This is due to the fact that the increase of entanglement with time requires an exponential increase of virtual bond dimensions [7]. In [9], the authors determine the computational difficulty of

Matrix Product States (MPS) ground states, for which the ground state is a MPS of size. For these Hamiltonians,[9] shows that finding the ground state (or a polynomial-accuracy approximation) is an NP-hard problem. This implies that finding an MPS approximation of these ground states can not be computed by classical computers. Also, [3] proves that quantum states can not be approximated by convex combinations of tensor networks with low bond dimension. Therefore, the virtual bond dimension bounds the tensor networks to small systems.

2 Method

This chapter presents an overview of the project in section 2.1 and the employed neural network architecture - Restricted Boltzmann Machine in section 2.2. In section 2.3, the custom cost function is detailed, together with the mathematical explanation of its gradients, which are computed taking into consideration complex differentiability.

2.1 Overview

As mentioned in the previous chapter, the disadvantages of the tensor networks and stochastic reconfiguration methods have demanded the introduction of new methods for solving the quantum many-body problem. As the problem can be summarized dimensional reduction and feature extraction, [1] investigated the use of artificial neural networks, as the networks can easily be modified and adapted to describe and analyze a quantum system. The authors used a Restricted Boltzmann Machine (RBM) to both find an unknown ground state of a given Hamiltonian H and extend it to the time-dependent Schrödinger's equation 1.1. Their results proved the versatility of their method, which was then extended in the next years.

In [7] and [8], the research has also been extended to a Convolutional Neural Network (CNN), for which a RBM is a special case of a fully connected single layer CNN with a fixed activation function [8]. Also, in [7], the RBM architecture demonstrates similar results to the stochastic reconfiguration. Therefore, this project is based on [7], replicating the results of the RBM on the Ising chain and extending it to have Graphics Processing Unit (GPU) support. By employing GPU support, the goal is to be able to analyze larger systems.

Firstly, it's important to discuss the goal of the project. The goal is to solve the time-dependent Schrödinger 1.1 by approximating a time step of a ordinary differential equation (ODE) by using using gradient descent on neural network quantum state at the next step. In this case, the variational ansatz is $\psi[\theta]$, where the term θ is a complex vector containing time-dependent variational parameters [7]. Therefore, it is possible to find the gradients of ψ with respect to θ and derive an ODE for θ [7]. For each time step, the network parameters are optimized to minimize an error between $\psi[\theta_{n+1}]$ and $\psi[\theta_n]$. The cost function can be compactly in least-squares form as

$$C(\theta) = ||A\psi[\theta] - b||^2, \tag{2.1}$$

where A is the sparse submatrix of $I + \frac{i\Delta t}{2}H$ corresponding the spin configuration $\sigma^{(j)}$ and

vector b has entries $b_j = ((I - \frac{i\Delta t}{2}H)\psi[\theta_n])(\sigma^{(j)})$ [7]. The cost function and its gradients will be further discused in subsection 2.3.

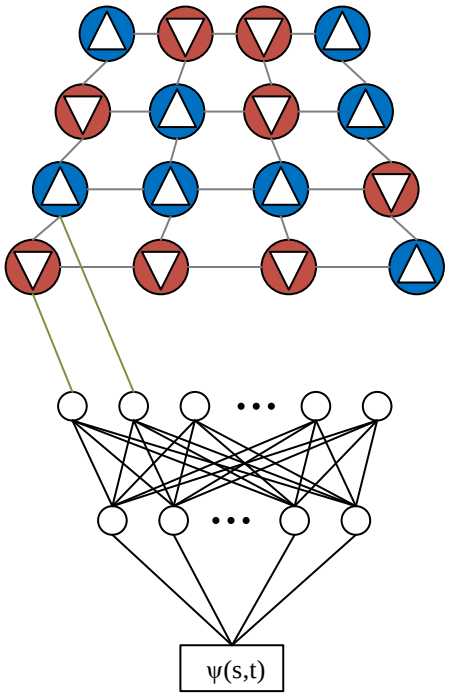

Figure 2.1: Schematic illustration of the proposed method.

Fig. 2.1 shows a schematic illustration of the proposed method's pipeline. Given spin a configuration s generated randomly, blue and red referring to the spin $\uparrow$ and $\downarrow$ state, functions as the input to an Artificial Neural Network (ANN) whose output at the end is the corresponding wave function amplitude ψ_s. The neural-network proposed is the RBM, which consists of two layers of neurons. The input of the visible layer is the spin configuration s with the system size L, where the quantum Hilbert space dimension is 2^L.

The RMB ansatz is applied to a one- dimensional transverse-field Ising model, which consists of a chain of spins that interact with their nearest neighbors and are subject to an external magnetic field, called h. Its Hamiltonian is given by Eq.1.2. This system undergoes a phase transition from the ferromagnetic to the paramagnetic regime at h = J in one dimension and at h= J = 3:04438(2) in two dimensions [7]. Therefore, the coupling constant J is set to 1.

The initial state was found by performing a Hamiltonian quench with respect to the field strength h [7]. Firstly, the network parameters are optimized to represent the ground state for h = 1.5 and then changed to h = 0.75 for the real time evolution. The ground state, which is the state of the lowest energy [2], is important to the initialization of RBM weights. By definition, finding the ground state of a quantum many body system is analogous to finding

the eigenfunctions which result in the lowest eigenvalues for a particular Hamiltonian [2]. For a small system, it's possible to simply use iterative methods to calculate the eigenvalues and eigenvectors of the Hamiltonian matrix, which would give the ground state. However, in the case of this project, the ground state was generated using Stochastic Reconfiguration (SR) [10]. The main idea of SR is to modify the parameters of a trial wave function in such a way that it approaches the ground state along a path dictated by the projection $1 - \epsilon H$, where ϵ is chosen such that $1 - \epsilon H \geq 0$ [11]. The ground state generation with SR proves to be good enough for delivering the expected results.

The optimizable neural network parameters(a - the bias of the 'visible' layer, b - the bias of the 'hidden' layer, w - the weights between the 'hidden' and 'visible' layer) are complex-valued parameters. In this project, we represent complex-valued parameters as two real-valued sets of parameters that are connected according to the algebraic rules induced by split-complex arithmetic [12]. Therefore, we can rewrite each operation as a combination of algebraic operations between the real and imaginary parts. In subsection 2.3, the reasoning behind representing the complex parameters in such a way will be detailed. Therefore, we have: $a = a_{real} + i * a_{imag}$, $b = b_{real} + i * b_{imag}$, $w = w_{real} + i * w_{imag}$, where a_{real}, a_{imag}, b_{real}, b_{imag}, w_{real}, w_{imag} are real-valued parameters. The cost function, written having a_{real}, a_{imag}, b_{real}, b_{imag}, w_{real}, w_{imag} as terms, will be derived with respect to them. Therefore, the gradients are calculated with respect to each term, which will give the gradient step for the optimization.

In the next part, the main algorithm for optimization inside the RBM neural network will be detailed.

Data: $a_{initial}$, $b_{initial}$, $w_{initial}$
Result: $\psi_t list$
for *i in range (time_steps)* **do**
  add last ψ_t to list;
  while *optimizing_steps < limit* **do**
    $Energy_{past}$, ψ_{past} - calculated with $a_{previous}, b_{previous}, w_{previous}$;
    variable_list = [a_real, a_imag, b_real, b_imag, w_real, w_imag];
    new_gradients = gradient of **cost_function** w.r.t. **variable_list**;
    update a, b, w with new_gradients in Adam optimizer;
    calculate loss;
  end
  re-initiliaze a, b, w with $a_{initial}$, $b_{initial}$, $w_{initial}$;
end

Algorithm 1: Training sequence in RBM structure.

2.2 Restricted Boltzmann Machine

As mentioned previously, the neural network architecture proposed is the RBM, which has been shown to represent the ground state of Hamiltonians with high accuracy [7]. Boltzmann machines are named after the Boltzmann distribution (also known as Gibbs distribution), which is a part of statistical mechanics.

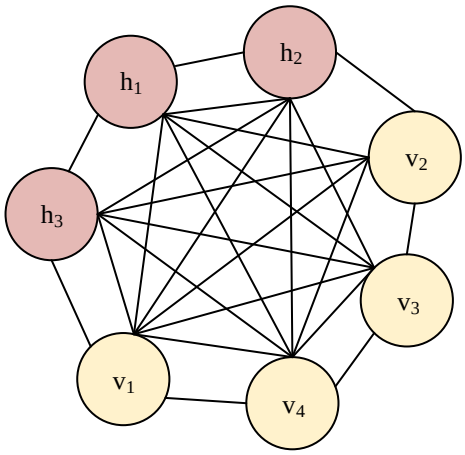

Figure 2.2: Boltzmann Machine architecture.

In Fig. 2.2, we have a visual representation of a Boltzmann machine architecture. Compared with neural networks, which do not have any connections between the input nodes, a Boltzmann Machine has connections among the input nodes. It is shown in Fig.2.2 that all the nodes are connected to all other nodes irrespective of whether they are visible or hidden nodes, which allows them to share information among themselves.

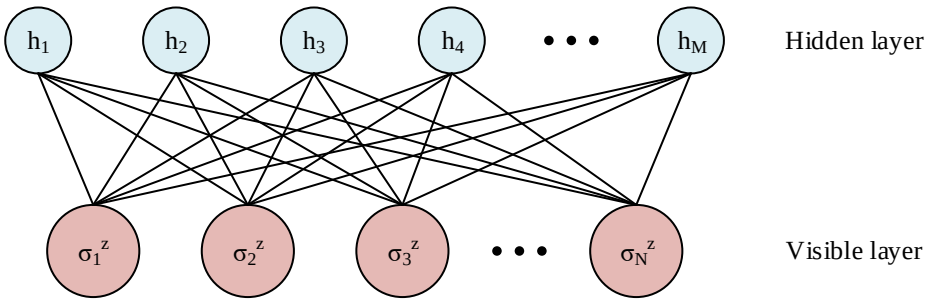

Figure 2.3: Restricted Boltzmann Machine architecture.

Restricted Boltzmann Machines are a variant of Boltzmann machines with a difference that their neurons form a bipartite graph, which means there are no connections between nodes of the same layer. In Fig. 2.3 we have a visual representation of the RBM artificial network, which is constituted of one visible layer of L nodes, corresponding to the physical spin variables in a

chosen basis (S = $\sigma_1, .., \sigma_L$), and a single hidden layer of M auxiliary spin variables ($h_1, ..., h_M$). The 'visible' and the 'hidden' layer are fully connected with each other but have no intra-layer connections [7]. The construction of the RBM corresponds to a variational expression for the quantum state:

$$\psi(\sigma) = \sum_{h_i} e^{\sum_j a_j\sigma_j + \sum_i b_i h_i + \sum_{i,j} w_{ij} h_i \sigma_j}, \tag{2.2}$$

where a_j, w_{ij} and b_i are the optimizable network parameters and h_i are the auxiliary spin variables, which only take $\pm$ 1 values. To be specific, a is the bias of the 'visible' layer, b the bias of the 'hidden' layer and w the weights between the 'hidden' and 'visible' layer.
The ansatz from Eq.2.2 can be transformed by sumation over h_i:

$$\begin{aligned}
\psi(\sigma) &= \sum_{h_i} e^{\sum_j a_j\sigma_j + \sum_i b_i h_i + \sum_{i,j} w_{ij} h_i \sigma_j} \\
&= \sum_{h_i} e^{\sum_j a_j\sigma_j} e^{\sum_i h_i (b_i + \sum_j w_{ij}\sigma_j)} \\
&= e^{\sum_j a_j\sigma_j} \prod_i e^{(b_i + \sum_j w_{ij}\sigma_j)} + e^{-(b_i + \sum_j w_{ij}\sigma_j)} \\
&= e^{\sum_j a_j\sigma_j} \prod_i 2cosh(b_i + \sum_j w_{ij}\sigma_j),
\end{aligned} \tag{2.3}$$

where $cosh(x) = \frac{e^x + e^{-x}}{2}$

One of the advantages of using an RBM is that constructing the Hamiltonian is not necessary at every point but instead is possible to use the analytical expression for the local energy $\frac{H|\psi(\sigma)\rangle}{\psi(\sigma)}$. In the following lines, the analytical expression for the local energy will be provided, taking into consideration the RBM architecture and the Transverse-field Ising model, which gives the Hamiltonian model from Eq. 1.2. For numerical stability, the $cosh(x)$ is $exp(log(cosh))$, therefore, the logarithmized Eq. 2.3 becomes:

$$\log\psi(\sigma) = \sum_j a_j\sigma_j + \sum_i \log\left(2\cosh\left(b_i + \sum_j w_{ij}\sigma_j\right)\right). \tag{2.4}$$

Therefore, taking into consideration Eq.2.4, Eq.2.3 is, in a numerical stable form:

$$\begin{aligned}
\psi(\sigma) &= \exp\left(\sum_j a_j\sigma_j + \sum_i \log\left(2\cosh\left(b_i + \sum_j w_{ij}\sigma_j\right)\right)\right) \\
&= \exp\left(\sum_j a_j\sigma_j\right) \prod_i 2\cosh\left(b_i + \sum_j w_{ij}\sigma_j\right).
\end{aligned} \tag{2.5}$$

To compute the local energy for the RBM and Transverse-field Ising model, it is necessary to apply the Hamiltonian of Eq.1.2 and the formula for $\psi(\sigma)$ of Eq.2.5.

$$
\begin{aligned}
E_{local} &= \frac{H\left|\psi\left(\sigma\right)\right\rangle}{\psi\left(\sigma\right)} = -h\frac{\sum\limits_{k}\sigma_k^x\left|\psi\left(\sigma\right)\right\rangle}{\psi\left(\sigma\right)} \\
&= -h\frac{\sum\limits_{k}\exp\left(-a_k\sigma_k+\sum\limits_{j\notin k}a_j\sigma_j\right)\prod\limits_{i}2\cosh\left(b_i-w_{ik}\sigma_k+\sum\limits_{j\notin k}w_{ij}\sigma_j\right)}{\exp\left(\sum\limits_{j}a_j\sigma_i\right)\prod\limits_{i}2\cosh\left(b_i+\sum\limits_{j}w_{ij}\sigma_i\right)} - J\sum_{\{i,j\}}\frac{\sigma_i^z\sigma_j^z\psi\left(\sigma\right)}{\psi\left(\sigma\right)} \\
&= -h\frac{\sum\limits_{k}\left(-2ak\sigma_k\right)\prod\limits_{i}2\cosh\left(b_i-w_{ik}\sigma_k+\sum\limits_{j\notin k}w_{ij}\sigma_j\right)}{\prod\limits_{i}2\cosh\left(b_i+\sum\limits_{j}w_{ij}\sigma_j\right)} - J\sum_{\{i,j\}}\frac{\sigma\sigma\prime\psi\left(\sigma\right)}{\psi\left(\sigma\right)} \\
&= -h\frac{\sum\limits_{k}^{N}e^{-2ak\sigma_k}\prod\limits_{i}\cosh\left(b_i+\sum\limits_{j\neq k}w_{ij}\sigma_j-w_{ik}\sigma_k\right)}{\cosh\left(b_i+\sum\limits_{j}w_{ij}\sigma_j\right)} - J\sum_{\{i,j\}}\sigma\sigma\prime.
\end{aligned}
\tag{2.6}
$$

It is essential to mention that $\sigma_i^z\sigma_j^z$ becomes $\sigma\sigma\prime$ due to the fact that the spin configuration is flipped for the nearest neighbors.

2.3 Cost function and gradient methods

This section covers the topic of the cost function in subsection 2.3.1. Also, it tackles the calculation of gradients for the optimization step while considering complex differentiability in subsections 2.3.2 and 2.3.3.

2.3.1 Cost function

As mentioned in the previous section, it is possible to find the gradients of ψ with respect to θ and use the chain rule to derive an ODE for θ [7]. Therefore, the cost function for our system can be written similar to the paradigm of neural network training, as in [7], in contrast with SR [10]. Furthermore, the network parameters are optimized to minimize an error between $\psi[\theta_{n+1}]$ and $\psi[\theta_n]$, similar to the paradigm of minimizing an error between the ground truth labels and the predicted labels in simple neural networks. The error is defined by

$$\left\| \psi\left[\theta_{n+1}\right] - \Phi^{\Delta t}\left(\psi\left[\theta_n\right]\right) \right\|, \tag{2.7}$$

where $\Phi^{\Delta t}$ is the discrete flow of a numerical ODE method applied to the time-dependent Schrödinger Eq.1.1 [7].

For the ODE $y'(t) = f(t, y(t))$, with time-step Δt, the implicit midpoint rule is:

$$y_{n+1} = y_n + \Delta t\, f\left(t_n + \frac{\Delta t}{2}, \frac{1}{2}\left(y_n + y_{n+1}\right)\right), \tag{2.8}$$

which leads, in the specific case of Eq. 1.1, to:

$$\psi\left[\theta_{n+1}\right] \approx \psi\left[\theta_n\right] - i\,\Delta t\, H\left(\frac{\psi\left[\theta_{n+1}\right] + \psi\left[\theta_n\right]}{2}\right). \tag{2.9}$$

It's important to mention that the implicit midpoint method has important advantages for our system: it preserves the form of Hamiltonian dynamics and it doesn't contain any intermediate terms that would complicate the network optimization [7].
Therefore, the cost function for a single implicit midpoint rule time step becomes:

$$C\left(\theta_{n+1}\right) = \sum_{j=1}^{N} \left| \left(\left(I + \frac{i\Delta t}{2} H\right) \psi\left[\theta_{n+1}\right] - \left(I - \frac{i\Delta t}{2} H\right) \psi\left[\theta_n\right] \right) \left(\sigma^{(j)}\right) \right|^2, \tag{2.10}$$

which can be represented in the least squares form shown in Eq.2.1, a more compact form. As the goal is to minimize the cost function with respect to the network parameters of gradient descend, this cost function should be evaluated at every time step when updating the parameters. Also, partial derivatives of $C(\theta)$ with respect to θ should be calculated for every gradient descend, and parameters update step. Because $C(\theta)$ is not a holomorphic function and network parameter θ (therefore a, b, w) is complex-valued, direct derivation is not possible. Therefore, other mathematical techniques have to be employed for the derivation of the cost function. This topic will be discussed in the next subsection.

2.3.2 Complex differentiability

As mentioned previously, the cost function set to minimize is real-valued. Therefore, some information about real-valued complex functions and differentiability will be added, as it plays a crucial role in the optimization process.

In complex theory, a complex number z ϵ $\mathbb{C}$ is a number that can be represented in the form $z = x + i * y$, where x and y are real numbers and i is a symbol called 'the imaginary unit', which satisfies the equation $i^2 = -1$.
A complex function f has the form: $f: \mathbb{C} \rightarrow \mathbb{C}$, $f(z) = u(z) + i * v(z)$, where $u, v: \mathbb{R} \rightarrow \mathbb{R}$. The complex function f is called complex-differentiable at x, if and only if the following limit exists [13]:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{2.11}$$

A function that is complex differentiable everywhere is called entire [13]. A complex-valued function of one or more complex variables that is entire and complex differentiable is called holomorphic.

To give some context, the cost function C, discussed in the previous subsection, is a function of a single complex number z and its complex conjugate z^*, which can be written as C(z,z^*) [14]. Due to the fact that C is a mapping from a complex number to a real number, it is in general a non-holomorphic function, namely $\frac{\partial C}{\partial z^*} \neq 0$ [14]. Therefore, the Wirtinger or Dolbeault operators are employed to compute gradients of the cost function with respect to complex-valued parameters, which are defined as:

$$\frac{\partial}{\partial z} = \frac{1}{2}\left(\frac{\partial}{\partial x} - i\frac{\partial}{\partial x}\right), \frac{\partial}{\partial z^*} = \frac{1}{2}\left(\frac{\partial}{\partial x} + i\frac{\partial}{\partial x}\right), \tag{2.12}$$

with $z = x + i * y$. For real-valued functions, like the cost function 2.1, the partial derivatives with respect to x and y can be obtained from the Wirtinger derivative [7] via:

$$\frac{\partial f}{\partial x} = 2Re\left(\frac{\partial f}{\partial z}\right), \frac{\partial f}{\partial y} = -2Im\left(\frac{\partial f}{\partial z}\right). \tag{2.13}$$

The Wirtinger operators act on real-differentiable function $f: U \rightarrow \mathbb{C}$ (where $U \epsilon \mathbb{C}$ is a subset of $\mathbb{C}$), for which being holomorphic is not necessary. Therefore, we can apply the Wirtinger operators to our non-holomorphic cost function. In case the function is indeed holomorphic, the Cauchy-Riemann equations imply that the Wirtinger derivative $\frac{\partial}{\partial z}$ is equal to the complex derivative of the function and the conjugated Wirtinger derivative vanishes [7]:

$$\frac{\partial f(z)}{\partial z} = f'(z), \frac{\partial f(z)}{\partial z^*} = 0, \ f \ holomorphic. \tag{2.14}$$

To apply the chain rule, the basic principle of backpropagation, to non-holomorphic functions, the property of the Wirtinger operators that many non-holymorphic functions are differentiable in their real and imaginary parts is exploited. Therefore, having the Wirtinger operator, the chain rule is straightforward to compute and verify:

$$\frac{\partial}{\partial z}(g \circ f) = \left(\frac{\partial g}{\partial w} \circ f\right) \cdot \frac{\partial f}{\partial z} + \left(\frac{\partial g}{\partial w^*} \circ f\right) \cdot \frac{\partial f*}{\partial z}. \tag{2.15}$$

Combined with the chain rule, the gradients of the cost function in Eq. 2.10 [7] are:

$$\begin{aligned} \frac{\partial C(\theta)}{\partial \theta_l} &= \frac{\partial C}{\partial (A\psi)} \frac{\partial (A\psi)}{\partial \psi} \frac{\partial \psi}{\partial \theta_l} \\ &= \left\langle A\psi[\theta] - b | A \frac{\partial \psi[\theta]}{\partial \theta_l} \right\rangle. \end{aligned} \tag{2.16}$$

2.3.3 Optimization technique

Based on the previous section, where the importance and usage of Wirtinger operators is highlighted, and taking into consideration the fact that the cost function 2.1 is non-holomorphic, the gradient calculation should employ the Wirtinger derivatives.

However, although using the Wirtinger operators is valid for our project, a simpler and equivalent approach is possible: split a complex number into a tuple of two real numbers and rewrite the functions in terms of the tuple accordingly [14]. Therefore, for $a = a_{real} + i * a_{imag}$, $b = b_{real} + i * b_{imag}$, $w = w_{real} + i * w_{imag}$ the optimization is done with respect to variables a_{real}, a_{imag}, b_{real}, b_{imag}, w_{real}, w_{imag}. The cost function and the adjancent calculation are re-written with respect to the six variables mentioned earlier. For example, variable a (the bias of the visible layer) becomes $a = tf.complex(a_{real}, a_{imag})$. Variables b and w have a similar shape: $b = tf.complex(b_{real}, b_{imag})$, $w = tf.complex(w_{real}, w_{imag})$.

Because the cost function 2.1 is written with respect the six variables a_{real}, a_{imag}, b_{real}, b_{imag}, w_{real}, w_{imag}, the gradients are also calculated with respect to each of the variables and updated in the same manner. This is possible due to automatic differentiation, which was employed to make the computation easier.

Advantages and disadvantages

This section highlights the advantages and disadvantages of using the optimization method presented earlier: splitting a complex number into a tuple of two real numbers and rewriting the functions in terms of the tuple [14].

Firstly, the method was employed for the purpose of this thesis. Splitting a complex number into real and imaginary parts takes advantage of automatic differentiation (of Tensorflow

[15]), making the computation easier. Therefore, because of the system size, using a built-in differentiation in Tensorflow [15] proves to be faster than a hand-written function that calculates the Wirtinger derivative.

However, in [14], the authors compare the two approaches. They consider we consider a simple example in which they take a complex function of two complex numbers, $z = a_z + i * b_z$ and $w = a_w + i * b_w$, with a_z, b_z, a_w, b_w real numbers:

$$g(z, w) = zw. \tag{2.17}$$

For $y = u + i * v$, another complex number, the adjoint function of g is defined as:

$$g(u, v) = ((ua_w + vb_w, va_w - ub_w), (ua_z + vb_z, va_z - ub_z)), \tag{2.18}$$

which is fairly cumbersome to computer ever for this case. However, as this example already shows, this approach results in deeper nested functions in the backward process, which would usually consume more memory and reduce the efficiency of the computation [14]. Therefore, according to [14], this method is not efficient computationally for deep neural networks. In the case of this thesis, the two methods are not compared computationally, but due to the fact that RBM has only two layers (hidden and visible layers), the computational issue mentioned in [14] should not arise.

Implementation details

This section covers the implementation details in Tensorflow [15], focusing on the gradient calculation and variable updates.

Firstly, it was essential to define an optimizer suitable for the task and that gives good results in general. Therefore, the Adam optimizer (built in the Keras package) was chosen. Adam optimization is a Stochastic Gradient Descent (SGD) method that is based on the estimation of first-order and second-order momentum.

```
opt = tf.keras.optimizers.Adam(learning_rate=learning_rate)
```

Figure 2.4: Initialize Adam optimizer with chosen learning rate.

Secondly, defining the variable list is important for the automatic differentiation. In our case, there are the six variable mentioned earlier: a_{real}, a_{imag}, b_{real}, b_{imag}, w_{real}, w_{imag}.
Tensorflow [15] provides the **tf.GradientTape API** for automatic differentiation, which is computing the gradient with respect to the chosen input, in our case, the variable list. The operations are recorded inside the "tape", where the cost function is placed. Once the operations are recorded, **GradientTape.gradient(target, sources)** calculates the gradient of

```
var_list = [self.a_real, self.a_imag, self.b_real, self.b_imag,

self.w_real, self.w_imag]
```

Figure 2.5: Variable list defined.

the target with respect to the chosen variables. The Adam optimizer then uses the new gradients to update the variables.

```
with tf.GradientTape() as tape:
      cost_fn = self.cost()

grads_list = tape.gradient(cost_fn, var_list)
opt.apply_gradients(zip(grads_list, var_list))
```

Figure 2.6: Gradient calculation and variable update.

3 Results and performance comparison

3.1 Results

For the simulations, three systems with variable sizes were considered, for which L is number of visible states, M is the number of hidden states and n the number of sampled spin configurations. The systems are:

1. $L = 5,\ M = 25,\ n = 1000$
2. $L = 7,\ M = 35,\ n = 2000$
3. $L = 12,\ M = 60,\ n = 2000$

The coupling constant J has been set to 1, while the ground state was represented for $h_{initial} = 1.5$ and then changed h to 0.75 for the real time evolution [7]. As mentioned previously, the optimizer is Adam, which has been initialized with learning rate such as **1e-3** and **1e-4**. The number of optimization steps varies from 500 per time step to 5.000 per time step.

The code was run in two Anaconda environments, one CPU-based and one GPU-based. For the CPU-based environment, there are the following packages: Tensorflow 2.3.0, Numpy 1.19.2, Python 3.8.5 etc. For the GPU-based environment, there are the following packages: Tensorflow 2.0.0, CUDA 10.0.130, cudnn 7.6.5, Python 3.7.0.

3.1.1 Training loss

This section presents the training loss decrease for different learning rates and optimization steps. All the results shown in Fig.3.1, Fig.3.2, Fig.3.3 and Fig.3.4 are for the system with L=12.

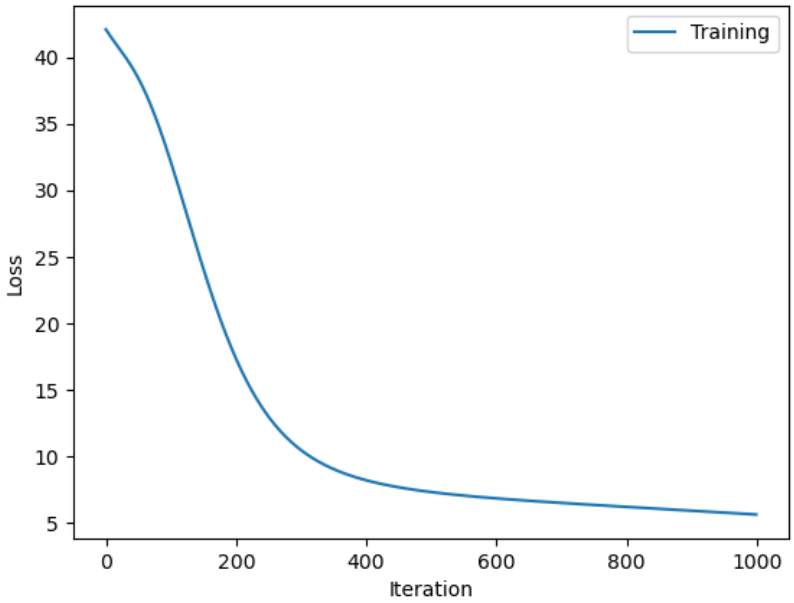

Figure 3.1: Plot of training loss for 1e-4 learning rate and 1.000 optimization steps.

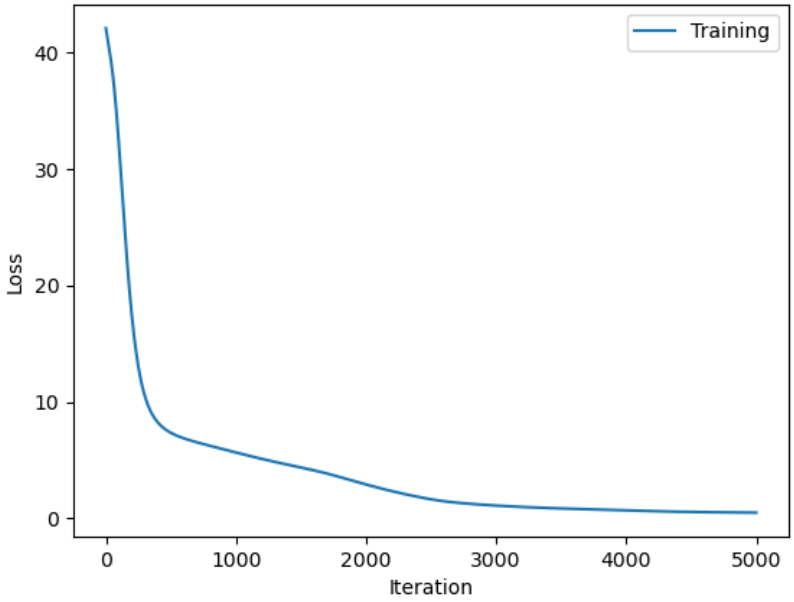

Figure 3.2: Plot of training loss for 1e-4 learning rate and 5.000 optimization steps.

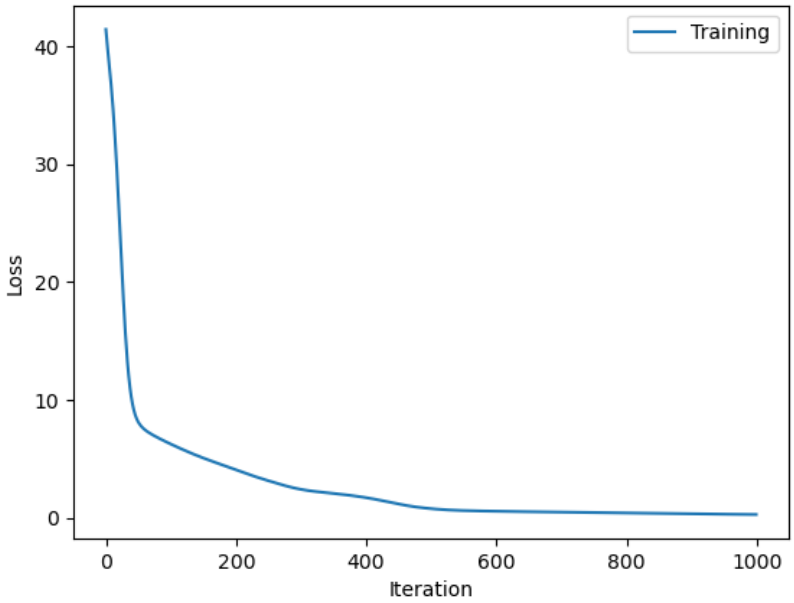

Figure 3.3: Plot of training loss for 1e-3 learning rate and 1.000 optimization steps.

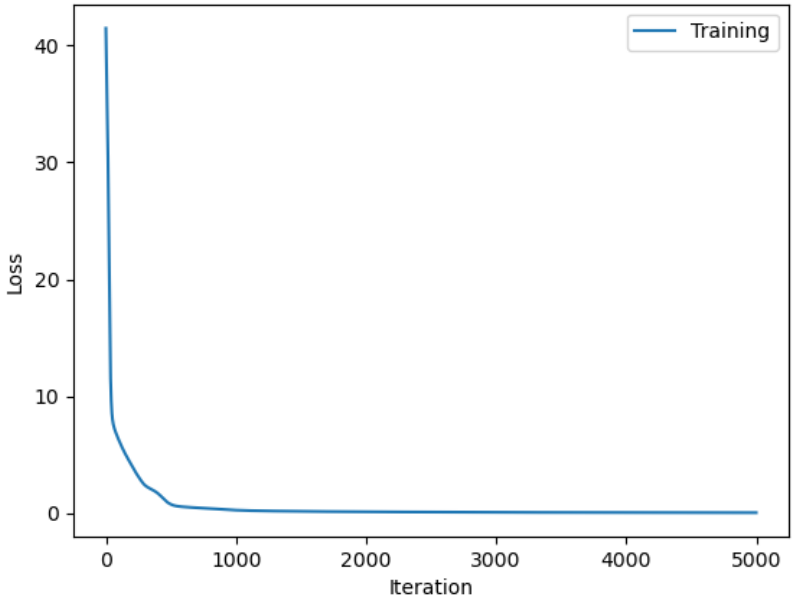

Figure 3.4: Plot of training loss for 1e-3 learning rate and 5.000 optimization steps.

Table 3.1 shows a comparison between the training hyperparameters (learning rate and optimization steps) for the system with L = 12 and their impact. The last column of the table

Table 3.1: Final loss values for different learning rates and optimization steps, for the system L=12.

Learning rate	Optimization steps	Final loss value
1e-4	500	$\approx$ 7.2863625
1e-4	1.000	$\approx$ 5.6571192
1e-4	2.000	$\approx$ 2.90747203
1e-4	3.000	$\approx$ 1.09232501
1e-4	5.000	$\approx$ 0.4844158
1e-3	500	$\approx$ 0.7859934
1e-3	1.000	$\approx$ 0.2871249
1e-3	2.000	$\approx$ 0.145365
1e-3	3.000	$\approx$ 0.1010354
1e-3	5.000	$\approx$ 0.0715654

displays the final loss value, showcasing how fast the loss can decrease.

3.1.2 Time evolution error

As a measure of accuracy, the time evolution error is calculated as in [7]. This is shown in the subsequent plots. As in [7], there are three different wave functions: the exact one ($\psi(t)$); the one obtained by the exact midpoint time-evolution, ($\psi_\Delta(t)$); and the one represented by the network, ($\psi_N(t)$). Therefore, the error of the network with respect is to the exact state is:

$$\epsilon(t) = \psi(t) - \psi_N(t). \tag{3.1}$$

This error is the sum of the error due to the the midpoint method and the error due to the network optimization:

$$\epsilon(t) = \epsilon_\Delta(t) + \epsilon_N(t). \tag{3.2}$$

The plots compare network error with the midpoint error, which both increase time-dependent. For the method to have good results, the error should be as close to zero to zero as possible, therefore it is important to have many optimization steps.

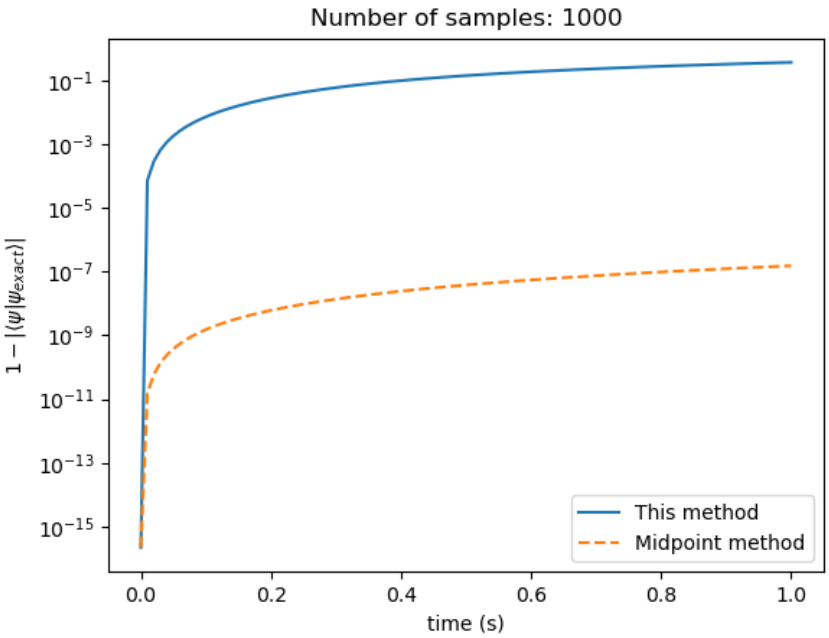

Figure 3.5: Time evolution error for 5 visible states, learning rate of 1e-3 and 1.000 optimization steps per time step.

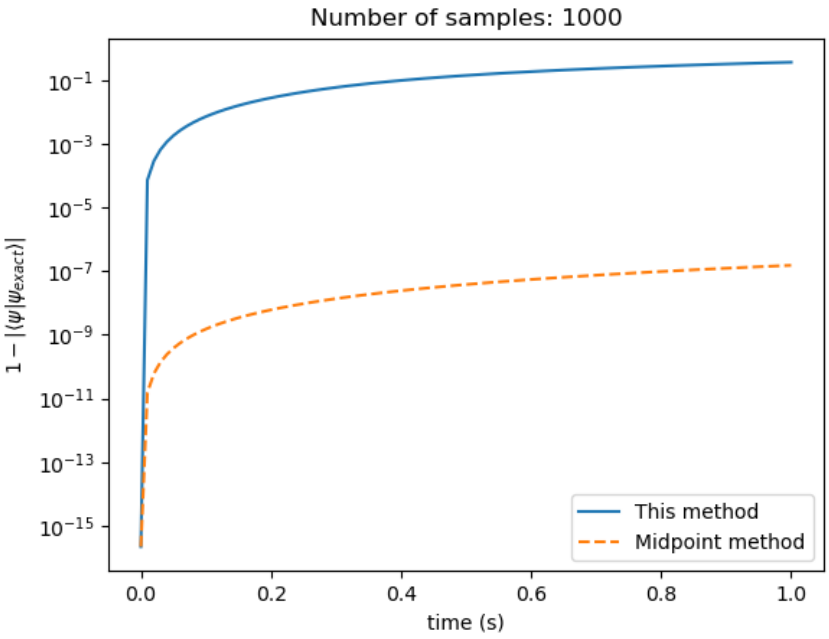

Figure 3.6: Time evolution error for 5 visible states, learning rate of 1e-3 and 5.000 optimization steps per time step.

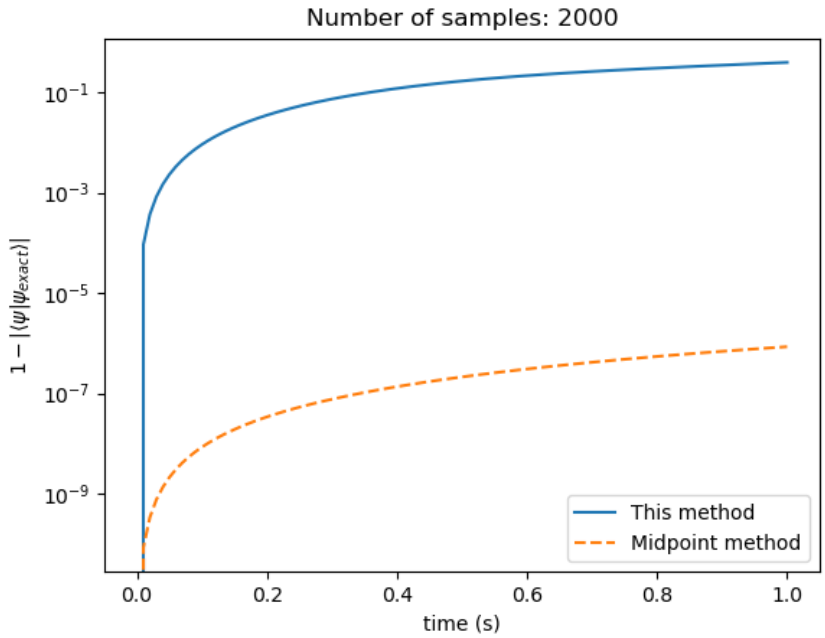

Figure 3.7: Time evolution error for 7 visible states, learning rate of 1e-3 and 1.000 optimization steps per time step.

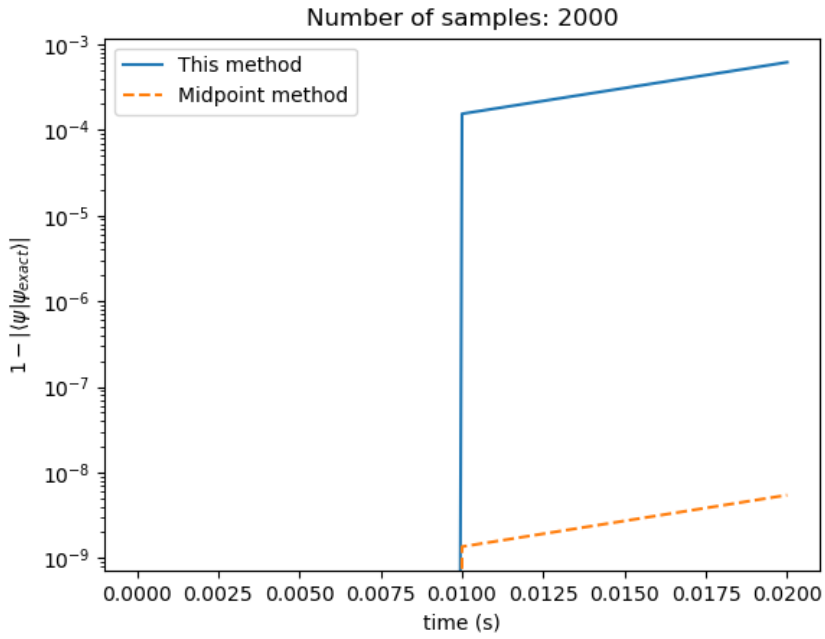

Figure 3.8: Time evolution error for 12 visible states, learning rate of 1e-4 and 5.000 optimization steps per time step.

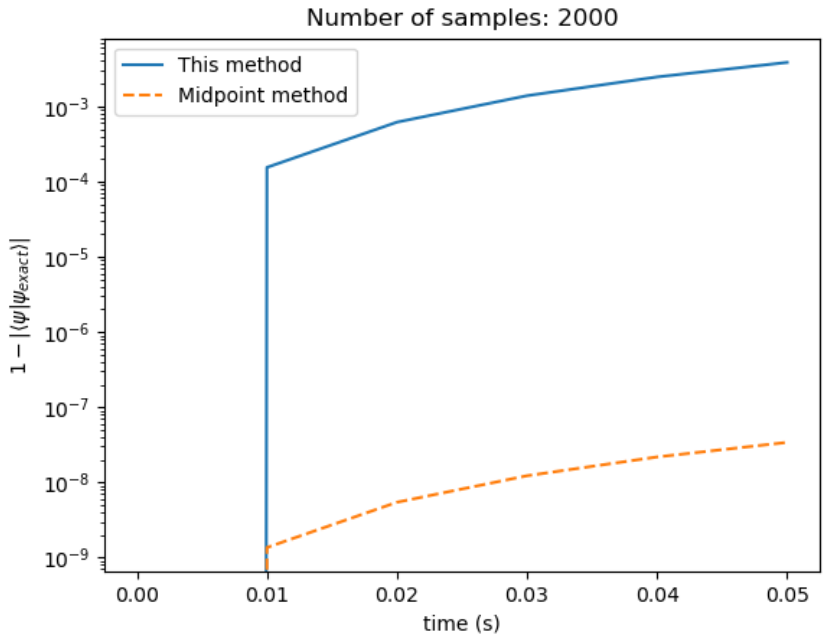

Figure 3.9: Time evolution error for 12 visible states, learning rate of 1e-3 and 1.000 optimization steps per time step.

By analyzing the final loss values and the time evolution errors for the experimented scenarios, the conclusion is that, for good results, there is a need for a low learning rate and a large number of optimization steps (5.000 or more) per time step. Therefore, this emphasis the importance of fast computation, required in the case of a slow-converging loss and thousands of optimization steps.

3.2 GPU performance

This section covers the computation improvements of using a GPU to run the large computations on, compared to a CPU. For these experiments, a personal computer with GeForce MX130, 2GB RAM GPU was used. As mentioned previously, a GPU-based environment was used, with the following packages: Tensorflow 2.0.0, CUDA 10.0.130, cudnn 7.6.5, Python 3.7.0.

The table 3.2 shows a comparison between the GPU and CPU performances for the three systems (L=5, L=7, L=12) and different optimization steps. The improvements for the GPU-based system are significant, especially in larger sizes. Therefore, this has shown that using a GPU is a viable option, even necessary for achieving results in extensive systems.

Table 3.2: Performance summary and comparison between GPU and CPU performances per time step.

L - visible states	M - hidden states	Optimization steps	Sample size	CPU	GPU
5	25	500	100	16.246 s	9.3745 s
5	25	1000	1000	32.562 s	18.849 s
5	25	5000	1000	162.46 s	93.745 s
7	35	500	2000	30.6178 s	19.6185 s
7	35	1000	2000	61.2556 s	39.242 s
7	35	5000	2000	306.178 s	196.185 s
12	60	500	2000	1,634.2653 s	817.689 s
12	60	1000	2000	3,268.5507 s	1,636.380 s
12	60	5000	2000	1,6342.6535 s	8,181.929 s

4 Conclusion

Throughout this thesis, it was demonstrated that methods for neural network optimization constitutes an accurate approach to describe the real time evolution of quantum wave functions. The neural network architecture used for the wave-function ansatz was the RBM, which has been shown to preserve the physicality of our system. The neural network quantum states approximated the implicit midpoint rule method for solving the time-dependent Schrödinger equation, while optimizing the network with respect to a custom cost function .

As a concrete example, this thesis studied the applicability of the transverse-field Ising model on a one-dimensional lattice of different sizes L=5, L=7, and L=12. The motivation of the thesis was to find a strategy to reduce the exponential complexity of the many-body wave function. Therefore, the use of the GPU was employed in delivering viable results. Simulation results have shown that using a GPU is a viable option, even necessary for achieving results in extensive systems.

Although the method has been proven accurate, the project can be extended towards a GPU parallelization technique for reducing the computation time even further. Also, the neural network architecture and the energy function can be generalized by employing the use of CNNs or Deep Boltzmann machine (DBM)s. Nevertheless, the method employed in the thesis has promising results for its simulations, comparable to the state-of-the-art.

5 Future work

This thesis has investigated how the neural networks can be used to describe the real time evolution of quantum wave functions, focusing on the RBM architecture. However, the approach can be extended.

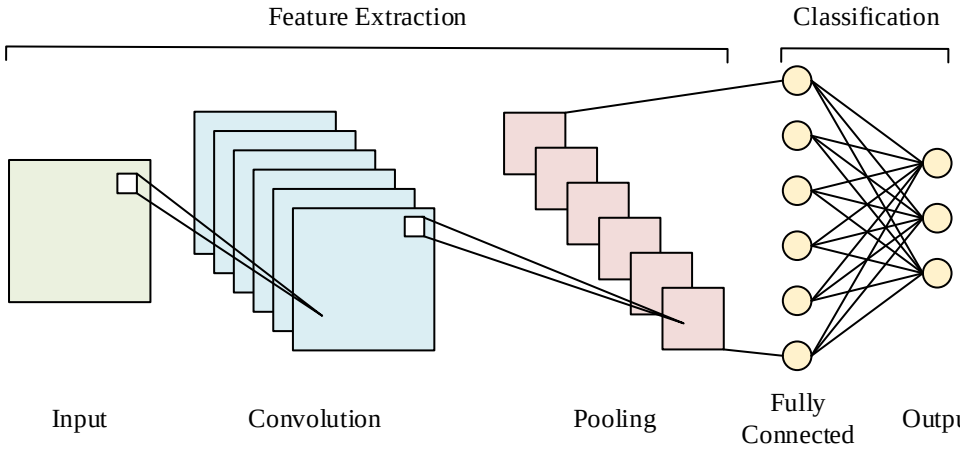

Figure 5.1: Schematic diagram of a basic convolutional neural network (CNN) architecture.

To begin with, the neural network architecture can be generalized to a CNN, as researched in [7] and [8]. As mentioned in [8], CNNs include the RBMs as a special case of a fully connected single layer CNN with a fixed activation function. The hidden layer size, which specifies the size of an RBM, corresponds to the number of channels in a a CNN architecture, and the filter size equals the linear extend of the system [8]. In contrast with the simple architecture of RBMs, CNNs are deep neural networks with sparse connectivity and feature extraction layers. In Fig. 5.1, a schematic diagram of a basic CNN is shown, where convolution (feature extraction) layers and pooling (feature selection) layers are highlighted. Considering the time evolution governed by the two-dimensional Ising model on a LxL lattice, the deep nature of CNNs could be exploited for large systems. According to [7], the ansatz for the wave function is $\psi(\sigma) = exp(CNN(\sigma))$, where $CNN(\sigma)$is the output of a CNN.

Moreover, due to the power limitations of RBMs, there are extensions of their architecture, which preserve physical system properties and applicability in quantum many-body systems. For example, DBMs [16] have at least two hidden layers and no intra-layer connection, in contrast with RBMs. In [16], reinforcement learning has been used to train DBMs to approximate ground states.

Furthermore, in this thesis, only one GPU has been employed. The computation time in table 3.2 could be improved by integrating more resources. Future work aims at GPU parallelization [17], in order to mitigate the computation time for large lattices. In [8], the authors use a parallel implementation of the time evolution algorithm.

List of Figures

List of Tables

Acronyms

ANN Artificial Neural Network. 5

CNN Convolutional Neural Network. 4, 23, 24

DBM Deep Boltzmann machine. 23, 24

GPU Graphics Processing Unit. 4, 21, 23, 25

MPS Matrix Product States. 3

ODE ordinary differential equation. 4, 10

RBM Restricted Boltzmann Machine. 4–9, 13, 23, 24

SGD Stochastic Gradient Descent. 13

SR Stochastic Reconfiguration. 6, 10

Bibliography

[1] G. Carleo and M. Troyer. "Solving the Quantum Many-Body Problem with Artificial Neural Networks". In: *Science* 355 (Feb. 2017), p. 602. DOI: 10.1126/science.aag2302.

[2] D. J. Griffiths. *Introduction to Quantum Mechanics (2nd ed.)* Prentice Hall. ISBN 978-0-13-111892-8., 2004.

[3] M. Navascues and T. Vertesi. *Bond dimension witnesses and the structure of homogeneous matrix product states*. 2018. arXiv: 1509.04507 [quant-ph].

[4] E. Stoudenmire and D. Schwab. "Supervised Learning with Quantum-Inspired Tensor Networks". In: (May 2016).

[5] J. Han, J. Lu, and M. Zhou. "Solving high-dimensional eigenvalue problems using deep neural networks: A diffusion Monte Carlo like approach". In: *Journal of Computational Physics* 423 (2020), p. 109792. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2020.109792. URL: https://www.sciencedirect.com/science/article/pii/S0021999120305660.

[6] R. Abu Saleem, M. I. Radaideh, and T. Kozlowski. "Application of deep neural networks for high-dimensional large BWR core neutronics". In: *Nuclear Engineering and Technology* 52.12 (2020), pp. 2709–2716. ISSN: 1738-5733. DOI: https://doi.org/10.1016/j.net.2020.05.010. URL: https://www.sciencedirect.com/science/article/pii/S1738573320301637.

[7] I. López-Gutiérrez and C. Mendl. "Real time evolution with neural-network quantum states". In: (Dec. 2019).

[8] M. Schmitt and M. Heyl. "Quantum many-body dynamics in two dimensions with artificial neural networks". In: (Dec. 2019).

[9] N. Schuch, I. Cirac, and F. Verstraete. "Computational Difficulty of Finding Matrix Product Ground States". In: *Physical Review Letters* 100.25 (June 2008). ISSN: 1079-7114. DOI: 10.1103/physrevlett.100.250501. URL: http://dx.doi.org/10.1103/PhysRevLett.100.250501.

[10] S. Sorella. "GREEN FUNCTION MONTE CARLO WITH STOCHASTIC RECONFIGURATION". In: *Physical Review Letters* 80 (1998), pp. 4558–4561.

[11] C.-Y. Park and M. Kastoryano. "Geometry of learning neural quantum states". In: *Physical Review Research* 2 (May 2020). DOI: 10.1103/PhysRevResearch.2.023232.

[12] T. Anderson. "Split-Complex Convolutional Neural Networks". In: (2017).

[13] N. Guberman. "On Complex Valued Convolutional Neural Networks". In: *CoRR* abs/1602.09046 (2016). arXiv: 1602.09046. URL: http://arxiv.org/abs/1602.09046.

[14] C. Guo and D. Poletti. "A scheme for automatic differentiation of complex loss functions". In: *CoRR* abs/2003.04295 (2020). arXiv: 2003.04295. URL: https://arxiv.org/abs/2003.04295.

[15] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[16] X. Gao and L. Duan. "Efficient representation of quantum many-body states with deep neural networks". In: *Nat Commun* 662 (2017). DOI: 10.1038/s41467-017-00705-2.

[17] S. Pal, E. Ebrahimi, A. Zulfiqar, Y. Fu, V. Zhang, S. Migacz, D. Nellans, and P. Gupta. "Optimizing Multi-GPU Parallelization Strategies for Deep Learning Training". In: *IEEE Micro* 39.5 (2019), pp. 91–101. DOI: 10.1109/MM.2019.2935967.