Ludwig-Maximilians-Universität München
Faculty of Physics

DMRG-inspired Algorithms for Neural Network Compression

Bachelor's Thesis
Submitted by

Aaron Lee Sander

Internal Supervisor: Prof. Dr. Jan von Delft
External Supervisor: Dr. Nicola Pancotti (MPQ)

Munich, September 11, 2020

Ludwig-Maximilians-Universität München
Fakultät für Physik

Von DMRG inspirierte Algorithmen für das Komprimieren neuronaler Netze

Bachelorarbeit
vorgelegt von

Aaron Lee Sander

Interner Betreuer: Prof. Dr. Jan von Delft
Externer Betreuer: Dr. Nicola Pancotti (MPQ)

München, 11. September 2020

Summary

The overall goal of this thesis from the beginning was to take an interdisciplinary approach and apply the mathematically rigorous ideas of quantum mechanical tensor networks to computer science problems. I wanted to make sure that any results would be understandable by both physicists and computer scientists such that we could bridge the gap between physical theory and application.

In the course of this thesis, I learned the theory behind tensor networks as well as created a Python library containing various tensor network operations. This library includes the ground state calculation of a Hamiltonian, which was not included in this thesis but used for my own understanding, as well as the quantum state compression. This was primarily for learning purposes in order to better understand the use of tensor networks in detail such that they could be reasonably applied later. All of this code was written with education in mind rather than optimization.

Next, we looked into various applications of this compression to computer science with the primary focus being on neural networks, graph embedding, and finite automata. For the sake of writing this thesis, this was narrowed down to applying it to neural networks. Many other applications appeared in these adjacent fields, but would have not been able to be completed within the required timeframe.

I then modified my code such that I could extend these ideas already understood in physics to neural networks. I also forked and modified the code used in [1] such that I could further improve on it. From the various analyses of the quantum state compression algorithm on both standard neural networks and a neural network trained with layers of Matrix Product Operators (MPO Net), we were able to first justify our approach as well as see that the layers of a neural network have different levels of compressibility.

Finally, I was able to apply various search algorithms to optimize the bond dimensions needed, further reducing the number of parameters necessary while minimizing the amount of loss in accuracy from our original network. The result of this is that we can reasonably compress a standard neural network by decomposing it and applying this algorithm as well as improve upon the work done in [1].

There are several topics and improvements included in the conclusion which I was unable to get to in the course of this thesis, but which can hopefully be explored further in the near future.

Code and Figures

This thesis is accompanied by a Github repository containing Python code and Jupyter notebooks for the tensor network operations described in its contents. All code was written by myself (other than the MPO Net, see below) and is primarily focused on being educational and easy to read rather than optimized. This code and all the figures used in this thesis can be found in the repository at:

github.com/aaronleesander/DMRG-inspired-Algorithms-for-Neural-Network-Compression

The repository used in [1] was also forked from which I used several modified versions of the FC2 and LeNet5 models. This repository can also be found at:

github.com/zfgao66/deeplearning-mpo

Acknowledgements

First, I would like to thank my advisor Dr. Nicola Pancotti for the initial ideas and guidance throughout this thesis and special thanks for treating me not only as a student but as a colleague and collaborator. I would also like to thank Prof. Dr. Jan von Delft for supervising and handling the administrative side of this work.

I would like to thank my parents, Sherry and JR, for their constant love and support throughout my studies, regardless of the ocean between us. I would also like to thank my "adopted" German family, Barbara, Alex, and Max for giving me a second home especially during the first few months when I moved to Germany.

I would like to thank all my friends who have been by my side through all the ups and downs of living abroad. Special thanks to Jacob Rutherford for the technical computer science discussions which were taken into account in this thesis as well as Matt Lamsey for his expert proofreading abilities.

Finally, I would like to thank my late German teacher Dr. Marilyn Carico for laying the foundations for me to be able to study here in the first place.

Without all of these people, this thesis as well as my entire Bachelor's degree would not have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Tensor Networks

Notation

The following notation choices have been made in this thesis:

- M: Tensor where the canonicality is unknown or unimportant for the calculation
- A: Tensor in left-canonical form
- B: Tensor in right-canonical form
- d: Total numbers of states in a system
- l: Index of a site in a tensor network
- L: Total Number of sites in a tensor network
- σ_l: Index of physical dimension at site l with length d
- a_l: Index of bond dimension between site l and $l+1$
- D_l: Length of bond dimension between site l and $l+1$
- $\sum_{\{a_l\}}$: Contraction along a set of dimensions

1.1 Theory

1.1.1 Introduction to Tensor Networks

Tensor Networks are both a visual representation of tensor operations as well as a language to better explain complicated tensor problems by avoiding cumbersome Einstein notation. Tensors themselves are simply a generalization of familiar mathematical objects to higher dimensions and are characterized by their rank, defining the number of dimensions which make up the tensor. Note that the perspective of tensors needed to understand this thesis will be explained analogously to how they could be used in computer science. The mathematical properties of tensors are much more complicated and other fields such as condensed matter physics where tensor networks originally come from will look at them from a different angle. In this thesis, we will only cover tensor operations that are necessary for our topic. A deeper explanation of various types of tensor operations can be found in [2] and [3].

Many will already be familiar with the lower rank tensors albeit under different names e.g. a rank-0 tensor is a scalar, rank-1 is a vector, rank-2 is a matrix, and anything above that is simply a "rank-N tensor". Since this thesis also intersects computer science, it is important to note that the concept of a tensor is already familiar to many computer scientists as a multidimensional array. However, the concepts in this thesis can also be defined and extended to infinite dimensions.

For these lower-rank tensors, such as matrices, operations such as multiplication are well-defined and follow a sort of algorithm in calculating. These operations do not translate in an obvious way to higher-rank tensors, the details of which will be discussed in this thesis. We can however translate this concept into the language of tensor networks which helps us visualize these operations.

The main properties of tensor networks which will appear frequently in this work are the following:

1. Each leg represents a dimension. Labels can be symbolic labels or dimension lengths.

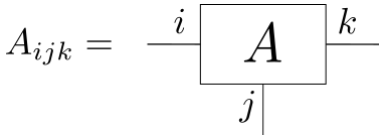

Figure 1.1: Rank-3 tensor with dimensions $(i \times j \times k)$

2. Legs are not static objects and can be reshaped, combined, or broken apart.

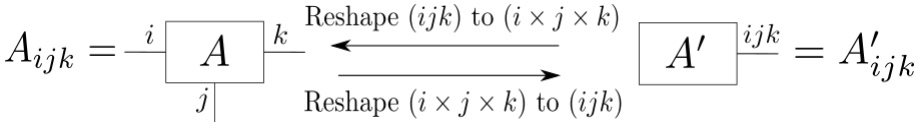

Figure 1.2: Rank-3 tensor reshaped into a rank-1 tensor and vice-versa

3. Legs can be contracted between two tensors to create a larger tensor.

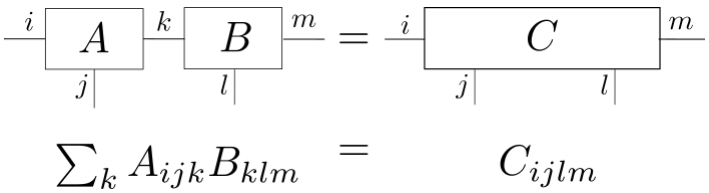

Figure 1.3: Two rank-3 tensors contracted along the k dimension to form a rank-4 tensor

1.1.2 Matrix Product States

Consider a quantum state in the local Hilbert Space $\mathcal{H}_l := \mathbb{C}^d$ where $d \in \mathbb{N}$ is the local Hilbert space dimension. A multi-particle quantum state of L particles would thus exist in a Hilbert Space with d^L possible states for the total system

$$\mathcal{H} := \mathcal{H}_l^{\otimes L} := (\mathbb{C}^d)^{\otimes L} = \mathbb{C}^d \underbrace{\otimes \ldots \otimes}_{\text{L-1 times}} \mathbb{C}^d \tag{1.1}$$

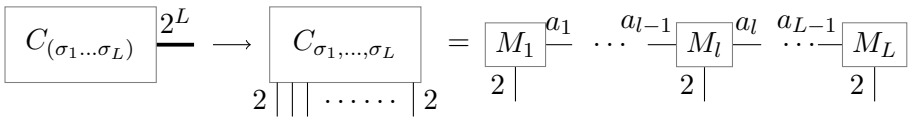

Figure 1.4: Matrix Product States of a system of L qubits with non-periodic boundary conditions. Arrow implies a reshape from a rank-1 tensor with 2^L elements to a rank-L tensor with 2 elements in each dimension.

where each product state in some local basis can be written as

$$|\sigma_1 \ldots \sigma_L\rangle = |\sigma_1\rangle \otimes \cdots \otimes |\sigma_L\rangle = \underbrace{\begin{pmatrix} \vdots \\ \vdots \end{pmatrix}}_{d^L \text{ elements}} \tag{1.2}$$

Even with the simplest two-state system such as a system of qubits where $d^L := 2^L$, this space becomes both computationally expensive and analytically too large to work with as the number of particles grows as the tensor needed to represent it grows exponentially.

Say that we have a system of L qubits such that physical dimension $d = 2$. This gives us the state in Hilbert space $\mathcal{H}$ where $|\sigma_l\rangle \in \{|0\rangle, |1\rangle\}$ is the state of the l^{th} particle and $C_{\sigma_1 \ldots \sigma_L}$ is a rank-L tensor with 2^L total elements

$$|\Psi\rangle = \sum_{\{\sigma_l\}} C_{\sigma_1,\ldots,\sigma_L} |\sigma_1 \ldots \sigma_L\rangle \tag{1.3}$$

Considering the definition of the Hilbert space defined in Eq. (1.1), we can decompose this state into local tensors at each site using $L - 1$ Singular Value Decompositions (or more generally, Schmidt Decompositions). This allows us to write our state as a Matrix Product State (MPS)

$$\begin{aligned} |\Psi\rangle &= \sum_{\{a_l\}} \sum_{\{\sigma_l\}} M^{\sigma_1}_{a_1} M^{\sigma_2}_{a_1,a_2} M^{\sigma_3}_{a_2,a_3} \ldots M^{\sigma_L}_{a_{L-1}} |\sigma_1 \ldots \sigma_L\rangle \\ &= \sum_{\{a_l\}} \sum_{\{\sigma_l\}} M^{\sigma_1}_1 \ldots M^{\sigma_L}_L |\sigma_1 \ldots \sigma_L\rangle \end{aligned} \tag{1.4}$$

Each tensor M_l has an index σ_l corresponding to the physical dimension at site l as well as bond dimensions a_l connecting neighboring sites. This is shown in Fig. 1.4. These bond dimensions represent entanglement between particles where $a_l = 1$ represents no entanglement between sites l and $l + 1$ and higher values represent more entanglement.

If the boundary conditions are periodic, we can also write the MPS with the trace

$$|\Psi\rangle = \sum_{\{\sigma_l\}} \mathbf{Tr}[M_1 \ldots M_L] |\sigma_1 \ldots \sigma_L\rangle \tag{1.5}$$

Overall, this means we can decompose a larger tensor into a non-unique tensor network of L tensors where the bounds are rank-2 and the inner tensors are rank-3 [3] [4] .

1.1.3 Matrix Product Operators

Here we extend the concept of Matrix Product States to Matrix Product Operators (MPO) which are used to map one MPS onto another, derived from operators in quantum mechanics

[4][5]. Note that by simply reshaping or combining legs of an MPO, we can apply all the same operations to this structure exactly as we can to an MPS. Unlike an MPS however, MPOs are made up of rank-3 tensors on the bounds and rank-4 tensors in the interior.

Similar to the Matrix Product State, we can represent an operator as

$$\hat{O} = \sum_{\{\sigma_l\}} \sum_{\{\tilde{\sigma}_l\}} C_{(\sigma_1,\ldots,\sigma_L)(\tilde{\sigma}_1,\ldots,\tilde{\sigma}_L)} \left|\sigma_1 \ldots \sigma_L\right\rangle \left\langle\tilde{\sigma}_1 \ldots \tilde{\sigma}_L\right| \tag{1.6}$$

where C is a matrix of size $(d^L \times d^L)$. This can be written as a Matrix Product Operator, exactly analogous to an MPS, such that

$$\begin{aligned} \hat{O} &= \sum_{\{a_l\}} \sum_{\{\sigma_l\}} \sum_{\{\tilde{\sigma}_l\}} W_{a_1}^{\sigma_1,\tilde{\sigma}_1} W_{a_1,a_2}^{\sigma_2,\tilde{\sigma}_2} \ldots W_{a_{L-1}}^{\sigma_L,\tilde{\sigma}_L} \left|\sigma_1 \ldots \sigma_L\right\rangle \left\langle\tilde{\sigma}_1 \ldots \tilde{\sigma}_L\right| \\ &= \sum_{\{a_l\}} \sum_{\{\sigma_l\}} \sum_{\{\tilde{\sigma}_l\}} W_1^{\sigma_1,\tilde{\sigma}_1} \ldots W_L^{\sigma_L,\tilde{\sigma}_L} \left|\sigma_1 \ldots \sigma_L\right\rangle \left\langle\tilde{\sigma}_1 \ldots \tilde{\sigma}_L\right| \end{aligned} \tag{1.7}$$

where W_l has bond dimensions a_{l-1} and a_1, but in contrast to a MPS, now has two physical dimensions σ_l and $\tilde{\sigma}_l$. This is visible in Fig. 1.5.

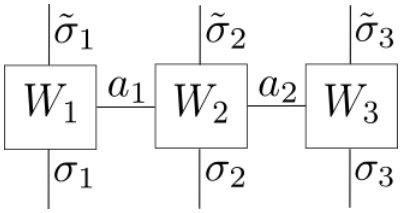

Figure 1.5: 3-site Matrix Product Operator according to Eq. (1.7)

1.1.4 Area Law and Matrix Product States

Many important states such at the ground state of a local non-critical Hamiltonian follow a famous concept known as an area law. It has been shown that matrix product states also fulfill a 1-Dimensional area law which can be seen as an important justification for their use in physics as well as in our application [6].

We consider a bipartite Hilbert Space $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$ with any state $|\Psi\rangle \in \mathcal{H}$. Its corresponding density matrix is

$$\rho = \left|\Psi\right\rangle \left\langle\Psi\right|$$

From here, we can calculate the Von Neumann entanglement entropy [7] for the state $|\Psi\rangle$ as follows

$$S_{A|B} = -Tr_A(\rho_A \log_2 \rho_A) = -\sum_{a=1}^{D} \lambda_a \log_2 \lambda_a \tag{1.8}$$

where λ_a are the eigenvalues of $\rho_A = Tr_A \rho$. The partial trace is the equivalent of a marginal distribution [8].

As the size of systems A and B grow, the entropy scales based on the boundary between the two partitions according to an area law as shown in Fig. 1.6. Thus, we can say

$$S_{A|B} \sim \partial A \tag{1.9}$$

The Von Neumann entropy is maximized if $\lambda_a = \frac{1}{D}\ \forall a$ [7]. This means we can set an upper bound on our entanglement entropy based on the size of our subsystem A.

$$S_{A|B} = -\sum_{a=1}^{D} \lambda_a \log_2 \lambda_a \leq -\sum_{n=1}^{D} \frac{1}{D} \log_2 \frac{1}{D} = -\log_2 \frac{1}{D} = \log_2 D$$

or more concisely

$$S_{A|B} \leq \log_2 D \tag{1.10}$$

Since the entropy scales according to Eq. (1.9), a 1-Dimensional system will have a constant boundary and therefore a constant entanglement entropy as the system grows [6].

Therefore, we can conclude that there is a bond dimension D in which our system is reasonably approximated by a Matrix Product State given by rearranging Eq. (1.10)

$$2^{const} \leq D \tag{1.11}$$

As D increases, we are able to search a larger area of the Hilbert space as shown in Fig. 1.7.

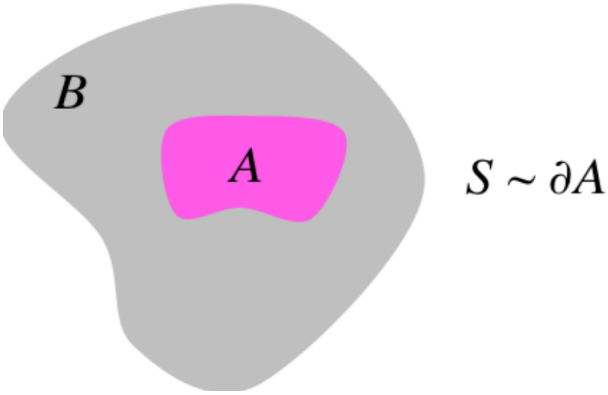

Figure 1.6: The entanglement entropy between systems A and B grows according to the boundary between them [9].

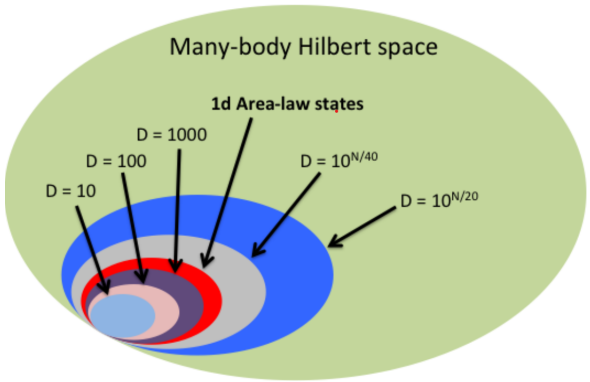

Figure 1.7: As our max bond dimension D increases, we slowly expand to cover a larger search area of our Hilbert space [9].

1.1.5 Finite Automata

Finite automata is an important concept in theoretical computer science as well as an essential part of hardware and software design. Finite automata give a description of the states of

a system and the inputs which allow transitions from one state to another, the complexity of which can be as simple as describing what occurs when a light switch is pressed or as complicated as accepting or rejecting inputs to a regular expression [10]. The state transitions also allow a form of memory where previous states and the paths taken through the system can have an impact on our decisions later.

As described in [11], finite automata and MPSs/MPOs have a direct connection, allowing us to easily describe one as the other and vice versa. The requirement is simply that our system has exponential decaying interactions such as nearest-neighbor interaction as our MPS/MPO requires a locality due to our site-wise tensors.

First, we can use the concept of finite automata to write quantum states such as the W-state or GHZ-state [12] as an MPS by hand. We can describe these as a finite automaton which parses all possible "quantum bit strings" and only accepts the eigenstates of the chosen superposition state while rejecting all others.

Likewise, we can look at each site of an MPS as a decision in a finite automaton. The tensor at each site represents all the possible transitions at that point in the automaton following an input from the physical dimension. Note that our first tensor in our tensor network is a matrix while the inner sites are rank-3 tensors. Looking at this from the point of view of finite automata, the first site does not depend on any previous input, so it is purely a matrix describing all the possible transitions for a given input such that element (i, j) represents the probability of a transition from state i to state j. However, the inner tensors depend on the previous inputs such that we have an extra dimension along which we can "slice" to create the transition matrix. Where along the tensor this slice lies is based on the values from earlier such that we fulfill the same criteria as a finite automaton.

Overall, this allows finite automata or similarly, decision trees and other graphs, to be represented as an MPS/MPO and vice versa.

1.2 Forms and Operations

1.2.1 Singular Value Decomposition

In order to decompose a tensor into an MPS, we apply the Singular Value Decomposition [4]. Be aware that we will always use the "Compact SVD" (Fig. 1.8) where only non-zero singular values are kept. This contrasts with a "Full SVD" where even zero singular values are kept and thus all of our matrices in the decomposition are square. During calculations, these zero singular values cause rows and columns full of zeros in our tensors, so it is computationally expensive for no useful reason. The number of non-zero singular values is known as the Schmidt rank which we call r [13].

$$M = U\Lambda V^\dagger \tag{1.12}$$

- M is a rank-2 tensor (matrix) with dimensions $(m \times n)$. The Schmidt rank is $r = \min(m, n)$. Therefore, our maximum number of singular values is equal to the shortest leg of M.
- U contains the left eigenvectors of M such that $U^\dagger U = \mathbb{1}$ with dimensions $(m \times m)$ for a full SVD and $(m \times r)$ for a compact SVD.
- Λ is a diagonal (albeit not necessarily square) matrix containing the singular values in descending order $s_1 > s_2 > \ldots$. This matrix has dimensions $(m \times n)$ for a full SVD and

$(r \times r)$ for a compact SVD. This may be S or Σ in other literature.

- $V^\dagger$ contains the right eigenvectors of M such that $VV^\dagger = \mathbb{1}$ with dimensions $(n \times n)$ for a full SVD and $(r \times n)$ for a compact SVD.

Note that the Full SVD will lead to equal bond dimensions along our entire MPS, while a Compact SVD will have a bond constrained by the Schmidt rank at each site. Also important to note is that the overall time complexity of the SVD of a matrix is given by $\mathcal{O}(mn^2)$ where n is the shortest dimension [4].

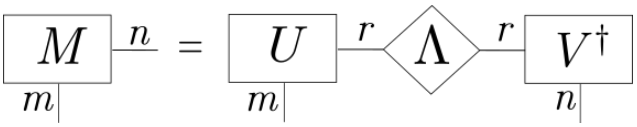

Figure 1.8: Compact SVD in tensor network notation. Note that the legs can be moved around the tensor freely.

1.2.2 Left- and Right-Canonical Form

It is useful for our tensor network to be put in what is called a canonical form [4] where all tensors have a form exploiting the left and right eigenvector matrices in the SVD:

- Left Canonical Form: $A_l^\dagger A_l = \mathbb{1}$ by exploiting U
- Right Canonical Form: $B_l B_l^\dagger = \mathbb{1}$ by exploiting $V^\dagger$

This form will allow us to simplify many calculations by using the unitary property as well as gives us a recognizable form during calculations.

As noted before, our MPS will be made of rank-2 and rank-3 tensors. For our rank-2 tensors, checking this condition is simply matrix multiplication by contracting the physical dimension. The rank-3 tensors, however, fulfill this condition by contracting both the physical dimension and left bond dimension (for A tensors) or right bond dimension (for B tensors) as seen in Fig. 1.9.

Mathematically, this definition of left and right canonical is

$$\sum_{a_{l-1}} \sum_{\sigma_l} A^{\sigma_l}_{a_{l-1},a_l} A^{\sigma_l}_{a_{l-1},a_l} = \delta^{a_l}_{a_l} \tag{1.13}$$

$$\sum_{a_l} \sum_{\sigma_l} B^{\sigma_l}_{a_{l-1},a_l} B^{\sigma_l}_{a_{l-1},a_l} = \delta^{a_{l-1}}_{a_{l-1}} \tag{1.14}$$

The usefulness of these forms will be evident later when, for example, calculating the scalar product or norm of a state. By knowing that a given tensor, when contracted with itself, results in an identity matrix, we can effectively ignore that site to focus on non-canonical sites.

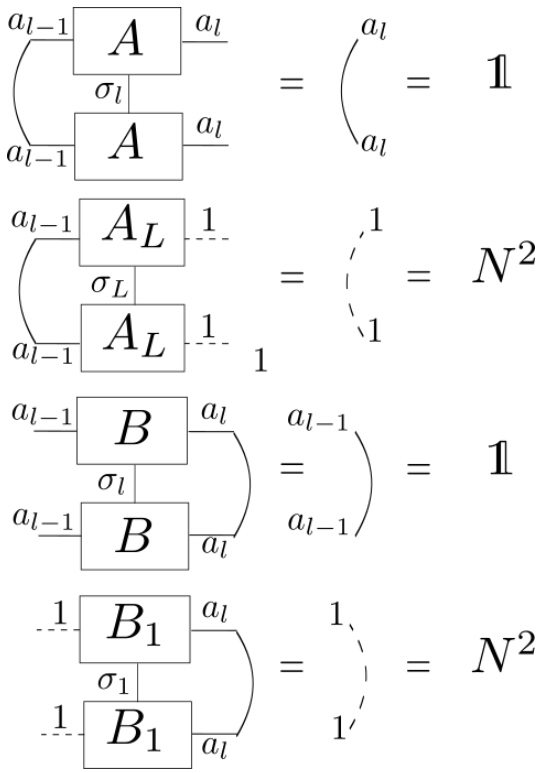

Figure 1.9: Procedure for checking canonical form. Dotted line indicates a dummy index. The last A tensor and first B tensor are related to the norm where N is the normalization constant. These are only canonical if $N = 1$.

1.2.3 Vidal's Notation

Let us assume that we have a left-canonical MPS created according to the algorithm in Sec. 1.4.3, however this time, at each SVD decomposition we remember the Singular Values Λ_l [4]. Note that we only need to *remember* the Singular Values matrix, it will still be multiplied into the new C' tensor as done in the algorithm. We can now use these values to put our MPS into Vidal notation where we will have two types of tensors in our network:

1. Γ tensors which sit on each site such that Γ_l sits on the l^{th} site

2. Λ tensors which sit on each bond such that Λ_l sits between Γ_l and Γ_{l+1}

An MPS fully decomposed with this method is found in Fig. 1.10.

From the algorithm for creating a left-canonical MPS, we now have at each site l a tensor A_l and we know its related Singular Values $\Lambda^{[l]}$ where $[l]$ signifies that it is not a tensor dimension, but that it is associated with that site. We can now find Γ_l by rearranging the following equation

$$A^{\sigma_l}_{a_{l-1},a_l} = \sum_{a_{l-1}} \Lambda^{[l-1]}_{a_{l-1},a_{l-1}} \Gamma^{\sigma_l}_{a_{l-1},a_l} \tag{1.15}$$

Rearranging this definition, we can calculate each Γ_l using the inverse of the previous site's singular values

$$\Gamma^{\sigma_l}_{a_{l-1},a_l} = \sum_{a_{l-1}} (\Lambda^{[l-1]}_{a_{l-1},a_{l-1}})^{-1} A^{\sigma_l}_{a_{l-1},a_l} \tag{1.16}$$

Likewise for a right-canonical MPS, we have

$$B^{\sigma_l}_{a_{l-1},a_l} = \sum_{a_l} \Gamma^{\sigma_l}_{a_{l-1},a_l} \Lambda^{[l]}_{a_l,a_l} \tag{1.17}$$

Notice that this gives us a common relation between left-canonical A tensors and right-canonical B tensors such that we can convert between them easily regardless of the initial normalization of the MPS. We can do this by simply choosing which neighboring bond to multiply a given Γ with. This is also charted in Fig. 1.12.

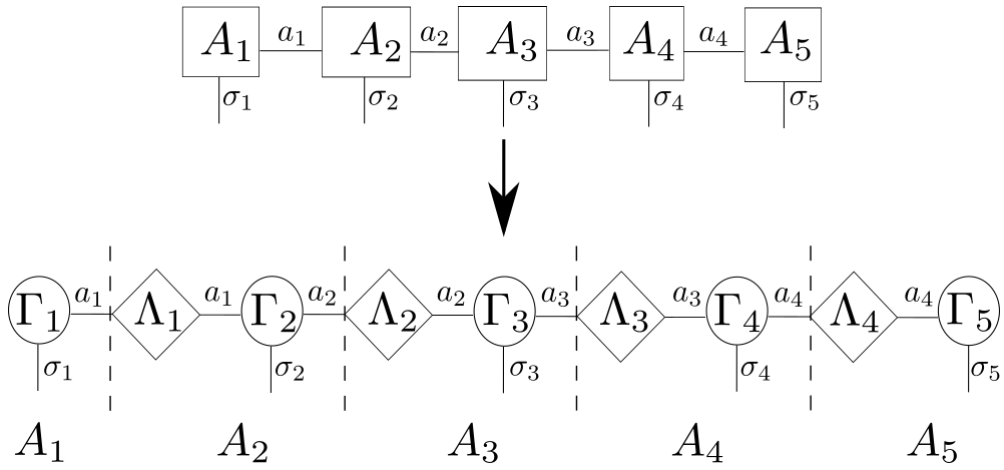

Figure 1.10: Decomposition of a 5-site left-canonical MPS into Vidal's Notation (Form: $A_1A_2A_3A_4A_5$)

1.2.4 Site Canonical Form (Mixed Canonical Form)

Once we have a state broken into Vidal's notation, we have the option of rebuilding it not only into left-canonical and right-canonical forms, but also site canonical (mixed canonical). This is a method of mixing A and B tensors, allowing us to essentially zoom in and focus on a given site.

Given our network of Γ and Λ tensors, we will choose a site l and build A tensors from the left side at sites $i \in [1, l-1]$ and B tensors from the right side at sites $i \in [l+1, L]$. This will however leave a site and its surrounding bonds such that we can create $M_i = \Lambda^{[i-1]}\Gamma^{[i]}\Lambda^{[i]}$ or more rigorously

$$M^{\sigma_i}_{a_{i-1},a_i} = \sum_{a_i} \sum_{a_{i-1}} \Lambda^{[i-1]}_{a_{i-1},a_{i-1}} \Gamma^{\sigma_i}_{a_{i-1},a_i} \Lambda^{[i]}_{a_i,a_i} \tag{1.18}$$

From this, we can create a state of the following form by exploiting Vidal's notation. The usefulness of this form will be clear during computations later.

$$|\Psi\rangle = \sum_{\{a_l\}} \sum_{\{\sigma_l\}} A_1 \dots A_{l-1} M_l B_{l+1} \dots B_L \, |\sigma_1 \dots \sigma_L\rangle \tag{1.19}$$

The tensor network notation for this is found in Fig. 1.11. All possible conversions between canonical forms can be seen in Fig. 1.12.

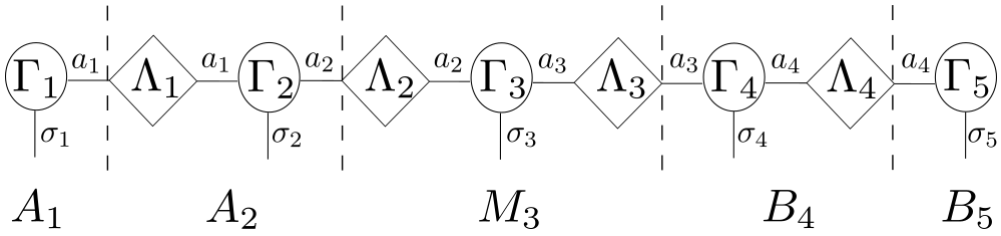

Figure 1.11: 5-site MPS in Site Canonical Form at the 3rd site (Form: $A_1A_2M_3B_4B_5$)

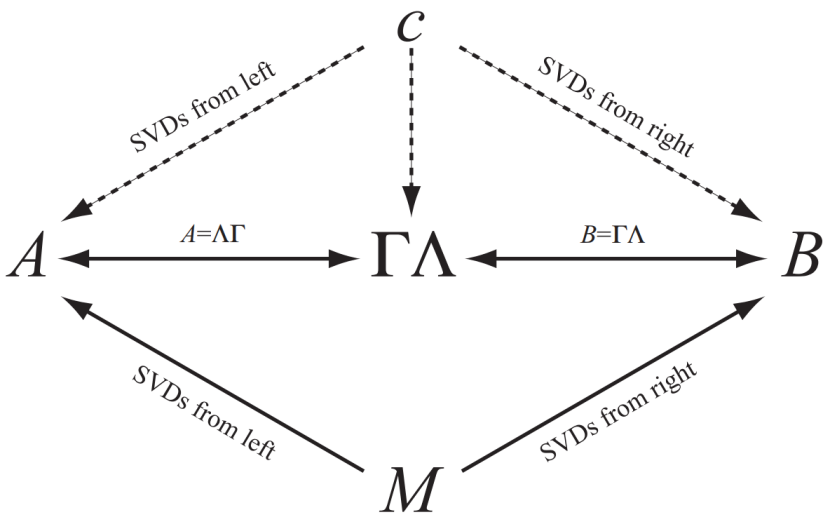

Figure 1.12: Various operations for converting between tensor types [4].

1.2.5 Scalar Product

Let us say we have two states $|\Psi\rangle$ and $|\tilde{\Psi}\rangle$ such that

$$|\Psi\rangle = \sum_{\{\sigma_l\}} C_{\sigma_1,\ldots,\sigma_L} |\sigma_1 \ldots \sigma_L\rangle$$

$$|\tilde{\Psi}\rangle = \sum_{\{\sigma_l\}} \tilde{C}_{\sigma_1,\ldots,\sigma_L} |\sigma_1 \ldots \sigma_L\rangle$$

where C and $\tilde{C}$ are rank-L tensors with one leg for each physical index σ_l.

This means if we want to find the scalar product, we have to contract each individual leg such that

$$\langle\tilde{\Psi}|\Psi\rangle = \sum_{\{\sigma_l\}} \tilde{C}^{\dagger}_{\sigma_1,\ldots,\sigma_L} C_{\sigma_1,\ldots,\sigma_L}$$

However, if each state is in the form of an MPS such that

$$|\Psi\rangle = \sum_{\{a_l\}} \sum_{\{\sigma_l\}} M^{\sigma_1}_{a_1} M^{\sigma_2}_{a_1,a_2} M^{\sigma_3}_{a_2,a_3} \ldots M^{\sigma_N}_{a_{L-1}} |\sigma_1 \ldots \sigma_L\rangle$$

$$|\tilde{\Psi}\rangle = \sum_{\{a'_l\}} \sum_{\{\sigma_l\}} \tilde{M}^{\sigma_1}_{a'_1} \tilde{M}^{\sigma_2}_{a'_1,a'_2} \tilde{M}^{\sigma_3}_{a'_2,a'_3} \ldots \tilde{M}^{\sigma_N}_{a'_{L-1}} |\sigma_1 \ldots \sigma_L\rangle$$

the scalar product can now be done site-wise. This means locally by contracting each smaller tensor first along σ_l, reshaping the bond dimensions into a single left and a right leg, and then finally contracting the bonds between each site until we have a scalar (see Fig. 1.13).

$$\begin{aligned}\langle\tilde{\Psi}|\Psi\rangle &= \sum_{\{a_l\}}\sum_{\{a'_l\}}\sum_{\{\sigma_l\}} \tilde{M}^{\dagger\sigma_1}_{a'_1} M^{\sigma_1}_{a_1} \tilde{M}^{\dagger\sigma_2}_{a'_1,a'_2} M^{\sigma_2}_{a_1,a_2} \dots \tilde{M}^{\dagger\sigma_N}_{a'_{L-1}} M^{\sigma_N}_{a_{L-1}} \\ &= \sum_{\{a_l a'_l\}} T^{[\sigma_1]}_{(a_1 a'_1)} T^{[\sigma_2]}_{(a_1 a'_1)(a_2 a'_2)} \dots T^{[\sigma_L]}_{(a_{l-1} a'_{l-1})}\end{aligned} \tag{1.20}$$

Note that a zipper contraction as done in [4] is another possible method and most likely more efficient since $a_l a'_l$ can get very large. However, the method above is what was used in this thesis.

We can also use this to find the norm of a state. Let us assume that we have a state in left-canonical form such that

$$|\Psi\rangle = \sum_{\{a_l\}}\sum_{\{\sigma_l\}} A^{\sigma_1}_{a_1} A^{\sigma_2}_{a_1,a_2} A^{\sigma_3}_{a_2,a_3} \dots A^{\sigma_N}_{a_{L-1}} |\sigma_1 \dots \sigma_L\rangle$$

Now, the benefit of this form becomes obvious. Remembering the canonical form in Sec. 1.2.2, each tensor besides the last one (unless normalized) will contract to $\mathbb{1}$, thus the scalar product for finding the norm simplifies to the tensors at the final site

$$\begin{aligned}\langle\Psi|\Psi\rangle &= \sum_{\{a_l\}}\sum_{\{a'_l\}}\sum_{\{\sigma_l\}} A^{\dagger\sigma_1}_{a'_1} A^{\sigma_1}_{a_1} A^{\dagger\sigma_2}_{a'_1,a'_2} A^{\sigma_2}_{a_1,a_2} \dots A^{\dagger\sigma_N}_{a'_{L-1}} A^{\sigma_N}_{a_{L-1}} \\ &= \sum_{\{a_l\}}\sum_{\{a'_l\}}\sum_{\{\sigma_l\}} \delta^{a_1}_{a'_1} \delta^{a_2}_{a'_2} \dots A^{\dagger\sigma_N}_{a'_{L-1}} A^{\sigma_N}_{a_{L-1}} \\ &= \sum_{a_{L-1}}\sum_{\sigma_N} A^{\dagger\sigma_N}_{a'_{L-1}} A^{\sigma_N}_{a_{L-1}}\end{aligned}$$

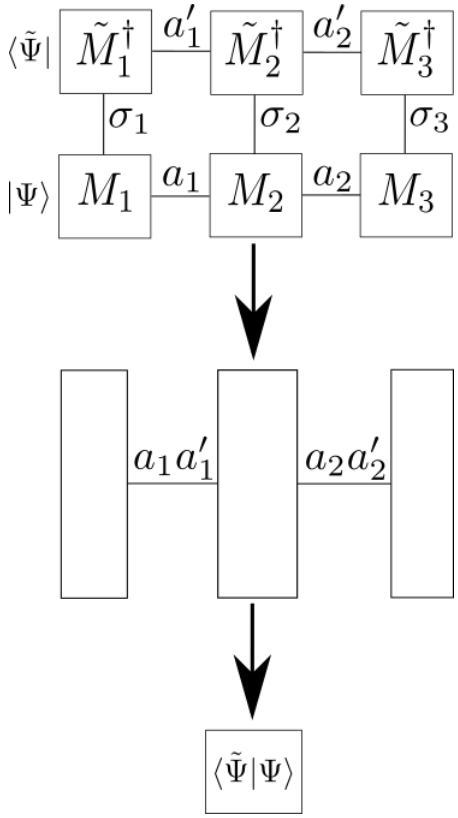

Figure 1.13: 3-site MPS contracted locally and then fully contracted to give the overlap of two MPSs. Similarly, a zipper method can be used as defined in [4].

1.3 Euclidean Distance and Cosine Similarity

Using the scalar product, we can define two metrics that will be used to quantify our results later on. First, we have the Euclidean distance [4] between two states such that

$$\| \, |\Psi\rangle - |\tilde{\Psi}\rangle \, \|^2 = \langle\Psi|\Psi\rangle - \langle\tilde{\Psi}|\Psi\rangle - \langle\Psi|\tilde{\Psi}\rangle + \langle\tilde{\Psi}|\tilde{\Psi}\rangle \tag{1.21}$$

Second, we can define the cosine similarity, a metric often used in machine learning and data analysis that is derived from the Euclidean dot product formula [14] [15]

$$similarity = \frac{\langle\tilde{\Psi}|\Psi\rangle}{\langle\Psi|\Psi\rangle \, \langle\tilde{\Psi}|\tilde{\Psi}\rangle} \tag{1.22}$$

For these metrics, a Euclidean distance of 0 and a similarity of 1 occur only if the states are the same state.

1.4 Applications

1.4.1 Matrix Product State of the W-state

This section is primarily used to build into using tensors and what an MPS means practically and what the elements within each tensor looks like.

Here we look at a common entangled quantum state called the W-state [12] and transform it first into a finite automaton and finally into an MPS [11]. The W state is a system of L

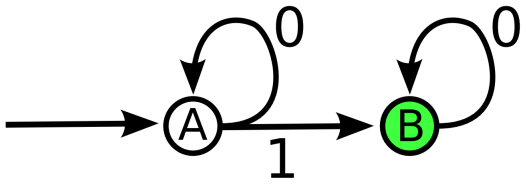

Figure 1.14: Finite Automaton aceepting eigenstates of the W-state [11]

qubits where only one qubit can be in its excited state at any time with all qubits having equal probability of being excited.

The L-qubit generalization is thus written as

$$|W\rangle = \frac{1}{\sqrt{L}}(|10\ldots0\rangle + |01\ldots0\rangle + \cdots + |00\ldots1\rangle)$$

This can be written as a finite automaton as shown in Fig. 1.14 which accepts only the eigenstates of the W-state, and rejects all other possibilities. We have 2 states in our finite automaton:

- State A: $|1\rangle$ has not passed through yet (We always begin here)
- State B: $|1\rangle$ has passed through

First, we need to construct tensors corresponding to the possible inputs. Each site l of the MPS will correspond to the l^{th} qubit/input to pass through the finite automaton. There are 4 possible transitions:

1. A to A (Caused by an input of $|0\rangle$, represented by matrix element (1,1))
2. A to B (Caused by an input of $|1\rangle$, represented by matrix element (1,2))
3. B to A (Impossible, always 0, represented by matrix element (2,1))
4. B to B (Caused by an input of $|0\rangle$, represented by matrix element (2,2))

Therefore, at each site we have a transition matrix given by

$$T = \begin{pmatrix} A \to A & A \to B \\ B \to A & B \to B \end{pmatrix} \tag{1.23}$$

Taking this into account, we can look at each site where we create the tensor of possible transitions and squeeze any dimensions with only impossible transitions (i.e. dimensions containing all 0s).

$$M_1 = \begin{pmatrix} |0\rangle & |1\rangle \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} |0\rangle & |1\rangle \end{pmatrix}$$

$$M_l = \begin{pmatrix} |0\rangle & |1\rangle \\ 0 & |0\rangle \end{pmatrix}$$

$$M_L = \begin{pmatrix} 0 & |1\rangle \\ 0 & |0\rangle \end{pmatrix} = \begin{pmatrix} |1\rangle \\ |0\rangle \end{pmatrix}$$

$$\sqrt{L}\,|W\rangle = \begin{pmatrix} |0\rangle & |1\rangle \end{pmatrix} \otimes \begin{pmatrix} |0\rangle & |1\rangle \\ 0 & |0\rangle \end{pmatrix}^{\otimes L-2} \otimes \begin{pmatrix} |1\rangle \\ |0\rangle \end{pmatrix} \tag{1.24}$$

We can index each tensor $M_l = M^{\sigma_l}_{a_{l-1},a_l}$ where a_{l-1} is the row (left bond dimension), a_l is the column (right bond dimension), and σ_l is the dimension *into* the tensor. This means for example for the first site

$$M^{[1]}_{11\sigma} = |0\rangle$$

and indexing *into* this gives us the elements

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

1.4.2 Matrix Product Operator of the Quantum Ising Model

We can also look at the Quantum Ising Model [16] which is an L-site 1D Hamiltonian with nearest-neighbor spin interaction such that

$$H = \sum_{l}^{L} \sigma_l^z \sigma_{l+1}^z + g\sigma_l^x \tag{1.25}$$

with j^{th} Pauli matrix of the l^{th} lattice position $\sigma_l^j \in M(\mathbb{C}, 2 \times 2)$ and parameter g.

Applying the exact same method used for decomposing the W-state into an MPS, we can represent this non-uniquely as

$$H = \begin{pmatrix} \mathbb{1} & \sigma^z & g\sigma^x \end{pmatrix} \otimes \begin{pmatrix} \mathbb{1} & \sigma^z & g\sigma^x \\ 0 & 0 & \sigma^z \\ 0 & 0 & \mathbb{1} \end{pmatrix}^{\otimes L-2} \otimes \begin{pmatrix} g\sigma^x \\ \sigma^z \\ \mathbb{1} \end{pmatrix}$$

Notice that in comparison to the MPS of the W-state, the entries are themselves (2×2) matrices. This means each elements has an extra 2 dimensions along which we can index.

For a better understanding of the concept of higher-dimensional tensors, let us consider the inner tensor which we can call A_{ijkl}. The matrix that is visible in the equation is the first two indices given by A_{ij}. If we index this to the first element, we have

$$A_{11} = \mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Now as an example, we can index further to A_{1122}

$$A_{1122} = \mathbb{1}_{22} = 1$$

This intuition is helpful for understanding how to use tensors in further applications.

1.4.3 Decomposing an Arbitrary Tensor into an MPS

Say we are given an arbitrary tensor T which is reshaped into a vector C of length N. We want to decompose it into a left-canonical MPS where the physical dimensions at each site correspond to an ordered sequence $\sigma = (\sigma_1, \ldots, \sigma_l, \ldots, \sigma_L)$.

While the physical dimensions are arbitrary in this algorithm and can be pre-selected, it is best to decompose the tensor into as many sites as possible in order to fully utilize the power of tensor networks. Having lower physical dimensions will also lower the Schmidt rank during SVD and will lower the bond dimensions that come naturally from decomposition. The bond dimensions from this algorithm will be related to the previous physical dimensions such that $a_l = \prod_1^l \sigma_l$ up to the point of symmetry in the bonds after which it will fall symmetrically.

To find the optimal σ, we can perform a prime factorization such that each physical dimension is a prime factor of N

$$N = \prod_{l=1}^{L} \sigma_l \tag{1.26}$$

where $\sigma_l \in \mathbb{Z}_p$. In this thesis, we also reorder these primes for symmetry.

A concrete example is that a matrix of (28×28) can be reshaped into a vector of length 784. The prime factorization of 784 is $\sigma = \{2, 2, 2, 2, 7, 7\}$. We then rearrange this into an ordered list for symmetry such that $\sigma = (2, 2, 7, 7, 2, 2)$.

Once we have chosen the physical dimensions we wish to have, can use the following algorithm:

1. Reshape C into a matrix $(\sigma_1 \times \frac{N}{\sigma_1})$ since the SVD can only be applied to matrices.

$$C_N \to C_{\sigma_1, \frac{N}{\sigma_1}}$$

2. Apply SVD to C such that $C = U\Lambda V^\dagger$. Here the bond dimension a_1 has length r_1 which is the Schmidt rank of the matrix (equivalent to σ_1).

$$C_{\sigma_1, \frac{N}{\sigma_1}} = \sum_{a_1} U_{a_1, a_1} \Lambda_{a_1, a_1} V^\dagger_{a_1, \frac{N}{\sigma_1}}$$

3. Set $A_1 = U^{\sigma_1}_{a_1}$. Create a new tensor $C' = \Lambda V^\dagger$

$$\sum_{a_1} U_{a_1, a_1} \Lambda_{a_1, a_1} V^\dagger_{a_1, \frac{N}{\sigma_1}} := \sum_{a_1} A^{\sigma_1}_{a_1} C'_{a_1, \frac{N}{\sigma_1}}$$

4. Reshape C' to $(a_1\sigma_2 \times \frac{N}{\sigma_1\sigma_2})$.

$$C'_{a_1, \frac{N}{\sigma_1}} \to C'_{a_1\sigma_1, \frac{N}{\sigma_1\sigma_2}}$$

5. Apply SVD to C' such that $C' = U\Lambda V^\dagger$.

$$C'_{a_1\sigma_1, \frac{N}{\sigma_1\sigma_2}} = \sum_{a_2} U_{a_1\sigma_1, a_2} \Lambda_{a_2, a_2} V^\dagger_{a_2, \frac{N}{\sigma_1\sigma_2}}$$

6. Reshape U to $(a_1 \times a_2 \times \sigma_2)$ where a_2 has length r_2, the Schmidt rank of the second matrix (equivalent to $a_1\sigma_1$).

$$U_{a_1\sigma_1, a_2} \to U_{a_1, a_2, \sigma_2}$$

7. Set $A_2 = U^{\sigma_2}_{a_1, a_2}$. Create a new tensor $C'' = \Lambda V^\dagger$.

$$\sum_{a_2} U_{a_1, a_2, \sigma_2} \Lambda_{a_2, a_2} V^\dagger_{a_2, \frac{N}{\sigma_1\sigma_2}} := \sum_{a_2} A^{\sigma_2}_{a_1, a_2} C''_{a_2, \frac{N}{\sigma_1\sigma_2}}$$

   Now in total we have

$$C_N = \sum_{a_2} \sum_{a_1} A^{\sigma_1}_{a_1} A^{\sigma_2}_{a_1, a_2} C''_{a_2, \frac{N}{\sigma_1\sigma_2}}$$

8. Repeat until the tensor is fully decomposed.

Likewise, we can perform this algorithm "flipped" to create a right-canonical MPS by updating B based on $V^\dagger$, and setting $C' = U\Lambda$.

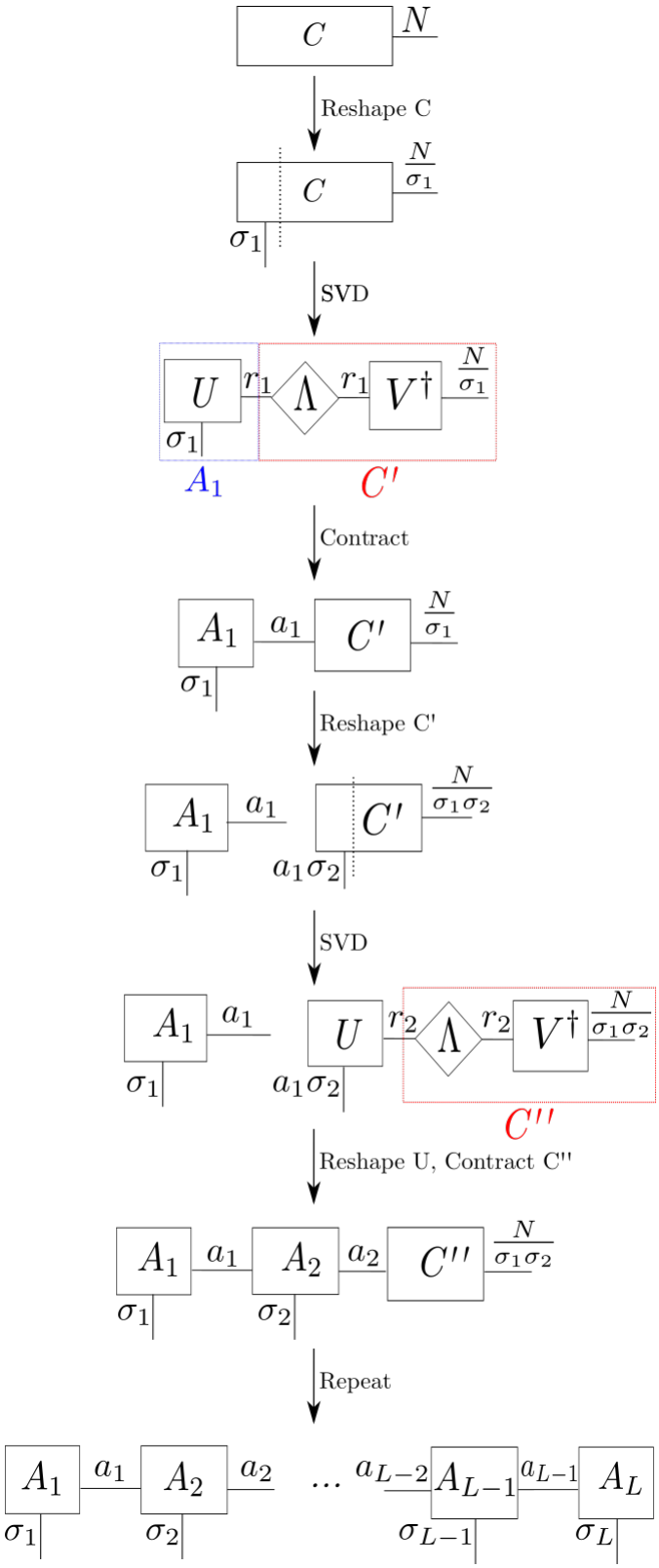

Figure 1.15: Tensor network notation for the decomposition of an arbitrary vector

Chapter 2

DMRG-Inspired Quantum State Compression

2.1 Purpose

We can effectively reduce the bond dimension of an MPS or MPO by minimizing the Euclidean distance between the original tensor network of bond dimension D and an approximation with bond dimension D' inspired by the ideas from the Density Matrix Renormalization Group [4]. We will always use $D' < D$ such that our approximation is also a compressed version of the original state.

2.2 Single Site Optimization

We begin with states $|\Psi\rangle$ with max bond dimension D and $|\tilde{\Psi}\rangle$ with max bond dimension D' defined by

$$|\Psi\rangle = \sum_{\{a_l\}} \sum_{\{\sigma_l\}} M^{\sigma_1}_{a_1} M^{\sigma_2}_{a_1,a_2} M^{\sigma_3}_{a_2,a_3} \dots M^{\sigma_N}_{a_{L-1}} |\sigma_1 \dots \sigma_L\rangle$$

$$|\tilde{\Psi}\rangle = \sum_{\{a'_l\}} \sum_{\{\sigma_l\}} \tilde{M}^{\sigma_1}_{a'_1} \tilde{M}^{\sigma_2}_{a'_1,a'_2} \tilde{M}^{\sigma_3}_{a'_2,a'_3} \dots \tilde{M}^{\sigma_N}_{a'_{L-1}} |\sigma_1 \dots \sigma_L\rangle$$

where $M^{\sigma_l}_{a_{l-1},a_l} =: M_l$ from here on.

We now minimize the Euclidean distance as defined by Eq. (1.21) by taking an element-wise partial derivative for a given matrix $\tilde{M}^\dagger_l$

$$\begin{aligned}
\frac{\partial}{\partial \tilde{M}^\dagger_l}(\| \, |\Psi\rangle - |\tilde{\Psi}\rangle \, \|_2^2) &= \frac{\partial}{\partial \tilde{M}_l}(\langle\Psi|\Psi\rangle - \langle\tilde{\Psi}|\Psi\rangle - \langle\Psi|\tilde{\Psi}\rangle + \langle\tilde{\Psi}|\tilde{\Psi}\rangle) \\
&= \frac{\partial}{\partial \tilde{M}_l}(\langle\tilde{\Psi}|\tilde{\Psi}\rangle - \langle\tilde{\Psi}|\Psi\rangle) \\
&= (\tilde{M}^\dagger_1 \dots \tilde{M}^\dagger_{l-1} \tilde{M}^\dagger_{l+1} \dots \tilde{M}^\dagger_L)(\tilde{M}_1 \dots \tilde{M}_l \dots \tilde{M}_L) \\
&- (\tilde{M}^\dagger_1 \dots \tilde{M}^\dagger_{l-1} \tilde{M}^\dagger_{l+1} \dots \tilde{M}^\dagger_L)(M_1 \dots M_l \dots M_L) \\
&= 0
\end{aligned} \tag{2.1}$$

However, since the elements of a matrix are scalars, this is equivalent to simply removing a site entirely from the tensor network of $\langle\tilde{\Psi}|$ which is a linear function of matrices.

We now set each part of Eq. (2.1) equal to each other and contract all sites around the site we wish to optimize. These sides will be called the left environment and the right environment around site l where

$$\begin{aligned}
\tilde{L}_{a'_{l-1},a'_{l-1}} &= \sum_{\{a'_l\}}\sum_{\{a'_l\}}\sum_{\{\sigma_l\}} \tilde{M}_1^\dagger \tilde{M}_1 \dots \tilde{M}_{l-1}^\dagger \tilde{M}_{l-1} \\
\tilde{R}_{a'_l,a'_l} &= \sum_{\{a'_l\}}\sum_{\{a'_l\}}\sum_{\{\sigma_l\}} \tilde{M}_{l+1}^\dagger \tilde{M}_{l+1} \dots \tilde{M}_L^\dagger \tilde{M}_L \\
L_{a_{l-1},a'_{l-1}} &= \sum_{\{a_l\}}\sum_{\{a'_l\}}\sum_{\{\sigma_l\}} \tilde{M}_1^\dagger M_1 \dots \tilde{M}_{l-1}^\dagger M_{l-1} \\
R_{a_l,a'_l} &= \sum_{\{a_l\}}\sum_{\{a'_l\}}\sum_{\{\sigma_l\}} \tilde{M}_{l+1}^\dagger M_{l+1} \dots \tilde{M}_L^\dagger M_L
\end{aligned} \tag{2.2}$$

Applying this to Eq. (2.1), we see that

$$\sum_{\{a_{l-1}\}}\sum_{\{a'_l\}} \tilde{L}_{a'_{l-1},a'_{l-1}} \tilde{M}_l \tilde{R}_{a'_l,a'_l} = \sum_{\{a_{l-1}\}}\sum_{\{a_l\}} L_{a_{l-1},a'_{l-1}} M_l R_{a_l,a'_l}$$

Notice that $\tilde{L}$ and $\tilde{R}$ are the scalar product of all sites of $\langle\tilde{\Psi}|\tilde{\Psi}\rangle$ before and after site l, respectively. Knowing this, we can exploit the site-canonical form at our given site to reduce the environments to the identity. This means that all sites in $\tilde{L}$ are A tensors and all sites in $\tilde{R}$ are B tensors such that we start with an initial compressed state of the form

$$|\tilde{\Psi}\rangle = \sum_{\{a'_l\}}\sum_{\{\sigma_l\}} \tilde{A}_1 \dots \tilde{A}_{l-1} \tilde{M}_l \tilde{B}_{l+1} \dots \tilde{B}_L \, |\sigma_1 \dots \sigma_L\rangle$$

We can directly update tensor $\tilde{M}_l$ as a contraction of the original tensor M_l and its environment, resulting in the correct bond dimensions through the legs hanging from the missing site

$$\tilde{M}_l = \sum_{\{a_{l-1}\}}\sum_{\{a_l\}} L_{a_{l-1},a'_{l-1}} M_l R_{a_l,a'_l} \tag{2.3}$$

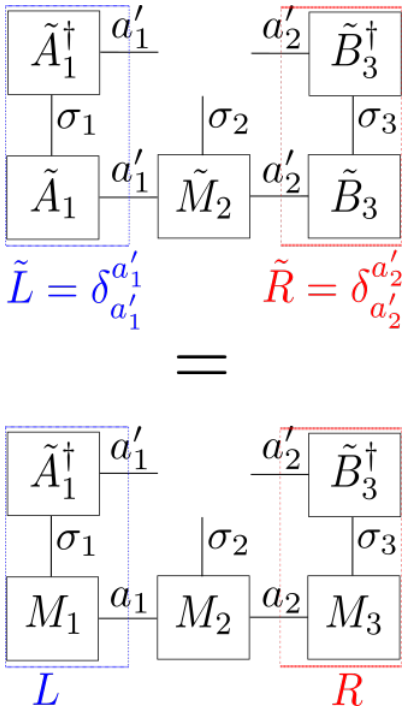

Figure 2.1: Optimization of the 2nd site of a 3-site MPS. The left and right environment of $\langle\tilde{\Psi}|\Psi\rangle$ at the top reduces to $\tilde{M}_2$ according to Eq. (2.3).

2.3 Optimization Sweep

In order to efficiently minimize the Euclidean distance between our original MPS and compressed MPS, we can optimize site-wise in a sweep from left to right (sites 1 to L) and then back right to left (sites L to 1) [4].

We begin by using a compressed MPS as prescribed in the site-wise optimization, but rather than having A tensors, we begin with a right-normalized state of B tensors. Since the sites will be optimized, the values within the original compressed state are not necessarily important, so we usually generate a random MPS with the desired bond dimension.

Therefore, we begin with a compressed state in site-canonical form on the first site

$$|\tilde{\Psi}\rangle = \sum_{\{a_l'\}} \sum_{\{\sigma_l\}} \tilde{M}_1 \tilde{B}_2 \dots \tilde{B}_L \, |\sigma_1 \dots \sigma_L\rangle \tag{2.4}$$

We then optimize $\tilde{M}_1$ according to Eq. (2.2) and Eq. (2.3). The updated site will be called $\tilde{M}_1'$.

In order to move to the next site, we apply a compact SVD to $\tilde{M}_1'$ such that

$$|\tilde{\Psi}\rangle = \sum_{\{a_l'\}} \sum_{\{\sigma_l\}} (U_1 \Lambda_1 V_1^\dagger) \tilde{B}_2 \dots \tilde{B}_L \, |\sigma_1 \dots \sigma_L\rangle$$

Now we have two things which we can exploit to help maintain the structure needed for site optimization:

1. In an SVD, U is left canonical. Therefore, the first site will be set to $\tilde{A}_1 = U_1$.
2. We can see that $\Lambda_1 V_1^\dagger \tilde{B}_2 = \Lambda_1 V_1^\dagger (\Gamma_2 \Lambda_2)$ due to the inherent form of a B tensor. We can update $\Gamma_2' = V_1^\dagger \Gamma_2$ such that $\Lambda_1 V_1^\dagger \tilde{B}_2 = \Lambda_1 \Gamma_2' \Lambda_2$. Notice that this is the form of

an M tensor such that $\tilde{M}'_2 = \Lambda_1 \Gamma'_2 \Lambda_2$. Working in reverse, this just means we multiply the remaining tensors to update the next site such that $\tilde{M}'_2 = \Lambda_1 V_1^\dagger \tilde{B}_2$

By applying the points above, we now have a compressed state

$$|\tilde{\Psi}\rangle = \sum_{\{a'_l\}} \sum_{\{\sigma_l\}} \tilde{A}_1 \tilde{M}_2 \tilde{B}_3 \dots \tilde{B}_L \, |\sigma_1 \dots \sigma_L\rangle$$

We continue this site-wise, setting $\tilde{A}_i = U_i$ and $\tilde{M}'_i = \Lambda_{i-1} V_{i-1}^\dagger \tilde{B}_i$, until we finally have a state

$$|\tilde{\Psi}\rangle = \sum_{\{a'_l\}} \sum_{\{\sigma_l\}} \tilde{A}_1 \tilde{A}_2 \dots \tilde{M}_L \, |\sigma_1 \dots \sigma_L\rangle$$

Now we repeat the process in the opposite direction, essentially doing everything in reverse. Analogous to the points in the left-right sweep, we exploit the following:

1. In an SVD, $V^\dagger$ is right canonical. Therefore, the last site will be set to $\tilde{B}_L = V_L^\dagger$.
2. We set $\tilde{M}'_{L-1} = \tilde{A}_{L-1} U_L \Lambda_L$.

After a full left-right then right-left sweep, we now end up with the form of our beginning state according to Eq. (2.4), but now with a Euclidean distance closer to the original state that we wish to approximate.

We calculate the Euclidean distance between the compressed MPS $|\tilde{\Psi}\rangle$ and original MPS $|\Psi\rangle$ after each sweep. If the change in distance between sweeps is less than a given threshold (e.g. $\epsilon < 10^{-8}$), then we have reached an optimal solution and stop sweeping. We can then increase our maximum bond dimension by 1 by padding the tensors with 0s along the bond dimension and continue sweeping until we reach a max bond dimension that gives us a suitable approximation.

Note that generally, we should calculate our metrics after each full back and forth sweep. Calculating it after a single site optimization is not generally done and causes inconsistencies where the distance does not decrease after each site, but does after each full sweep.

2.4 Application: Representing an Arbitrary Vector as an MPS

Here we can consider an arbitrary vector of length 4096 which could correspond practically to either the non-normalized quantum state of a 12-particle system in physics or a 4096-bit string in computer science. We can apply the algorithm described in Sec. 1.4.3 to decompose it into a 12-site MPS with physical dimension $\sigma_l = 2$ for all sites and bond dimensions $a = (2, 4, 8, 16, 32, 64, 64, 32, 16, 8, 4, 2)$. This is an exact representation of the vector meaning that a max bond dimension of 64 is required to represent it. We can, however, then apply our compression algorithm to analyze how well we can approximate this MPS, and therefore the vector, using a lower bond dimension. Here we analyze the cosine similarity and the Euclidean distance between our original MPS and all possible compressions. For this, we have chosen a threshold of $\epsilon = 10^{-8}$ between sweeps after which we move to a higher bond dimension. The results of this are shown in Table 2.1 and Fig. 2.2.

MaxBondDim	CosineSimilarity	EuclideanDistance
27	0.9505	0.22
41	0.9901	0.10
64	1.0	0

Table 2.1: Results for representing an arbitary vector of length 4096 as an MPS when passing the 95 % and 99 % similarity thresholds

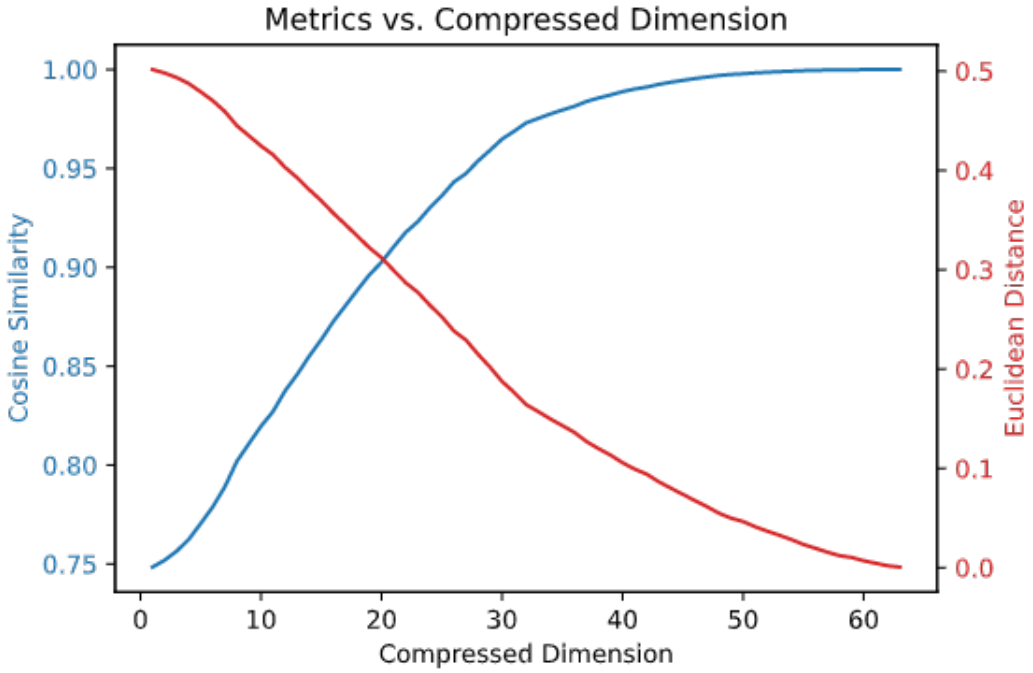

Figure 2.2: Cosine similarity and Euclidean distance between an MPS with max bond dimension 64 and its lower dimensional approximations

Chapter 3

Compressing Standard Neural Networks

3.1 Introduction to Supervised Learning

The work from this thesis will only be applied to models post-training, so specific in-depth details on training in supervised learning or machine learning in general will not be included. In order to understand the application, it is however important to understand the fundamental components of a neural network.

A neural network is made of a series of layers of neurons or perceptrons. The connections between each layer are described by a weights matrix such that the i^{th} neuron in the first layer has a weighted connection to the j^{th} neuron in the next layer contained as element W_{ji} [17]. Each neuron has a defined activation function based off of biological neurons in that the neuron only allows a signal to pass after a certain input threshold [18].

In supervised learning, we take an input (such as an image) and the known output (such as a vector classifying that image), and systematically adjust the weights and a bias offset in the system through a backpropagation algorithm until feeding an unknown input into the network will have a high accuracy of outputting a correct result [17].

Mathematically, a single layer network would thus be represented by a simple linear algebra problem described by

$$y = f(\mathbf{W}x + b) \tag{3.1}$$

with input x, output y, weights matrix $\mathbf{W}$, bias b, and activation function f. Generalizing this to more layers, we input the output from one layer into the next and so forth (beginning with layer 0)

$$y = f_2(\mathbf{W_2} f_1(\mathbf{W_1} f_0(\mathbf{W_0}x + b_0) + b_1) + b_2) \dots \tag{3.2}$$

3.2 Fully Connected 2-Layer Network (FC2)

The proof of concept for our method will be applied to a very simple network called FC2 [1]. Here all neurons connect to all others in two layers which is applied to the MNIST dataset for handwriting identification for the numbers 0-9. The input data from MNIST are grayscale (28×28) images where each pixel is a value between 0-255. This is flattened to a length 784 vector. The output data is a length 10 vector of 0s and a single 1 identifying the digit (i.e. to represent 0, the 1 will be in index 0 or to represent 9, the 1 will be in index 9) [19].

Our first layer will have 256 neurons and the Rectified Linear Unit (ReLU) activation function [20]. This function sets all negative values to 0 and allows all positive values through without changes. This layer can be represented by a 784×256 weights matrix W_0, accepting the MNIST input vector, and a length 256 bias vector b_0.

Our second layer will have 10 neurons and a linear activation function which simply lets all values through to form our output. This layer can be represented by a 256×10 weights matrix W_1 and a length 10 bias vector b_1.

In total, this simple network requires 203530 total parameters to represent, calculated by $size(\mathbf{W_0}) + size(b_0) + size(\mathbf{W_1}) + size(b_1) = 784 * 256 + 256 + 256 * 10 + 10 = 203530$. Assuming each parameter is a 32-bit float, this is roughly 814 kB just to store the values within the model, not including the activation function or any other operations in the model.

Most networks have significantly more complicated input/output data as well as more layers and more total neurons. A recent example is the GTP3 language model which in its smallest form has 125 million parameters and its largest 175 billion parameters [21].

The possible application of the MPS/MPO structure to machine learning can also be seen through several analogies with the physical theory previously discussed. The bond dimensions in an MPS/MPO describe entanglement between sites which can be seen as correlation between neighboring values in a machine learning model. There is also likely to be exponential decaying interactions in the form of nearby data being more similar and related to each other than distant data. For example, nearby pixels in image recognition such as MNIST are related and more similar to each other than distant pixels.

With this being said, we have two goals for this chapter:

1. Represent each layer of a trained neural network as an MPO rather than a large tensor.
2. Compress each MPO to significantly reduce the number of parameters necessary.

3.3 Metrics

There are several ways that we will quantify our results in the next few chapters. We wanted to generalize our results so that we have a practical machine learning result as well as a mathematical result.

When compressing an MPS/MPO, we will use the Euclidean distance and cosine similarity as described in Sec. 1.2.5. This gives us a mathematical result of how well the compressions approximate the original. We specifically are interested in the bond dimensions which pass the 0.95 and 0.99 cosine similarity thresholds.

Next we will consider the compression ratio which is the total parameters in our compressed MPS/MPO versus the original parameters in either the layer or the full network. In Chapters 3 and 4, we analyze the compression of individual layers, so this ratio is

$$CompressionRatio = 100 * \frac{N_i}{N_{0,i}}$$

where N_i is the compressed number of parameters of the i^{th} layer and $N_{0,i}$ is the original number of parameters of that layer.

However, in Chapter 5, we will analyze the compression ratio relative to the whole original network.

$$CompressionRatio = 100 * \frac{\sum_i N_i}{\sum_i N_{0,i}}$$

An important note is that we have chosen the parameters to be defined this way in our MPS/MPO layers, however, due to the SVD decomposition of the larger tensors, these parameters are not fully independent of each other. This means that there will be compression ratios larger than 100%, however this may not be fully accurate as many of the parameters counted will have interdependency. A short example is just that if we have a matrix M of size (2×2), then an SVD decomposition will return $M = USV^\dagger$ with respective dimensions $(2 \times 2)(2 \times 2)(2 \times 2)$ totaling 12 parameters. This does however mean that we would be overcounting and thus other results could be better depending on how you define the number of independent parameters. Therefore, our results will be considered more as an upper bound to using this method.

We will also look at the overall loss of the network. Here we will find the chance in accuracy between our original network's accuracy α_0 and our compressed network's accuracy α such that

$$L = \alpha_0 - \alpha$$

The figures and tables contained in the rest of this thesis will use multiple combinations of these metrics in order to give us a full picture of relationships between all the values. We are particularly interested in the impact on neural network loss as a function of compression ratio, compressed max bond dimension, cosine similarity, and Euclidean distance

3.4 Compressing a Standard Neural Network

3.4.1 Generalization

In this section, we will begin with a pre-trained model capable of some accuracy α_0. From here, we can extract the weight matrices of each layer and apply the method described in Sec. 1.4.3. Then we can reshape the result and represent our layer as an MPO.

Note that applying the above SVD operation to decompose a tensor will create an MPS, so we will need to pull the "combined dimension" corresponding to each side of an MPO site which we will then reshape into the correct MPO. We must first do the prime decomposition of both dimensions of the weights matrices and apply some modifications so that our layers will align properly:

1. The list of factors need to be equal length to represent the physical dimensions on each side of a site. Otherwise, our MPOs at each layer will have a different number of sites and not connect properly. To do this, we combine some prime factors into composite factors from the longer list or pack 1s for the shorter list.

2. The order is an open-choice. In this thesis, we focused on symmetry such that interior points receive more input. More analysis on the ordering has not been performed.

The following procedure is illustrated in Fig. 3.1. Let us say we have a weights matrix W with dimensions $(N \times M)$. First, we find the prime factorization of N to be used as the input physical dimensions of the layer $\sigma^{[0]}$ and the prime factorization of M for the output physical dimensions of the layer $\sigma^{[1]}$. $\sigma^{[i]}$ is an arbitrary sequence with elements $\sigma_l^{[i]}$ corresponding to the dimensions at each site l of the MPO.

We first adjust each list such that they are both length L defining the total sites in the MPO needed to represent the layer such that $\sigma^{[0]} = (\sigma_1^{[0]}, \ldots, \sigma_L^{[0]})$ and $\sigma^{[1]} = (\sigma_1^{[1]}, \ldots, \sigma_L^{[1]})$. We now close our matrix into a single tensor of length NM. In order to decompose it, we have

to pull off the combined physical dimension at each site. Our MPS is then decomposed to have legs $\sigma = (\sigma_1^{[0]}\sigma_1^{[1]}, \ldots, \sigma_L^{[0]}\sigma_L^{[1]})$. Once decomposed, we can then reshape the MPS into an MPO of the correct form by "opening" the legs at each site through a reshape. When moving to the next layer, we must use the previous output physical dimensions.

This method can be applied to higher-rank weight tensors by combining dimensions together. For example, an $(L \times M \times N)$ tensor can be reshaped into $(LM \times N)$ and then the above method repeated.

We can then compress each layer and compare the accuracy lost when replacing the original MPO.

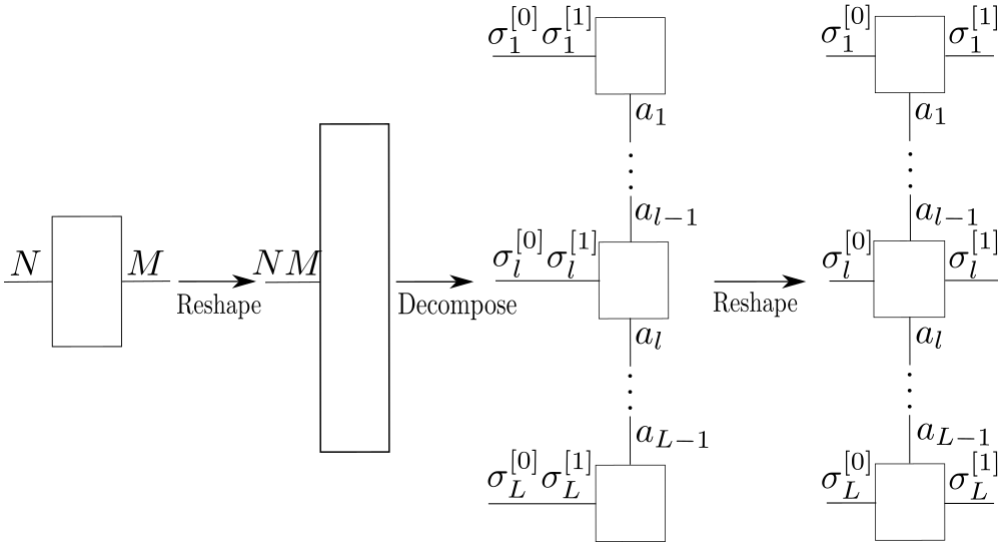

Figure 3.1: Decomposition of a matrix into a Matrix Product Operator

3.4.2 Application: Pre-Trained FC2 Model

Let us consider the FC2 network as previously described in Sec. 3.2

The first layer W_0 has dimensions (784×256) which have prime factorizations $\{2, 2, 2, 2, 7, 7\}$ (length 6) and $\{2, 2, 2, 2, 2, 2, 2, 2\}$ (length 8) respectively.

To solve the length problem, we simply combine values in the longer list until we have the same length as the shorter list. This means we can simply modify $\{2, 2, 2, 2, 2, 2, 2, 2\}$ to $\{2, 2, 2, 2, 4, 4\}$.

We then arrange these values into a symmetrical, ordered list such that finally we have $\sigma^{[0]} = (2, 2, 7, 7, 2, 2)$ and $\sigma^{[1]} = (2, 2, 4, 4, 2, 2)$. We reshape W_0 into a vector of length 200704 and decompose it into an MPS with physical dimensions $\sigma = \sigma^{[0]}\sigma^{[1]} = (4, 4, 28, 28, 4, 4)$. The physical dimensions are then reshaped to form the correct shape MPO with input dimensions $N = \sum_l \sigma_l^{[0]}$ and output dimensions $M = \sum_l \sigma_l^{[1]}$.

Likewise for the second layer, W_1 has dimensions (256×10). Here, we must use the same $\sigma^{[1]}$ from the last layer for the input so that it connects properly to the previous layer. The prime factorization of 10 is however $(2, 5)$, but we have a 6-site MPO. Therefore, we pack this list with 1s such that $\sigma^{[2]} = (1, 1, 2, 5, 1, 1)$. Again, the choice for the order of 2 and 5 are arbitrary. Now as previously done, we reshape W_1 into a vector of length 2560 and decompose it into an MPS with physical dimensions $\sigma = \sigma^{[1]}\sigma^{[2]} = (2, 2, 8, 20, 2, 2)$. Then we reshape it to have the correct MPO shape with input dimensions $N = \sum_l \sigma_l^{[1]}$ and output

dimensions $M = \sum_l \sigma_l^{[2]}$.

In this case, we have extracted the weights matrices from a pre-trained network and applied the algorithm from Sec. 1.4.3 to decompose it. The bond dimensions of our MPOs are directly related to the Schmidt rank as discussed in Sec. 1.2.1. The dimensions of the legs can be very high after decomposing due to large physical dimensions. For our FC2 model, W_0 naturally decomposes into an MPS/MPO with bond dimensions $a = (4, 16, 448, 16, 4)$ and W_1 into $a = (2, 4, 32, 4, 2)$. This is shown in Fig. 3.2.

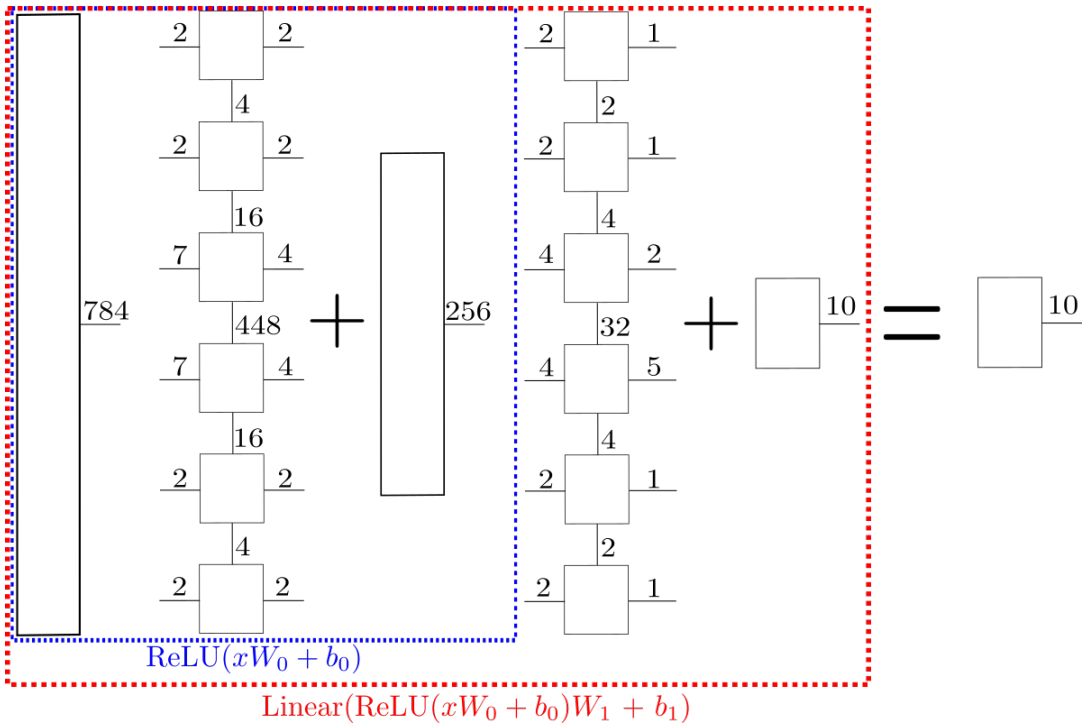

Figure 3.2: Standard FC2 model decomposed into MPOs

However, applying our algorithm, we can compress this much further without significant loss of accuracy, while also bringing the number of parameters necessary to a level less than the original model. In order to test this, we train an FC2 model up to an arbitrary accuracy since we are only concerned with loss from this initial value. We then compress it with the thresholds $\epsilon = 10^{-1}$ for the first layer since high bond dimensions are significantly slower to compress and $\epsilon = 10^{-8}$ for the second layer. The results are found in Table 3.1, Table 3.2, Fig. 3.3, Fig. 3.4, Fig. 3.5, and Fig. 3.6.

This application will be further optimized in Sec. 5.5 where we will apply new methodology that will be discussed in the next chapter.

Layer	MaxBondDim	CosineSimilarity	EuclideanDistance	CompressionRatio
0	184	0.950	12.09	84.13 %
0	276	0.990	5.40	126.06 %
0	488	1.00	0	204.44 %
1	21	0.955	1.64	93.44 %
1	28	0.993	0.66	124.06 %
1	32	1.00	0	141.56 %

Table 3.1: Results for representing a standard FC2 network as an MPO. Thresholds were 0.95 and 0.99 cosine similarity. Compression ratio is relative to **the parameters of the corresponding layer** of the FC2 network.

Layer	MaxBondDim	Loss	CompressionRatio
0	32	4.89 %	14.56 %
0	75	0.89 %	33.75 %
0	488	0 %	204.44%
1	8	4.16 %	36.56%
1	19	1.0 %	84.69%
1	32	0 %	141.56%

Table 3.2: Effect on loss when compressing each layer individually. Thresholds were bond dimensions causing a loss less than 1 % and 5 %. Compression ratio is relative to **the parameters of the corresponding layer** of the FC2 network.

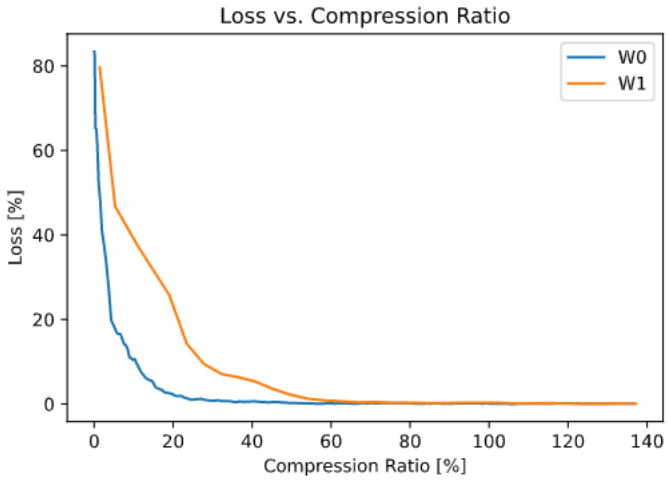

Figure 3.3: Loss vs. Compression Ratio of each layer with original bond dimensions $D = 448$ for layer 0 and $D = 32$ for layer 1. Compression ratio is relative to **the parameters of the corresponding layer**.

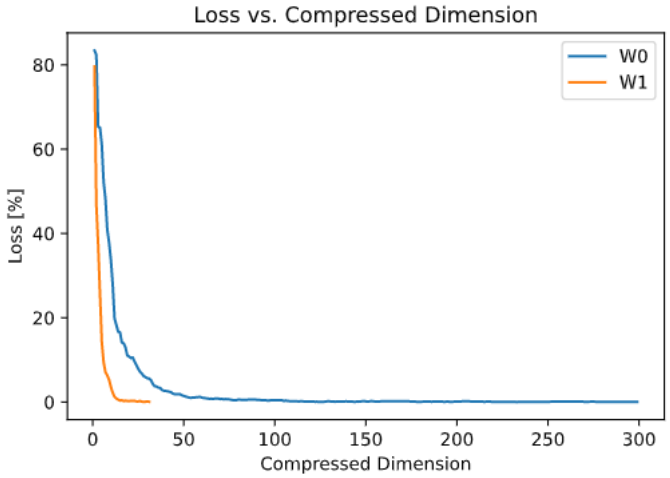

Figure 3.4: Loss vs. Compressed Bond Dimension of each layer with original bond dimensions $D = 448$ for layer 0 and $D = 32$ for layer 1

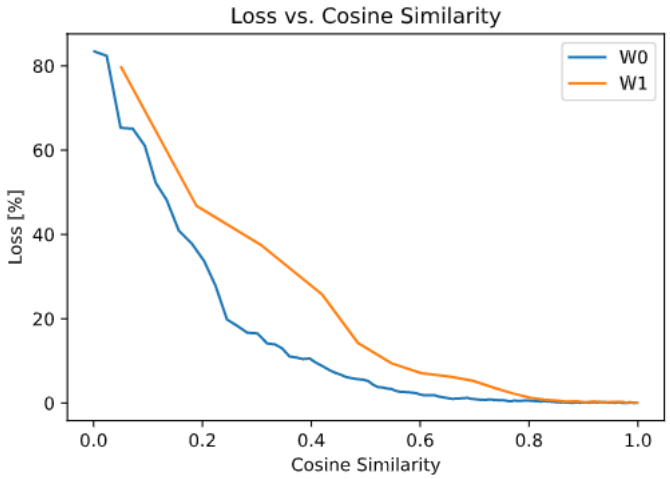

Figure 3.5: Loss vs. Cosine Similarity of each layer compared to the original bond dimensions $D = 448$ for layer 0 and $D = 32$ for layer 1

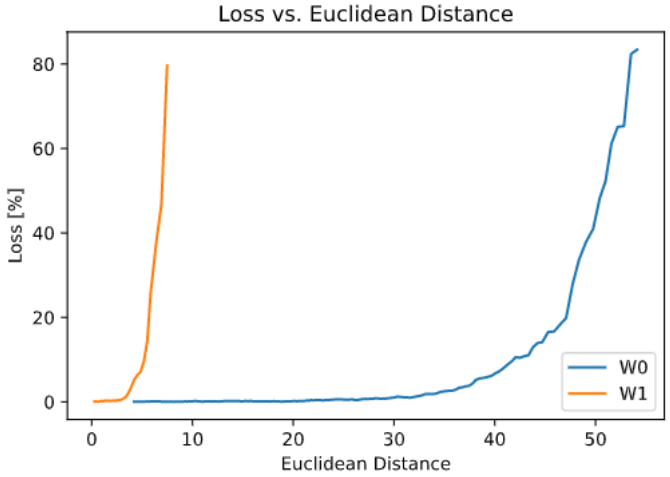

Figure 3.6: Loss vs. Euclidean Distance of each layer compared to the original bond dimensions $D = 448$ for layer 0 and $D = 32$ for layer 1

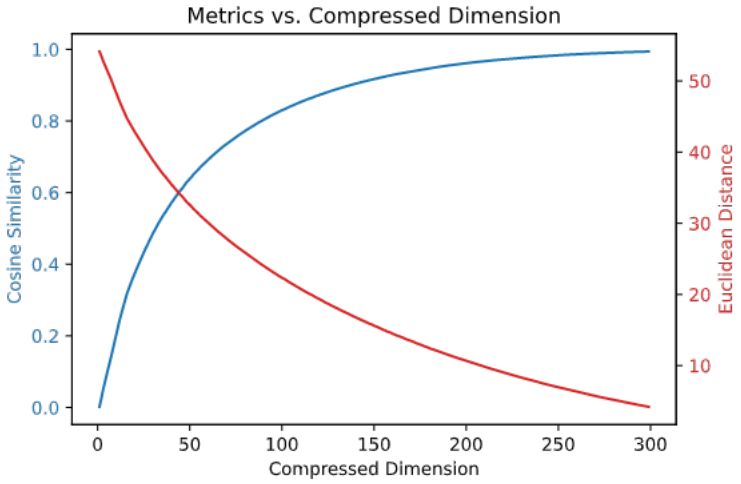

Figure 3.7: Metrics when compressing the second layer W_0 of our FC2 model from its original max bond dimension 448 (up to bond 300)

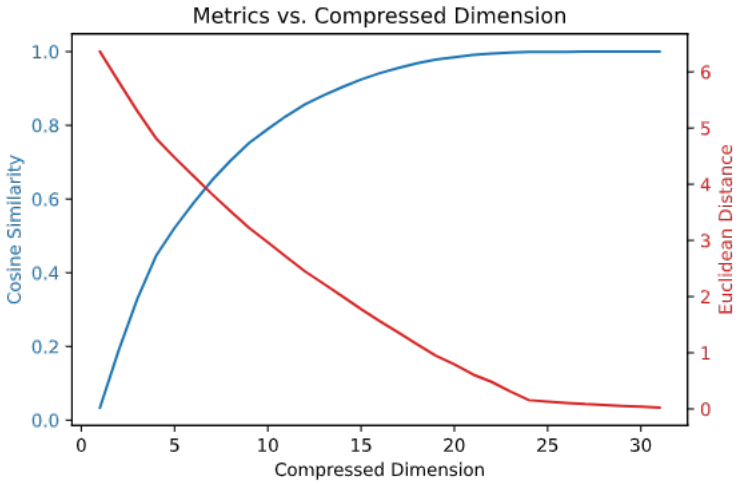

Figure 3.8: Metrics when compressing the second layer W_1 of our FC2 model from its original max bond dimension 32

Chapter 4

Compressing the MPO Net

4.1 Background

All results in this section are improvements built off of [1] where they were able to directly train neural networks with an MPO structure for each layer. The paper itself is mostly physics-oriented, but this concept has also been considered in computer science such as in [22] and [23].

Rather than training a single tensor in each layer like a standard neural network, the MPO Net trains several smaller tensors at once corresponding to the sites of the MPO. The bond dimension of each layer is fixed before training. The advantage of this is that we can train our model at a much lower bond dimension than that of the decomposition seen in Sec. 3.4. Note that when training, all bond dimensions are the same, however, our algorithm causes changing bond dimensions due to the compact SVD. The architecture can be seen in Fig. 4.1.

Note that from here on compression ratios will now consider both the originally trained MPO Net as well as the standard FC2 network in Sec. 3.4. Our algorithm will mostly be applied as an improvement to the results found [1] for the original MPO Net.

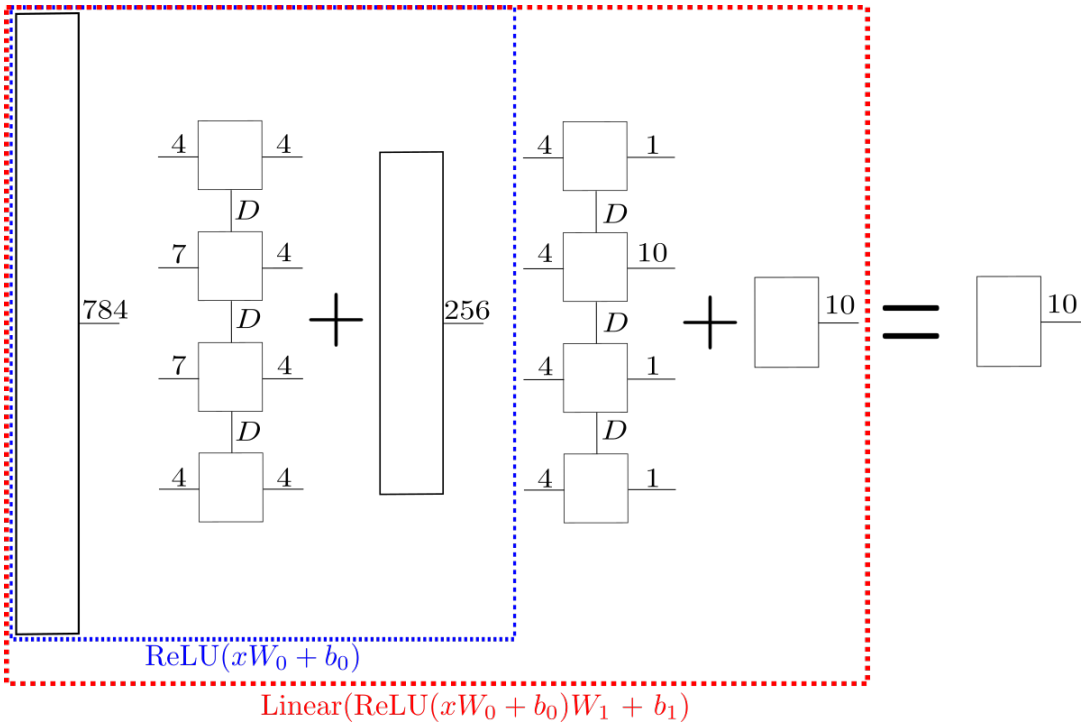

Figure 4.1: FC2 MPO Net used by [1]. Note that all bond dimensions are the same and the sites are not prime factorized as we did in Sec. 3.4.

4.2 Compression vs. Training at a Given Bond Dimension

Here we analyze the loss of accuracy when compressing a previously trained model with MPOs of dimension D to D'. In this test, all layers were trained with the same bond dimension. A noticeable result of training at different bond dimensions is that there are diminishing returns above a certain D such that any new parameters cause little change in the accuracy of the model.

We began with the hypothesis that since an MPO with a higher bond dimension has more parameters and theoretically holds more information, we could compress it from bond dimension D to D' while having a higher accuracy model than originally training it at D'. To test this, we train several models with bond dimensions from $D = 1$ to $D = 8$ for both layers. We then compress each layer individually in each model. The loss has been normalized according to the best loss overall such that the results are independent of the accuracy of the trained models.

We notice that the compressed MPO does not give better accuracy compared to originally training the model at that bond dimension. However the result shows that for dimensions relatively close to our original D, we can reproduce the accuracy. This could possibly be useful in that a trained model of a higher bond dimension also has direct access to lower bond dimension models, meaning the total parameters (and thus speed) and accuracy are adjustable. For some applications, this could be seen as a coarse-fine setting.

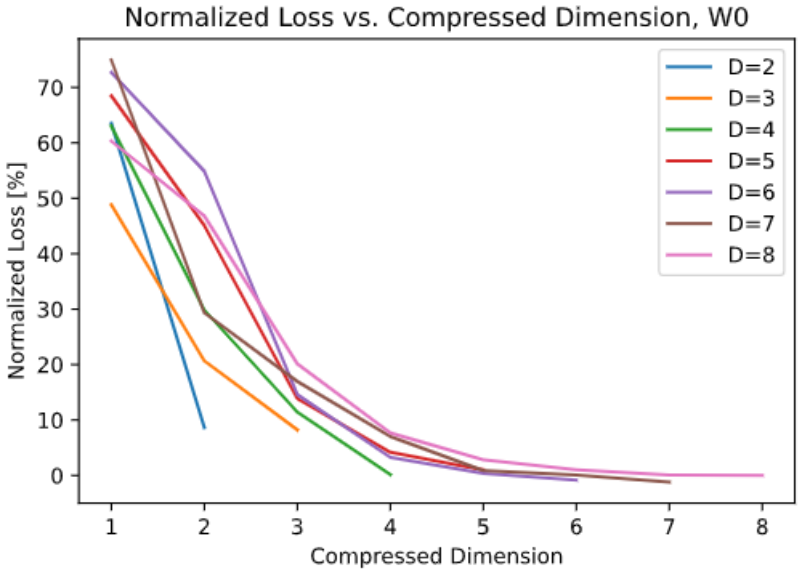

Figure 4.2: Comparison between trained and compressed models for the first layer of our FC2 model

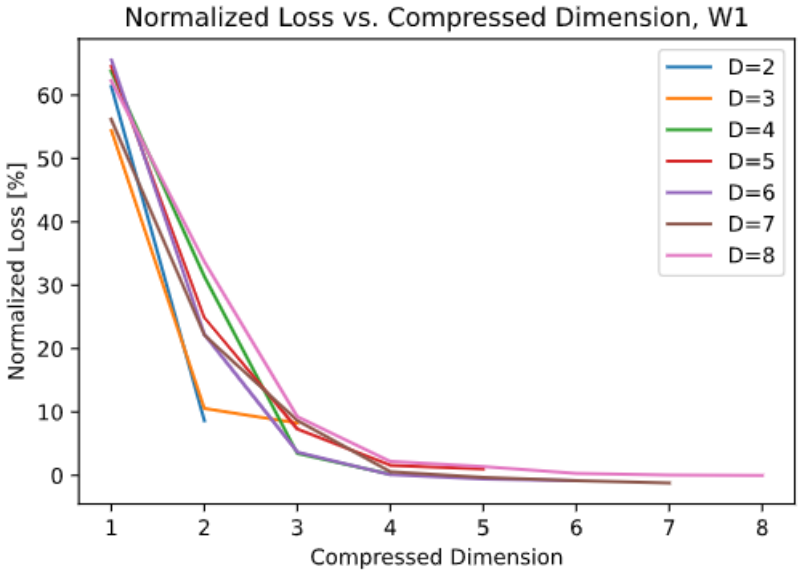

Figure 4.3: Comparison between trained and compressed models for the second layer of our FC2 model

4.3 Compressing Individual Layers

Here we trained a model with bond dimension $D = 16$ and compared the effects of compression on each individual layer. The threshold for the compression tests was $\epsilon = 10^{-8}$. The results are visible in Tables 4.1 and 4.2 as well as Fig. 4.4, 4.5, 4.6, and 4.7.

These results show a significant improvement over beginning with a standard neural network such as in Sec. 3.4 as we can reach a much smaller bond dimension. We can see in Table 4.2 and Fig. 4.4 that we can improve on the compression of the original MPO Net as well as significantly improve in comparison to a standard FC2 network as done in Sec. 3.4.

With less than 1 % loss of accuracy, we can compress our first layer to roughly 33 % of the original MPO Net parameters and 2.4 % of the standard FC2 network parameters.

Likewise, we can compress our second layer to roughly 10 % of the original MPO Net parameters and 43 % of the standard FC2 network parameters.

Be aware that in this test we held one layer constant while compressing the other. Therefore, we cannot compress both down to this value and expect the loss to be the same since the layers affect each other. Optimizing how much we compress each layer will be explored in the next section.

Layer	MaxBondDim	CosSim	EucDist	CompRatio(FC2 MPO)	CompRatio(FC2)
0	13	0.955	7.94	66.54 %	4.92 %
0	15	0.989	3.97	88.09 %	6.52 %
0	16	1.00	0	100 %	7.4 %
1	7	0.963	1.78	11.1 %	49.38 %
1	10	0.993	0.75	15.73 %	70.0 %
1	16	1.00	0	100 %	71.56 %

Table 4.1: Results when compressing each individual layer of the FC2 MPO Net. Thresholds were 0.95 and 0.99 cosine similarity. Compression ratio is relative to **the parameters of the corresponding layer** in the trained FC2 MPO Net and in the original FC2 neural network..

Layer	MaxBondDim	Loss	CompRatio(FC2 MPO)	CompRatio(FC2)
0	7	4.30 %	19.99 %	1.48 %
0	9	0.98 %	32.49 %	2.4 %
0	16	0 %	100 %	7.4 %
1	4	2.78 %	6.46 %	28.75 %
1	6	0.69 %	9.55 %	42.5 %
1	16	0 %	100 %	71.56 %

Table 4.2: Effect on loss when compressing each layer individually. Thresholds were bond dimensions causing a loss less than 1 % and 5 %. Compression ratio is relative to **the parameters of the corresponding layer** in the trained FC2 MPO Net and in the original FC2 neural network.

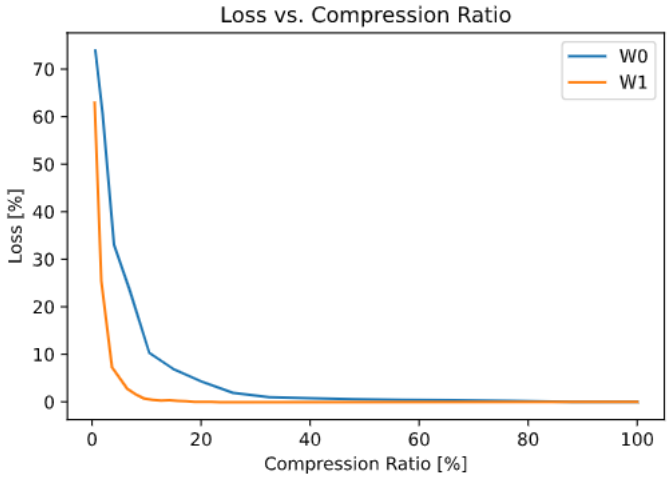

Figure 4.4: Loss for each layer by compression from original trained MPO Net. Compression ratio is relative to **the parameters of the corresponding layer** in the FC2 MPO Net

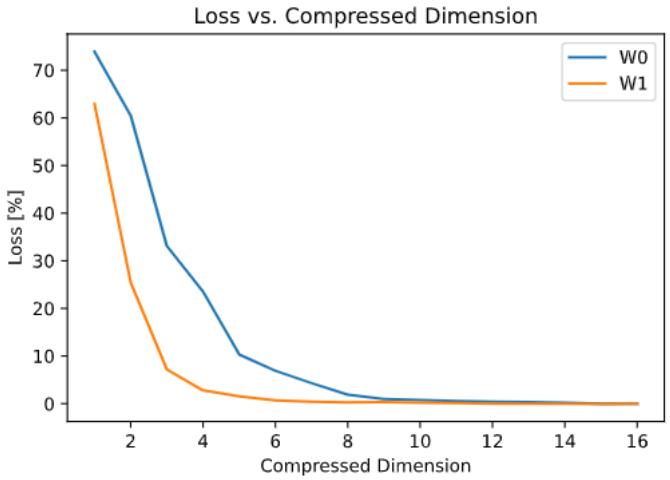

Figure 4.5: Loss for each layer during compression by max bond dimension

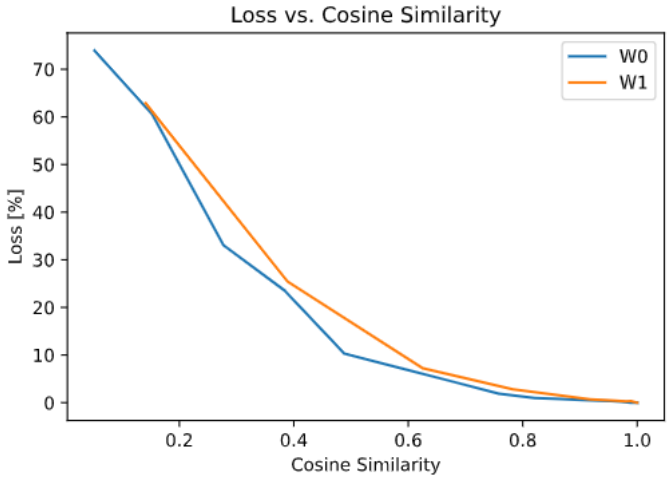

Figure 4.6: Loss relative to cosine similarity between a layer's MPO and its approximation

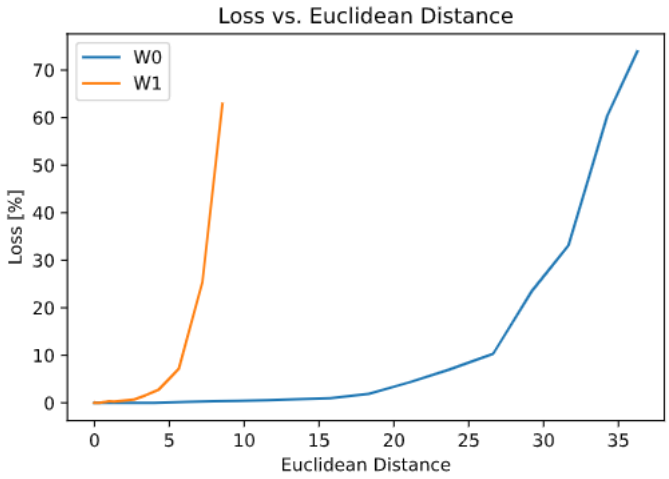

Figure 4.7: Loss relative to Euclidean distance between a layer's MPO and its approximation

Chapter 5

Algorithm for Optimizing Bond Dimensions

5.1 Graphical Model

We notice that each layer in the model has a different effect on the accuracy as it is compressed. The consequence of this is that rather than reducing the bond dimension of all layers equally, we can find an optimal solution which minimizes the number of parameters needed while keeping the loss under a given threshold.

In order to do this, we start with a trained model with N layers where all layers have bond dimension D. We define the total parameters of this model as ω and the loss as L defined as the difference in accuracy in each configuration compared to the initial accuracy $\alpha_0 - \alpha$. We then need to search N-dimensional space in order to find a solution with minimum loss and minimum parameters. In order to avoid local solutions, we have applied two different search methods for this. Note that in this thesis, we have only applied this algorithm to the 2-layer FC2 network, however it should generalize to deeper networks based on the theoretical background as well as preliminary results from applying it to other networks.

To begin, we will consider all the possible dimension combinations as a graph. First, we begin with a graph where all nodes can be labeled by the bond dimensions of the N layers i.e. $(D_0, \ldots, D_i, \ldots, D_N)$. We begin at an initial node with our starting max bond dimensions $(D_0, \ldots, D_i, \ldots, D_N)$. We wish to compress only one layer at a time in order to search step-by-step for a solution. In an N-dimensional space, this means we have N possible steps at each node corresponding to N possible layers to compress. At each step, we can consider the possible compressions for each layer $D_i \to D_i - 1$. This is visualized in Fig. 5.1.

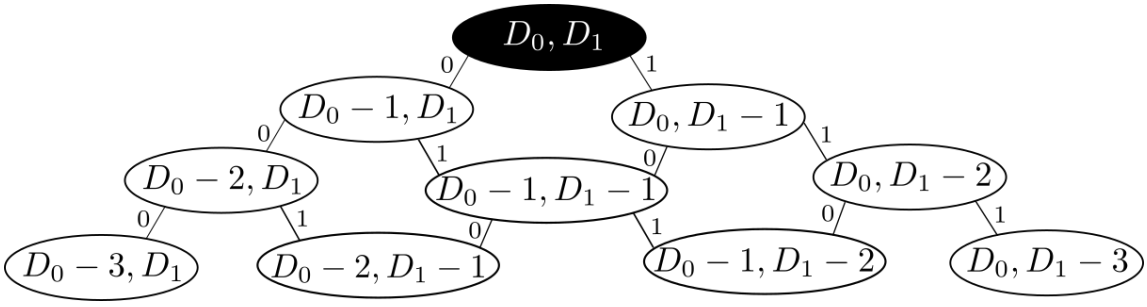

Figure 5.1: Sample section of a graph for a 2 layer network when compressing each layer individually. 0 means to compress layer 0 and 1 means to compress layer 1.

5.2 Depth-First Search

Here we will combine two concepts into a practical method for searching our higher dimensional space (i.e. deep models). We will use a depth-first search in that we can only access nodes below our current node and cannot traverse the graph horizontally [17]. We then combine it with a sort of gradient descent in which our decision along which path to take is directly related to some optimization value.

At each node, we calculate the new loss value and new number of parameters in each possible direction such that we have the term $\frac{\Delta L}{\Delta \omega}$ for each layer. We then update the layer whose compression minimizes this term such that we have the least amount of loss and greatest amount of parameter compression. For example, if the i^{th} layer minimizes it, our first step away from the initial node will be to $(D_0, \ldots, D_i - 1, \ldots, D_N)$. We have reached a minimum once we are at a node where all possible compressions lead to $L > \epsilon$ where ϵ is a pre-defined threshold for how much accuracy loss we are willing to sacrifice during compression. This algorithm will return a single solution, but it may be a local solution rather than the global one. This is visualized in Fig. 5.2.

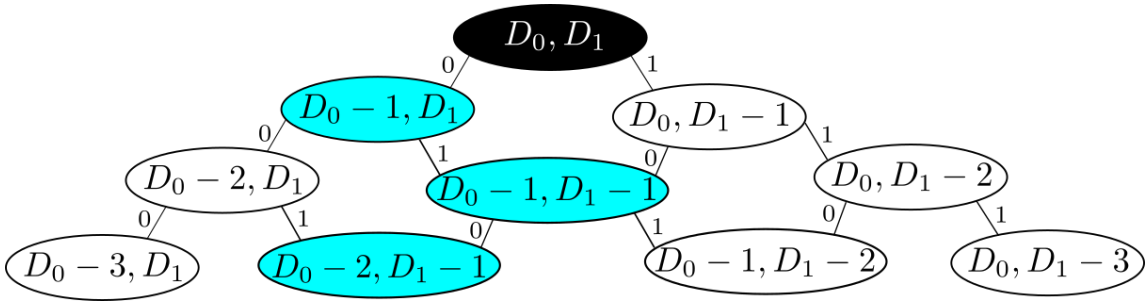

Figure 5.2: Example path for a depth-first search through our compression graph. Any direction is possible and is chosen by our term $\frac{\Delta L}{\Delta \omega}$ for each compression.

5.3 Permutation Search

Another method is to only consider one layer at a time where we compress it until we reach our condition $L > \epsilon$. Once we reach this, we simply move to another layer and do the same. We continue this with as many layers as possible until we reach a local solution. In some cases, this method does give a better solution than the depth-first search. However, there are $N!$ total permutations in which we can move between layers. The combination of these two methods gives us access to a possible $N! + 1$ solutions (assuming no two solutions are the same). The feasibility of searching for all of these and checking for a global solution is dependent on how deep the network is i.e. the value of N and the resources available. This is visualized in Fig. 5.3.

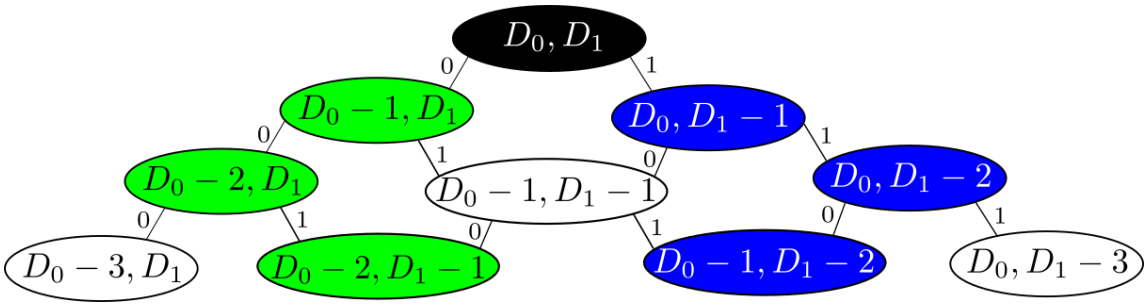

Figure 5.3: Two possible paths through our graph using a permutation search. One layer is compressed completely until a certain threshold is reached, then we switch to the other layer.

5.4 Application: Optimal Solution for FC2 MPO Net

For a smaller network such as FC2, we can visualize these search algorithms to build intuition for applying it to larger networks. In this example, we were able to calculate the full graph for all possible combinations of max bond dimension by layer. This is used purely as an example since higher-dimensional space will first not be easy to visualize and second we cannot reasonably calculate all the nodes beforehand.

Here we trained a network with $D = 8$ for both layers then used the depth-first search and permutation searches to find the combination of compressed bond dimensions with loss less than two thresholds, 1 % and 5 %. The overall graph of possible combinations can be found in Fig. 5.4. The results are found in Table 5.1, Fig. 5.5, Fig. 5.6.

Threshold	Dim0	Dim1	Loss	CompRatio(FC2 MPO)	CompRatio(FC2)
5 %	4	6	4.81 %	28.33 %	0.87 %
1 % (Optimal)	6	7	0.99 %	49.10%	1.51%
1 %	7	5	0.94 %	57.69%	1.77%
-	8	8	0 %	100 %	3.07 %

Table 5.1: Results for optimizing the bond dimensions of the FC2 MPO network trained at a max bond dimension 8. The values included are the 2 results from our algorithm as well as the optimal solution for the 1 % threshold (see Fig. 5.5 and Fig. 5.6). Compression ratio is relative to **the total network parameters** in the trained FC2 MPO Net and in the original FC2 neural network.

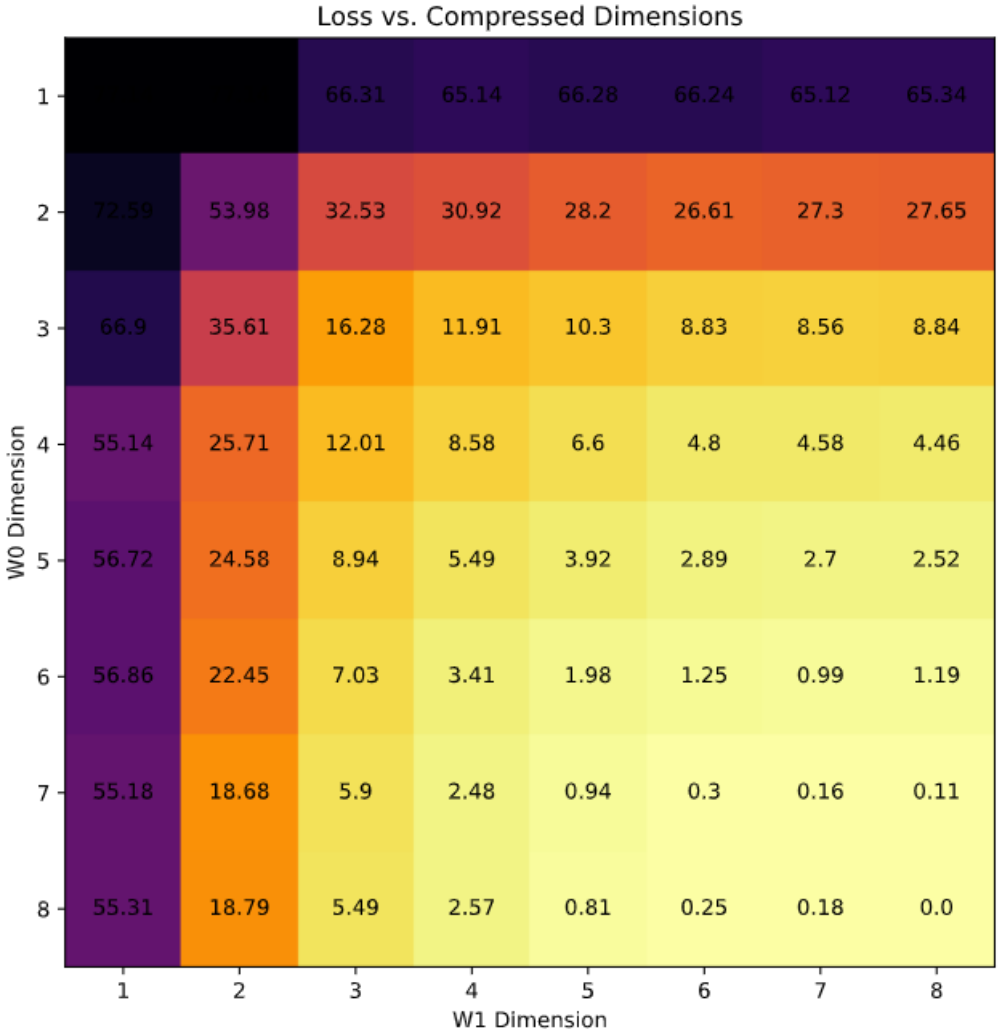

Figure 5.4: Heatmap of all layer combinations and their loss [%] compared to the original model

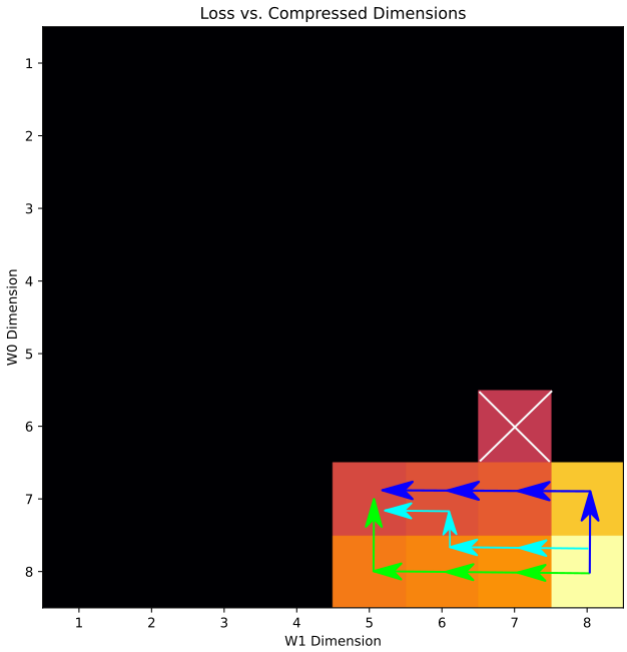

Figure 5.5: Paths through the graph in Fig. 5.4 corresponding to the depth-first search (Cyan) and permutation searches (Green and Blue). Threshold was set to 1 % and any nodes above this threshold are blacked out. The white X corresponds to the global solution. No path found the global solution.

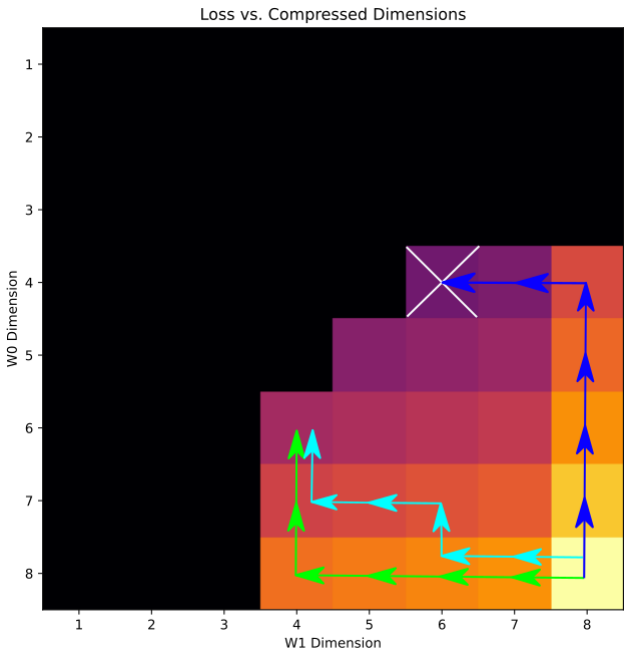

Figure 5.6: Paths through the graph in Fig. 5.4 corresponding to the depth-first search (Cyan) and permutation searches (Green and Blue). Threshold was set to 5 % and any nodes above this threshold are blacked out. The white X corresponds to the global solution. Optimizing layer 0 and then layer 1 found the global solution.

5.5 Application: Optimal Solution for Standard FC2

Now that we have built intuition into how this algorithm works, we will apply it to our standard FC2 network from Sec. 3.4.2. Since the bond dimensions are so high, we cannot reasonably graph it into a heatmap as done before, but we will still apply the depth-first search algorithm to it. During compression we also see two interesting things:

1. The accuracy increases up to roughly 0.1 % at several nodes. The compression is likely fixing overfitting to some degree.
2. There are several combinations with 0 % loss that we saw when compressing for a higher threshold. However, our algorithm could not reach them because they are "hidden" behind nodes where the loss is above the 0 % threshold. A random walk or some form of exploration in addition to our original algorithm could help find these points.

The results of this are found in Table 5.2.

Threshold	Dim0	Dim1	Loss	CompressionRatio
5 %	28	26	4.81 %	14.06 %
1 %	64	29	0.99 %	30.10 %
0.1 %	166	24	0.07 %	74.78 %
0 %	169	24	-0.02 %	76.11 %

Table 5.2: Results for optimizing the bond dimensions of the standard FC2 network after being decomposed into MPO layers. Compression ratio is relative to **the total network parameters** in the original standard FC2 neural network.

Chapter 6

Future Outlook

6.1 Qubit-Inspired Feature Vectors

As noted in this thesis, tensor networks benefit from more sites such that we require smaller tensors. In order to truly maximize the number of sites, we could modify our input such that each value is a vector in itself. For example, the MNIST dataset has pixel values between 0-255 which could be normalized to 0-1. Then each value is a (2×1) row vector or ket that is a superposition of the values 0 and 1. Each pixel is then applied to its own site with physical dimension 2, expanding our total sites from 6 as used in this thesis to 784. These methods and related ones are discussed in [24] [25] and [26]. This could also see a significant boost in efficiency as it better utilizes the properties of tensor networks.

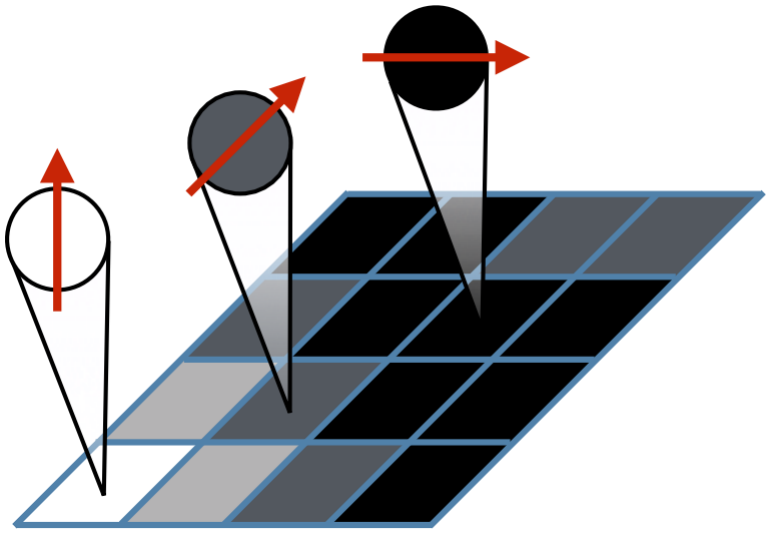

Figure 6.1: Qubit-inspired feature vectors where each pixel is mapped to a 2-dimensional superposition between white and black [26]

6.2 Speed-up of Neural Networks

The MPO Net is trained such that an entire input vector is used to train each site of the MPO individually. Note that if this is modified to use feature vectors as described, we could take advantage of the smaller number of parameters during application. This means that a given input vector is split into smaller parts to train each site by using tensor contraction. A neural

network which utilizes tensor network contractions during training would not only require less memory, but would also see a significant speed boost. If the compression is already on the magnitude of a few percent of a typical network, the ability to do quicker calculations on smaller tensors could be extremely useful.

If we were to train a neural network as done in [26], utilize the properties from [1], and compress it using our method, we will essentially have a neural network built entirely from a single algorithm inspired by the Density Matrix Renormalization Group.

Two of these possible architectures are seen in Fig. 6.2 and 6.3.

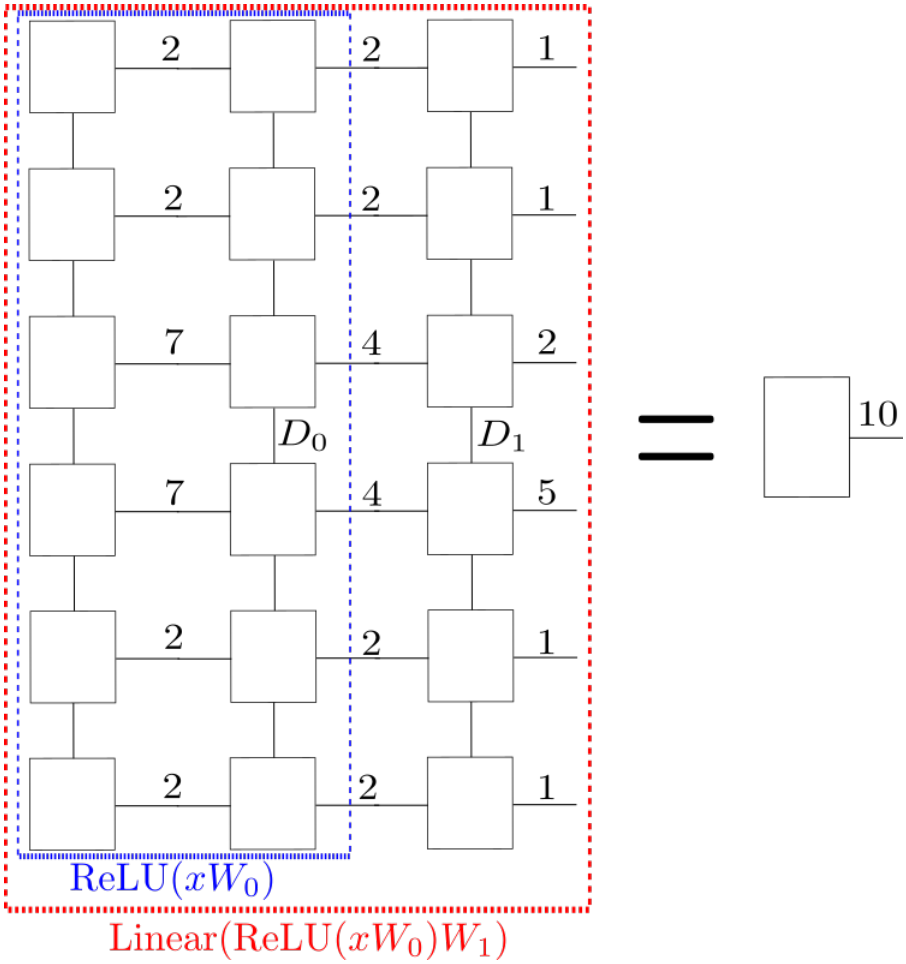

Figure 6.2: An ideal architecture for the FC2 model where the input data is now an MPS rather than a length 784 vector. This allows us to utilize the speed of tensor contractions, but the input data needs to be put in this form before training. Biases are not included.

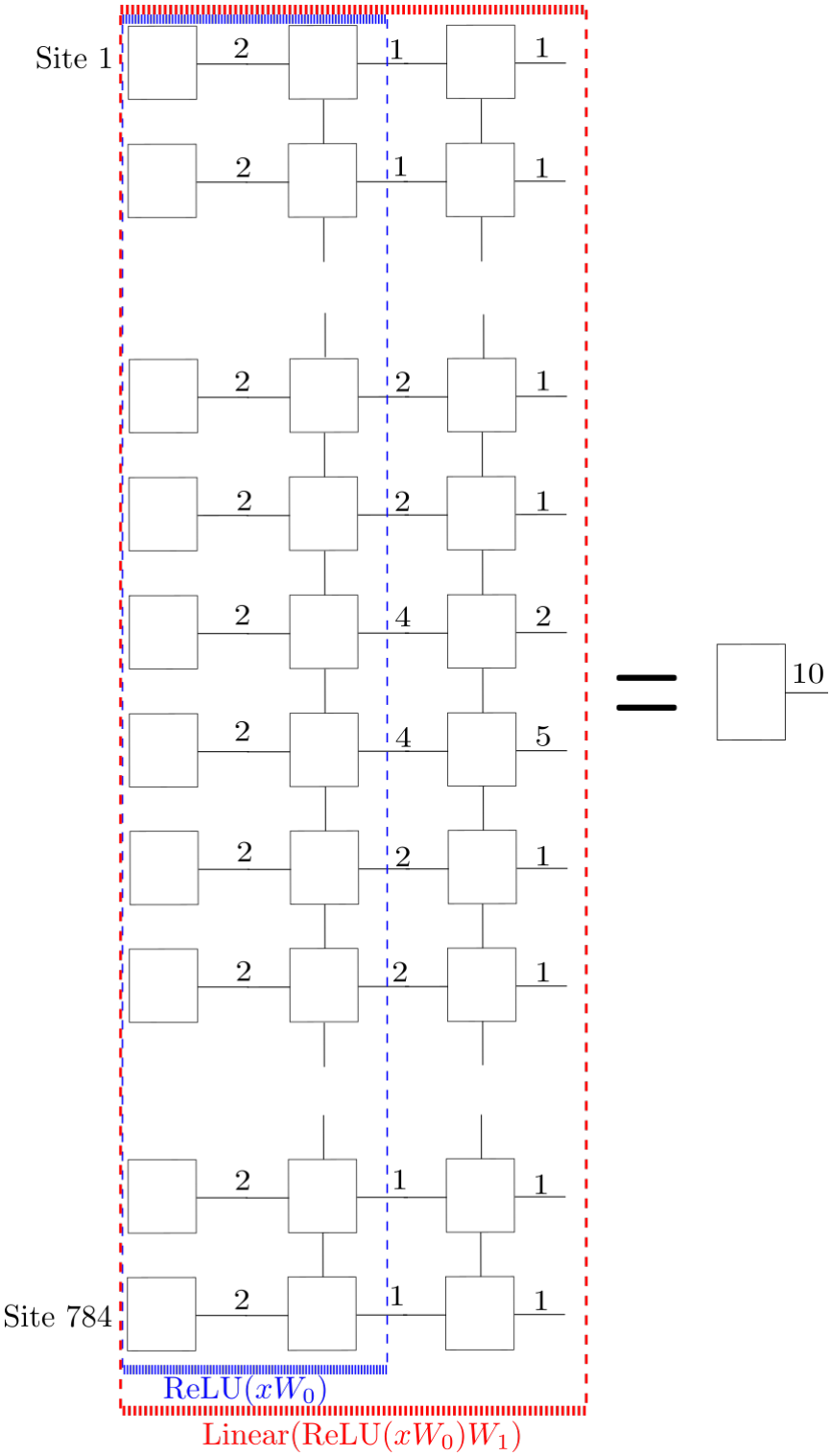

Figure 6.3: An ideal architecture for the FC2 model where the input data is now a qubit for each pixel rather than a length 784 vector. This allows us to utilize the speed of tensor contractions as well as the advantage of having more sites, but the input data needs to be put in this form before training. Biases are not included.

6.3 Neural Networks as Finite Automata

As described previously, an MPS/MPO can be described as a finite automaton. We have also shown that neural networks can be represented by an MPS/MPO. This opens the possibility that we can describe a neural network as a finite automaton where our compression algorithm essentially reduces the total number of states and transitions necessary at each layer. This could see huge benefits in applying a mathematically rigorous understanding to machine learning.

Bibliography

[1] Z.-F. Gao, S. Cheng, R.-Q. He, Z. Xie, H.-H. Zhao, Z.-Y. Lu, and T. Xiang. Compressing deep neural networks by matrix product operators. *Physical Review Research*, 2(2): 023300, 2020.

[2] J. Biamonte and V. Bergholm. Tensor networks in a nutshell. *arXiv preprint arXiv:1708.00006*, 2017.

[3] J. von Delft. Lecture notes in tensor networks. Ludwig-Maximilians Universität München, July 2020.

[4] U. Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of physics*, 326(1):96–192, 2011.

[5] B. Pirvu, V. Murg, J. I. Cirac, and F. Verstraete. Matrix product operator representations. *New Journal of Physics*, 12(2):025012, 2010.

[6] M. B. Hastings. An area law for one-dimensional quantum systems. *Journal of Statistical Mechanics: Theory and Experiment*, 2007(08):P08024, 2007.

[7] J. von Neumann. *Mathematische Grundlagen der Quantenmechanik*. Springer-Verlag, 1932. doi: 10.1007/978-3-642-61409-5.

[8] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester. *A Modern Introduction to Probability and Statistics: Understanding why and how*. Springer Science & Business Media, 2005.

[9] R. Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014.

[10] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to automata theory, languages, and computation, 2nd edition*. ACM SIGACT News, 2001. doi: 10.1145/568438.568455.

[11] G. M. Crosswhite and D. Bacon. Finite automata for caching in matrix product algorithms. *Physical Review A*, 78(1):012356, 2008.

[12] W. Dür, G. Vidal, and J. I. Cirac. Three qubits can be entangled in two inequivalent ways. *Physical Review A*, 62(6):062314, 2000.

[13] N. Lord, G. H. Golub, and C. F. V. Loan. *Matrix Computations*. The Mathematical Gazette, 1999. doi: 10.2307/3621013.

[14] G. Sidorov, A. Gelbukh, H. Gómez-Adorno, and D. Pinto. Soft similarity and soft cosine measure: Similarity of features in vector space model. *Computación y Sistemas*, 18(3): 491–504, 2014.

[15] A. Singhal et al. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[16] K. Huang. *Statistical Mechanics, 2nd Edition.* Wiley, 1987. ISBN 0471815187. doi: citeulike-article-id:712998.

[17] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach Third Edition.* Pearson, 2010. ISBN 9780136042594. doi: 10.1017/S0269888900007724.

[18] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, 2000.

[19] Y. LeCun, C. Cortes, and C. Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[20] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[21] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[22] V. Khrulkov, O. Hrinchuk, L. Mirvakhabova, and I. Oseledets. Tensorized embedding layers for efficient model compression. *arXiv preprint arXiv:1901.10787*, 2019.

[23] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov. Tensorizing neural networks. In *Advances in neural information processing systems*, pages 442–450, 2015.

[24] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

[25] I. Glasser, N. Pancotti, and J. I. Cirac. From probabilistic graphical models to generalized tensor networks for supervised learning. *IEEE Access*, 8:68169–68182, 2020.

[26] E. Stoudenmire and D. J. Schwab. Supervised learning with tensor networks. In *Advances in Neural Information Processing Systems*, pages 4799–4807, 2016.

Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Arbeit selbständig verfasst zu haben und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel benutzt zu haben.

Aaron Lee Sander München, 11. September 2020