Approximating Matrix Product States with Machine Learning

APPROXIMATING MATRIX PRODUCT STATES WITH MACHINE LEARNING

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Bachelor of Arts

in

Physics

by

Sam Greydanus

DARTMOUTH COLLEGE

Hanover, New Hampshire

May 31, 2017

Examining Committee:

_____

James Whitfield, Chair

_____

Miles Blencowe

_____

Lorenza Viola

Abstract

Obtaining compressed representations of entangled systems is an important open problem in quantum mechanics. The Density Matrix Renormalization Group (DMRG) algorithm introduced by S. R. White in 1992 [35] has been successful at solving one-dimensional cases but does not generalize well to arbitrary dimensions. We explore the possibility of using neural network models to solve ground state problems in place of DMRG. In experiments on a system of four spin-$\frac{1}{2}$ particles interacting by the 1D Heisenberg Hamiltonian, we show that this approach can approximate ground state energies and Matrix Product State coefficients to a mean percent error of less than 5%. Furthermore, we use deep learning to obtain MPS coefficients for low-lying energy states directly from the system Hamiltonian. Our findings suggest that neural networks, which generalize well to arbitrary dimensions, could be useful tools for solving 2D and 3D systems where DMRG fails.

Acknowledgements

First, a heartfelt thanks to Professor James Whitfield for advising me. He spent countless hours helping me grasp difficult concepts. Thanks to him, I learned to see the subatomic world with new clarity. I always left his office having learned something interesting.

I thank Professor Kristina Lynch for guiding me through the technical aspects of completing a thesis. I thank my friends for keeping me sane throughout the stressful and often exhausting process. Finally, I thank my parents, James and Judy Greydanus, for supported me in all my endeavors, be they raising the perfect show pig or writing an honors thesis about quantum entanglement.

Contents

Chapter 1

Introduction

1.1 Statement of thesis

Simulating entangled quantum systems is difficult because the cost of simulation grows as $\mathcal{O}(d^N)$ for N sites with d states each. Many interesting entangled systems have low Schmidt rank[1]; algorithms such as the Density Matrix Renormalization Group (DMRG) can simulate these systems in $\mathcal{O}(Nm^3)$ time where m is a used-defined upper bound on the dimensionality of the system. DMRG, and algorithms like it, do not generalize well to 2D and 3D systems.

This thesis explores the possibility of using deep learning algorithms to perform the same simulations, also in $\mathcal{O}(Nm^3)$ time. The generality of the deep learning approach means that it can extend naturally to 2D and 3D systems.

In experiments on a four-particle, spin-$\frac{1}{2}$ system, we used the deep learning approach to calculate ground state energies and Matrix Product State (MPS) coefficients to a mean percent error of less than 5%. We also had success in estimating

[1] see Appendix A.

the MPS coefficients of very low-energy states directly from system Hamiltonians.

1.2 Background

Quantum theory is notorious for violating classical intuitions. Indeed, well after its initial formulation in the 1920's, physicists questioned its legitimacy. The Einstein-Podolsky-Rosen (EPR) paradox uncovered one its most bizarre consequences: entanglement. Though this "spooky action at a distance" was initially meant to discredit quantum theory, it soon grew into a field of its own [9].

Entanglement tells us that we cannot describe the behavior of one particle in an entangled system without first describing the entire system. It is an inherent property of the quantum mechanics; we would not have quantum devices without it [8]. However, it can be frustrating from a theoretical standpoint. It complicates even the simplest calculations and makes predicting the behaviors of quantum systems an immense challenge. To better understand how entangled systems behave, researchers usually build computer simulations of entangled systems to study their properties. The ability to accurately simulate entangled systems will help us build the next generation of quantum computers, clocks, and ciphers.

Deep learning represents the state of the art in computer vision, translation, and artificial intelligence [19, 37, 21]. It enables computers to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. This property is what enables deep learning to perform complex nonlinear transformations on high-dimensional data [11].

Researchers have used deep learning to accelerate drug discovery, identify ex-

otic particles in high-energy physics data, simulate fluid turbulence, shape microfluid flow, and predict the quantum mechanical properties of small organic molecules to within chemical accuracy [25, 3, 30, 28, 10]. These examples illustrate how deep learning has already become a valuable tool for scientific research.

1.3 Overview

Chapters 2 and 3 will review entanglement and the techniques physicists have developed to study it. We will provide simple physical examples of entangled systems, introduce useful formalism such as the tensor product and the Matrix Product State (MPS), and show how these mathematical objects can describe the properties of entangled systems efficiently.

Chapter 4 will describe the Density Matrix Renormalization Group (DMRG) algorithm which is the standard way of obtaining MPS coefficients for an entangled system. With the goal of demonstrating the strengths and weaknesses of this algorithm, we will begin with a mathematical formalism, then provide pseudocode, and finally show results from an implementation of DMRG in Python.

In Chapter 5 we explore how deep learning can improve our understanding of entangled systems. After introducing formalism for multilayer neural network models, we will use them to solve for ground state energies and eigenstates of entangled systems and obtain MPS coefficients. Finally, in Chapter 6 we provide closing remarks and discuss what place, if any, deep learning has as a tool for approximating quantum systems.

Chapter 2

Entanglement

2.1 Definition

A system is said to be entangled when its components cannot be described independently of each other. In other words, knowledge of the whole system is necessary to understand the parts. We use the vague words *system* and *components* intentionally here. In quantum mechanics, the components are usually particles (such as electrons) and the system is usually several particles with coupled spins. Yet the formalism applies to a much broader range of problems. Consider the red-blue stick example.

Imagine a stick with one red end and one blue end. Without looking at the ends, break the stick into two pieces and toss one end aside at random. Now, if you look at the remaining piece and it is red, you can immediately reason that the end you tossed aside was blue. In this example, the two pieces of the red-blue stick are said to be classically correlated. Quantum entanglement is similar to classical correlation, but stronger. Consider the following modification:

What if both ends of the stick are blue? Without looking at either end, you could reason that the piece you tossed aside was blue. Since looking at the piece in your hand does not tell you anything new about the color of the piece you tossed aside, this system is *not* entangled.

The red-blue stick is an example of entanglement in the classical (statistical) sense. A quantum red-blue stick would have the same amount of red paint and blue paint, but each end would start out with both some mixture - or superposition - of red and blue. When you throw away one end and observe the color of the other, its color would become either red or blue *as* you observed it.

2.2 The Einstein-Podolsky-Rosen (EPR) Paradox

The suspicious reader might point out that the quantum red-blue stick example violates relativity. When you observe the color of the piece in your hand to be, say, red, the color of the piece you tossed aside must change to blue *instantaneously*. Yet relativity says that nothing can travel faster than the speed of light, not even information.

This is the heart of the Einstein-Podolsky-Rosen (EPR) paradox [9]. It is a good example of how quantum entanglement violates classical intuitions. To resolve the EPR paradox, one must avoid working in the reference frame of a single piece of the stick and instead consider the system as a whole. Repeating the experiment many times, you will notice a correlation between the colors of the two pieces, but this correlation will appear to be a property of the full system, rather than one piece acting upon the other.

The EPR paradox is an important introduction to quantum entanglement be-

cause it shows why one must model an entangled system as a whole. Much of the formalism that follows is based on the idea that each component should be treated on an equal footing.

2.3 Two spin-$\frac{1}{2}$ particles

The simplest physical example of entanglement is a system of two spin-$\frac{1}{2}$ particles (e.g. electrons). Each electron can have a spin of either +$\frac{1}{2}$ ($\uparrow$) or -$\frac{1}{2}$ ($\downarrow$). The combined system has four states: $\{\uparrow\uparrow, \uparrow\downarrow, \downarrow\uparrow, \downarrow\downarrow\}$.

Since electrons are indistinguishable, we must always superimpose $\uparrow\downarrow$ with $\downarrow\uparrow$ (as in equations 2.2 and 2.4). This gives us four possible states: a triplet and a singlet, in terms of angular momentum number S_z. In $|S, S_z\rangle$ notation,

$$|1,1\rangle = \uparrow\uparrow \qquad \text{triplet} \tag{2.1}$$

$$|1,0\rangle = \frac{1}{\sqrt{2}}(\uparrow\downarrow + \downarrow\uparrow) \qquad \text{triplet} \tag{2.2}$$

$$|1,-1\rangle = \downarrow\downarrow \qquad \text{triplet} \tag{2.3}$$

$$|0,0\rangle = \frac{1}{\sqrt{2}}(\uparrow\downarrow - \downarrow\uparrow) \qquad \text{singlet} \tag{2.4}$$

In the system above, only equations 2.2 and 2.4 represent entangled states. The others are product states (all up or all down) because we can know the state of one subsystem without knowledge, and without affecting, the state of the other.

2.4 Formal aspects

From now on, I will refer to the components of an entangled system as *sites* (I will use N to denote their total number) and the set of values available to each site as *states* (d).

How can we represent systems with an arbitrary number of sites? How can we represent sites with an arbitrary number of states? In order to describe entanglement in the general case, we need to introduce a powerful operation called the *tensor product*.

2.4.1 Tensor product

The tensor product is denoted by the $\otimes$ symbol and operates on two matrices to produce a third.

$$A^{[m\times n]} \otimes B^{[p\times q]} = C^{[mp\times nq]} \tag{2.5}$$

For example, if A and B are both 2×2 matrices as shown

$$A^{[2\times 2]} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \qquad B^{[2\times 2]} = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \tag{2.6}$$

Then the tensor product looks like

$$A^{[2\times2]} \otimes B^{[2\times2]} = C^{[4\times4]} \tag{2.7}$$

$$= \begin{pmatrix} A_{11}\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} & A_{12}\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \\ A_{21}\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} & A_{22}\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \end{pmatrix} \tag{2.8}$$

$$= \begin{pmatrix} A_{11}B_{11} & A_{11}B_{12} & A_{12}B_{11} & A_{12}B_{12} \\ A_{11}B_{21} & A_{11}B_{22} & A_{12}B_{21} & A_{12}B_{22} \\ A_{21}B_{11} & A_{21}B_{12} & A_{22}B_{11} & A_{22}B_{12} \\ A_{21}B_{21} & A_{21}B_{22} & A_{22}B_{21} & A_{22}B_{22} \end{pmatrix} \tag{2.9}$$

Now, for two sites $\left|\psi^d_{\chi=1}\right\rangle$ and $\left|\psi^d_{\chi=2}\right\rangle$ where each has d states, we can represent the full system with equation 2.10.

$$|\psi_{sys}\rangle = \left|\psi^d_{\chi=1}\right\rangle \otimes \left|\psi^d_{\chi=2}\right\rangle \tag{2.10}$$

For example, if $\left|\psi^{d=2}_{\chi=1}\right\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\left|\psi^{d=2}_{\chi=2}\right\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, then the system can be described by equation 2.11.

$$|\psi_{sys}\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \tag{2.11}$$

We can combine operators, such as the Hamiltonians $H_{\chi=1}$ and $H_{\chi=2}$ of the two

sites, in the same way.

2.4.2 The curse of dimensionality

The curse of dimensionality refers to a set of problems which arise when working with very high dimensional data. With the tensor product, we can show that the dimension of the system scales exponentially with the number of sites. It is not long before the curse of dimensionality severely limits our ability to compute an entangled system's properties.

Imagine we want to compute the ground state, ψ_0 of a two-particle system. The state ψ_0 is given by the eigenvector, $\lambda_0 = E_0$ that corresponds to the smallest eigenvalue - and lowest energy - of the system Hamiltonian, H_{sys}. Since $H_{sys} = H_1 \otimes \mathbb{1} + \mathbb{1} \otimes H_2 + H_{12}$ (where H_{12} is the interaction Hamiltonian), it is just a 4×4 matrix and can be easily diagonalized.

Now consider a system of 21 spin-$\frac{1}{2}$ particles. The dimensionality of H_{sys} becomes $2^{21} \times 2^{21}$, or 4194304×4194304; it takes a sparse Hermitian eigensolver running on a 2014 MacBook a full minute to find the ground state (see Figure 2.1). Systems for which $d > 30$ are not practical to compute. Simply saving a *sparse* H_{sys} for $d = 40$ fills 4TB of memory!

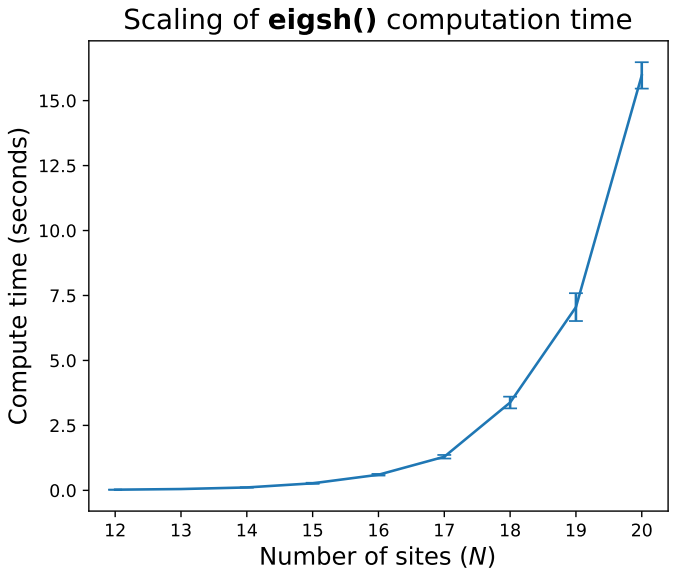


Figure 2.1: Runtimes of a sparse Hermitian eigensolver on a 2014 MacBook. Error bars denote one standard deviation.

In order to simulate larger entangled systems, we need a representation that does not scale exponentially with system size: enter Matrix Product States.

Chapter 3

Matrix Product States

3.1 The MPS Ansatz

Consider a 1D system of entangled states such as the one on the right side of Figure 3.1. Since the system is entangled, it is impossible to decompose it into a tensor product of two subspaces (as expressed by the inequality $|\psi\rangle \neq |\psi\rangle_A \otimes |\psi\rangle_B$). However, when the bond dimension (Schmidt rank) is low, the system on the left side of Figure 3.1 is a good approximation. This system can be expressed as a product state, so it is a far more efficient representation.

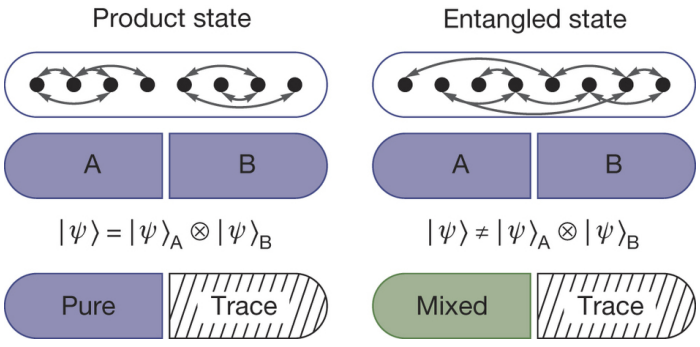


Figure 3.1: The MPS ansatz, taken from [16].

Matrix Product States (MPS) are a generalization of this idea. Using a series of $N-1$ Schmidt decompositions[1], one can re-express the d^N coefficients of a state ψ in terms of about $(2d^2+d)N$ parameters [8]. Letting d be the local dimension of a given site and $A[i]^{s_i}$ be the matrix of MPS coefficients when the i-th site is in state s_i in range=$(1,d)$. The matrix product state is then defined by equation 3.1.

$$|\psi_{mps}\rangle = \sum_{s_1,\ldots,s_N=1}^{d} Tr(A[1]^{s_1}A[2]^{s_2}\ldots A[N]^{s_N})\,|s_1,\ldots s_N\rangle \tag{3.1}$$

Thus, we will define d different $A[i]^{s_i}$ matrices for each site i. Each of these matrices has dimension $[m \times m]$, where m is defined by the user. These matrices contain coefficients which, when multiplied and traced over, reconstruct the full state, ψ.

3.2 Simple examples

To obtain a better intuition of how MPS coefficients combine to reconstruct a state ψ, consider these example MPS states from Eckholt (2005) [8].

Pure State $|\mathbf{000\ldots0}\rangle$ With dimension $d=1$, we have the 1×1 matrices:

$$A[i]^0 = 1 \quad A[i]^1 = 0 \tag{3.2}$$

GHZ State $|\mathbf{000\ldots0}\rangle + |\mathbf{111\ldots1}\rangle$ With dimension $d=2$, we have the 2×2 matri-

[1]See Appendix A for an explanation of Schmidt decompositions.

ces (see Figure 5.7b for a visual representation):

$$A[i]^0 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad A[i]^1 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \tag{3.3}$$

W State $|\mathbf{100}\rangle + |\mathbf{010}\rangle + |\mathbf{001}\rangle$ With dimension $d = 3$, we have the 3×3:

$$A[1]^0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad A[2]^0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} A[3]^0 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$A[1]^1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad A[2]^1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} A[3]^1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

3.3 Expectation values

How does MPS speed up computations on entangled systems? Imagine that we want to compute the expectation value of operator O for a $d = 2$ system. The operator for the full system would be

$$\begin{aligned} O = &(O_1 \otimes \mathbb{1} \otimes \mathbb{1} ... \otimes \mathbb{1}) \\ &+ (\mathbb{1} \otimes O_2 \otimes \mathbb{1} ... \otimes \mathbb{1}) \\ &+ (\mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1} ... \otimes O_N) \end{aligned}$$

Now, define E_{O_i} as the *transfer matrix* which corresponds to local operator O_i:

$$E_{O_i} := \sum_{s_i, s_i'=1}^{d} \langle s_i' | O_i | s_i \rangle \left(A[i]^{s_i} \otimes (A[i]^{s_i'})^\dagger \right) \tag{3.4}$$

With these definitions, Eckholt (2005) [8] showed that the expectation value in the MPS picture is given by equation 3.5.

$$\langle \psi_{mps} | O | \psi_{mps} \rangle = Tr[E_{O_1} ... E_{O_N}] \tag{3.5}$$

In other words, computing the expectation value of a system in the MPS representation corresponds to multiplying a set of transfer matrices and tracing the result. This computation scales as $\mathcal{O}(d^2 m^6 N)$ where m is a user-defined parameter that limits the size of the A matrices and N is the total number of sites.

Computing the expectation value of an observable using the full system Hamiltonian corresponds to multiplying a vector ψ_{sys} of dimension d^N with a matrix. The computational complexity of this operation is $\mathcal{O}(d^{2N})$.

The MPS approximation is valid for 1D quantum systems (with low Schmidt rank) and that it offers an immense speedup for tasks such as computing the expectation values. However, we have not yet seen how to calculate the coefficients of the A matrices. In the next section, I will introduce one method for computing these coefficients, the Density Matrix Renormalization Group (DMRG) algorithm.

Chapter 4

Density Matrix Renormalization Group

4.1 The DMRG algorithm

The idea behind the Density Matrix Renormalization Group (DMRG) algorithm is to find accurate approximations of the ground state and low-lying energies of interacting quantum systems [35]. The algorithm is a generalization of the Wilson numerical renormalization group [36]. Both techniques are useful because they can integrate out unimportant degrees of freedom by applying a succession of renormalization group transformations to a system. In other words, only the m principal eigenvectors of a subsystem $\psi_{N=l}$ are used to construct subsystem $\psi_{N=l+1}$ in each step of DMRG (where m is a user-defined value).

4.1.1 Formalism

Consider a 1D system of particles with local interactions. In the DMRG picture, we start by treating the system as a collection of separate pieces – *free sites* – and slowly combine them to form a *block*. When the block contains the entire system, we can use it to calculate ground states, ground state energies, and expectation values of the system. The free sites are particles that have not yet been added to H_{sys}.

Base case. The DMRG algorithm begins with a block composed of a single site as shown in Figure 4.1.

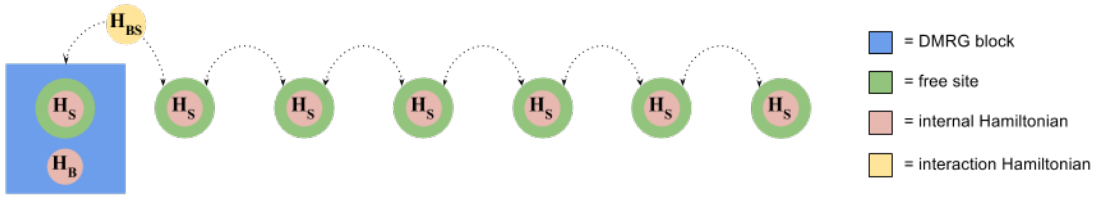


Figure 4.1: DMRG begins with a block of size $N = 1$

Enlarge. The next step of DMRG is to enlarge the system by adding an adjacent free site to the block. If H_B is the Hamiltonian of the block, H_S is the Hamiltonian of the free site, and H_{BS} is the interaction Hamiltonian between the free site and the block, then the Hamiltonian of the enlarged block will be

$$H_E = H_B + H_S + H_{BS} \tag{4.1}$$

where the dimension of H_E is a factor of d larger than the dimension of H_B (where d is the dimension of H_S). Figure 4.2 shows this step.

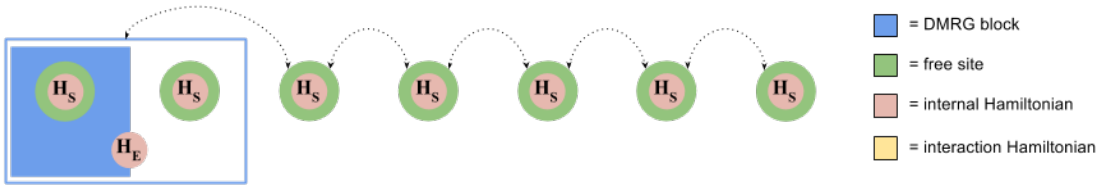


Figure 4.2: The *enlarge* step consists of adding a free site to the block.

Build $\mathbf{H_{super}}$. To compute an accurate ground state, the next step is to build a *superblock* Hamiltonian, H_{super}, by reflecting the enlarged block and merging the two systems as shown in Figure 4.3 and equation 4.2, creating a block with $2(l+1)$ sites.

$$H_{super} = H_E + H_{E'} + H_{SS'} \tag{4.2}$$

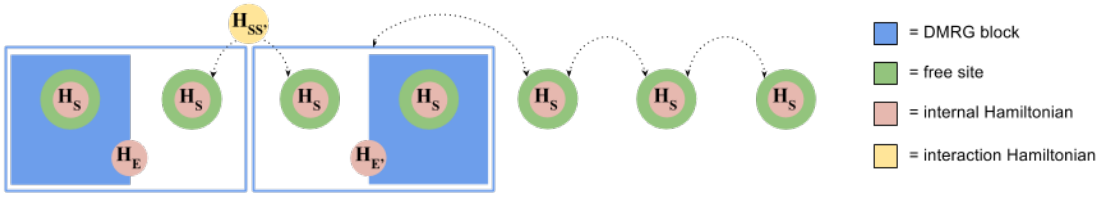


Figure 4.3: Building H_{super}, as given by equation 4.2.

Decompose $\mathbf{H_{super}}$. This step of DMRG solves for the ground state energy of the block by diagonalizing H_{super} (which is of dimension $[2md \times 2md]$) and saving the smallest eigenvalue, E_0, and its corresponding eigenvector, ψ_0. Equation 4.4 describes this process. In practice, one can use a sparse Hermitian eigensolver such as scipy's `eigsh()` function [31].

$$U\Lambda U^{\dagger} = H_{super} \quad where \quad \lambda_i = \Lambda_{ii} \tag{4.3}$$

$$\text{keep} \quad E_0 = min(\lambda_1, \lambda_2 ... \lambda_N), \quad |\psi_{E_0}\rangle \tag{4.4}$$

The reduced density and transformation matrices. The density matrix is a partial trace over ψ_{E_0} as shown in equation 4.5. In practice, this can be accomplished by reshaping ψ_{E_0} and taking the dot product of the result with its complex conjugate [1].

$$\rho_{N=l+1}^{[md \times md]} = Tr_R \left|\psi_{E_0}\right\rangle \left\langle\psi_{E_0}\right| \tag{4.5}$$

The reduced density matrix is useful for constructing the transformation matrix, O, which DMRG uses to reduce the dimensionality of the enlarged block while preserving as much information as possible about the system's lowest energy states.

The transformation matrix is constructed from the largest m eigenvectors of $\rho_{N=l+1}$, where m is a user-defined value which places an upper bound on the dimensionality of ψ_B and H_B. Large values of m produce more accurate results but are more expensive to compute. Small values of m produce less accurate results and are more efficient.

The construction of $O^{[md \times m]}$ looks like

$$U \Lambda U^\dagger = \rho_{N=l+1}^{[md \times md]} \quad where \quad \lambda_i = \Lambda_{ii} \tag{4.6}$$

$$(\lambda'_1, \lambda'_2 ... \lambda'_{(md)}) = \texttt{sort\_descending}(\lambda_1, \lambda_2 ... \lambda_{(md)}) \tag{4.7}$$

$$O^{[md \times m]} = \left[u_{\lambda'_1} : u_{\lambda'_2} : \cdots : u_{\lambda'_m}\right] \tag{4.8}$$

$$\tag{4.9}$$

[1] Available at https://github.com/greydanus/psi0nn.

where $u_{\lambda_i'}$ is the eigenvector which corresponds to eigenvalue λ_i'.

Rotate and truncate. Having constructed the transformation matrix, all that remains is to transform every operator S_E of the enlarged block into a new basis of dimension m (equation 4.10

$$(S_E)^{[m\times m]} = (O^\dagger)^{[m\times md]}(S_E)^{[md\times md]}(O)^{[md\times m]} \tag{4.10}$$

Estimate error. Transforming the DMRG block into a smaller-dimensional space destroys information if the eigenvalues λ_i of $\rho_{N=l+1}^{[md\times md]}$ are nonzero for $i > m$. A simple metric for the truncation error the sum of these eigenvalues (equation 4.11

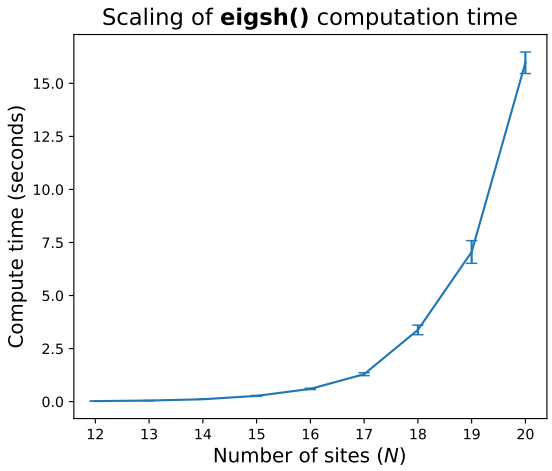
$$\epsilon := \sum_{i>m} \lambda_i = 1 - \sum_{i<m} \lambda_i \tag{4.11}$$

.

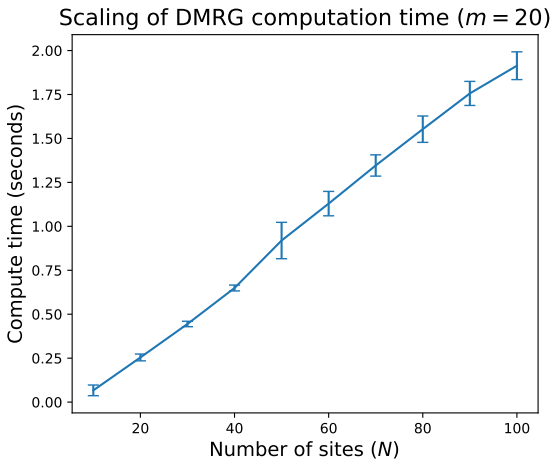
Putting it together. Everything to this point constitutes a single step of DMRG. Having enlarged the block and transformed it to a new basis, the next step is to enlarge the block with a second free site and repeat the steps described above.

There are two variants of DMRG: infinite and finite. Infinite DMRG consists of simply repeating the basic DMRG step until the superblock Hamiltonian contains L free sites, where the user chooses L. Finite DMRG involves performing additional computations on the block once it reaches size L, and generally makes final results more accurate. For the details of finite DMRG, refer to Subsection 4.2.2.

Comparing runtimes for the DMRG approach with exact diagonalization in Figure 4.4, we see that DMRG is much more efficient. We used a 1D chain of spin-$\frac{1}{2}$ particles interacting via the Heisenberg Hamiltonian.



(a) Exponential scaling in d for exact diagonalization.



(b) Linear scaling in d for DMRG.

Figure 4.4: Scaling of runtime with number of sites d (block size), computed on a 2014 MacBook. DMRG can quickly compute the ground state of a $d = 100$ system whereas exact diagonalization cannot.

Pseudocode for infinite DMRG is provided in Section 6.2

4.2 Results

4.2.1 Infinite algorithm

We implemented the infinite DMRG algorithm in Python[2]. For a case study, we used the Heisenberg Hamiltonian with coupling terms $J = J_z = 1$ and local fields set to zero.

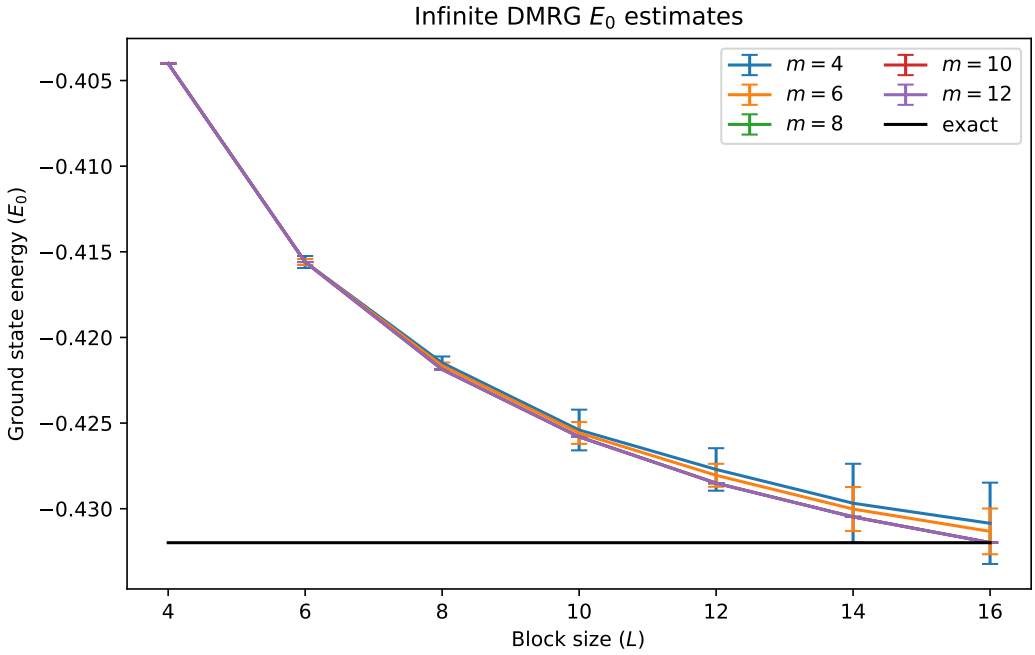
$$H_{Heis} = J_z(S_1^z \otimes S_2^z) + \frac{J}{2}\left(S_1^+ \otimes S_2^- + S_1^- \otimes S_2^+\right) \tag{4.12}$$

We used a total of $L = 16$ sites and tested the algorithm's approximations of

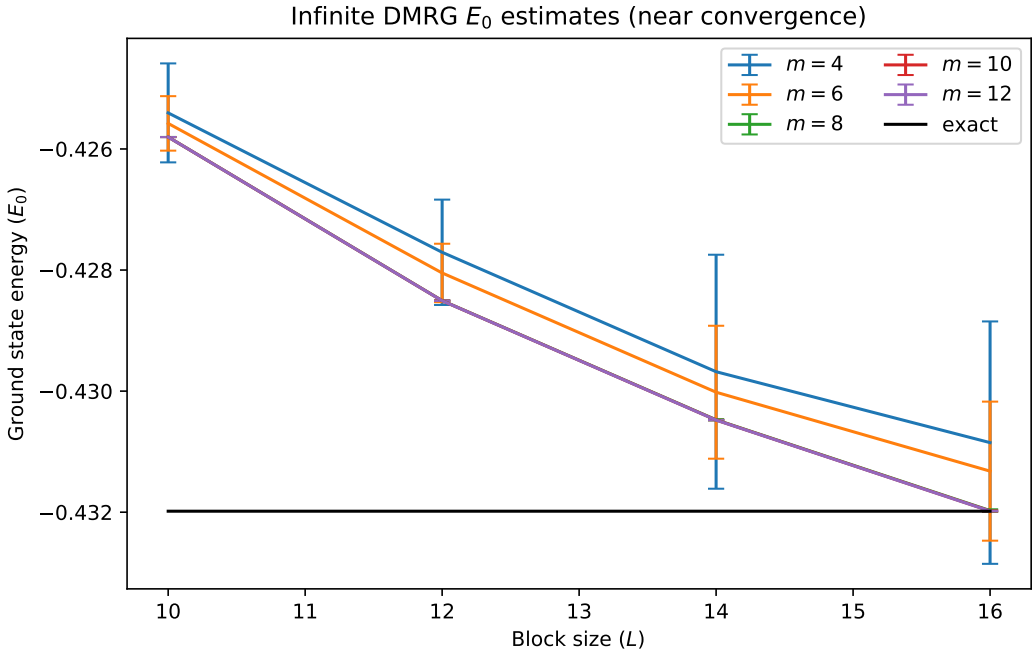
[2]Available at https://github.com/greydanus/psi0nn.

E_0 for $m = \{4, 6, 8, 10, 12\}$ against a ground truth E_0 computed using exact diagonalization. We chose a relatively small system of 16 sites because it allowed us to compare the results directly against exact diagonalization.

Figure 4.5a shows convergence of infinite DMRG for the various values of m and Figure 4.5b is a closer view of DMRG steps for the largest block sizes. Estimates of E_0 begin to diverge for large block sizes in Figure 4.5b as the dimensionality of the physical system grows much larger than the user-defined bond dimension m.



(a) Infinite DMRG, full space.



(b) Infinite DMRG, near convergence.

Figure 4.5: Convergence of the infinite DMRG algorithm to the ground state energy E_0 of a 16-site Heisenberg Hamiltonian. Error bars indicate the accumulation of truncation error (ϵ) after each step of DMRG.

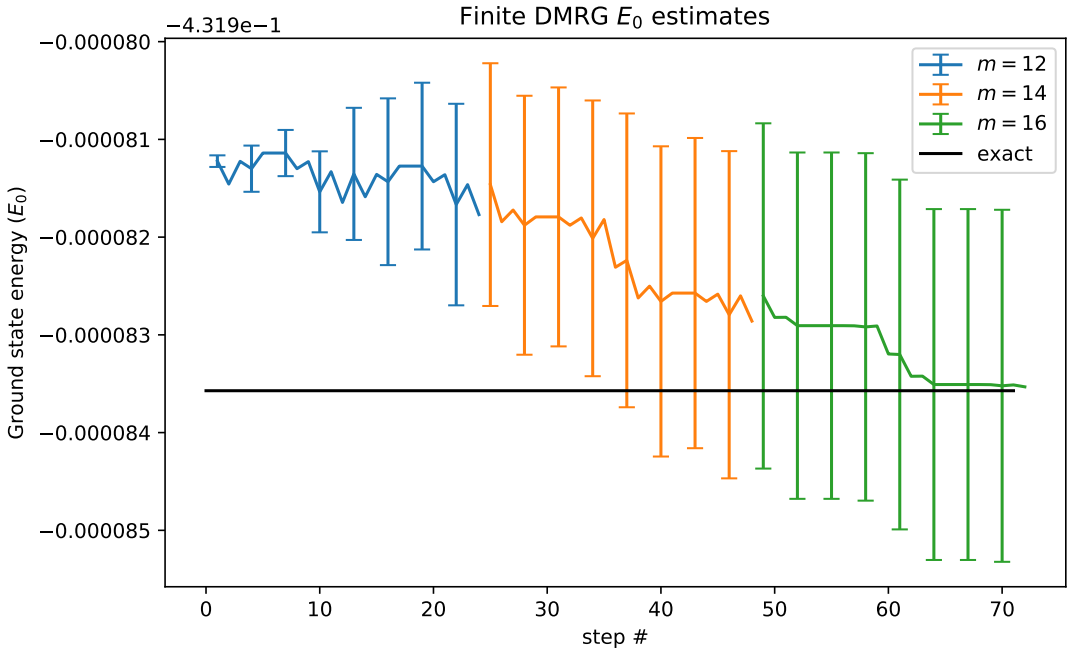
4.2.2 Finite algorithm

Finite DMRG begins with the infinite algorithm. After building the system to desired block size L, the block is split into a left block (initially with all of the sites) and a right block (initially with no sites). Using the same steps described above, sites are transferred from the left block to the right block. Soon, all sites will be in the right block. At this point, the sites are transferred back into the left block. This entire process constitutes a single *sweep* of the finite DMRG algorithm.

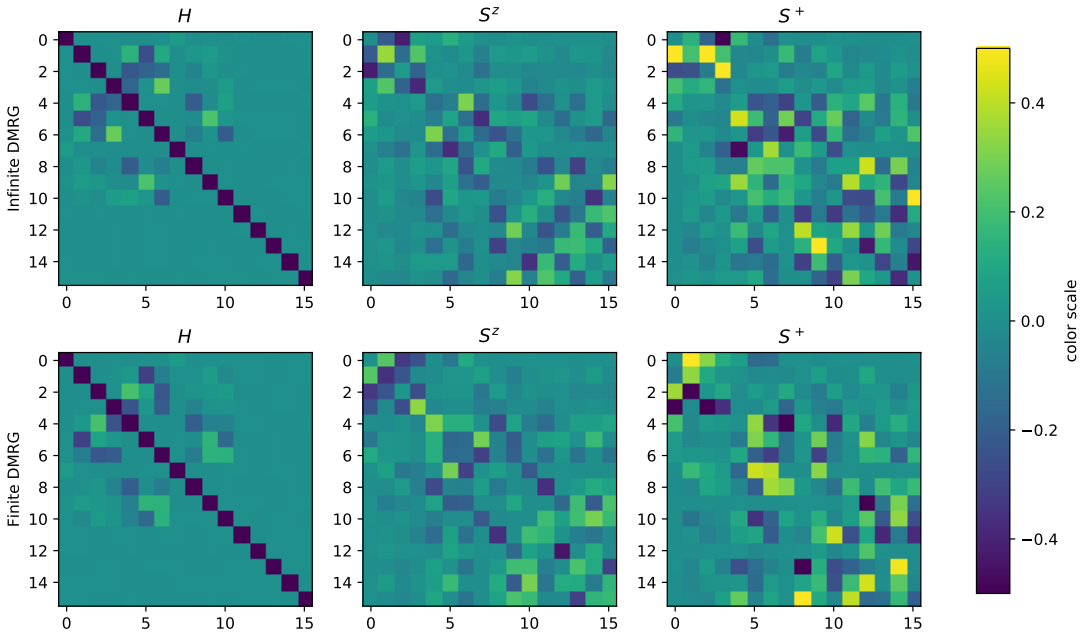
The user can change the block dimensionality, m, between each of the sweeps. For example, Figure 4.6a shows results from three sweeps of finite DMRG, each with a larger m. The initial state of the system in Figure 4.6a was the same as the final state of the system in Figure 4.5b. Each sweep of DMRG increases the accuracy of the ground state energy estimate, $\hat{E}_0$, especially if m is increased as well.

Qualitatively, each sweep of finite DMRG spreads information about local interactions to nearby sites that are indirectly affected by these interactions. For example, an interaction between sites χ_1 and χ_2 might affect the state of χ_3 in a secondary way. Each sweep of finite DMRG adds more information about this secondary interaction to the compressed system, producing a more accurate final picture. These changes correspond to small changes in operators such as the system Hamiltonian H, spin operator S^z, and raising operator S^+. A qualitative picture of these changes is shown in Figure 4.6b.

4.2 Results



(a) Finite DMRG convergence. Error bars represent only the errors introduced by finite DMRG.



(b) Comparison of block operators before and after finite DMRG. Note, for example, the small differences along the off-diagonal elements of the Hamiltonian.

Figure 4.6: Convergence of the finite DMRG algorithm to the ground state energy E_0 of a 16-site Heisenberg Hamiltonian *after* running the infinite algorithm.

4.3 DMRG in the MPS picture

The Density Matrix Renormalization Group (DMRG) algorithm is particularly interesting because it can be reexpressed in terms of Matrix Product States [27, 33]. In fact, DMRG is one of the most common ways of obtaining MPS coefficients for 1D systems [33]. The objective of this section is to provide a sketch of what DMRG looks like in the MPS picture. For a more thorough treatment of this subject, refer to Schollwock (2011) [27].

Connecting DMRG and MPS comes down to reorganizing the $O^{[md\times m]}$ matrices obtained after adding each site (see equation 4.10). Instead of keeping the $O^{[md\times m]}$ transformation matrix as a single $[md \times m]$ matrix, we can instead think of it as d matrices of dimension $[m \times m]$ labeled by s_i ranging from 1 to d. Notationally, we have $O_{s_i j,k} = A^{s_i}_{j,k}$. The site indexing, i, of the A matrices must be included since the transformation matrix is also site dependent. Hence $O[i]^{[md\times m]} \mapsto \{(A[i]^{s_i})^{[m\times m]}\}_{s_i=1}^{d}$ where the $\{\}$ contains the set of d matrices (each of dimension $[m \times m]$) that parameterize site i.

In this chapter, we introduced the DMRG algorithm and showed how it can be used to efficiently obtain ground state energies and matrix product states of 1D systems. In the next chapter, we will explore whether neural networks can perform the same tasks.

Chapter 5

Measuring Entanglement with Neural Networks

"Last year, the cost of a top, world-class deep learning expert was about the same as a top NFL quarterback prospect," said Peter Lee, Vice President of Microsoft Research, in 2014 [32]. Lee's claim is a testament to deep learning's massive surge in popularity over the past several years. Technology companies tend to proselytize the merits of each new technology, but deep learning has also made a name for itself by powering breakthrough results in computer vision, translation, and several areas of basic science (see Section 1.2).

Deep learning is exciting because it enables computers to learn complex multivariate information from raw data. In this chapter, we will introduce the mathematics of neural networks, the building blocks of deep learning, and train them to measure properties of entangled systems.

5.1 Previous work

Deep learning is already an established tool in quantum chemistry. A 2017 paper by Justin Gilmer showed that deep neural networks can predict the quantum mechanical properties of small organic molecules to within chemical accuracy [10]. They can do this several orders of magnitude more quickly than algorithms based on Density Functional Theory (DFT), with little cost in accuracy.

Neural networks can also solve the Schrödinger equation. Mills et al. [20] trained a convolutional neural network to compute the energy spectrum of a single electron, given an arbitrary 2D potential. The training data was generated using an actual Schrödinger equation solver.

Of particular interest is a 2017 paper by Carleo and Troyer which showed that neural networks can determine the ground state and "describe the unitary time evolution of complex, interacting quantum systems" [6]. More specifically, the authors used a neural network to determine ground states of 1D Ising and Heisenberg models for lattices of up to 80 sites. They also used it to solve for the ground state of a 2D (10×10) Heisenberg Hamiltonian.

These are just a few examples of successful research projects that combine deep learning and quantum mechanics. They suggest that neural networks might perform well on other problems in quantum mechanics, in particular, the task of calculating MPS coefficients.

5.2 Neural networks

5.2.1 Formalism

Neural networks can be thought of as multivariate function approximators. Indeed, Hornik et al. [14] proved that multilayer feedforward networks are a class of universal approximators. In other words, for an arbitrary Borel measurable function $\mathbf{y}(\mathbf{x})$, it is possible to make the approximation shown in equation 5.1.

$$\mathbf{y}(\mathbf{x}) \approx \hat{\mathbf{y}}(\mathbf{x}) \quad where \quad \hat{\mathbf{y}}(\mathbf{x}) = f_{NN}(\mathbf{x}, \theta) \tag{5.1}$$

This is accomplished by choosing values for the network's trainable parameters, θ, that minimize the difference between the functions $\mathbf{y}$ and $\hat{\mathbf{y}}$. Think of this process as an optimization problem

$$\arg\min_{\theta} \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) \tag{5.2}$$

where $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ is a loss function; it measures how well $\hat{\mathbf{y}}$ approximates $\mathbf{y}$ [11]. Data scientists use many different loss functions depending on the training objective. A common loss function is L_2 *Loss*:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2} \sum_{i} (\mathbf{y_i} - \hat{\mathbf{y}}_\mathbf{i})^2 \tag{5.3}$$

The basic neural network is a series of linear transformations followed by nonlinear, element-wise "activation" functions such as the sigmoid, hyperbolic tangent, or RELU (REctified Linear Unit) functions [23, 26]. If $\mathbf{x}$ is a finite-dimensional input vector, σ is the element-wise activation function, and $\mathbf{W}$ and $\mathbf{b}$ are matrices of trainable parameters such that $\theta = [\mathbf{W}_1, \mathbf{W}_2, \ldots \mathbf{W}_n, \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]$, then the neural

network function is given by equations 5.4-5.8

$$\hat{\mathbf{y}}^{[1\times s]} = f_{NN}(\mathbf{x}, \theta) = \sigma(\mathbf{z}_n) \tag{5.4}$$

$$= \sigma(\mathbf{h}_n^{[1\times r]} \cdot \mathbf{W}_n^{[r\times s]} + \mathbf{b}_n^{[1\times s]}) \tag{5.5}$$

$$where \quad \mathbf{h}_n^{[1\times r]} = \sigma(\mathbf{h}_{n-1}^{[1\times q]} \cdot \mathbf{W}_{n-1}^{[q\times r]} + \mathbf{b}_{n-1}^{[1\times r]}) \tag{5.6}$$

$$\dots \tag{5.7}$$

$$where \quad \mathbf{h}_2^{[1\times n]} = \sigma(\mathbf{x}^{[1\times m]} \cdot \mathbf{W}_1^{[m\times n]} + \mathbf{b}_1^{[1\times n]}) \tag{5.8}$$

Here the superscripts denote matrix dimensions and the subscripts denote the layer of the neural network. The neural network above has a total of n layers. Networks for which $n > 1$ are said to be deep; in state-of-the-art computer vision models, n can exceed 100 [12, 15]. For the purposes of this project, though, we used $n = \{2, 3\}$.

We can optimize θ using a weight-update algorithm such as stochastic gradient descent. The basic idea is to take a partial derivative of the loss function with respect to every θ_i using the chain rule. For example, imagine we wanted to compute the gradient on $W_n[i, j]$ from equation 5.5 where $\sigma(z)$ is the sigmoid function, $\sigma(z) = \frac{1}{1+e^{-z}}$.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_n[i,j]} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}_n} \frac{\partial \mathbf{z}_n}{\partial \mathbf{W_n}[i,j]} \tag{5.9}$$

$$= \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}_n^{[1\times s]}[j]} \frac{\partial}{\partial \mathbf{W_n}[i,j]} (\mathbf{h}_n^{[1\times r]}[i] \cdot \mathbf{W}_n^{[r\times s]}[i,j] + \mathbf{b}_n^{[1\times s]}[j]) \tag{5.10}$$

$$= \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}_n^{[1\times s]}[j]} \mathbf{h}_n^{[1\times r]}[i] \tag{5.11}$$

for efficiency of notation (and also computation), we can vectorize the calculation:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_n}^{[r\times s]} = (\mathbf{h}_n^\intercal)^{[r\times 1]} \cdot \left(\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}}\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}_n}\right)^{[1\times s]} \quad \textit{vectorize} \tag{5.12}$$

$$= (\mathbf{h}_n^\intercal)^{[r\times 1]} \cdot \hat{\mathbf{y}}(1-\hat{\mathbf{y}})^{[1\times s]} \cdot \left(\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}}\right) \quad \textit{because} \quad \frac{\partial \sigma}{\partial z} = \sigma(1-\sigma) \tag{5.13}$$

$$= (\mathbf{h}_n^\intercal)^{[r\times 1]} \cdot \hat{\mathbf{y}}(1-\hat{\mathbf{y}})^{[1\times s]}(\hat{\mathbf{y}}-\mathbf{y})^{[1\times s]} \quad \textit{derivative of equation 5.3} \tag{5.14}$$

Similarly, we can compute derivatives on any θ_i in the network. In the field of machine learning, this process is called backpropagation because it allows one to backpropagate errors through the network [13]. Once every gradient has been computed, we can update the θ according to equation 5.15 where α is a user-defined constant called the *learning rate*.

$$\theta = \theta + \alpha \cdot \nabla\theta \tag{5.15}$$

Equation 5.15 is the learning rule for stochastic gradient descent. There are other, more complex learning rules based on the same idea such as Adam [18]. We found that Adam worked better in practice, and used it in all experiments.

In practice, it is possible to define the input variable $\mathbf{x}$ as matrix of dimension $[b, m]$ where m is the dimensionality of each input example and b is the number of different inputs. Each of these $[b, m]$ matrices is called a *batch*, so b is called the *batch size*. Computing the gradients for a batched input reduces noise and accelerates learning, so neural networks are often trained on batched inputs where b is in range=$(1, 32)$.

To summarize, the idea of neural networks is to feed a set of training examples ($\mathbf{x}$) through the model, compare the outputs ($\hat{\mathbf{y}}$) to their target values ($\mathbf{y}$) obtain

gradients on all trainable parameters ($\nabla\theta$), and then move each parameter slightly in the direction of its gradient. A visual representation of this process is shown in Figure 5.1.

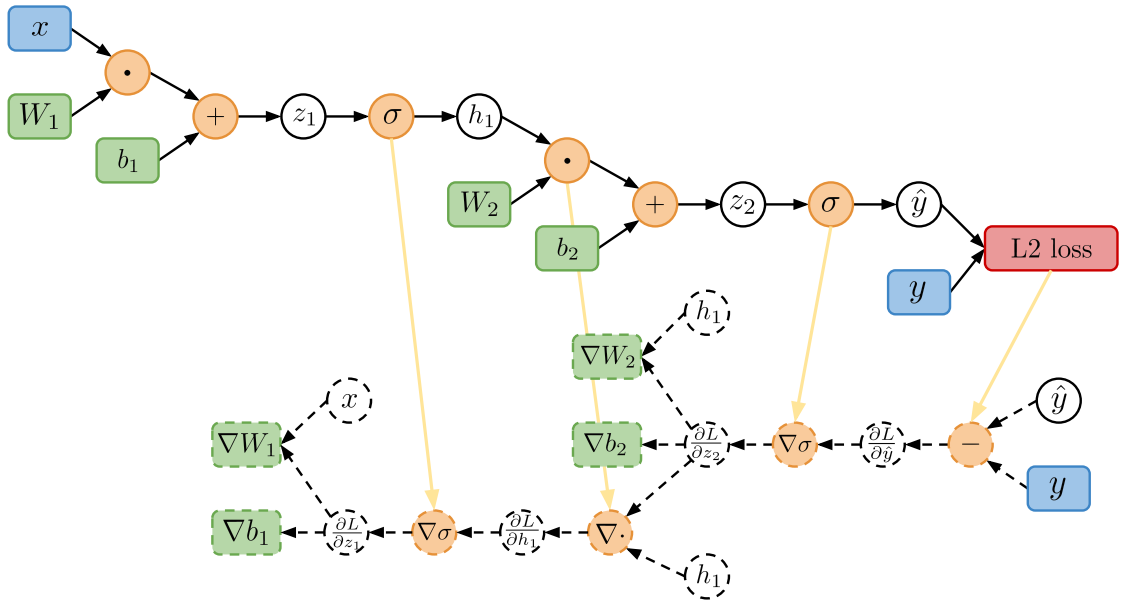


Figure 5.1: Graphical representation of a full forward-backward pass on a 2-layer neural network. **Blue**: input parameters. **Green**: trainable parameters. **Orange**: operations. **White**: intermediate variables. **Yellow**: similarities between forward-backward passes.

Most modern deep learning frameworks compute gradients automatically, a process called *automatic differentiation*[1] [4]. These frameworks include TensorFlow, PyTorch, Theano, and Chainer [1, 24, 2, 29]. We chose PyTorch for this project because of its lightweight structure, smooth interface with Python, and high-performance linear algebra library.

Pseudocode for training a neural network is provided in Section 6.2

[1]Stanford's CS231n course notes (http://cs231n.github.io) is an approachable introduction to computing gradients with backpropagation and understanding automatic differentiation [17].

5.3 Results

5.3.1 Training examples

We studied a 1D system of four spin-$\frac{1}{2}$ particles with interactions as described in equation 5.16. We chose this class of interactions because it is very general and has a tensor product structure. Furthermore, the Heisenberg Hamiltonian is a subclass of the training Hamiltonian, which allows us to compare our neural network results to the DMRG results we obtained in Chapter 4.

$$H_{sys} = J_{xz}(\hat{S}_1^{xz} \otimes \hat{S}_2^{xz}) + \frac{J}{2}\left(\hat{S}_1^{+} \otimes \hat{S}_2^{-} + \hat{S}_1^{-} \otimes \hat{S}_2^{+}\right) \tag{5.16}$$

In equation 5.16, J_{xz} and J are chosen from the random normal distribution $\mathcal{N}(\mu = 0, \sigma = 1)$. The terms $\hat{S}^{xz}$ and $\hat{S}^{+}$ are defined as shown in equation 5.17 where α_1, α_2, β, and γ are also drawn from $\mathcal{N}$.

$$\hat{S}^{xz} = \begin{pmatrix} \alpha_1 & 0 \\ 0 & -\alpha_2 \end{pmatrix} + \begin{pmatrix} 0 & \beta \\ \beta & 0 \end{pmatrix} \tag{5.17}$$

$$\hat{S}^{+} = (\hat{S}^{-})^{\dagger} = \gamma S^{+} \tag{5.18}$$

We omitted S^y because it introduces complex values; to operate on complex numbers with neural networks we would be forced to double the input and output dimensions so as to include both real and imaginary components.

Some example Hamiltonians produced by equation 5.16 are shown in Figure 5.2. It should give some sense of the diversity of possible H_{sys} Hamiltonians. Mak-

ing the problem space large forced our neural network models to find solutions that generalized well.

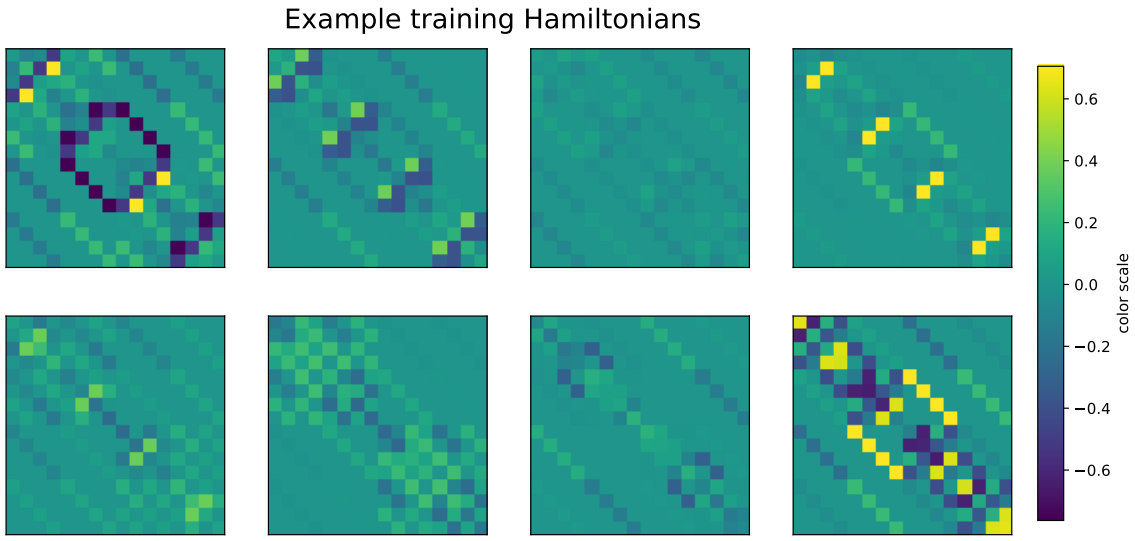


Figure 5.2: Examples of a random Hamiltonians produced by equation 5.16.

5.3.2 Tasks

We trained neural network models to perform three tasks related to solving ground state problems for entangled systems. Informal names for each model and the training objectives are summarized below:

1. `H2e0`: approximate $H_{sys} \rightarrow E_0$

   Given the Hamiltonian, H_{sys}, of an interacting system, a neural network was trained to estimate the ground state energy, E_0, of the system.

2. `psi2psi`: approximate $\psi \rightarrow \psi_{MPS}$

   Given a random (but normalized) state vector, ψ, a neural network was trained to estimate the MPS coefficients that will best reconstruct the original

state vector, ψ.

3. `H2psi0`: approximate $H_{sys} \rightarrow \psi_0$

   Given the Hamiltonian, H_{sys}, of an interacting system, a neural network was trained to estimate the ground state eigenvector, ψ_0 of the system.

Table 5.1 summarizes the training parameters of each of these neural network models. All of the nonlinear activation functions, σ were rectified linear activations [23] unless otherwise specified. We used L_2 loss, as described in equation 5.3, for training models on all three tasks. Code is available online.[2]

Table 5.1: Neural network settings. Here, n corresponds to number of layers, h to size of hidden layers, α to learning rate, x to the batched input, and y to the target matrix. Brackets denote matrix dimensions. For example, $[x]$ corresponds to $[b, m]$ where b is the batch size and m is the input dimension.

Name	n	h	α	x	$[x]$	y	$[y]$	train steps
`H2e0`	3	512	3×10^{-4}	H_{sys}	$[16 \times 256]$	E_0	$[16 \times 1]$	300000
`psi2psi`	3	512	3×10^{-4}	ψ	$[1 \times 16]$	ψ_{mps}	$[1 \times 32]$	300000
`H2psi0`	3	512	3×10^{-4}	H_{sys}	$[16 \times 256]$	ψ_0	$[16 \times 16]$	300000

Input data was synthesized at runtime and targets were computed by exact diagonalization. All H_{sys} were constructed according to equation 5.16. The ψ vectors in Task 2 were drawn from a random uniform distribution of range=(0,1) and normalized so that $\psi^\dagger \psi = 1$.

[2] Available at https://github.com/greydanus/psi0nn.

5.3.3 Task 1: approximate $H_{sys} \rightarrow E_0$

$$\arg\min_{\theta} \frac{1}{2}(E_{NN} - E_0)^2 \tag{5.19}$$

We trained our first model according to the optimization objective in equation 5.19. For the Hamiltonians constructed according to equation 5.16, ground state energies generally fell within range=(-3,0). Our model was able to estimate ground state energies to within ± 0.015, a mean percent error of about 1.1%. Full results are summarized in Table 5.2.

Table 5.2: Results for Task 1.

Type of energy	Mean energy	Mean error	Mean % error
E_0	-1.297 ± 0.01	N/A	N/A
$\hat{E}_{NN}$	-1.288 ± 0.01	0.015 ± 0.01	1.13 ± 1
mean(E_0)	-1.297 (all)	0.644 ± 0.01	49.67 ± 1

We were interested in how the neural network performed on the Heisenberg Hamiltonian which, as mentioned earlier, is a subspace of the Hamiltonians constructed by equation 5.16. The Heisenberg Hamiltonian has two coupling terms, J and J_z as shown in equation 4.12. First, we varied J and plotted ground state energy estimates against exact ground state energies computed with exact diagonalization. Figure 5.3 shows this for four different constant values of J_z. Next, we varied J_z and plotted the same quantities for four different values of J as shown in Figure 5.4.

The combination of Figures 5.3 and 5.4 gives a picture of how the neural network learns to solve the Heisenberg subspace. The approximations improve with

number of training steps; the model converges upon excellent solutions for over 100000 training steps. Adjusting h (number of hidden units), n (number of layers), α (learning rate), and other neural network hyperparameters could produce additional gains in accuracy.

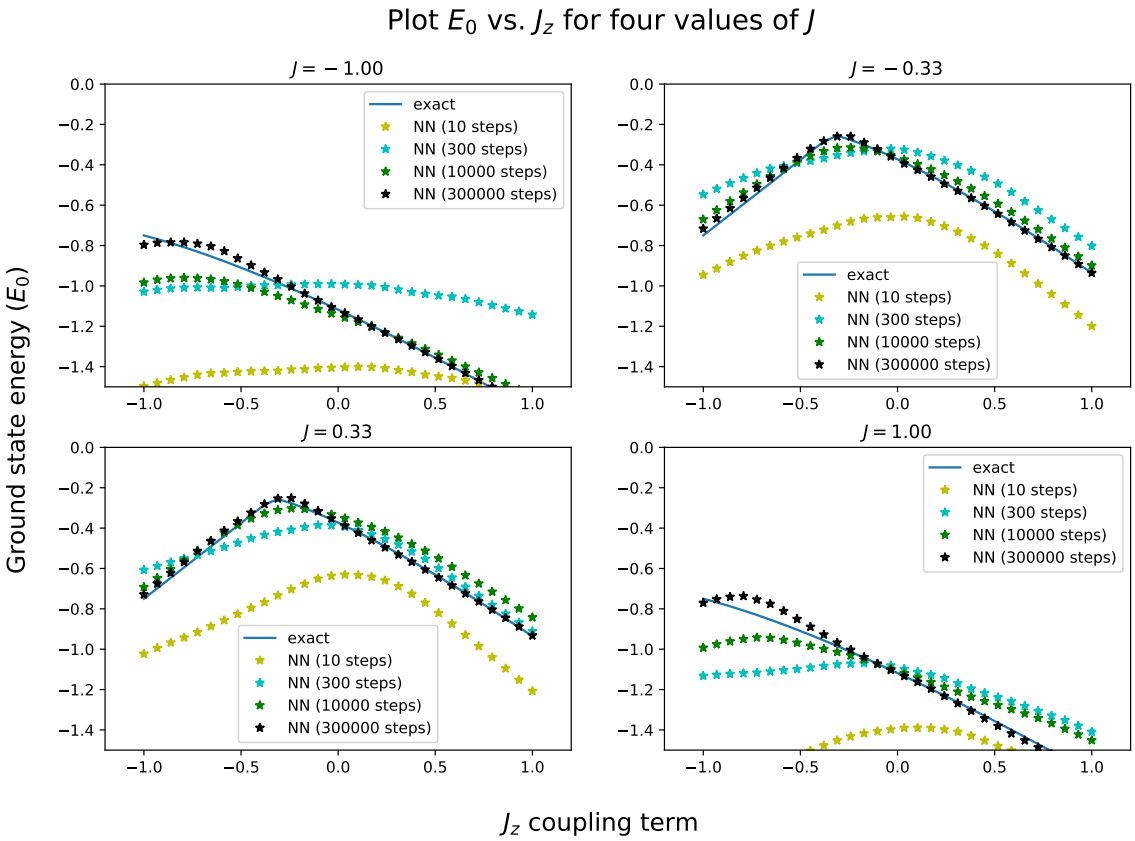


Figure 5.3: A 3-layer neural network learns to estimate the ground state energies of the Heisenberg Hamiltonian for various values of the coupling constant J.

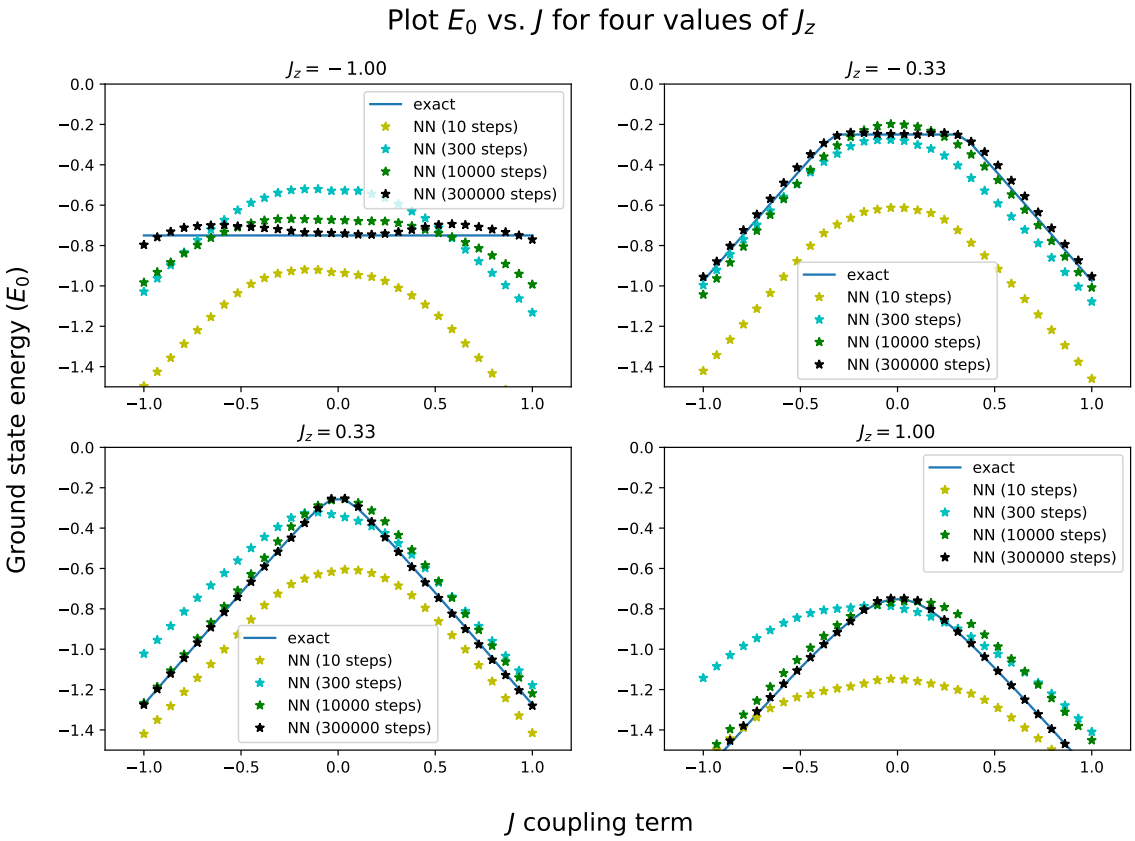


Figure 5.4: A 3-layer neural network learns to estimate the ground state energies of the Heisenberg Hamiltonian for various values of the coupling constant J_z.

5.3.4 Task 2: approximate $\psi \rightarrow \psi_{MPS}$

$$\arg\min_{\theta} \frac{1}{2} \sum_{i} (\psi_i - \psi_{i,MPS})^2 \tag{5.20}$$

We were also interested to see if a neural network could map a full-space state vector, ψ to its MPS representation, ψ_{MPS}. In order to do this, we trained our neural network on the *autoencoder*-like objective shown in equation 5.20.

In deep learning, an autoencoder is a type of neural network which is trained to reconstruct its input as accurately as possible. This is made difficult by making the dimensions of its hidden layers smaller than the dimension of the input data. These neural networks learn to create very efficient representations of the input data [34]. The *encoder* part of the autoencoder is the set of progressively lower-dimensional layers from the input layer to the middle; the *decoder* is the set of progressively higher-dimensional layers from the middle to the output layer [11].

In our model, the encoder is a neural network which takes as input the vector ψ and outputs estimates of the MPS coefficients $\hat{\psi}_{NN,MPS}$. The decoder portion is simply a PyTorch implementation of equation 3.1. An L_2 loss is then computed according to equation 5.20. As the decoder has no trainable parameters, PyTorch simply sends gradients backwards through equation 3.1, obtains gradients on the estimated MPS coefficients, and trains the neural network encoder as usual. This structure is summarized in Figure 5.5.

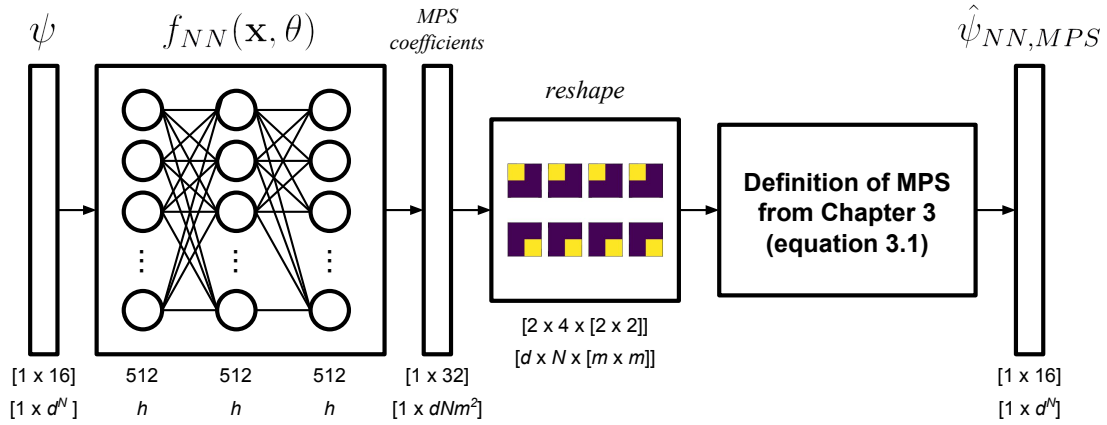


Figure 5.5: Overview of the autoencoder structure we used to train a neural network on the $\psi \rightarrow \psi_{MPS}$ task.

We obtained an average reconstruction error of ± 0.035 which corresponds to a 3.45% mean percent error. We were also interested in whether the model could reconstruct the GHZ state and obtain the MPS coefficients listed in Chapter 3.

Our model was able to reconstruct the 4-particle GHZ state with average error of ± 0.049 (4.9% mean percent error) as shown in Figure 5.6.

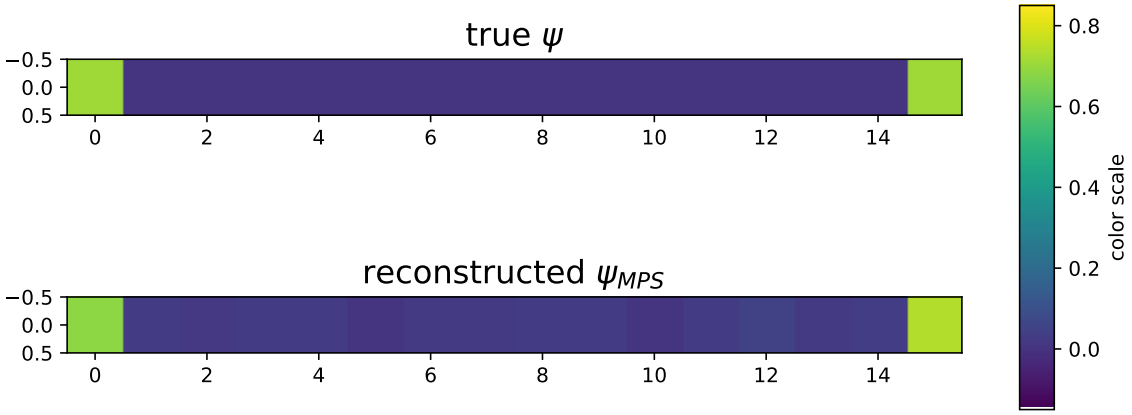
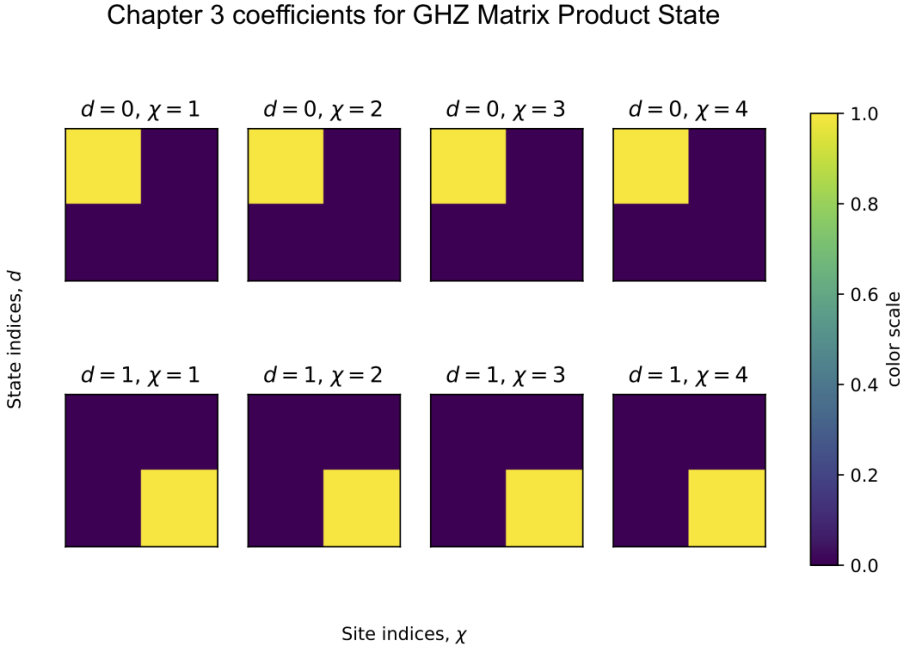


Figure 5.6: Training a neural network to approximate MPS coefficients, then reconstructing the original state from those coefficients.

Interestingly, the estimated MPS coefficients were generally different from those listed in Chapter 3, as shown in Figure 5.7. We did, however, observe a close correspondence between the expected MPS coefficients and neural network estimates earlier in training (training step 20000), as shown in Figure 5.8. In both cases, the reconstructed ψ had a mean percent error of less than 5%

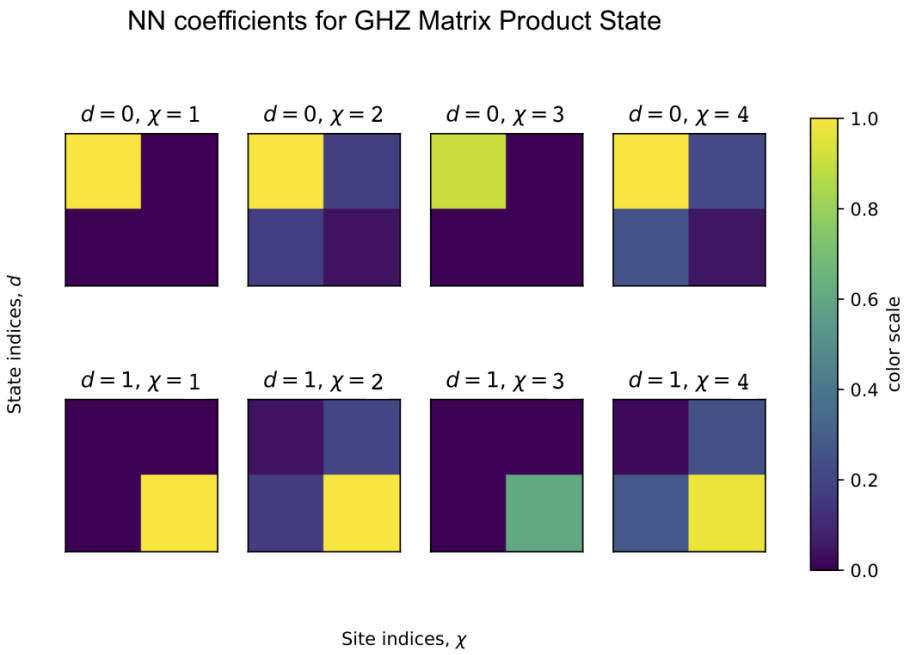
The MPS representation is not unique. For example, there are both right canonical and left canonical forms of MPS [27]. We hypothesize that the neural network simply used a different MPS ”basis” than the ”exact” example taken from Chapter 3. This is probably what has happened in Figure 5.7a; if the matrices are rounded to zero or one, then they multiply correctly to produce the GHZ state.

(a) NN prediction, at training step 64000, for MPS coefficients of the GHZ state.

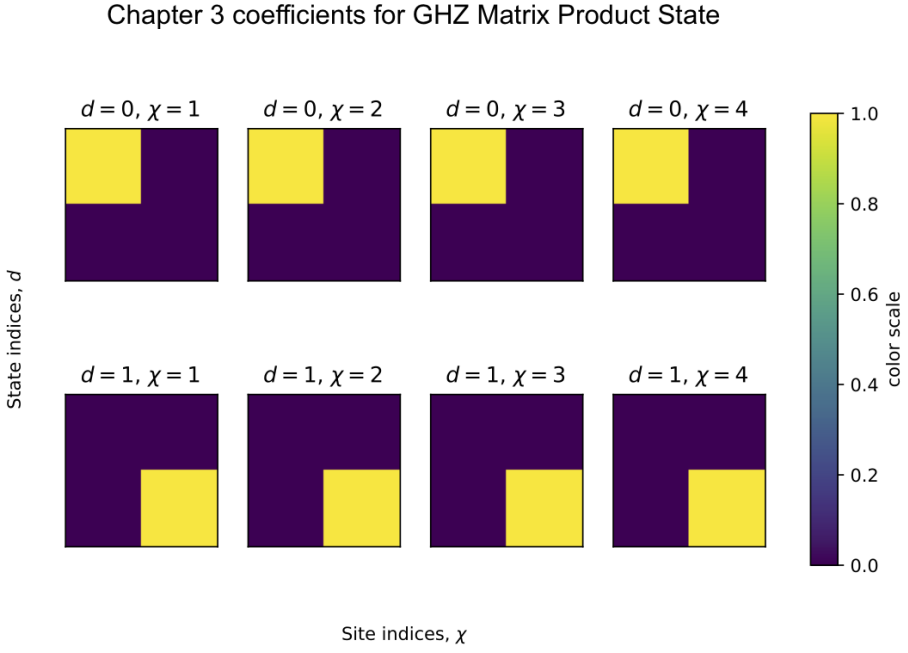


(b) MPS coefficients of the GHZ state as described in Chapter 3.

Figure 5.7: Using the GHZ state as a case study, we observe an interesting difference between MPS coefficients predicted by a neural network and the expected coefficients.



(a) NN prediction, at training step 20000, for MPS coefficients of the GHZ state.



(b) MPS coefficients of the GHZ state as described in Chapter 3.

Figure 5.8: In this case, we observe a close correspondence between MPS coefficients predicted by a neural network and the expected coefficients.

5.3.5 Task 3: approximate $H_{sys} \rightarrow \psi_0$

$$\arg\min_{\theta} \frac{1}{2}(E_{\psi_{NN}} - E_0)^2 \tag{5.21}$$

The final training objective is shown in equation 5.21. We made only one significant adjustment to the neural network before training on this task: we normalized the output vector so that $\psi_{NN}^{\intercal}\psi_{NN} = 1$. The loss function was a L_2 loss (see equation 5.3) between the true ground state energy E_0 and the energy of the ground state predicted by the model, E_{NN}, computed via equation 5.22.

$$E_{\psi_{NN}} = \langle \psi_{NN} | \, H_{sys} \, | \psi_{NN} \rangle \tag{5.22}$$

We also combined this task to solve for ground states in the MPS picture, producing the modified training objectives shown in equations 5.23 and 5.24.

$$\arg\min_{\theta} \frac{1}{2}(E_{\psi_{NN,MPS}} - E_0)^2 \tag{5.23}$$

$$E_{\psi_{NN,MPS}} = \langle \psi_{NN,MPS} | \, H_{sys} \, | \psi_{NN,MPS} \rangle \tag{5.24}$$

Results for testing the model on 5000 Hamiltonians (computed according to equation 5.16) are summarized in Table 5.3 and Figure 5.9.

Table 5.3: Locating neural network estimates of ground states for H_{sys} with respect to the systems' low-energy eigenstates.

Type of energy	Mean	Mean error	Mean % error
E_0	-1.310 ± 0.01	N/A	N/A
$E_{\psi_{NN}}$	-1.284 ± 0.01	0.026 ± 0.01	1.99 ± 0.1
E_1	-1.258 ± 0.01	N/A	N/A
$E_{\psi_{NN,MPS}}$	-1.251 ± 0.01	0.051 ± 0.01	3.94 ± 0.1
E_2	-1.021 ± 0.01	N/A	N/A
$E_{\psi_{random}}$	0.0112 ± 0.01	1.321 ± 0.01	100.86 ± 0.1

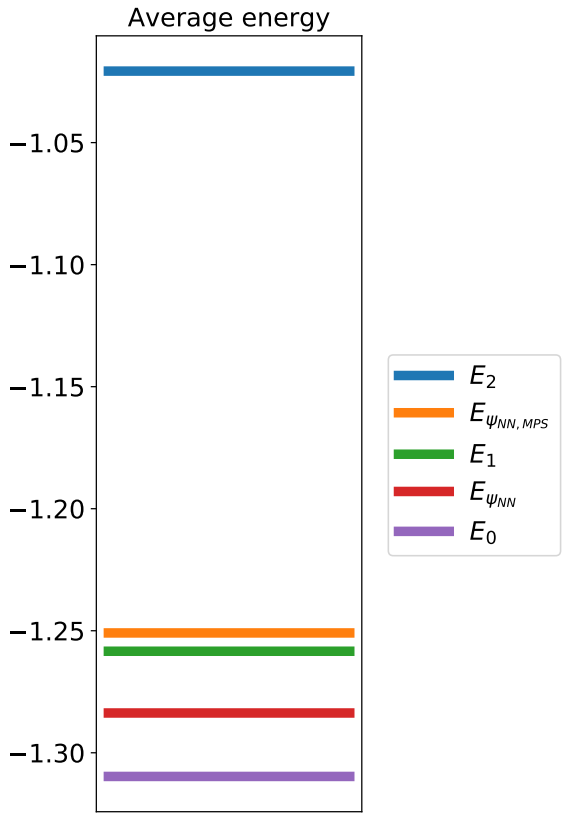


Figure 5.9: Comparison of the average energy of neural network predictions against the three lowest eigenstates of H_{sys}.

Qualitatively, the exact ground states often looked similar to the neural networks' predictions such as the example shown in Figure 5.10.

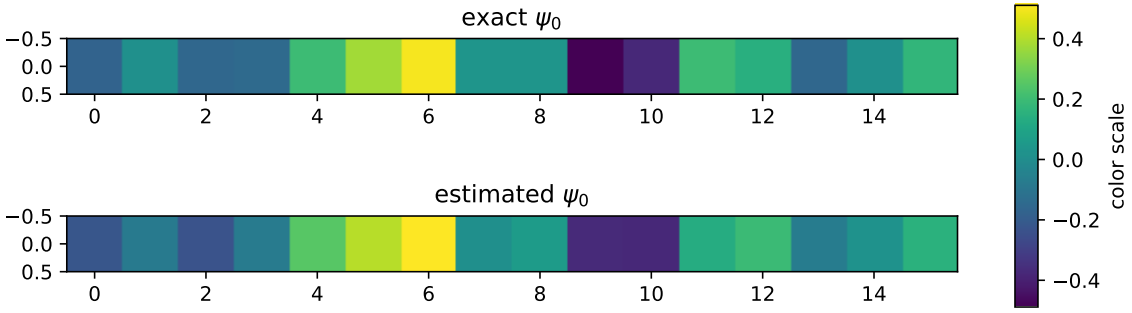


Figure 5.10: Comparison between the actual ground state and the low-energy state, ψ_{NN}, estimated by our model. Note the close correspondence between most values.

There were other situations, though, where ψ_{NN} looked nothing like ψ_0. We hypothesized that, in these cases, the neural network was producing a linear combination of different low-energy states. We tested this hypothesis by comparing ψ_{NN} to both ψ_0 and ψ_1. As shown in Figure 5.11, we found cases where ψ_{NN} looked more similar to ψ_1 than ψ_0. These preliminary findings support our hypothesis.

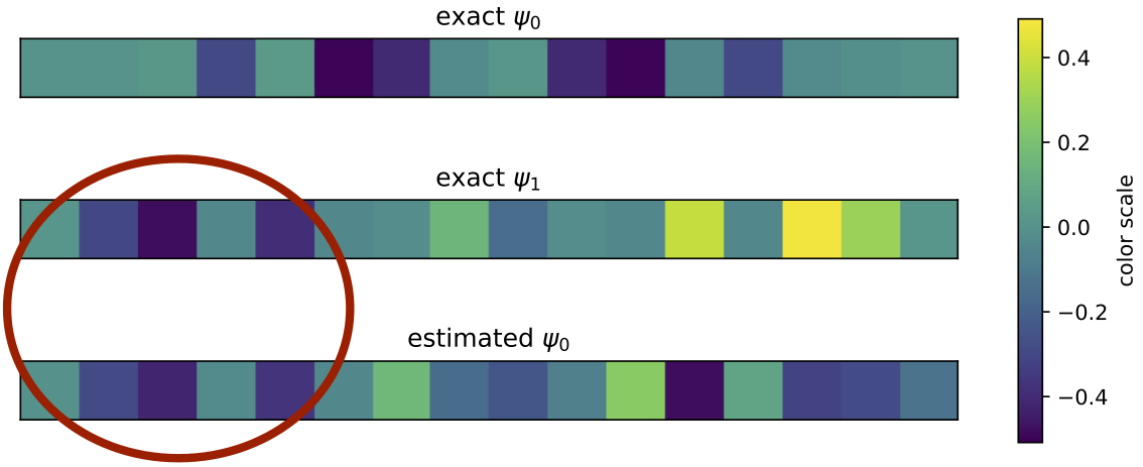


Figure 5.11: Comparison between ψ_0, ψ_1, and ψ_{NN}. Note the close correspondence between ψ_1, and ψ_{NN} in the circled area.

We were interested to see whether a neural network could solve the same ground state problem *and* produce MPS coefficients for the solution, $\hat{\psi}_{NN,MPS}$, instead of the full state solution $\hat{\psi}_{NN}$. In other words, we used the autoencoder structure described in Task 2 but used the system Hamiltonian as an input and minimized the energy of the output, $\hat{\psi}_{NN,MPS}$ according to equation 5.22. A schema of this model is shown in Figure 5.12.

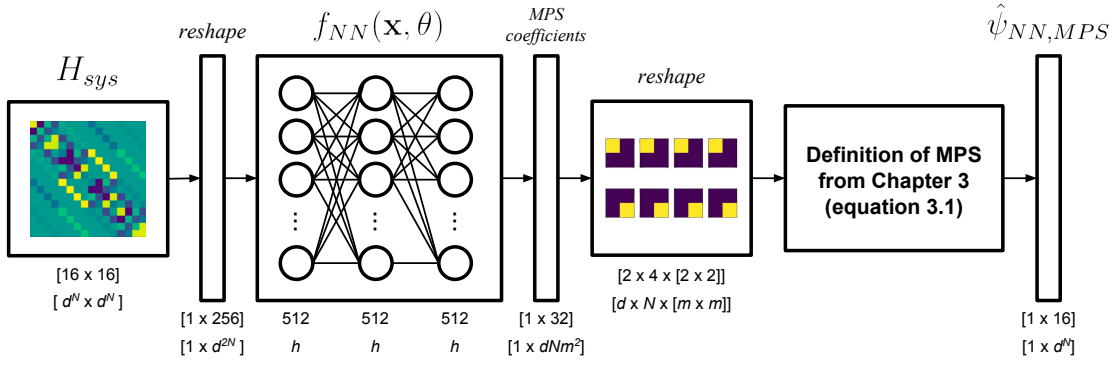


Figure 5.12: Overview of the autoencoder structure we used to train a neural network on the $H_{sys} \rightarrow \psi_{MPS}$ task.

This model was also successful in finding low-energy states of the system. For some test samples, it produced near-exact ground states as shown in Figure 5.13. For other samples, it produced energies close to the first excited state. Future work will analyze why this occurs and lower the average energy obtained using NN output $\psi_{NN,MPS}$ below the first excited state.

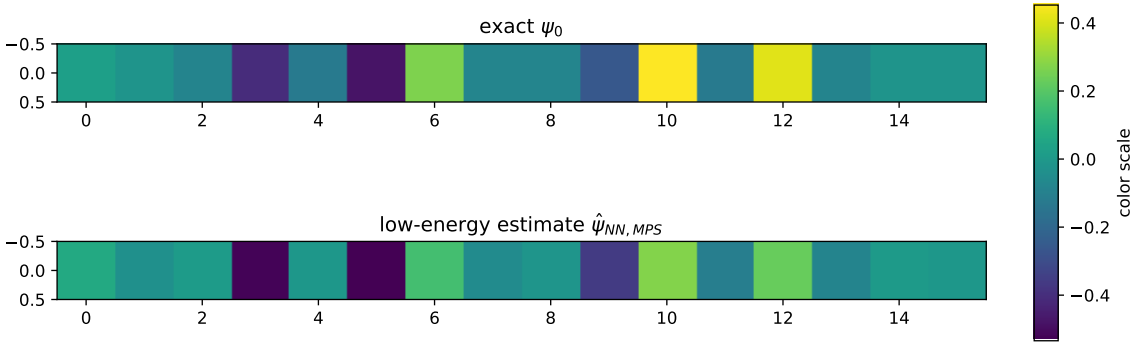


Figure 5.13: Comparison between the actual ground state, ψ_0, and the low-energy state, $\psi_{NN,MPS}$, estimated by our model. Note the close correspondence between most values.

5.4 Discussion

In this section, we shifted our attention to a new computational tool, deep learning, and explored its ability to measure quantum entangled systems. Given its role in obtaining state of the art results for other open problems in quantum mechanics, this direction seemed promising [5, 7, 10, 5, 7, 20, 6].

We showed that neural networks can produce accurate estimates of ground state energies for systems of several interacting spin-$\frac{1}{2}$ particles. We also found that they can transform arbitrary quantum states into their MPS representations. Finally, we showed that they can be trained to produce states with energies well

below the first excited state. As our approach was quite general, we speculate that these tasks could be extended to larger systems and to 2D and 3D systems.

Though we used full system Hamiltonians, which grow[3] as d^N, as inputs to our models, we could just as easily have input *only* the 6 coupling terms – $\{\alpha_1, \alpha_2, \beta, \gamma, J, J_z\}$ – for each site-site interaction, which grows as $6N$.

These results confirm the findings of Carleo and Troyer [6] using a different type of deep learning architecture (Carleo and Troyer used Restricted Boltzmann Machines). They also extend deep learning to the domain of Matrix Product States; it appears that neural networks can solve ground state problems in the MPS representation with little to no cost in accuracy (see Table 5.3).

[3]Where d is the number of states per site and N is the number of sites.

Chapter 6

Closing remarks

In this thesis, we explored the fundamental quantum phenomenon of entanglement and explained why it makes simulating quantum systems difficult. In order to avoid the exponential growth of computation time imposed by entanglement, we introduced Matrix Product States (MPS) and showed how to use them to perform calculations efficiently over a large number ($N > 30$) of entangled sites.

Next, we explored one of the most popular algorithms for obtaining MPS coefficients, the Density Matrix Renormalization Group (DMRG). We showed that it can produce quick and accurate estimates of ground state energies for large systems because its computational cost grows linearly with the number of sites, N. We even showed that DMRG and MPS are two sides of the same coin; the formal definition of DMRG can be rewritten in terms of the A coefficient matrices of MPS.

Finally, the new results contained in this thesis showed that deep learning – a tool that has already produced impressive results in other areas of quantum research – can perform measurements on entangled systems. In particular, we obtained good estimates of ground state energies for Hamiltonians with tensor

product structure and learned to transform quantum states into their MPS representations. We also computed low-energy states, well below the first excited state, directly from system Hamiltonians.

Our findings suggest that deep learning is indeed a valuable tool for measuring entangled systems. We believe that changing the input data from a system Hamiltonian to a list of local interaction Hamiltonians, training larger neural networks, and extending our model to solve 2D and 3D systems could produce interesting, and potentially state-of-the-art, solutions to large quantum many-body problems.

Appendix A: Linear algebra

This appendix is adapted from previous work by Professor James Whitfield.

6.1 Singular value decomposition

The singular value decomposition is similar to the eigenvalue decomposition but it can be applied to matrices that are not square. Singular values are frequently used to characterize norms of matrices. For instance, the operator norm is the largest singular value and the trace norm is the sum of the singular values.

The singular value decomposition of T says that

$$T = V\Sigma W^\dagger \tag{6.1}$$

with the following spatial decomposition: $[n \times m] = [n \times n][n \times m][m \times m]$. Here $[n \times m]$ denotes the space of matrices with n columns and m rows. V and W are unitary and Σ is diagonal. The diagonal elements of Σ are the singular values and, they are uniquely specified by T.

The singular values of matrix T are the eigenvalues of $\sqrt{T^\dagger T}$. Since $T^\dagger T$ is positive semi-definite, the square root function is well defined. Additionally, since

$T^\dagger T$ is normal ($[T, T^\dagger] = 0$) we have $T^\dagger T = V \Lambda V^\dagger$ with Λ a diagonal matrix and V a unitary matrix. We define the singular matrix as $\Sigma \equiv \sqrt{\Lambda}$.

For arbitrary matrix, T, we now give constructions for the input and output unitary matrices, V and W, which will prove that the SVD exists for all matrices. If T is non-singular, we can invert the diagonal singular value matrix, Σ^{-1} by taking the reciprocal of the singular values. Letting $W = T^\dagger V \Sigma^{-1}$, W is unitary and $T = V \Sigma W^\dagger$.

When T is singular, the inverse of Σ is no longer available as some of the singular values are zero. Instead, the pseudo-inverse is taken where only the inverse of non-zero diagonal elements is used. As before, the output matrix W is defined using the pseudo-inverse. To satisfy the unitary constraint, arbitrary orthonormal vectors are included as columns of W. Thus, the singular value decomposition exists for all matrices regardless of shape and the matrices need not satisfy special properties.

The vector form of the SVD can be easily seen as $TW = V\Sigma$ and $V^\dagger T = \Sigma W^\dagger$ imply

$$T|w_i\rangle = \sigma_i |v_i\rangle \tag{6.2}$$

$$\langle v_i|T = \langle w_i|\sigma_i \tag{6.3}$$

The vectors $\{|v_i\rangle\}$ are called the left singular vectors and $\{|w_i\rangle\}$, the right singular vectors.

6.2 Schmidt decomposition

The Schmidt decomposition discussed in Chapter 3 is essentially a restatement of the SVD we have discussed so far. We now construct the Schmidt decomposition using the SVD. Given an arbitrary orthonormal basis for H_A and H_B denoted as $\{|i\rangle\}$ and $\{|j\rangle\}$ respectively, we wish to ascertain the Schmidt vectors and the Schmidt coefficients. Since we can span $H_A \otimes H_B$ by the tensor product of the bases, we can write the state ψ as, $|\psi\rangle = \sum_{ij} T_{ij}|i\rangle|j\rangle$, where T is $[n \times m]$. Here n is the dimension of H_A and m is the dimension of H_B. Applying the SVD to matrix T yields $T = V\Sigma W^\dagger$. Evaluating the $(i,j)^{th}$ matrix element and inserting resolutions of the identity we get,

$$T_{ij} = \sum_{l=1}^{m}\sum_{k=1}^{n} V_{ik}\Sigma_{kl}W^\dagger_{lj}.$$

Inserting this into the expression for ψ

$$\begin{aligned}
|\psi\rangle &= \sum_{ij} T_{ij}|i\rangle|j\rangle \\
&= \sum_{k=1}^{n} \sigma_k \delta_{kl} \left(\sum_i V_{ik}|i\rangle\right)\left(\sum_j W^\dagger_{lj}|j\rangle\right) \\
&= \sum_{k=1}^{\chi} \sigma_k |a_k\rangle|b_k\rangle
\end{aligned}$$

with $|a_j\rangle \equiv \sum_{k=1}^{n} V_{ik}|i\rangle$ and $|b_j\rangle \equiv \sum_{k=1}^{m} W^*_{jl}|j\rangle$. The singular values are the Schmidt coefficients and the number of non-zero singular values is the Schmidt number.

Appendix B: Pseudocode

6.3 Finite DMRG

The Finite DMRG algorithm discussed in Chapter 4 follows the basic structure shown below. There are many implementations online; code for this thesis is available at https://github.com/greydanus/psi0nn.

Algorithm 1 Infinite Density Matrix Renormalization Group (DMRG)

procedure INFINITE DMRG
block $\leftarrow$ single free site
while current_block_length $< \frac{1}{2}$ max_block_length:
    $S'_E \leftarrow S_B + S_S + S_{BS}$ **for** S **in** operators *enlarge block*
    $H_{super} \leftarrow H_E + H_{E'} + H_{SS'}$ *build* H_{super}.
    $E_0, \psi_{E_0} \leftarrow$ `eigsh(`H_{super}`, k=1)` *decompose* H_{super}.
    compute reduced density matrix, ρ:
        $\rho \leftarrow Tr_R \left|\psi_{E_0}\right\rangle \left\langle\psi_{E_0}\right|$
    $evals$, $evecs \leftarrow$`svd(`ρ`)` (where e'vecs are sorted by descending e'vals)
    $O \leftarrow$ `concatenate(`$evecs[:m]$`)` *construct transformation matrix*
    $S_E \leftarrow O^\dagger S'_E O$ **for** S'_E **in** operators *rotate and truncate block*
    estimate truncation error, ϵ:
        $\epsilon \leftarrow 1 - \sum_{i<m} evals[i]$
return $block$, E_0, ψ_{E_0}, ϵ

6.4 Neural network

We can summarize the training process of a neural network as shown below. Reference equations 5.4-5.15 and Figure 5.1 for additional clarity. This pseudocode was adapted from Mori, 2005 [22].

Algorithm 2 Training a neural network with backpropagation

procedure BACKPROP
$W \leftarrow$ random intial values
for e = 1 to N (all examples):
    $x \leftarrow$ input for example *e*.
    $y \leftarrow$ output for example *e*.
    run x forward through network, computing all $\{\hat{y}_i\}$,$\{z_i\}$
    for all weights (j, i) (in reverse order):
        compute $\nabla_i \leftarrow \begin{cases} (y_i - \hat{y}_i) \cdot \mathcal{L}'(z_i) & \text{if } i \text{ is output node} \\ \mathcal{L}'(z_i) \sum_k W_{i,k} \nabla_k & \text{otherwise} \end{cases}$
        $W_{j,i} \leftarrow W_{j,i} + \alpha \cdot \nabla_i$

Bibliography

[1] Martn Abadi, Ashish Agarwal, and Paul Barham. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. Technical report, 2015.

[2] Rami Al-Rfou and Guillaume Alain. Theano: A Python framework for fast computation of mathematical expressions. *arXiv Preprint (1605.02688v1)*, 2016.

[3] P Baldi, P Sadowski, and D Whiteson. Searching for Exotic Particles in High-Energy Physics with Deep Learning. *ArXiv Preprint (1402.4735v2)*, 2014.

[4] Atilim G Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *ArXiv Preprint (1502.05767v2)*, 2015.

[5] Jrg Behler and Michele Parrinello. Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces. *Physical Review Letters*, 98, 2007.

[6] Giuseppe Carleo and Matthias Troyer. Solving the Quantum Many-Body Problem with Artificial Neural Networks. *Science*, 2017.

[7] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Aln Aspuru-Guzik, and Ryan P Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. *Advances in Neural Information Processing Systems*, 28:2224–2232, 2015.

[8] Maria Eckholt. Matrix Product Formalism. *Master's Thesis*, pages 5–35, 2005.

[9] A Einstein, B Podolsky, and N Rosen. Can the Quantum-Mechanical Description of Physical Reality Be Considered Complete? *Physical Review*, 47:777–780, 1935.

[10] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural Message Passing for Quantum Chemistry. *arXiv Preprint (1704.01212v1)*, 2017.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *Computer Vision and Pattern Recognition*, 2015.

[13] Hecht-Nielsen. Theory of the backpropagation neural network. In *International Joint Conference on Neural Networks*, pages 593–605. IEEE, 1989.

[14] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1 1989.

[15] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens Van Der Maaten.

Densely Connected Convolutional Networks. *Computer Vision and Pattern Recognition*, 2016.

[16] Rajibul Islam, Ruichao Ma, Philipp M Preiss, M Eric Tai, Alexander Lukin, Matthew Rispoli, and Markus Greiner. Measuring entanglement entropy through the interference of quantum many-body twins. *Nature*, 2015.

[17] Andrej Karpathy. CS231n Convolutional Neural Networks for Visual Recognition, 2015.

[18] Diederik P Kingma and Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, 2015.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems*, pages 1097–1105, 2012.

[20] K Mills, M Spanner, and I Tamplyn. Deep learning and the Schrodinger equation. *Preprint (arXiv:1702.01361v1)*, 2017.

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518, 2015.

[22] Greg Mori. Neural Networks, 2005.

[23] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27 th International Conference on Machine Learning*, 2010.

[24] Adam Paszke, Sam Gross, and Soumith Chintala. PyTorch, 2017.

[25] Bharath Ramsundar, Steven Kearnes, Patrick Riley, Dale Webster, David Konerding, and Vijay Pande. Massively Multitask Networks for Drug Discovery. *Preprint (arXiv:1502.02072)*, 2015.

[26] Jrgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[27] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326(1), 2011.

[28] Daniel Stoecklein, Kin Gwn Lore, Michael Davies, Soumik Sarkar, and Baskar Ganapathysubramanian. Deep Learning for Flow Sculpting: Insights into Efficient Learning using Scientific Simulation Data. *Scientific Reports*, 7:46368, 4 2017.

[29] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a Next-Generation Open Source Framework for Deep Learning. *Annual Conference on Neural Information Processing Systems (NIPS)*, 2015.

[30] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating Eulerian Fluid Simulation With Convolutional Networks. *Preprint (arXiv:1607.03597v2)*, 2016.

[31] Stefan van der Walt, S Chris Colbert, and Gael Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering*, 13(2):22–30, 3 2011.

[32] Ashlee Vance. The Race to Buy the Human Brains Behind Deep Learning Machines - Bloomberg, 2014.

[33] F Verstraete, J I Cirac, and V Murg. Matrix Product States, Projected Entangled Pair States, and variational renormalization group methods for quantum spin systems. *Advances in Physics*, 57(2), 2009.

[34] Vincent Pascal and Hugo Larochelle. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion Pierre-Antoine Manzagol. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

[35] Steven White. Density Matrix Formulation for Quantum Renormalization Groups. *Physical Review Letters*, 69(19):2863–2866, 1992.

[36] Kenneth G Wilson. The renormalization group: Critical phenomena and the Kodo problem. *Reviews of Modern Physics*, 47(4):773–840, 1975.

[37] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, and Mohammad Norouzi. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *Preprint (arXiv:1609.08144v2)*, 2016.