Dynamic Topology Reconfiguration of Boltzmann Machines on Quantum Annealers

by

Jeremy Liu

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Science)

August 2020

Acknowledgements

I would like to thank Ke-Thia Yao for working closely with me these past few years, for securing helpful resources, and for guiding me in research. I would also like to thank Robert Lucas for his valuable feedback and suggestions for focusing the dissertation.

I also thank my committee members for the collaborative work we completed and for taking the the time to review this work.

Table of Contents

List Of Figures

Abstract

Boltzmann machines have useful roles in deep learning applications, such as modeling data, initializing weights for other types of networks, or extracting efficient representations from high-dimensional data. However, practical deployments of Boltzmann machines feature crippled topologies that exclude looping connectivity since such connectivity creates complex distributions that are difficult to sample from. We have used an open-system adiabatic quantum annealer to sample from complex distributions and implement Boltzmann machines with looping connectivity. Further, we have created policies mapping Boltzmann machine variables to the quantum bits of an annealer. These policies, based on correlation and entropy metrics, dynamically reconfigure the topology of Boltzmann machines during training and improve performance.

Chapter 1

Introduction

In this work we are concerned with the development of an information-theoretic approach for reconfiguring Boltzmann machine (BM) connectivity to improve performance. Our examination of BMs is tied to its realization on adiabatic quantum annealing hardware. The D-Wave open system adiabatic quantum annealer has a chimera topology that limits the number of quantum bits (qubits) and connections between its qubits, creating a series of variable-to-qubit mapping decisions that have performance implications. Our implementation of Boltzmann machines on quantum annealers differs from those typically used by other researchers. Some choose to implement full Boltzmann machines that represent both visible and hidden units on the annealer [3, 2] while some discuss connectivity topologies but do not implement them on hardware [13, 16]. The Boltzmann machines we implement on quantum annealers represent only the hidden units and their intra-layer connections. As such, deciding how to map BM variables to annealer qubits is equivalent to choosing a BM connectivity topology because we can only represent a small subset of all possible connections. We want to make variable-to-qubit mapping decisions such that our eventual choice of topology generates better results than a "default" or random topology would, improving overall BM training results. Our results show that mapping decisions designed to lower entropy in BM topology leads to better outcomes.

Here we offer motivation and context for why we choose to study this topic. We start with the aphorism that machine learning is informed by both algorithms and hardware. Algorithms tell us how to solve a particular problem within the constraints defined by our choice of hardware. Hardware development, itself, may sometimes be driven by promising theoretical algorithms that just need the correct equipment for implementation. For us the latter includes research into creating hardware suitable for implementing neuromorphic networks or general quantum computers. Neuromorphic computing research draws inspiration from biological brains that feature highly connected networks and produce results that are still beyond current machine learning approaches, the hope being that mimicry of these highly connected structures can enable similarly effective computations or problem solving. In the same vein, quantum algorithms have already been proven to be superior to classical algorithms in certain areas but have yet to find quantum hardware capable of implementing them[1].

Our work focuses on Boltzmann machines, with an eye towards how we can alter them for more efficient usage assuming current and future technological developments. In recent years we have had access to D-Wave's adiabatic quantum annealers, and our primary machine (called DW2x) is located at the USC-Lockheed Martin Quantum Computing Center (QCC). Also called adiabatic quantum optimization (AQO[2]) devices, they are capable of solving specific optimization problems. We can recast our BMs as these sort of optimization problems and use AQO machinery to implement them, but this comes with its own set of challenges due to the physical constraints of the hardware. Our work considers these constraints, how best to work around them, and how we can apply learned principles to future generations of similar hardware. However, one point we would like to stress is that the applicability of our work is not restricted to a specific set

[1]There has been a notable recent finding from Google showing evidence of quantum supremacy [4]. Although the specific application found within the paper is not generally useful, it is encouraging that quantum supremacy, in some form, has been empirically observed.

[2]We use several different terms to describe computations involving quantum effects. These include AQO, adiabatic quantum computation (AQC), quantum-enabled optimization (QEO), or quantum annealing as a shorthand for open-system adiabatic quantum annealing. The field does not yet have a broad consensus for standard terminology. For this document we generally use the terms quantum annealing or open-system adiabatic quantum annealing.

of hardware, either quantum or classical. The annealer we use can be simulated using classical methods, and our findings can also be easily applied to BMs implemented on classical machines. We mention quantum annealing hardware because it is novel and because it defines the restrictions of the problem we solve.

To summarize our work going forward: **we use an entropy-based metric to dynamically reassign BM variables to qubits, subject to quantum annealing hardware restrictions, in order to improve data modeling results**.

We have mentioned Boltzmann machines several times, and we now provide a short explanation of what they are and why we are interested in them. A Boltzmann machine is a generative model of data composed of many connected computational units. The connections between units are called weights, and each weight is a separate parameter that can be adjusted during training. We call BMs generative because we can sample a trained network to create a new data distribution that should be very similar to the data it was trained on. While BMs can sometimes be complicated in structure, with many layers and connections in a deep configuration, the BMs we are concerned with are overall very simple in structure and are not necessarily neural networks, though they have been used as the building blocks of more complex types of networks such as autoencoders [27]. In this sense they can be seen as a sort of fundamental network whose properties are worth exploring. We chose to focus on BMs because they are generative, structurally simple, and relevant to other types of networks.

We find quantum annealing hardware interesting because it is a new technology that has not seen widespread usage, yet continues to improve with successive generations. Furthermore, our previously stated interest in BMs carries over to this area because, with some minor tweaks, we can use such hardware to implement BMs. In general, simulated annealing is a method for optimizing functions. Quantum annealing's theoretical advantage over classical simulated annealing is found in quantum tunneling phenomena, where in certain cases quantum annealing can find the global optimum of some function more quickly than simulated annealing could [51]. We use D-Wave

System's quantum annealing hardware in our work and accept all the constraints that come with it.

Questions of quantum supremacy and quantum advantage naturally arise when working with quantum hardware. Quantum supremacy is the idea that quantum algorithms can solve problems classical algorithms would be fundamentally unable to tackle; quantum advantage is a weaker version of supremacy where quantum algorithms solve problems faster than classical algorithms. Whether these concepts can be demonstrated empirically are still open questions in research and are not the focus of this work. We do not seek to prove or disprove any sort of supremacy or advantage; rather, we are content to work with the quantum annealing hardware available to us and to consider its other useful properties and limitations.

In particular we find D-Wave's quantum annealer appealing because it can be used to sample from complicated distributions, the sort that Boltzmann machines are fundamentally built from and which cannot be neatly analyzed. Whereas classical methods and hardware typically use Markov chain Monte Carlo (MCMC) methods [28, 42, 14, 21] to draw samples from the Boltzmann distributions we define, the same distribution defined on annealing hardware can be directly sampled. This is a capability that opens up new possibilities in designing BM network connectivity. Taking into consideration the cost of MCMC methods, and absent the ability to directly sample from a distribution, BMs have typically featured restricted and simplified connectivity patterns that make tractable the computations necessary for training. The availability of a quantum annealer and its ability to directly draw samples allows us to rethink how to define BM connectivity and to reexamine previous restrictions. But as mentioned before, this comes with its own set of challenges because the hardware is, itself, restricted in topology.

The first problem is that the annealer only has 1098 working qubits. The second problem is that the qubits are realized in a planar, 2-dimensional space and thus cannot feasibly feature unlimited connections among themselves. Together, these two factors severely restrict the size of problems we can place on the machine. As we will discuss later, this poses problems for

straightforward implementation of BMs on annealing hardware. One way to fit a dense graph problem into a sparser space, a situation we would face in a straightforward BM implementation, is called embedding. If done correctly, embedding would allow us to approximate denser problems with sparser ones, such as the annealer's chimera topology, and we could proceed to perform non-trivial experiments. Unfortunately, subgraph isomorphism is an NP-complete problem, which makes utilizing it to fit dense problems on a sparse annealer too difficult. Our implementation avoids the graph embedding problem by opting to simply use BMs with the default interconnection topology imposed by the hardware. Instead of trying to find a topology that approximates some target problem, we settle for using whatever topology happens to be available.

Chapter 2

Boltzmann Machines

In this chapter we will examine Boltzmann machines and how we have modified them for implementation on a quantum annealer. In Section 2.1, we introduce Boltzmann machines, define the equations that govern them, show how they are trained, and explain how dense topologies create tractability problems. In Section 2.2, we give several examples, including some from our own work, of how BMs are used in practice. Having shown the typical definition and usage of BMs, we then go to Section 2.3, which introduces the limited Boltzmann machine, our expanded version of Boltzmann machines that uses a richer topology than commonly used Boltzmann machines. We explain how limited Boltzmann machines differ from restricted Boltzmann machines, and we also briefly discuss how a quantum annealer can benefit this new formulation.

2.1 Basics

There exist many methods for explaining data distributions. Parameterized Gaussian distributions spring to mind, as do other functions like exponential, or binomial, and all their myriad extensions for multiple dimensions. A slight step up in terms of complexity includes constructions like Bayesian networks, where we begin to impose a sort of order upon the our data by creating latent variables to explain the observable ones. This linkage of variables to each other is a powerful tool

that forms the basis for graphical models of data. Boltzmann machines can be placed within the context of graphical models as a method for explaining and emulating an observed set of data.

Boltzmann machines can be visualized as a set of interlinked nodes divided into two subsets. One subset of nodes is called the visible layer, and the other subset is naturally called the hidden layer. Units of the visible layer represent the data we observe and wish to model while units of the hidden layer represent latent variables. These hidden units/variables influence the distribution of states within our visible units but are not directly observable in the original data set.

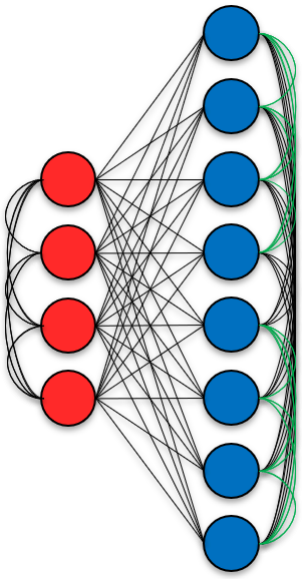


Figure 2.1: A Boltzmann machine that features full connectivity between all units. The units are represented as circles. The visible layer is composed of the red circles and the hidden layer is composed of the blue circles. Connections between units are bilateral, and we impose no restrictions on connectivity between the units in this case. For N units, we have $\frac{n(n-1)}{2} = \mathcal{O}(N^2)$ connections.

Figure 2.1 offers a visualization of a Boltzmann machine. It is essentially a type of undirected Bayesian network, a Markov random field, in which many different possible network states exist. Even in the binary case where units can be either on or off, which we assume throughout this work, we have $\mathcal{O}(2^N)$ possible network states given N units in the network. Given so many states, it is too daunting a task to describe each network configuration individually. Instead we must parameterize our model by assigning weights to the connections between each unit. We can call

Boltzmann machines energy-based models because our choice of parameterization is a function that assigns an energy to each network configuration:

$$E(x) = -(\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i) \tag{2.1}$$

Equation 2.1 has the connections between BM units represented as the matrix w where w_{ij} is the bilateral connection between units i and j (whose states are noted as s_i and s_j). Additionally, we have a bias term θ_i associated with the i-th unit that contributes to the energy of a network configuration so long as the i-th unit is "on." Each network configuration $x = (s_0, s_1, \ldots s_N)$ can be assigned an energy in this way, where lower energy states have a higher probability of appearing when we sample the network. We are particularly interested in the lowest energy network state we call the "ground" state. Viewed from another perspective, this is simply the definition of an Ising spin glass problem from physics where we have two-body interactions (our w_{ij}) and a transverse field (our θ_i).

As mentioned, the energy of a given network configuration is proportional to its probability of appearing when we sample the Boltzmann machine. This is also where BMs derive their name from because they follow a Boltzmann distribution:

$$P(v, h) = \frac{\sum_h e^{-E(v,h)}}{Z} \tag{2.2}$$

$$Z = \sum_v \sum_h e^{-E(v,h)} \tag{2.3}$$

Here our notation differs slightly from Eq. 2.1 in that we separate network state vector x into (v, h) where v represents the visible unit states and h represents the hidden unit states. This is done for convenience because we will often be referring to the visible and hidden units separately.

The denominator of Eq. 2.2 poses a particular problem for us. It is known as a partition function, Z, and is difficult to calculate because we must sum over all possible network states, which we previously noted as an exponential ($\mathcal{O}(2^N)$) endeavor. This makes sampling from such a distribution too difficult for practical purposes, so researchers instead developed restricted connectivity topologies [1] and took advantage of conditional independence using a fast, approximate training algorithm known as contrastive divergence [25].

Training a Boltzmann machine means adjusting the w_{ij} and θ_i such that, when we take random samples from the network, our distribution of states should appear as similar as possible to the data set we trained upon. We want to maximize the probability of producing the training data set, recalling that our visible units v represent that data. Eq. 2.1 is the expression we use, noting that we are unable to observe the hidden unit states of the training data so we sum over all possibilities. The contrastive divergence (CD) algorithm takes the derivative of the logarithm of Eq. 2.1 with regards to our w_{ij}:

$$\frac{\partial \log P(v)}{\partial w_{ij}} = \langle s_i s_j \rangle_{data} - \langle s_i s_j \rangle_{model} \tag{2.4}$$

The angular brackets $\langle \cdot \rangle$ are expected values according to either our data or the BM model, which we will explain later. For now we observe that we seem no better off than before since we cannot use exact inference to calculate the frequency of ($s_i = 1$) or ($s_j = 1$) we expect to see because Eq. 2.1 does not factorize. We still have to sum over all possibilities or use methods such as MCMC to find our terms. Therefore, researchers decided to make a compromise in BM connectivity to enable fast training by enforcing conditional independence between visible units and hidden units. This was achieved by forbidding any connections among units within the same layer.

Figure 2.2 gives a visualization of the resulting restricted Boltzmann machine (RBM). It appears much simplified in contrast to the full BM shown in Figure 2.1. In exchange, training

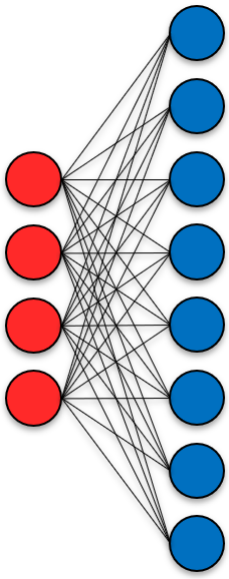


Figure 2.2: A restricted Boltzmann machine only allows connections from visible units to hidden units. This creates a $M \times N$ bipartite structure where we have $\mathcal{O}(MN)$ connections.

and sampling becomes tractable due to the conditional independence that bipartite connectivity introduces. To see why conditional independence is so important, consider the fully connected BM case. Should we want to calculate the probability that hidden unit i is "on," then we write $P(h_i = 1) = P(h_i = 1|v, h_{-i})$, where the $-i$ subscript means "all units except the i-th unit. This is an exponentially large space we have to cover. On the other hand, if we have the RBM's bipartite connectivity, we can reduce greatly this expression's complexity through factorization: $P(h_i = 1) = \prod_{j=1}^{n} P(h_i = 1|v_j)$. This is possible precisely because no hidden units are connected to any other hidden units. The companion case where we consider the probability some visible unit i is on follows similarly. On top of all this, we note that we start with a full array of visible unit states, which allows us to calculate all the hidden unit probabilities in parallel, which would let us calculate new visible unit probabilities, etc. Eq. 2.4 is simplified, becoming:

$$\frac{\partial \log P(v)}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \tag{2.5}$$

Given this new simplification, it is now appropriate to explain what $\langle\cdot\rangle_{data}$ and $\langle\cdot\rangle_{model}$ mean. $\langle\cdot\rangle_{data}$ are the expected values of v_i and h_j given that we clamp the visible states of the BM to the training data we observe. $\langle\cdot\rangle_{model}$ are the expected values of v_i and h_j when we do not clamp any values, instead performing the chain of visible states, then hidden states calculations mentioned in the previous paragraph. In contrastive divergence these two values are derived from what are called the positive and negative phases of the training process. The positive phase is straightforward as described described earlier: we clamp visible units to the states of observed data. This induces some distribution of hidden unit states. The negative phase then begins with this hidden unit state distribution and performs chain calculations, calculating a visible unit distribution, then a hidden unit distribution, etc. Typically we stop after just one iteration of this chain because it has been empirically observed to work well enough[1].

2.2 Usage

These simplified RBMs are often used as part of a pre-training procedure that chooses better starting parameters for more complex networks. Notably, they are found as part of autoencoders [27, 26]. Autoencoders are a type of feed-forward network that finds informative encodings of data. They are initially built by sequentially stacking RBMs together, then fine-tuned using backpropagation.

Figure 2.3 shows an autoencoder network. Its goal is to produce efficient encodings of data that lose minimal information. As mentioned, autoencoders are built from stacking RBMs together, which we can see by treating each adjacent pair of layers as a BM. They are accordingly trained as BMs normally would be before converting the entire stack of RBMs into a feed-forward network that is fine-tuned using backpropagation.

[1]This is where we call contrastive divergence an approximate training method. The proper way to calculate $\langle\cdot\rangle_{model}$ would be to sample from the network starting from a random configuration of visible unit states and forming a long sequence of alternating Gibbs sampling between the visible and hidden unit states.

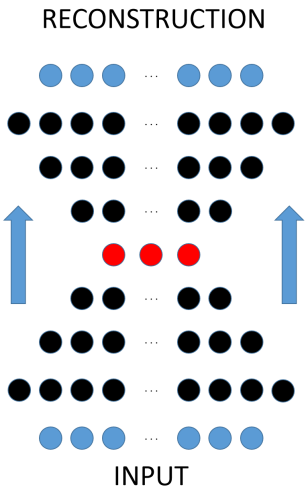
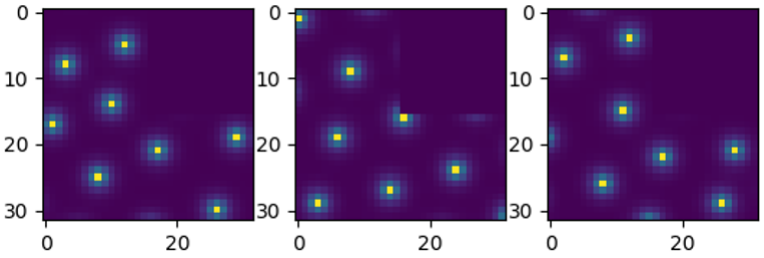


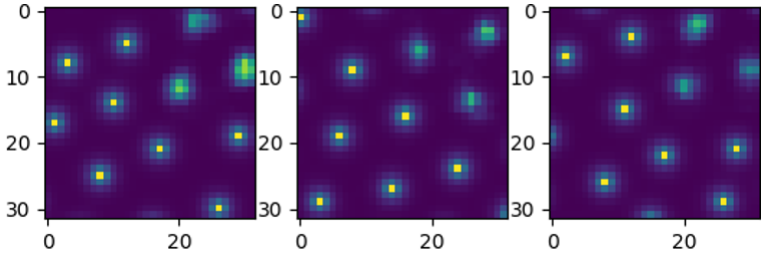
Figure 2.3: Autoencoders are feed-forward networks; edges are omitted from the diagram to avoid clutter. Autoencoders take some input vector and successively "squeeze" it down in dimensionality to produce an encoding (shown as the red units above). The network then attempts to reconstruct the original input based on the encoding, seeking to minimize the difference between the output and input. Each adjacent pair of layers can be treated as a Boltzmann machine, and pre-training (before applying backpropagation) proceeds using the usual contrastive divergence method to establish good starting parameters for the autoencoder.

Boltzmann machines can also be used to deal with noisy data or restore missing data due to their nature as generative models. In the course of our own work we obtained data generated from computer simulations of various materials being burned or thermalized. Our intent was to apply straightforward machine learning concepts to a field that has yet to significantly utilize it in its own research efforts. The data are composed of atoms with 3-dimensional spatial coordinates projected down into multiple image slices. Our work has shown phase formation, the results of thermalization processes, can be modeled even using something as simple as a Boltzmann machine.

See Figure 2.4 for an example of our results. It is a demonstration that BMs are capable of modeling physical phenomena and discovering the underlying rules that govern them. Even though Boltzmann machines have to use a restricted subset of connectivity and an approximate training algorithm, they are still capable of performing useful work, either as a standalone network or as the building block of larger ones.



(a) 32×32 pixel image projection of atoms in 3-dimensional space. A given space is divided into 3 image slices; for each slice, we removed the top-right portion of each image.



(b) A Boltzmann machine's completion of the missing data. The BM's results conform broadly with what we would expect to see in a crystal-like structure found throughout the molybdenum-oxide material.

Figure 2.4: Data gathered from a molecular dynamics simulation of a burning molybdenum-oxide material.

2.3 Limited Boltzmann Machines

Having discussed the orthodox model of Boltzmann machines, we now discuss what we call a limited Boltzmann machine, or LBM. Recall that Boltzmann machines are meant to be fully connected Markov random fields, but RBMs make a computationally necessary concession by restricting connectivity to a bipartite scheme between the visible units and hidden units. Our LBMs push back against this bipartite restriction by allowing a limited set of connections[2] to exist between units of the hidden layer. We do this for several reasons. First, we hypothesized that additional connectivity would give the LBM additional representational power. After all, LBM connectivity would be a superset of RBM connectivity, and in the worst case scenario where

[2]We cannot represent all-to-all connectivity between hidden units because the D-Wave hardware is not expansive enough to contain such a representation. Each qubit can only be connected to six other qubits at most. Future generations of hardware will expand this per-qubit connectivity limit. D-Wave has announced a new architecture, Pegasus, that increases the qubit degree from six to fifteen [9]. Our work is not predicated upon a specific degree of qubit connectivity, so in principle it can be applied to these future hardware generations as well.

additional connections are not helpful, LBMs can always set intra-layer connection weights to zero values and degenerate into RBMs. Therefore we believed that LBMs had a high chance to benefit from adding connectivity between hidden units.

The second reason we chose to focus on hidden unit connectivity was because we were interested in observing latent variables and encodings. In Section 2.2 we talked briefly about autoencoders and how they reduce input data into a set of progressively more compact encodings. When we view each pair of layers in an autoencoder as a Boltzmann machine, we observe that unit states of the hidden layer becomes the eventual encoding that is passed through the feed-forward network of the autoencoder. The hidden units act as a store of important or salient information, and studying how adding hidden unit connectivity to the fundamental Boltzmann machine affects its performance becomes relevant to other types of networks.

The third reason we chose to focus on hidden unit connectivity was to ensure we could solve larger problems and thus use more interesting data sets. We discuss the specifics of the hardware and our implementation in later chapters, so it suffices for now to say that had we chosen to add connectivity between our visible units, we would be severely restricted in our choice of data set, and we would also have to return to the NP-hard graph embedding problem we strongly wish to avoid.

Figure 2.5 shows a LBM that features connectivity between hidden units that conform to the physical limitations of our quantum annealing hardware. For comparison purposes against other Boltzmann machine connectivity models, referring to Figures 2.1 and 2.2 may be useful.

LBMs share the same problems of fully connected Boltzmann machines in that exact inference cannot be used to calculate the distribution of hidden unit states. Conditional independence, the convenient shortcut found in RBMs, is broken by the LBM's connectivity structure, making efficient training and evaluation difficult. However, we do have a sampling device at our disposal: an open-system adiabatic quantum annealer that just so happens to solve the sort of optimization problem Boltzmann machines are equivalent to. In place of using exact inference (see Section

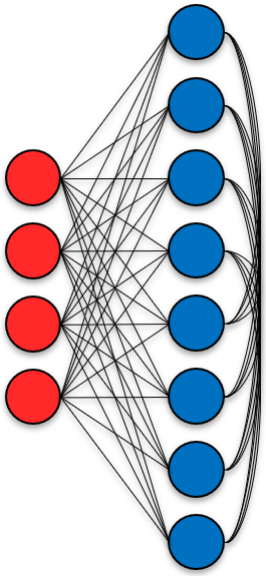


Figure 2.5: A limited Boltzmann machine (LBM). Connectivity within a LBM is a superset of connectivity within a RBM; see Figure 2.1 to compare. In addition to all the connections allowed to RBMs, LBMs also allow connections to exist between units of the hidden layer. Given a $M \times N$ grid of chimera cells, there would be $16MN + 4M(N-1) + 4N(M-1) = \mathcal{O}(MN)$ connections within a LBM. Note that this calculation was done assuming the implementation of D-Wave's quantum annealing machinery, where each qubit is restricted to having a maximum of 6 connections.

2.1), we can instead use this annealing hardware to draw samples from a complex distribution. The rest of the contrastive divergence training procedure would remain the same.

An important reason we cannot apply this approach to full Boltzmann machines is a limited number of qubits. With our implementation of LBMs on a quantum annealer, we do not have to use precious qubits to represent our data set. In contrast, were we to represent full Boltzmann machines on the annealer, we would only be able to solve problems of trivial size. Our usage of approximate training via contrastive divergence allows us to use qubits purely to represent hidden units and the connections between them on an annealer.

Chapter 3

Adiabatic Quantum Annealing Applied to Deep Learning Networks

We have covered the basic concepts of Boltzmann machines and the issues that arise when we manipulate the connectivity among visible and hidden units. Practical applications of Boltzmann machines running on classical machines use a restricted topology that allows only bipartite connectivity between visible and hidden units. Our research focuses on the novel direction of adding hidden-to-hidden connectivity back into this computational model, an addition that complicates the training procedure but can be tackled using a quantum annealer.

In the next few sections we introduce adiabatic quantum annealing and discuss how it can be applied to the BMs we have discussed. We also extend our discussion of limited Boltzmann machines (LBMs) that include expanded connectivity. We also include experimental results demonstrating that LBMs perform better than RBMs and begin an investigation of our overall topic at hand - how best to map Boltzmann machine variables to hardware qubits to improve data modeling results. Section 3.1 gives an overview of quantum computing and how adiabatic quantum annealing works. Section 3.2 describes the data we use in our experiments and the initial approach we took to achieve our research goals. Section 3.3 describes the open-system adiabatic quantum annealer we use. Following this, Section 3.4 shows how we implement Boltzmann machines on an adiabatic quantum annealer. Section 3.5 presents the results our initial experiments while

Section 3.6 discusses the findings and implications of those results. Section 3.7 gives context to adiabatic quantum annealing's place in machine learning by examining different neural network models (convolutional neural networks, or CNNs, and spiking neural networks, or SNNs) and their implementations on different hardware platforms.

While so far we have been only concerned with Boltzmann machines, we now also consider other types of neural networks as part of a broader survey of machine learning in non-traditional computing environments. Many-layered networks are called deep learning networks, and the restriction of intra-layer connections allows rapid training on graphical processing units. We will explain some current limitations of deep learning networks and offer approaches to help mitigate them. However, as our focus is on an adiabatic quantum computing approach in this survey, in Section 3.7 we leave only high-level descriptions of CNNs and SNNs to offer comparison and context for experiment designs. We conducted CNN experiments that used a high performance computing (HPC) environment to automatically discover good topologies for neural networks. Because HPC environments use classical machines, they remain restricted from using intra-layer connections. We also conducted experiments on SNNs, which use neuromorphic computing as a low-power alternative for representing neural networks. Rather than explicitly choosing one solution or another, these approaches can augment each other.

3.1 Quantum Computing Overview

Feynman first discussed quantum computing within the context of simulation, noting that simulating a quantum system using a classical computer seems to be intractable [20]. Interest in quantum computing surged with the introduction by Shor of a polynomial-time algorithm for factoring integers [48], giving an exponential speedup over the best known classical algorithm and threatening to break most modern encryption systems. As with Turing's early work, these theories for quantum computing were developed before quantum hardware was available. Different models

of quantum computing have since been developed in order to explore the power of quantum information processing. In the quantum circuit model (on which Shor's original algorithm relies), a sequence of unitary transformations is applied to a set of qubits, in a way analogous to the logical gates that are applied to classical bits in classical computing. In the measurement-based quantum computing model [39], a special quantum state is prepared beforehand, and a computation is performed by adaptively applying quantum gates to each qubit and measuring them. In the adiabatic quantum annealing model [30, 19], a quantum state encoding the solution of a problem is prepared using the adiabatic theorem of quantum mechanics. Adiabatic quantum annealing or optimization is a specialized form of quantum annealing that only solves an optimization problem. All three models have the same computational power but offer different trade-offs. Quantum information is extremely fragile, and any source of noise (like thermal fluctuations, unwanted interactions with an uncontrolled environment, etc.) can destroy the quantum features that are expected to provide a computational speedup. Each model is susceptible to noise and requires very large error-correction overhead to overcome the effects of that noise.

Adiabatic quantum computing (AQC) is an implementation of the ideas of quantum computing that relies on the adiabatic theorem of quantum mechanics. This result states that if a system is in the ground state of a particular Hamiltonian and the parameters of this Hamiltonian are changed slowly enough, the system will remain in the ground state of the time-dependent Hamiltonian. This idea was used by Farhi et al. [19] to propose an alternative to the quantum circuit model of quantum computing. The main idea is to start with a Hamiltonian whose ground state is easy to construct, and slowly change it into one whose ground state encodes the answer to a particular problem.

AQA is a special case of AQC. AQA only solves a particular optimization problem. As realized in the D-Wave, it finds the ground state of an Ising problem. This model describes a system of interacting magnetic moments subject to local biases. This problem was shown by Barahona [5] to be NP-hard, so many other optimization problems of practical interest can be recast in this form.

If we consider a set of spin variables $S_i = \pm 1$, the energy of the system is given by a quadratic expression of the form

$$E_{Ising}(s) = \sum_i h_i \, s_i + \sum_{i,j} J_{ij} s_i s_j \tag{3.1}$$

Solving this problem means finding a spin configuration that minimizes this energy function. In a quantum approach, we consider a quantum system of interacting spins described by the Ising Hamiltonian

$$H_{Ising} = \sum_i h_i \, \sigma_i^z + \sum_{i,j} J_{ij} \sigma_i^z \otimes \sigma_j^z \tag{3.2}$$

where h_i represent local magnetic fields and J_{ij} are couplings between spin pairs. This Hamiltonian is diagonal in the σ^z basis, and its ground state can be used to construct the corresponding configuration that minimizes the Ising energy above.

To solve this problem in the context of AQA we can choose an initial Hamiltonian of the form

$$H_0 = -\sum_i \sigma_i^x \tag{3.3}$$

that represents the effects of a transverse field applied to all spins. The ground state of H_0 consists in all spins being in the $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ state. If we consider the spins as little magnetic moments, this corresponds to all spins pointing in the x direction. Quantum mechanically this state is separable, easy to construct (just apply a strong magnetic field in the x direction), and when expressed in the computational basis it is an equal superposition of all possible states.

The computation is performed by slowly changing the relative weights of H_0 and H_{Ising} during the interval $[0, T]$

$$H(t) = (1 - \frac{t}{T})H_0 + \frac{t}{T} H_{Ising}. \tag{3.4}$$

This process is known as *quantum annealing*. The change must be slow compared to the time scale associated with the minimum energy gap of the time-dependent Hamiltonian, where we define the gap as the energy difference between the energies of the first excited state and the ground state[10, 44, 8]. If the change is too fast the system can transition to an excited state, and the state at the end of the annealing will not be the ground state of the Ising Hamiltonian. On the other hand, if the change is too slow the computation will take a long time and risk decoherence, where quantum properties are lost. The main challenges in adiabatic quantum annealing are to understand the connection between this energy gap (i.e., the runtime) and the size of the problem, and to find Hamiltonians that solve a given problem while possessing a larger gap [50]. But other issues are also important for practical implementations, in particular how unavoidable noise affects the system due to the system's interaction with the environment.

The adiabatic quantum annealing (AQA) model has seen hardware implementation, and we use a D-Wave Systems device at the USC-Lockheed Martin QCC for our experiments. One benefit of our D-Wave device is that it fields a much larger array of qubits than today's NISQ (noisy intermediate-scale quantum) circuit model systems do [53, 4, 49] - 1098 working qubits versus the 53 working qubits of Google's Sycamore chip. However, the D-Wave processor is still limited in many aspects, the most important being the fact that it operates at a finite temperature, and the effects of this noise in the performance of the device is still an active area of research. We typically refer to these types of devices operating at a finite temperature as "open system" adiabatic quantum annealers.

Quantum annealers are in principle designed to solve a particular optimization problem, typically finding the ground state of an Ising Hamiltonian. Unfortunately, thermal fluctuations due to interactions with a finite temperature reservoir, in addition to unwanted quantum interactions with other systems in the environment, tend to kick the system out of its ground state and into an excited state. These unavoidable features make the quantum annealer behave more like a sampler than an exact optimizer in practice. However, this seemingly counterproductive property

may be turned into an advantage since the ability to draw samples from complicated probability distributions is essential to probabilistic deep learning approaches such as the Boltzmann machine, which relies heavily upon sampling complex distributions in both training and output. The important implication here is that quantum annealers can help us overcome the problem of complex topologies mentioned before. BMs in their unrestricted form are impractical to train on classical machines, a fact that led to the development of RBMs that eliminate intralayer edges and introduce bipartite connectivity. Bipartite graphs allow the use of an algorithm known as contrastive divergence that approximates samples from a RBM in linear time, which is a critical tool for the practical usage of BMs because sampling is the core engine for training BMs[1]. Quantum annealing hardware allows us to partially pull back from this bipartite limitation. Quantum annealers provide a novel way to sample richer, more powerful topologies, and several approaches exploit this feature for different choices of graphs and topologies on D-Wave hardware [2, 6, 7].

3.2 Data and Initial Approach

Our initial set of experiments were performed in the context of a collaboration with ORNL and the University of Tennessee, Knoxville (UTK), with the goal of studying alternative approaches for solving machine learning problems using emerging technologies such as quantum devices or neuromorphic hardware. Consequently, our experimental designs took into consideration the needs of our different platforms. The platforms we considered were adiabatic quantum annealing, high performance computing, and neuromorphic computing.

Adiabatic quantum annealing, high performance computing, and neuromorphic computing differ significantly from each other in how they process data. As such, the amount of data each can support dictated our choice of deep learning problems that could be adapted to each of these three heterogeneous paradigms. The D-Wave device we used (DW2X) supported 1098 working

[1]Contrastive divergence is an approximate method and essentially replaces the laborious sampling process with a much faster exact inference process. A quantum annealer is appealing because we could pull a similar trick and replace exact inference with sampling via a quantum annealer.

qubits[2], which limited the size of problems we could solve. With this in mind we chose to examine two datasets we refer to as MNIST and neutrino data.

The MNIST data set, discussed in 3.7.3, is a well-known collection of hand-written digits extensively studied in the deep learning community. The dataset is composed of images, each of which contains a handwritten digit and an associated label identifying the digit. The digit images are only $28 \times 28 = 784$ pixels, which fits within the 1098 qubit D-Wave hardware and onto HPC and neuromorphic architectures[3]. Our later experiments used neutrino particle detection data down-sampled and adjusted to 32×32 pixels.

The neutrino scattering dataset was collected at Fermi National Accelerator Laboratory as part of the MINERvA experiment that is focused on vertex reconstruction [52]. In the Main Injector Experiment for v-A (MINERvA), many scintillator strips were arranged in planes orthogonal to the neutrino beam within the detector, aligned across three different orientations or "views." We utilized both the energy lattice and the time lattice information in the dataset. In particular, we represented the energy lattice as an image, where the intensity of each pixel in the image corresponds to the average energy over time in the detection event. The images show the trajectory of particles over time from the view of one particular plane. We also used the time lattice in one of our experiments. For the time lattice, each data point in a detection event corresponds to the time at which an energy level exceeds a certain threshold. Associated with each detection event is a number corresponding to a specific detection plate within the chamber, and this number indicates which plate a neutrino strikes. This number can then be utilized to determine in which detector region or segment the vertex of the event was located.

In BM experiments we used down-sampled and collated image data from one single plane. We did not use the original data because the quantum annealer has limited space for storing problems

[2]NASA uses a D-Wave 2000Q which features 2048 fully functioning qubits.

[3]The size of the data is not a strict limitation on our experimental design because we do not map visible units (representing data) onto qubits. Rather, we map hidden units to qubits, and we have total control over the number of hidden units we choose to use. When we were developing our experimental design, we did not yet have the previously described implementation, so we played safe and chose a small data set to start with.

and because BMs are not well-suited to handling temporal data. However, the SNN experiments did take advantage of temporal data because SNNs are designed to handle such data. Because the adiabatic quantum annealing portion of this project is of particular interest, we next provide a more detailed description of the process and of the annealing hardware.

3.3 D-Wave's Superconducting Adiabatic Quantum Annealer

The architecture and physical details of the quantum adiabatic processor we studied are described in detail in [23]. In essence, it is designed to represent the Ising Hamiltonian as an array of superconducting flux qubits with programmable interactions. The qubits are implemented using SQUIDs (Superconducting QUantum Interference Devices) composed of a Niobium loop elongated in one direction. Several loops and Josephson junctions are added to the design to both allow for the required controls to implement quantum annealing and to compensate for the slight differences between the physical properties of any two SQUIDs due to fabrication variations. The processor has a unit-cell structure composed of 8 qubits with four arranged horizontally and four vertically such that each qubit intersects orthogonal qubits. At these intersections another SQUID is placed to control the magnetic coupling between the corresponding horizontal and vertical qubits within the same cell. These are the only couplings allowed within the unit cell (i.e. horizontal qubits are not coupled to other horizontal qubits). This architecture results in a coupling graph that is fully bipartite at the unit cell level. The processor is then built by adjoining more unit cells in a square lattice such that the horizontal qubits in one cell are coupled to the horizontal qubits in the neighboring cells to the right and the left, and the vertical qubits are coupled to the vertical qubits on top and on the bottom. A visualization of this setup, also known as a (M, L) Chimera graph[4], is shown in Figure 3.1.

[4] (M, L) Chimera graphs have $M \times M$ cells and L-sized partitions with $2L$ units per cell. The DW2X implements a $(12, 4)$ Chimera graph with 1098 working qubits.

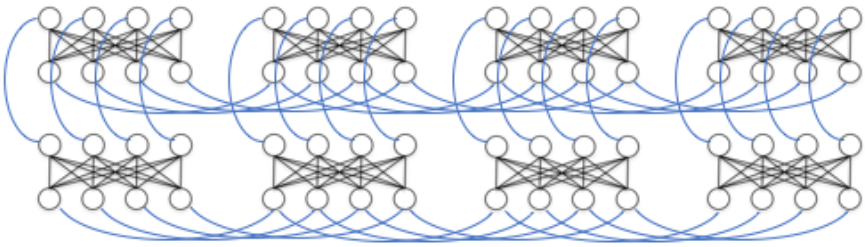


Figure 3.1: Chimera graphs are composed of 8-qubit cells featuring bipartite connectivity. Each cell's partition is connected to another partition in the adjacent cells

Programmable interactions and biases are used to implement the Ising Hamiltonian in Eq.3.2. The parameters h_i represent local magnetic fields while the parameters J_{ij} are the couplings between two spins. Their values are restricted to the range $[-2, 2]$ for the local local fields, and $[-1, 1]$ for the couplings . It is understood that the couplings J_{ij} are only nonzero when there is a physical coupler associated with that particular pair of qubits on the chip. A transverse field term can also be implemented on each qubit, resulting on a driver Hamiltonian of the form shown on Eq. 3.3. The adiabatic quantum annealing is implemented by combining the two Hamiltonians above and changing their relative weight adiabatically, such that ideally the system remains always in the ground state. In other words, the processor implements the Hamiltonian

$$H(t) = A(t)H_x + B(t)H_{Ising} \tag{3.5}$$

where the functions A and B satisfy $A(0) >> B(0)$ and $A(T) << B(T)$, for some final annealing time T. At $t = 0$, the system is in the ground state of the transverse field Hamiltonian H_x, corresponding to all the qubits being in the same eigenstate of σ^x, or in other words, a superposition of all possible states in the computational basis. For the closed system case (where there are no interactions with the environment), if the quantum annealing is done slowly enough, the adiabatic theorem of quantum mechanics guarantees that the state of the system at time T is with high probability the ground state of H_{Ising}. How slow is "slowly enough" depends on the details of the Hamiltonian, in particular the inverse of the energy gap between the ground

state and the first excited state, and this feature is the main factor in determining a lower bound on the run time of the device. However, real devices are not ideal closed systems, so unwanted interactions with the environment will try to kick the system out of its ground state.

The current generations of D-Wave machines are designed for experimental use and are not optimized for turnaround time, unlike relatively mature CPU or GPU platforms. Rather than directly competing against existing classical solutions to machine learning, we focus on showing it is viable to use a quantum annealer to help train a neural network with complex topologies using architectures and approximations that differ from what has been used before [2, 6, 7]. For this reason, instead of using clock timings, we measure error metrics against the number of training epochs. As quantum annealing technology becomes more developed, machine learning algorithms may see benefits from using this new type of hardware. Regardless, clock timings are still important to consider. We next describe the computational workflow for each problem using D-Wave machines and communication latency between a client machine and a D-Wave machine; later we describe the timings over various operations on the hardware.

Each problem is sent across a network using D-Wave's Solver API[5] (Matlab or Python) to the worker queue. Workers can concurrently process multiple requests and submit post-processed requests to the quantum processing unit (QPU) queue. Each request is then run sequentially on the QPU. Finally, the workers return the results back to the client. In one study D-Wave reported the mean turnaround time for each request was approximately 340ms. Timings can vary depending on network latency - request latency can be reduced by placing the client physically next to the annealer, for example.

Communication latency aside, we also look at how long it takes to define and solve a problem on D-Wave. Loading and defining a problem on D-Wave hardware takes around $t_d = 10$ms. Drawing a sample from the defined distribution via annealing takes around $t_a = 20\mu$s. Reading out the unit states from a sample takes around $t_r = 120\mu$s. We repeat the sampling and read-out

[5]This is now deprecated. D-Wave has switched over to a new framework called Ocean.

stages $k = 100$ times for each MNIST image or neutrino detection instance in our experiments. So for each data point within our datasets, it takes $T = t_d + k(t_a + t_r)$ time to process. Currently the problem definition time t_d and read-out time t_r dominate wall-clock timing, but we again stress we are looking to future developments and advancements in quantum annealing hardware that will reduce such overhead. We find the low annealing time particularly appealing because it scales well in algorithmic terms. That is, we can add additional hardware qubits or connectivity to produce more complex networks but the sampling time (annealing time t_a for our experiments) will not increase, which is not the case for simulating equivalent networks in software.

The number of physical couplers restricts the set of problems that can be natively implemented on the processor, and it represents one of the main limitations of the devices. Minor graph embeddings can overcome this limitation but at the expense of utilizing more than one qubit per graph node [11]. As we will show in the next chapter, our approach turns this problem on its head. Instead of trying to fit a problem into a particular topology, we start with our hardware topology using RBMs that have no intralayer couplings and study the advantages gained from adding additional couplers.

3.4 Implementing a Boltzmann Machine on D-Wave

We used D-Wave's adiabatic quantum annealer located at the USC-Lockheed Martin Quantum Computing Center. We implemented a Boltzmann machine to represent the MNIST digit recognition problem and neutrino particle detection problem. Deep learning using BMs has been proposed before, but as previously discussed, learning is intractable for fully connected topologies because we need to compute expected values over an exponentially large state space [1, 43]. RBMs address this by restricting network topology to bipartite connectivity to introduce conditional independence among "visible" units (representing the dataset and RBM output) given the "hidden" units (representing latent factors that control the data distribution), and vice versa,

though they lose some representational power in the process. The quantum annealing hardware gave us an opportunity to first implement a RBM to establish baseline performance and then ease some topology restraints to investigate how more complex topologies could improve our results.

Our RBM used 784 visible units to represent each pixel in a 28×28 MNIST digit image and 80 hidden units on a D-Wave adiabatic quantum computer. We added an additional 10 visible units as a digit classification layer where the unit with highest probability was chosen as the label. Similarly we used $32 \times 32 = 1024$ units to represent the neutrino data, 80 hidden units, and 11 classification units to represent the 11 collision sites in the neutrino detection chamber, where the classification unit with the highest probability was chosen as the BM's guess for which plate the particle struck. The BMs were trained over 25 epochs on a training set and then evaluated against a validation set.

Next, as mentioned above, we loosened some of the topology restrictions of RBMs. RBMs enforce bipartite connectivity (see Figure 2.2), meaning hidden units are not connected to one another. We partially removed this restriction and allowed some of our hidden units to communicate with each other. We called this semi-restricted BM a "limited" Boltzmann machine (LBM). LBMs can be viewed as a superset of RBMs, the only difference being a set of extra available connections between hidden units. The previously described superconducting quantum adiabatic processor has physical constraints that limit connectivity to a chimera topology, so LBMs remain a subset of BMs.

Because D-Wave hardware faces a physical constraint on the number of possible units and connections, we would have had to employ the minor embedding approach mentioned above if we wanted to represent all of a BMs units on hardware. This would result in a large overhead in the number of qubits required, restricting our approach to small BMs. However, we can still try to exploit the quantum features of the D-Wave by restricting the topology of our model and only embedding part of it in the device. In our implementation we chose to represent only the hidden units, used the annealer as a sampler for the interconnected hidden units to estimate

required quantities needed to update the weights, and left representation of the visible units to a classical machine. We were primarily interested in the interaction between hidden/latent units because they can represent abstract features extracted from the data. Figure 3.2 visualizes the extra connectivity we added to the LBM model and Figure 3.3 shows how we represented LBMs on the D-Wave's chimera topology.

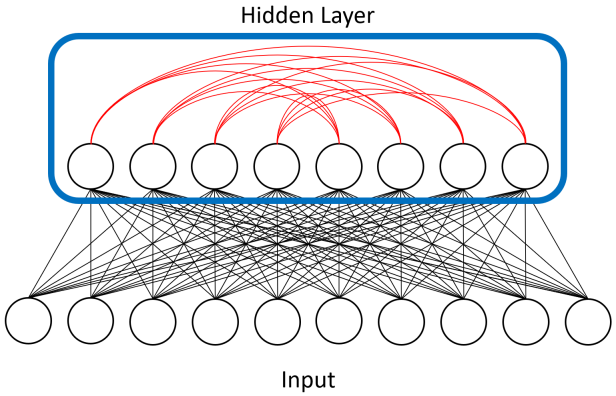


Figure 3.2: Our LBM model added connectivity between units in the hidden layer, shown in red. RBMs prohibit such intralayer connections because they add too much computational complexity for classical machines. We represented the hidden layer (units contained within the blue box) on the D-Wave device. The connections between hidden units were 4-by-4 bipartite due to the device's physical topology constraints.

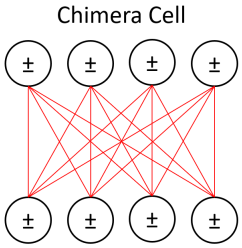


Figure 3.3: The hidden layer from Figure 3.2 is represented in one of D-Wave's chimera cells here, with the cell's bipartite connectivity made more obvious. The input/visible units of the LBM are left on a classical machine. Their contributions to the activity of the hidden units is reduced to an activity bias (represented with ± symbols) on those units. Figure 3.1 shows the overall chimera topology of the D-Wave device.

Using D-Wave hardware to adjust LBM parameters may help tackle the intractability issue because the quantum annealer does not rely on conditional independence between units within a layer. We give a short explanation of the training process for BMs to illustrate.

The configuration x of binary states s of units has an energy E defined by

$$E(x) = -\sum_i s_i b_i - \sum_{i<j} s_i s_j w_{ij} \tag{3.6}$$

where b is the bias of a unit and w_{ij} is the mutual weight between two units i and j. The partition function is $\sum_u e^{-E(u)}$, and the probability the BM produces a particular configuration x is

$$P(x) = e^{-E(x)} / \sum_u e^{-E(u)}. \tag{3.7}$$

$P(x)$ is difficult to compute in a full BM because it requires a sum over an exponentially large state space. If we want to determine the probability of some hidden unit i is on (equal to 1) without any guarantee of conditional independence, we would have to calculate $P(h_i = 1) = P(h_i = 1|v, h_{-i})$ where v is the state configuration of visible units and h is state configuration of the hidden units. But if we use RBMs to restrict ourselves to bipartite connectivity between v and h, this probability factorizes and we can write $P(h_i = 1) = \prod_{j=1}^{n} P(h_i = 1|v_j)$. Our first RBM baseline experiment used this standard procedure with 1-step Gibbs sampling. In our LBM experiment, we did not need to rely on conditional independence or Gibbs sampling because we used quantum annealing instead to approximate samples from the more complicated probability distribution.

Earlier we mentioned we only represented the hidden units on the annealer. This was done by reducing the influence of visible units on the hidden unit state distribution to a simple bias on the qubit representing a given hidden unit. Suppose we wanted to sample from the annealer to get an empirical measure of the hidden unit state distribution. In such a situation, we would already know the visible unit states as a part of our constrastive divergence training algorithm. The distribution of hidden unit states would then be defined by Equation 3.7. Critically, one part of $E(x)$ is fixed: the aforementioned visible unit states. If we look at the latter term of Equation 3.6, $\sum_{i<j} s_i s_j w_{ij}$, we notice that any of those two-body interactions involving a visible unit will

be fixed. There are two possibilities: in the first, a hidden unit configuration excludes such an interaction, in which case we are done. In the second case, a hidden unit configuration includes such an interaction, in which case $s_i s_j w_{ij}$ is contributed to the overall energy of that state. In effect, this is simply a bias on the hidden unit since we already know the visible unit state involved in all these interactions. It is this observation we make that allows us to represent only the hidden units on the annealer.

The training procedure for BMs compares the distribution of the data against the expected distribution according to the model and uses the difference to adjust the weight matrix w. Sampling from the model is difficult so we approximate using Markov Chain Monte Carlo (MCMC) sampling. The first "positive" phase of training locks the states of visible units to a configuration determined by the data - for example, a 28×28 pixel image from the MNIST dataset. The hidden unit distribution according to the data is found in this phase. The second "negative" phase unlocks the visible units and the system is allowed to settle. Sampling during this phase is difficult so we approximate samples using contrastive divergence with one step of MCMC and find the unit distributions according to the BM model. The weight matrix is then updated with the following equation:

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{reconstruction}) \tag{3.8}$$

where ϵ is the learning rate, $\langle v_i h_j \rangle_{data}$ is the product of visible and hidden unit state probabilities in the positive phase, and $\langle v_i h_j \rangle_{reconstruction}$ is the product of visible and hidden unit probabilities in the negative phase.

For the MNIST problem we used 6,000 images from the MNIST digit dataset to train the RBM and LBM. Each 28×28 image was represented with a 784-length vector with 10 units using 1-hot encoding to represent the class of digit. In training the labels were hidden and the BM attempted to reconstruct them to guess what the image label was. The classification unit with the highest

probability of being "on" was chosen as the BM's label guess. The neutrino experiment used the same setup except the images were 32×32 pixels and thus 1024 visible units. The weight matrices were randomly initialized from a standard normal distribution and updated using the rule in equation 3.8.

We wanted to further explore how connections between hidden units, referred to as couplers, contributed to problem solving in a LBM topology. To do so we limited the visible-to-hidden connectivity in the next experiment such that each hidden unit was only allowed to see a 4×4 box of pixels in the input images. These boxes did not overlap with each other. Reconstructing the input image became a much harder problem and the hope was that the addition of couplers would allow hidden units to trade information about input pixels in boxes they normally could not communicate with and improve results. This setup was somewhat inspired by CNN convolutional layers but we decided to make the "convolution" non-overlapping to use fewer qubits. In the future we will expand to use more qubits.

We believed this setup would make couplers relatively more important to the LBM because we reduced the ratio of visible-hidden connections to couplers. An input image with $32 \times 32 = 1024 = 2^{10}$ data points and 64 hidden units has $2^{10} \times 2^6 = 2^{16}$ visible-to-hidden connections for 168 couplers. But hidden units with only 4×4 boxes of pixel visibility would instead have $2^4 \times 2^6 = 2^{10}$ visible-to-hidden connections for 168 couplers.

3.5 Initial Results

We trained our RBM and LBM using the same parameters over 25 epochs (complete runs over all the training data). We followed common guidelines for choosing and adjusting hyperparameters [24]. We selected the learning rate ϵ to be 0.1 for weights between visible-to-hidden weights and 0.1 for hidden-to-hidden units for our experiments, excepting our first one shown in Figure 3.4. Setting ϵ too low means a BM learns slowly and may get trapped in local minima whereas setting

it too high can cause the network to travel wildly in parameter space and be unable to learn coherently.

Before implementing the RBM running on MNIST data we wanted to get initial results indicating there was some merit to the LBM topology. Using simulated data, we mapped a BM to a quantum annealing simulator and trained two configurations, one where intralayer connections were disabled and one that had random intralayer connections. 10 epochs of training a RBM and LBM in Figure 3.4 show the LBM has some advantage.

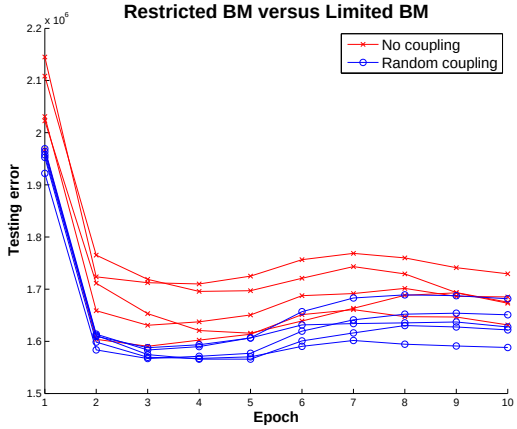


Figure 3.4: An initial experiment to demonstrate LBM utility. Reconstruction error (sum of squared error) of BMs trained on simulated data using no intralayer connections and using random intralayer connections with a small (0.0001) hidden-to-hidden weight learning rate. Here we show 5 RBMs (red) and 5 LBMs (blue), and the results suggest even just the presence of relatively static intralayer connections gives LBMs a performance advantage over RBMs. We obtained these results from the quantum annealing simulator provided by D-Wave.

As discussed, our first experiment was to establish a performance baselines in RBMs so we could later compare LBMs against them. Figure 3.5 displays reconstruction error (sum of squared error between the actual data and BM reconstruction data) and classification rate. This figure is included to confirm that the RBM did indeed learn to model the MNIST digit data distribution. Figure 3.6 contains a comparison of RBM performance and LBM performance on the MNIST digit recognition problem.

The RBM and LBM were both implemented on D-Wave and on MNIST images using the same number of hidden and visible units. For this test we trained over ten epochs. The RBM configuration, as discussed, had no intra-layer connections, whereas the LBM configuration had limited connections between the hidden nodes.

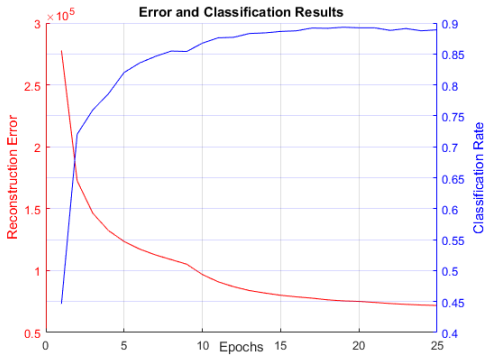


Figure 3.5: Reconstruction error and classification rate over 25 training epochs using 6,000 MNIST images for training and 6,000 for testing. Reconstruction error decreases as classification rate rises, confirming that the RBM learns the MNIST data distribution.

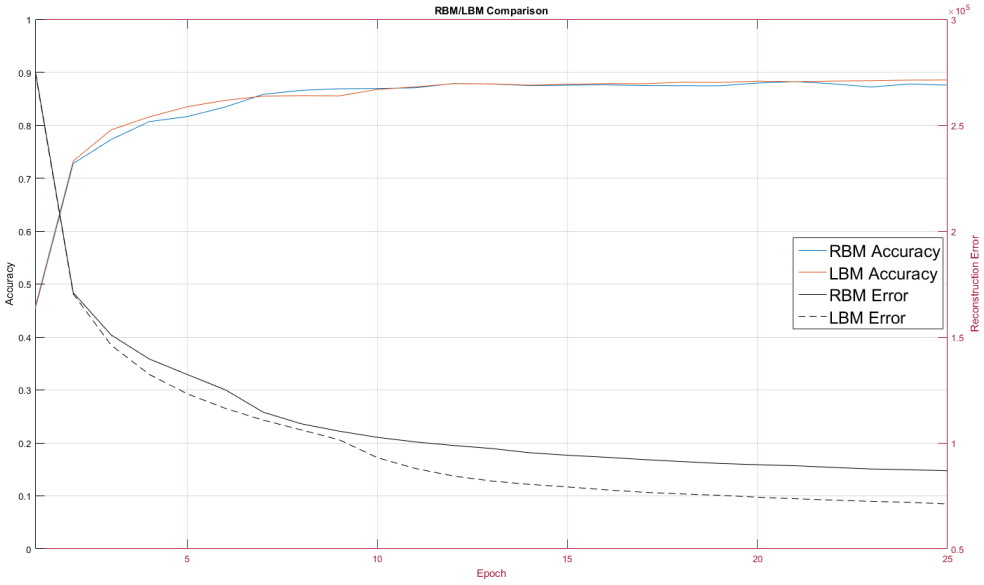


Figure 3.6: RBM and LBM performance on the MNIST digit classification task. The LBM tends to label the digits slightly better and produces lower reconstruction error than the RBM.

One quirk we found was LBM configuration initially performed worse than the RBM configuration. This was unexpected and we adopted a hybrid learning approach where the intralayer

connections were reassigned from a random normal distribution for the first 3 training epochs. Afterwards the intralayer couplers were allowed to evolve according to the standard training rule. Our choice of a 3-epoch delay for intralayer training was rather arbitrary; further exploration into the mechanics involved will be explored in future work where we will pre-train models as RBMs on classical machines and then later hand over training to a quantum annealer.

The LBM achieved a classification rate of 88.53 percent, seen in Figure 3.4, and was comparable to other RBM results on MNIST [15].

Our LBM setup mapped only the hidden units to the D-Wave hardware whereas most other work maps a whole BM. The latter approach requires down-sampling and graph embedding. We hope our approach scales better with problem size because we represent the visible input units on classical machines and still use contrastive divergence as a training method.

Our experiments on neutrino data and limited visible-to-hidden connectivity were run on both simulation software and D-Wave hardware. We used both because hardware has physical limits regarding parameter ranges and experiences parameter warping, so the inclusion of software results provides additional support if both environments produce comparable results. Parameters on the hardware for Ising models have 4 to 5 bit precision[6] and can only take on values within a small range, typically $[-2, 2]$ for h or $[-1, 1]$ for J. Software simulators do not have this limited precision and their parameters are not limited to any particular range.

We show the simulator results in Figure 3.7. Results from the simulator suggest the addition of couplers in this new setup improved performance, which led to our move to experiment on the quantum annealing hardware. Our experiments in Figure 3.8 were similar to the previous ones, albeit we first trained a RBM on a classical machine. We then took this lightly trained RBM model and moved it to the D-Wave hardware, used its semi-trained parameters to initialize the weights of the D-Wave RBM and LBM, enabled 168 couplers, then continued training for an

[6]Precision is determined by integrated control errors and is additionally dependent on the values we choose to program for the h, j matrices in our Ising problem. Differing desired values produces variable precision in the 4-5 bit range [37].

additional 20 epochs. We again performed the RBM experiment 5 times and the LBM experiment 5 times.

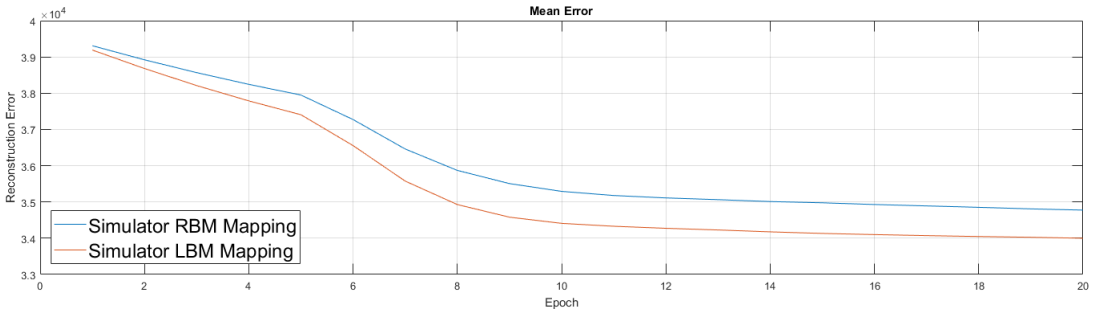


Figure 3.7: Comparison of RBM against LBM trained on neutrino data using a software simulator. Weights are randomly initialized from a normal distribution. The change in learning rate at epoch 5 is due to a change in the momentum parameter in the algorithm that is designed to speed the rate of training. The graph shows the mean performance of 5 different RBMs and 5 different LBMs and suggests the mean reconstruction error of RBM and LBM are significantly different.

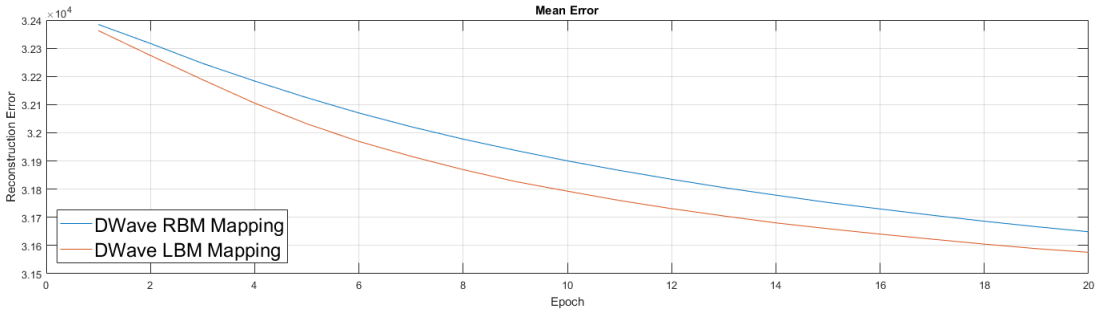


Figure 3.8: Another comparison of RBM against LBM run on neutrino data using D-Wave hardware. Both the RBM and LBM are initialized from the same pre-trained model. The pre-trained model is a RBM run for 3 epochs on a classical machine. The graph shows the mean performance of 5 different RBMs and 5 different LBMs, suggesting the performance difference between RBM and LBM persists on hardware.

In the LBM experiment we did not remap qubits in any scheme more complicated that a linear fashion. That is, we designated each qubit to oversee a 4×4 box in a horizontal order and simply assigned each qubit to unit cells according to this order. In Section 4.1 we will argue this is suboptimal and that we can improve our results even more by considering smarter remappings of qubits to take advantage of locality within image data. For now we leave the comparison as RBM results versus LBM results without any special qubit remapping.

One aspect of superconducting technology worth mentioning is power consumption. The energy consumption of a system such as the D-Wave hardware is dominated by the cooling of the processor. When programming the device, the control signals inject some energy into the system that can increase the temperature by a few milli Kelvin. This energy needs to extracted, resulting in a few pico Watts of power being dissipated in this step. But the actual computation requires a negligible amount of energy. The cooling requirement has remained flat for four generations of the D-Wave device and is not expected to change in the foreseeable future. While the energy consumption of quantum annealers is typically not a highlighted advantage over classical systems, power efficiency may eventually become an important reason for preferring quantum computing systems in the future.

3.6 Impact of Qubit Mapping and Spatial Locality

Up to now our experiments used a simple 1:1 mapping of hidden units to qubits by placing qubits in chimera cells in the order we defined them. This was done to keep implementation simple and straightforward. There are many other ways we could choose to map hidden units to physical qubits and it is worth considering how such choices impact data modeling results. We immediately think of exploiting data locality, especially since our data is composed of images. Our initial simple 1:1 mapping did not take advantage of such locality, so we will examine different possible types of mappings that produce better results and see how they reveal patterns within our data sets.

To draw out the relative importance of hidden-to-hidden connections in our LBMs, and to provide a setup for exploring how qubit mappings might exploit data locality, we proceeded to try a novel qubit mapping method. Recall that LBMs reintroduced limited connectivity between hidden units normally disallowed by RBMs. Our new experimental setup made a new modification to the LBM by deleting the majority of visible-to-hidden connections within the LBM. While our first set of experiments had allowed each hidden unit to exchange information with all visible

units, the newer setup restricted connectivity such that each hidden unit of the LBM could only see a 4×4 pixel square of image pixels. Figure 3.9 illustrates the connections between image pixels and hidden units of the LBM.

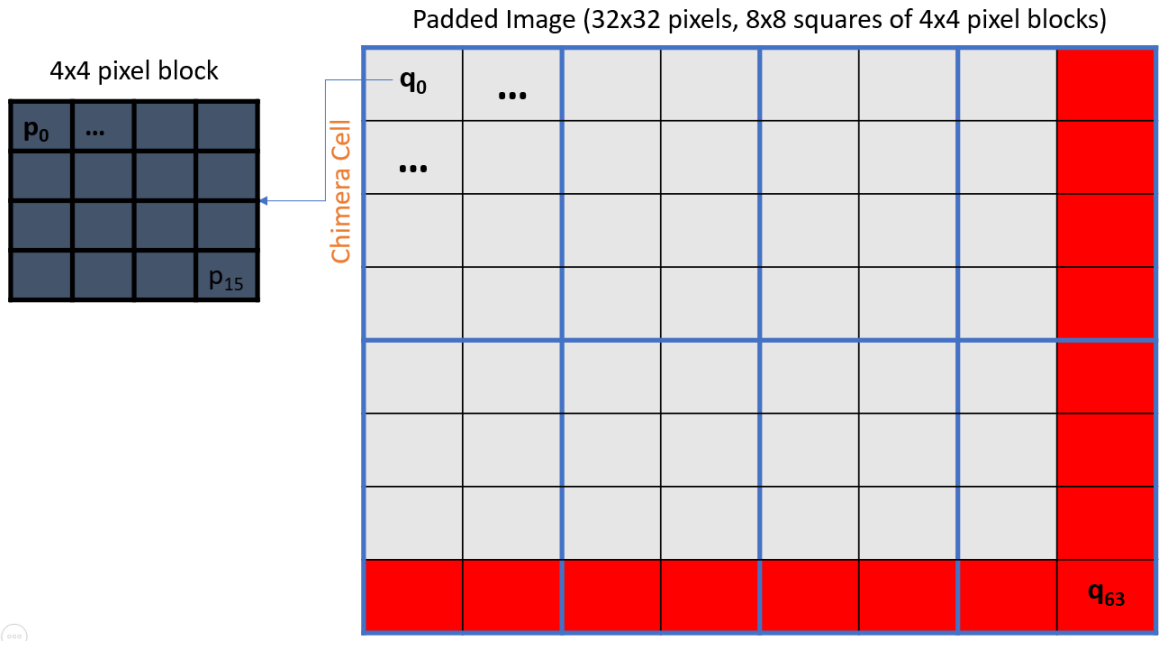


Figure 3.9: Qubits in the modified LBM are connected to a 4×4 square of pixels in the input image. Once each qubit is connected to a non-overlapping square, we then must decide how to map these logical qubits into the physical chimera cells. The chimera cells are represented here with blue outlines, and it can be seen that each cell has a 8 qubits as expected. Some boxes are in red because the original images were 28×28 pixels and we padded them with empty pixels to make the images 32×32 pixels.

After setting up the visible-hidden connectivity of the units, the next step was to think about how to map each qubit to the chimera cells. Our experiments had simply dropped the qubits into cells according to their logical order, but this is not the only way to place qubits in cells. Considering that we had just restricted our visible-hidden connectivity, our intuition was that the way we organized the qubits into chimera cells would now become relatively important because the couplers between hidden units could compensate for the new relative blindness of the hidden units. One way to take advantage of data locality in images was to place spatially-similar qubits together, the idea being nearby pixels have information relevant to each other and that hidden

units can exchange this relevant information either within their chimera cells or through chains connecting cells together. Figure 3.10 shows one such mapping scheme we refer to as a "box" mapping because we attempt to stitch together a square of pixel squares.

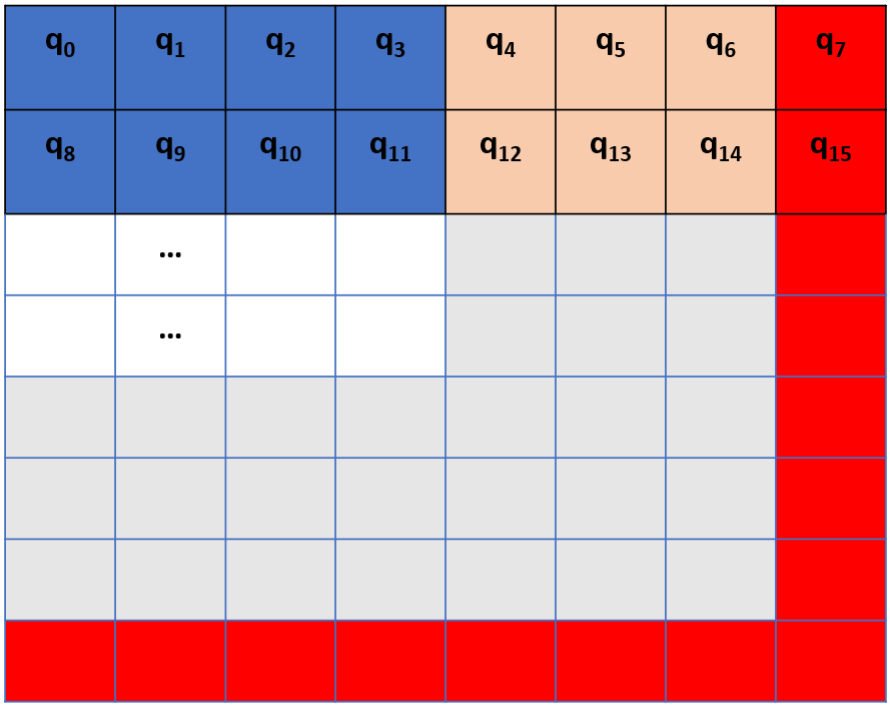


Figure 3.10: One way to map logical qubits ($q_0 \ldots q_6 3$) to chimera cells. The example here shows the "box" mapping where qubits representing closely adjacent pixels are grouped together in one chimera cell (namely qubits $q_0, q_1, q_2, q_3, q_8, q_9, q_{10}, q_{11}$).

In addition to our box mapping, we searched through a randomly generated assortment of other mappings. We felt that though the box mapping made intuitive sense, there should exist better solutions. In a box mapping, we only take advantage of purely local spatial information. Translated into the terms of the original input data, the particle collision dataset, a hidden unit that is connected to one collision plate's worth of image pixels is placed into a chimera cell that contains another hidden unit that might be connected to an adjacent collision plate's image pixels. It might be more informative to also place a hidden unit into the same chimera cell that can see a plate on the far end of the detection chamber. A particle ricochet on one end of the chamber might indicate the particle also hits a collision plate on the opposite side of the chamber, information

that a mix of local and global hidden unit connectivity can pick up on. Figure 3.11 shows some early results demonstrating that better mappings than the box mapping exist.

3.7 Alternative Approaches

We have mostly focused on quantum annealing so far, as it is the topic we are most interested in. However, HPC and neuromorphic approaches also deserve some exposition to give more context about the deep learning environment and considerations we faced when working with quantum annealing and Boltzmann machines. We have mentioned HPC and neuromorphic technology as two other platforms that can be utilized to benefit deep learning networks. Each has certain qualities that are not found in our adiabatic quantum annealing approach due to fundamental differences between the platforms. Quantum annealers can handle complex topology but are limited in number; HPC exploits massive parallelization for computation speed but still uses classical machines; and neuromorphic hardware is low power but tricky to train. We envision an integrated future where we can call upon the strengths of each platform to augment machine learning efforts and create a richer set of neural networks suitable for implementation on different machines. In this section we describe results from our HPC and neuromorphic efforts and how they can also contribute to training deep learning networks. We give an overview of HPC and neuromorphic computing before discussing networks tailored for implementation in those environments.

We tied a specific type of network to each of these computing environments. Our HPC environment implements CNNs and searches for good hyper-parameters. Hyper-parameters can be seen as a type of meta parameter that is not directly derived from data or the training process. In our case, the hyper-parameters of a CNN include the number of layers we use, the size of each layer, and the connectivity between layers. Our neuromorphic environment is concerned with SNNs and finding a network connectivity scheme that classifies images accurately. And of

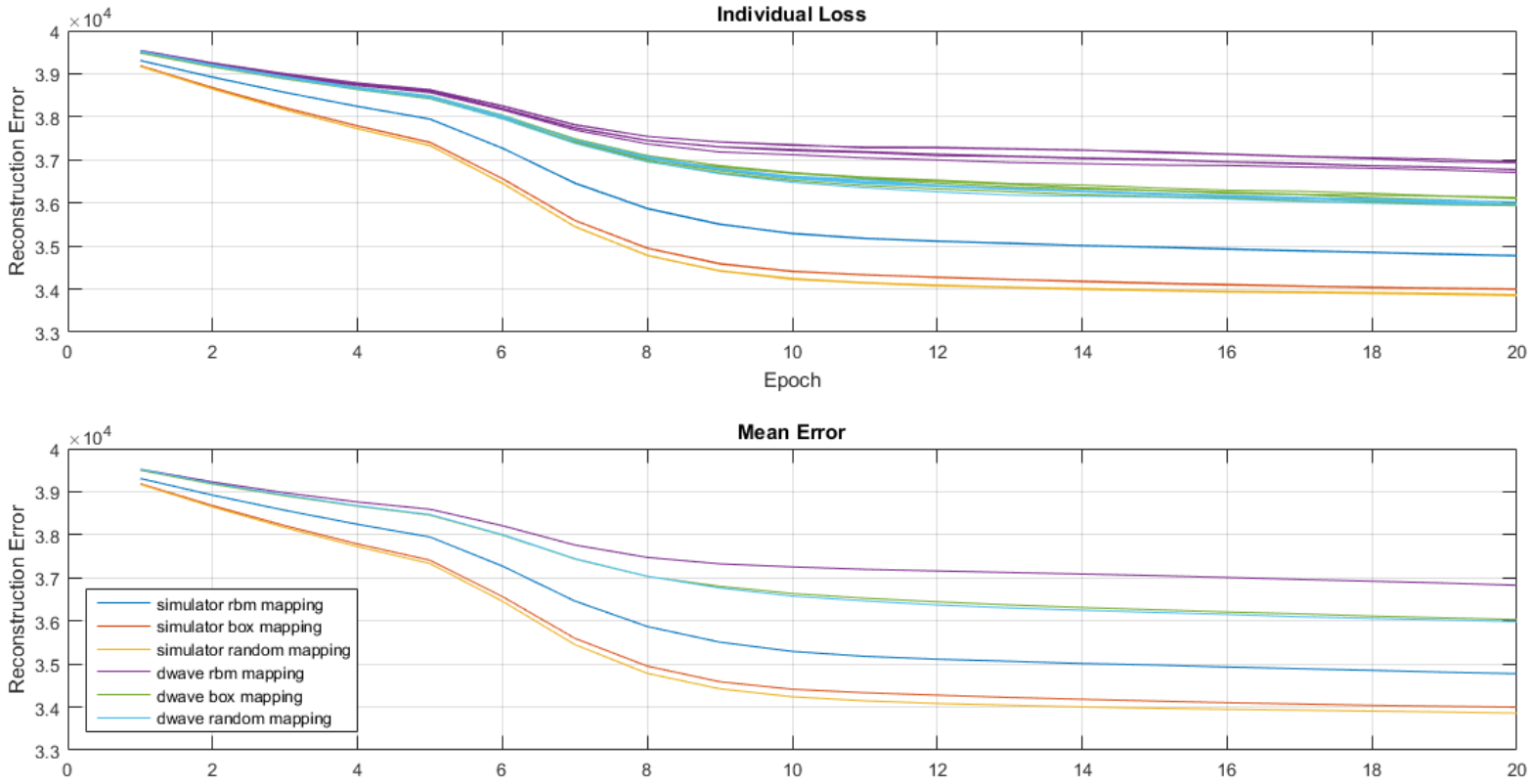


Figure 3.11: A comparison of different qubit mapping methods on both software simulator and the D-Wave processor. A plain RBM run on each platform is shown as baseline and are the weakest performers. The initial qubit-mapping scheme was the "box" mapping where qubits that cover adjacent pixels were grouped together in chimera cells. Some random mapping, featuring mixes of spatially adjacent qubits and more global connections, outperform the box mapping.

course, as we have been discussing up to now, our adiabatic annealing environment implements Boltzmann machines to model several data sets.

3.7.1 HPC Environment

We demonstrated [38] that improved network hyper-parameters can be found by using an evolutionary algorithm [59] and the Titan supercomputer, a collection of 300,000 cores and 18,000 Nvidia Tesla K20x GPUs. These results demonstrated that near optimal hyper-parameters for CNN architectures can be found for the MNIST handwritten digit dataset by combining evolutionary algorithms and high performance computing. The kernel size and the number of hidden units per layer were the hyperparameters that were optimized. This work utilized 500 nodes of Titan for 3 hours in order to evaluate $16,000$ hyperparameter sets.

An improved version of the aforementioned evolutionary algorithm has been developed such that not only can hyper-parameters of a fixed topology be optimized, but the topology of the network itself can be optimized [58]. This improved algorithm can evolve the number of layers and the type of each layer in addition to each individual layer's hyperparameters. This work has been applied to the MINERvA vertex reconstruction problem, which we have referred to as the neutrino particle detection problem in this paper, and has yielded improved results over standard networks. This approach is able to achieve an accuracy of 82.11% after evaluating nearly $500,000$ networks on Titan in under 24 hours utilizing $18,000$ nodes of Titan, which represents a significant improvement over the baseline network that achieved 77.88%. Manually designing a network to attain such an improvement could take weeks or months due to the limited ability of a human to design, evaluate, and interrogate the performance of their networks in order to propose improved designs.

These HPC results are relevant to our quantum annealing approach because efforts to apply AQA to deep learning networks can benefit from this ability to pick good hyper-parameters. When

we designed our RBM and LBM experiments, we manually chose learning rates and topologies. Future work can incorporate our HPC findings here to find optimal hyper-parameters for our deep learning networks before using a quantum annealer to further tune the networks. Our LBM experiment where we first trained a RBM on a classical machine before moving it to the annealer and adding intralayer connections seems particularly amenable to such a procedure.

Convolutional Neural Networks

We now discuss convolutional neural networks, or CNNs. These are highly suitable for implementation in HPC environments because they are massively parallel by design.

Our previously adiabatic quantum annealing approach relies on usage of Boltzmann machines, described in Chapter 2, and can be considered a type of deep neural network. CNNs are another type of deep learning network, and they have become widely used for analyzing image data [32]. As with other deep learning networks, CNNs contain many layers of neural units with many connections between different layers but no connections between units of a particular layer. They also use standard stochastic gradient descent and back-propagation combined with labeled data to train. What separates a CNN from other networks are its unique connectivity arrangement and different types of layers. See Figure 3.12 for a high-level diagram of the CNN architecture.

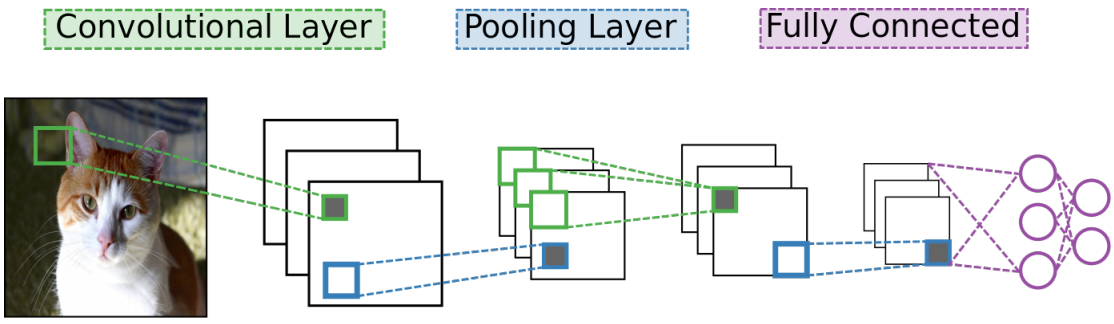


Figure 3.12: A convolutional neural network is composed of a series of alternating convolutional and pooling layers. Each convolutional layer extracts features from its preceding layer to form feature maps. These feature maps are then down-sampled by a pooling layer to exploit data locality. A perceptron [40], a simple type of classification network, is placed as the last layer of the CNN.

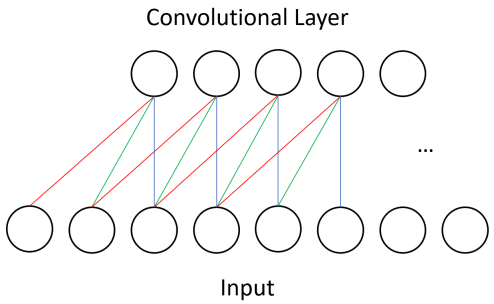


Figure 3.13: The connectivity in a CNN is sparse relative to the previously shown BM model. Additionally, the set of weights is shared between units, unlike in BMs. In this illustration we symbolize this with the red, green, and blue connections to show that each unit in the convolutional layer applies the same operation to different segments of the input.

One type of layer in CNNs is the convolutional layer. Unlike in other neural networks, a convolutional layer uses a kernel, or small set of shared weights, to produce a feature map of the input to the layer, and many convolutional layers operate in succession. Other networks would typically have every input unit connected to every processing unit in a layer whereas a CNN is satisfied with using convolution to produce sparse connections between layers - see Figure 2.2 for the relatively denser connectivity of a BM[7] and compare it against the sparse CNN connectivity shown in Figure 3.13. A kernel captures some feature from the input, and convolving a kernel with the data finds this feature across the whole input. For example, a kernel that detects diagonal lines can be convolved with an image to produce a feature map that can be interpreted as identifying all areas of an image that contain diagonal lines.

The second type of layer is the pooling layer. Pooling layers use the many feature maps produced by convolutional layers as input and subsample them to produce smaller feature maps to help take advantage of data locality within images. CNNs use alternating layers of convolutional and pooling layers to extract and abstract image features. Pooling operations makes feature detection in CNNs resilient to position shifts in images [46].

[7]Figure 2.1 features even denser connectivity than RBMs, but only RBMs are used in practical applications.

3.7.2 Neuromorphic Environment

A neuromorphic approach to computing offers the potential of low-power implementations of networks derived from the AQA and HPC portions of our work. AQA needs hardware to be cooled as much as possible and HPC needs thousands of CPUs/GPUs. The power consumption of either is far beyond what a neuromorphic solution requires to function.

For our neuromorphic comparison points we considered a two-phase experiment. The initial phase was to demonstrate the feasibility of a native spiking neuromorphic solution by implementing a SNN in a software-based simulation. The next phase was to collect energy estimates by simulating the characteristics of the corresponding SNN implemented on memristive neuromorphic hardware. In previous work [38] for the MNIST task we started by simulating a simple spiking neural network trained to classify MNIST images.

We used evolutionary optimization (EO) to generate an ensemble of networks that classified MNIST images with an accuracy of approximately 90%. The accuracy of the generated ensemble was comparable to some other non-convolutional spiking neural network approaches [15]. The network we considered for this experiment was one network in the ensemble. In particular, the network we chose is one that distinguishes between images of the digit 0 and images of other digit types. For the second phase of the experiment the energy consumption was also determined for a memristive implementation of this network. Here the synapses consisted of metal-oxide memristors and represented both a weight value and a delay value. Each synapse in the network had twin memristors to implement both positive and negative weights [45] and a synaptic buffer to control the delays and peripheral connections. The neurons used in the network are implemented using the mixed-signal integrate and fire approach.

The simulation of energy estimate leveraged the energy per spike values for each synapse and neuron phases gathered from low-level circuit simulation. The network was simulated with a clock speed of 16.67 MHz and the average power and energy calculated for the network was 304.3mW and

18.26nJ. We note that this estimate includes the digital programmable delays as well. However, if we consider the core analog neuromorphic logic, the energy per spike is 5.24nJ and the average power was 87.43mW, which is consistent with similar memristor-based neuromorphic systems [34]. In contrast, MNIST classification tasks on GPU, FPGA (field-programmable gate arrays), or even ASIC (application-specific integrated circuit) architectures were reported to be in the W range [18], far above neuromorphic implementations like the one we described or IBM's TrueNorth [57].

In previous work [47] we also applied this approach to estimating the energy usage of a memristive based implementation on the Fermi data. As opposed to the MNIST task in which we trained multiple SNNs to form an ensemble, we built a single SNN for the neutrino data with 50 input neurons and 11 output neurons where the 11 output neurons corresponded to the 11 class labels in the neutrino data. We used a single view of the data (the x-view) rather than all three views. Instead of interpreting the data as pixels in an image we utilized the time lattice of the data. In the time lattice each value in the x-view corresponds to the time at which the energy at that point exceeded a low threshold. We used these times to govern when spikes should appear as input in the SNN. This generated a natural encoding for SNN-style networks as opposed to the somewhat unnatural mapping of non-temporal data to an image format. We found a resulting network with 90 neurons and 86 synapses that reached approximately 80.63% accuracy on the testing set, comparable to the approximately 80.42% accuracy achieved by a CNN that was also restricted to the x-view [52]. We estimated the energy usage of a memristive based neuromorphic implementation of the network for the neutrino data to be approximately 1.66 μJ per classification. These results, more so than the MNIST results, demonstrate that leveraging the temporal nature of certain data may result in extremely efficient SNN solutions to certain tasks.

Spiking Neural Networks

SNNs differ from both BMs and CNNs by incorporating the extra dimension of time into how information is processed, making them suitable for implementation in a neuromorphic computing environment. BMs and CNNs do not have a sense of time built into their architectures - neural unit activity is iteratively calculated on a layer-by-layer basis. SNNs instead use integrate-and-fire neurons, units that collect activation potential over time and fire or "spike" upon reaching a threshold, after which they cannot fire during what is known as a refractory period. Additionally, synapses in an SNN can include programmable delay components, where larger delay values on the synapse correspond to longer propagation time of signals along that synapse. Additionally, there is not necessarily a division of units into well-organized layers in an SNN, and input is fed to the network over time.

SNNs have great potential in moving away from the traditional implementation of machine learning algorithms on the CPU of a von Neumann architecture. For example, the CPU/memory model, while useful on many diverse applications, has the drawback of high power requirements. Nature's biological neural networks have extremely low power requirements by comparison. There are many different ways to implement neuromorphic systems, but one of the more promising device types to include in neuromorphic systems is memristors. Development of memristive technology opens the potential of running spiking neural networks using low power consumption on neuromorphic architectures.

A key challenge associated with SNNs in general and SNNs for neuromorphic systems in particular is determining the correct training or learning algorithm with which to build the SNN. Though there have been efforts to map existing architectures like CNNs to equivalent spiking neuromorphic systems [17, 29], there is also potential to develop independent deep learning architectures that exploit the temporal processing power of SNNs.

3.7.3 Challenges

Complex networks pose enormous problems for deep learning, three of which we will identify. How we tackle each of these challenges is the basis of our collaboration with Oak Ridge National Laboratory (ORNL), where we seek relief from these issues through quantum adiabatic annealing, high performance computing, and neuromorphic computing.

The first of these challenges comes from complex network topology in neural networks. By complex network topology we mean bidirectional connections and looping connectivity between neural units, which slow training to a crawl. The training algorithms we know for such complex networks have greater than polynomial runtime, making them effectively intractable and untenable for practical purposes. Therefore, as described in the previous chapter, deep networks deployed on real-world problems (like the previously discussed CNN architecture) impose limitations on network topology. Removing intralayer connections or enforcing strict rules for network topology allows faster and tractable training algorithms to run. However, doing so takes away some of the representational power of the network [56], and these restricted or limited networks do not reflect models found in nature. While tractable models perform remarkably well on specialized classification tasks, we speculate that other more complex and generalized tasks may benefit from the additional representational power offered by complex networks. We believe quantum adiabatic computing offers part of a potential solution through its ability to sample from complex probability distributions such as those generated by neural networks containing intralayer connections.

The second challenge is automatically discovering optimal or near-optimal network hyperparameters and topologies. Hyperparameters in deep learning refer to the model parameters, i.e., the activation function used, the number of hidden units in a layer, the kernel size of a convolutional layer, and the learning rate of the solver. Currently the best deep learning models are discovered by creating, training, testing, and tuning many models on some well-known reference dataset and reporting the best model in the literature. But if the dataset has not been examined before, it is

difficult to know how to tune networks for optimal performance. GPU-based high performance computing provides an opportunity to automate much of this process - to train, test, and evolve thousands of deep learning networks to find optimally-performing network hyperparameters and network topologies.

The last challenge is power consumption, which we can help address through neuromorphic computing. Machine learning's computational needs have so far been met with power-hungry CPUs, and more recently GPUs. The switch from CPUs to GPUs has significantly sped up computation and lowered computation costs, but GPU efficiency in training networks still pales in comparison to the efficiency of biological brains. For an image recognition task, it might take many server farms and a hydroelectric dam to compete with a mundane human brain running on a bit of glucose. Neuromorphic computing offers a potential solution by developing specialized low-power hardware that can implement SNNs approximating trained networks derived from more orthodox architectures.

We focus on deep learning's challenges related to quantum adiabatic computing. Though high performance and neuromorphic computing are investigated by many other researchers, we generally only dive deeply into adiabatic quantum annealing and its application to Boltzmann machines. Our experiments use the MNIST dataset [33], an image dataset of handwritten digits compiled by NIST, and a neutrino particle detection dataset produced by Oak Ridge National Laboratory.

3.7.4 Summary and Discussion

We compared a standard benchmark problem, MNIST digit recognition, on three different platforms: quantum adiabatic optimization, HPC, and neuromorphic. Our results show each option offers a unique benefit. Quantum adiabatic computation opens up complex topologies for use in deep learning models that would normally prove intractable for classical machines. HPC allows

us to optimize CNNs on a large scale to find an optimal topology with its associated parameters. Neuromorphic lets us implement low power neural network solutions derived from other platforms. However, it is also clear that the MNIST problem is not ideally suited to showcase the capabilities of either the quantum or neuromorphic systems because it has been essentially solved using CNNs.

For example, the greater representational power of the quantum LBM approach is likely better utilized on a more complex dataset. Similarly, spiking neuromorphic systems may be better suited for use on datasets that include temporal components. In Figure 3.15 we propose an architecture we believe provides the ability to leverage the strengths of each of these computing platforms for future, more complex data sets.

The goal of this survey is to explore how to address some of the current limitations of deep learning, namely networks containing intralayer connections, automatically configuring the hyperparameters of a network, and natively implementing a deep learning model using energy efficient neuron and synapse hardware. We used quantum annealing, high performance computing, and neuromorphic computing to address these issues using three different deep learning models (LBM, CNN, and SNN).

The quantum adiabatic computing approach allows deep learning network topologies to be much more complex than what is feasible with conventional von Neumann architecture computers. The results show training convergence with a high number of intralayer connections, thus opening the possibility of using much more complex topologies that can be trained on a quantum computer. There is no time-based performance penalty due to the addition of intralayer connections, though there may be a need to sample more often in order to reduce potential errors.

HPC allows us to automatically develop an optimal network topology and create a high performing network. Many popular topologies used today are developed through trial and error methods. This approach works well with standard research datasets because the research community can learn and publish the topologies that produce the highest accuracy networks for these

data. But when the dataset is relatively unknown or not well studied, the trial-and-error approach loses its effectiveness. The HPC approach provides a way to optimize the hyper-parameters of a CNN, saving significant amounts of time when working on new datasets, perhaps even bootstrapping under-studied datasets into the regular publish-and-review iterative process.

Memristor-based hardware provides an opportunity to natively implement a low-power SNN as part of a neuromorphic computing environment. Such a network has the potential to feature broader connectivity than a CNN and the ability to dynamically reconfigure itself over time. Neuromorphic computers' benefits, including robustness, low energy usage, and small device footprint, can prove useful in a real-world environment today if we develop a mechanism for finding good network solutions for deployment on memristor-based devices that do not rely on conversions from non-spiking neural network types.

Platform	Algorithm	Contribution	Significance
Quantum	Limited Boltzmann	Development of LBM for quantum computers	Increased representational ability for deep learning, potentially tractable for quantum computer
HPC	Convolutional Neural Network	Evolutionary optimization design CNN network	Rapid design of high performing network
Neuromorphic	Spiking Neural Network	Native implementation of SNN on neuromorphic hardware	Very low power implementation of SNN

Figure 3.14: A comparison of the platforms, deep learning approaches, contributions, and significance of the result from the MNIST experiment.

We can use the three different architectures together to create powerful deep learning systems to go beyond our current capabilities. For example, current quantum annealing hardware is limited in the size and scope of problems it can solve but does allow us to use more complex networks. We can turn this into an opportunity by using a complex network as a higher level layer in a CNN as seen in Figure 3.15. Higher layers typically combine rich features and can benefit from increase intralayer connectivity; they also have smaller-sized inputs than lower layers, easing

the limited-scope issue of current quantum annealing hardware. Such an augmented CNN may improve overall accuracy.

The HPC approach of automatically finding optimal deep learning topologies is a fairly robust and scalable capability, though quite expensive in development and computer costs. The ability to use deep learning methods on new or under-studied datasets (such as the neutrino particle detection dataset) can provide huge time savings and analytical benefit to the scientific community.

The neuromorphic approach is limited by the lack of robust neuromorphic hardware and algorithms, but it holds the potential of analyzing complex data using temporal analysis using very low power hardware. One of the most compelling aspects of this approach is the combination of a SNN and neuromorphic hardware that can analyze the temporal aspects of data. The MNIST problem does not have a temporal component, but one can imagine a dataset that has both image and temporal aspects such as a video or our neutrino detection dataset. A CNN approach has been shown to perform well on the image side, so perhaps a SNN can provide increased accuracy by analyzing the temporal aspects as well.

For example, a CNN could analyze an image to detect objects within the image and output the location and/or orientation of those objects. This output can be used as input for an SNN. As each video frame is processed independently by the CNN, the output can be fed into the SNN, which can aggregate information over time and make conclusions about what is occurring in the video or detect particular events that occur over time, all in an online fashion. In this example the CNN could be trained independently using the labeled frames of the video as input images while the SNN could be trained independently utilizing different objects with their locations and orientations as input.

These experiments provide valuable insights into deep learning by exploring the combination of three novel approaches to challenging deep learning problems. We believe that these three architectures can be combined to gain greater accuracy, flexibility, and insight into a deep learning approach. Figure 3.15 shows a possible configuration of the three approaches that addresses the

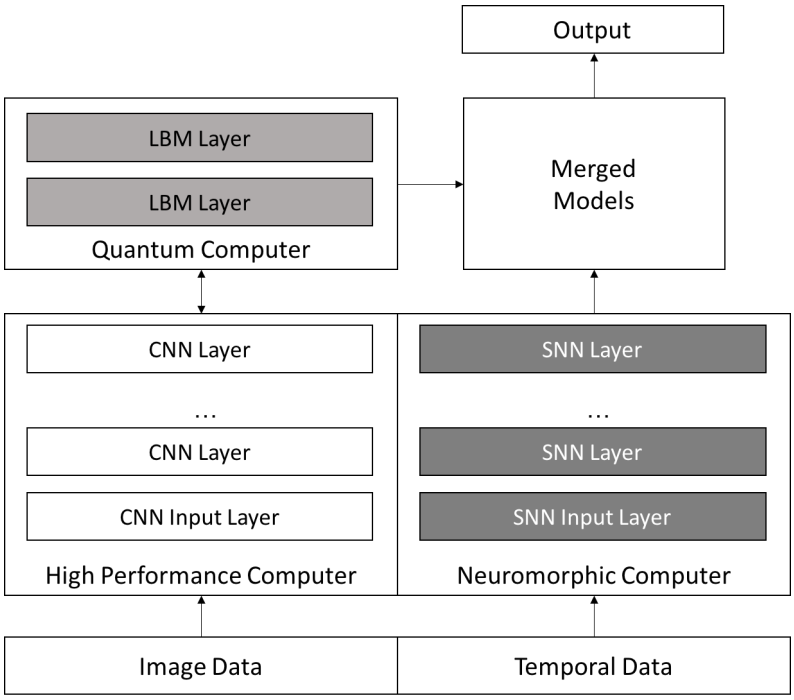


Figure 3.15: A proposed architecture that shows how the three approaches, quantum, HPC, and neuromorphic can be used to improve a deep learning approach. Image data can be analyzed using an HPC rapidly derived CNN with the top layers using an LBM on a quantum computer. The top layers have fewer inputs, and require greater representational capabilities which both play to the strength and limitations of a quantum approach. The temporal aspect of the data can be analyzed using a SNN, finally the image and temporal models will be merged to provide a richer and we believe a more accurate model, with an aim to be deployed in very low power neuromorphic hardware.

three deep learning challenges we discussed above. The high performance computer is used to create a high performing CNN on image type data. The final layer or two is then processed by the quantum computer using an LBM network that contains greater complexity than a CNN. The temporal aspects of the data are modeled using a SNN, and the ensemble models are then merged and an output produced. Our belief is that this approach has the potential to yield greater accuracy than existing CNN models.

Though inspired by biological neural models, deep learning networks make many simplifications to their connectivity topologies to enable efficient training algorithms and parallelization on GPUs. CNNs in particular have emerged as a standard high performance architecture on tasks such as object or facial recognition. While they are powerful tools, deep learning still has several

limitations. First, we are restricted to relatively simple topologies; second, a significant portion of network tuning is done by hand; and third, we are still investigating how to implement low power, complex topologies in native hardware.

We chose three different computing environments to begin to address the issues respectively: quantum adiabatic computing, high performance computing clusters, and neuromorphic hardware. Because these environments are quite different, we chose to use different deep learning models for each. This includes Boltzmann machines in the quantum environment, CNNs in the HPC environment, and SNNs in the neuromorphic environment. We chose to use the well-understood MNIST hand-written digit dataset and a neutrino particle detection dataset.

Our results suggest these different architectures have the potential to address the identified deficiencies in complex deep learning networks that are inherent to the von Neumann CPU/memory architecture that is ubiquitous in computing.

The quantum annealing experiment showed that a complex neural network, namely one with intralayer connections, can be successfully trained on the MNIST digit recognition and neutrino particle detection tasks. The ability to train complex networks is a key advantage for a quantum annealing approach and opens the possibility of training networks with greater representational power than those currently used in deep learning trained on classical machines. High performance computing clusters can use such complex networks as building blocks to compare thousands of models to find the best performing networks for a given problem. And finally, the best performing neural network and its parameters can be implemented on a complex network of memristors to produce a low-power hardware device capable of solving difficult problems. This is a capability that is not feasible with a von Neumann architecture and holds the potential to solve much more complicated problems than can currently be solved with deep learning on classical machines.

We proposed a new deep learning architecture based on the unique capabilities of the quantum annealing, high performance computing, and neuromorphic approaches presented in this paper. This new architecture addresses three major limitations we see in current deep learning methods

and holds the promise of higher classification accuracy, faster network creation times, and low power, native implementation in hardware.

Chapter 4

Finding Better Qubit Mappings

Previous results show that some mappings perform better than others. Given that better mappings exist, we naturally ask the question of how we might find those mappings. Previously we had relied upon intuition and knowledge of past problems that let us know locality-exploiting mappings would produce decent results. What we seek is a more disciplined and objective approach for finding these good qubit mappings, and a mathematical explanation that may explain our results.

Section 4.1 discusses static qubit mappings - mappings that are determined early in training and remain unchanged afterwards. Section 4.2 then describes how we utilized correlation to design better qubits mappings in a more disciplined manner. Section 5 moves on to examine dynamic qubit mappings - mappings that are evaluated and changed throughout training - and how we used entropy to create better performers. In Section 6, we offer a mathematical explanation for how entropy influences qubit mapping quality and draw connections to similar work conducted by other researchers.

4.1 Static Qubit Mappings

We will present the evolution of our work since obtaining the initial results in previous chapters regarding qubit mapping. First we offer a quick review of our qubit mapping topic.

Our initial implementation of Boltzmann machines on D-Wave's adiabatic quantum annealer reduces the visible unit states and their connections to hidden units to biases on the hidden units. The hidden units, meanwhile, are represented by qubits on the annealer and we allow them to utilize the couplings between qubits to represent connections between hidden units. This networked connectivity between qubits generates a complex unit state distribution that would require expensive MCMC methods to simulate classically. But since we use a quantum annealer, we can directly sample from the distribution to carry out our Boltzmann machine training procedure.

Previous chapters showed our work up to this point when we first began to consider how we would map logical qubits to hardware qubits[1]. Recall that a logical qubit is the index we assign to a hidden unit. A hardware qubit, on the other hand, is the annealer's qubit we choose to represent the logical qubit. The choice of which hardware qubits we use to represent our logical qubits is important because the hardware qubits are restricted in their connectivity options. Not only is a hardware qubit restricted to a maximum of 6 connections, its connections are restricted to its immediate neighbors in physical space. So when we make decisions about mapping logical qubits to hardware qubits, we cannot treat the hardware qubits as interchangeable. Each one is uniquely positioned in space and connectivity options.

Figures 3.7 and 3.8 show situations in which we do not consider doing anything special for our logical-to-hardware qubit mapping. That is, if we have a qubit logically indexed as the ith qubit, we assign the ith hardware qubit to represent it on the annealer. This is the most straightforward way to create a mapping and does not require any additional work or thought. When we generated those figures using LBMs trained on the neutrino scattering data set, it was sufficient to outperform RBMs, but we believed they way in which we assigned logical qubits to hardware qubits should have a significant impact on overall LBM performance. Our results shown in Figure 3.11 support this belief. Besides noting that our choice of qubit mappings affects results,

[1]From here on out we generally refer interchangeably to qubits and hidden units. They represent the same computational unit so we conflate the terms.

an important observation to take from this figure is that our attempts at manually creating a static qubit mapping were outperformed by creating random mappings. On the one hand, it was good to see confirmation of our hypothesis that qubit mapping schemes are worth investigating; on the other hand, our first thoughtful attempt at creating a mapping lost to a random method.

Nevertheless we were motivated by this observation and sought to develop better performing qubit mapping policies. The earliest of these policies we call static maps because once we create a map, it remains unchanged throughout the remainder of BM training. Later methods allow for dynamic mapping based on changing conditions, but for now we cover just the static methods.

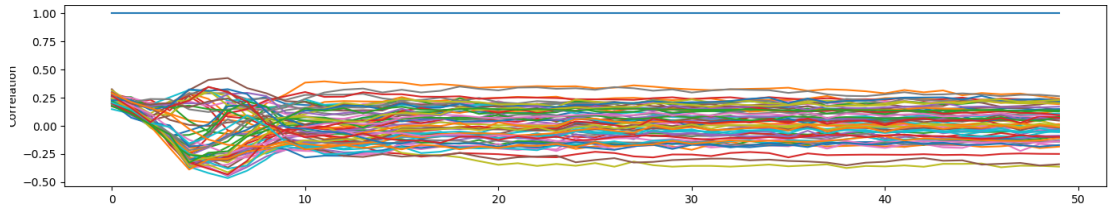
4.2 Correlation

The creation of line and box qubit mappings by hand relied on assumptions we could make due to domain knowledge of the data set. For instance, we know that spatial locality is important to image data, hence the focus on lines and boxes. But we wanted to go beyond manually creating qubit mappings and develop a more coherent policy based on some sort of metric that can be generalized. We decided that focusing on correlations between qubit activity could be a first path forward; eventually we would conclude that maximizing correlation within our chimera cells would produce the best results.

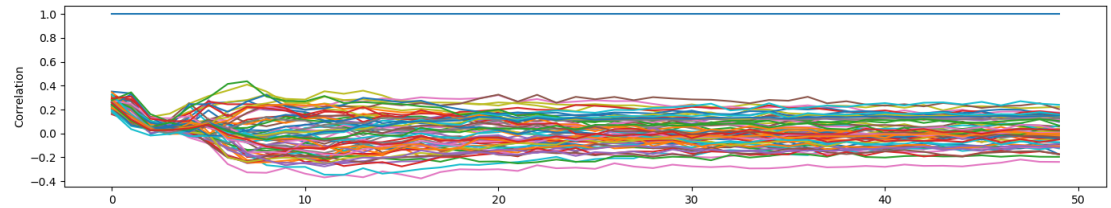
In order to calculate how qubits were correlated with each other based on activation, we recorded the hidden states of a Boltzmann machine induced by the BM's exposure to our input data patterns. This record of hidden unit states was the basis for generating our matrix of qubit correlation values. We used Pearson's correlation coefficient, which ranges within $[-1, 1]$ and is defined:

$$\rho_{X,Y} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} \tag{4.1}$$

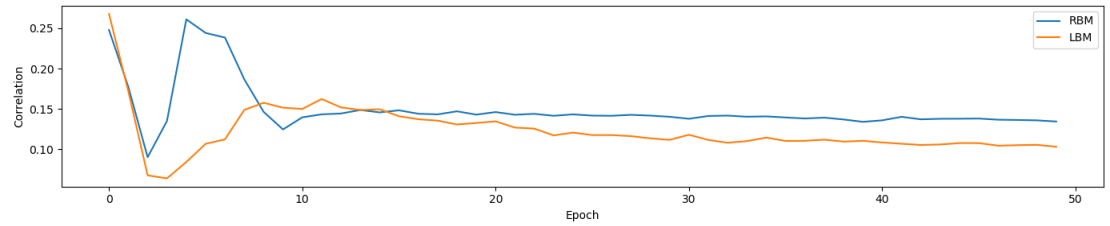
The random variables X and Y represent any pair of qubits in our Boltzmann machine. Equipped with this notion of correlation calculation, our immediate first step was to simply observe how correlation behaved in our Boltzmann machines. The idea was to train a BM as usual and additionally calculate a matrix of correlation coefficients after every training epoch. Then, for each qubit q, we could plot q's correlation with with all other qubits in the BM as training proceeded.



(a) Correlation values for the 20th qubit across training epochs in a RBM modeling MNIST digit data. Initially correlations are close together because random weights lead to random activity in the hidden units when exposed to input data. As training proceeds, we see the range of correlation values widen and settle around epoch 10.



(b) Correlation values for the 20th qubit across training epochs in a LBM modeling MNIST digit data. We broadly see the same behavior as in the RBM.



(c) The mean magnitude of all correlation values for the 20th qubit. Correlation magnitude rises early in training but begins a long downward trend.

Figure 4.1: Correlation values for RBM and LBM. These show results from examining one arbitrary qubit at index 20.

Figure 4.1 shows a sample of correlation observations. We calculated our matrix of correlation coefficients as described after each training epoch and plotted the results stemming from one arbitrary qubit. The correlation values behaved reasonably and within expectations. Initially

correlation magnitudes were clumped together due to a randomized initialization of parameter weights - random initialization should create random activity and subsequently little correlation. And as training progressed, the range of correlation values expanded, which also seemed natural because the point of training is to have a BM react selectively to input patterns.

The first trend we noticed was the settling of correlation values. Correlation magnitudes rapidly diverged in the first few epochs of training, some even completely reversing from correlated to anti-correlated. After around epoch 10, however, the correlation magnitudes seemed relatively much more stable. Our plot of mean magnitudes in Figure 4.1 shows a rather dramatic plateau at epoch 10 for the RBM.

The second trend was the downward shift of correlation magnitude as training continued past epoch 10. The initial few training epochs can be seen as chaotic since the BMs are still trying to find a foothold in parameter space. Once that foothold is found, however, the rest of training can be viewed as fine-tuning. We believed this might have happened around epoch 10 (Figure 3.11 also sees most of its improvement prior to epoch 10). Regardless of when this switch typically occurs in training, we were interested in where BM training would eventually lead our qubit correlations. Cursory examination of the plot showed that the longer we were to train (and achieve lower error values), the lower our correlation magnitudes would become. We interpreted this as a signal that overall correlation magnitude had some impact - whether good or bad - on BM performance.

Given this interpretation that correlation among hidden units could affect the results of our BM, we wanted to design a qubit assignment policy based o then magnitudes of correlation coefficients. A BM with k hidden units has $\mathcal{O}(k^2)$ correlation coefficients, but a quantum annealer only has $\mathcal{O}(k)$ couplers available to enable interaction between hidden units. In order to implement a BM on annealing hardware we would be forced to choose a subset of interactions to enable, therefore we should choose to enable the most beneficial interactions. Combined with our observation that BM performance seems to influence or be influenced by correlation magnitude, we created the following remapping procedure:

1. Find an available (unassigned) pair of qubits that have the highest correlation coefficient.

2. Place the qubits in the opposing partitions of an unused chimera cell.

3. For each of these "seed" qubits, find the 3 other qubits most correlated with the seed and place them in the opposing partition.

4. Repeat until all qubits and chimera cells are assigned.

We started with a RBM and partially trained it for 10 epochs; we chose 10 epochs due to our observation that qubit correlations seem to settle around that time for our choice of hyper-parameters on the MNIST digit data set. After performing this weak remapping procedure, we then converted the RBM to a LBM[2] using the new mapping and continued training. Our results showed that our efforts created a performance improvement as we had hoped for.

Figure 4.2 shows a plot comparing the performances[3] of different qubit mappings. As we had guessed, attempts to influence the correlation between qubits within a chimera cell did alter the performances of the Boltzmann machine.

In addition to the mapping procedure we just described, we also developed a variant policy and tested it on a new data set, a MoS_2 thermalization data set, to both check if our results held up across data sets and to have another policy to compare against. MoS_2 is a molybdenum-sulfide monolayer that produces a variety of different structures when thermalized (or burned). The thermalization process can be simulated and a data set of 3-dimensional spatial coordinates for each atom produced. We took this coordinate data set and projected it down into 2-dimensional image slices amenable to processing by a Boltzmann machine.

[2]Conversion of RBM to LBM is straightforward. Recall that LBM connectivity is a superset of RBM connectivity, so all we need to do is add previously non-existent couplers to our BM. The only thing a mapping affects is our choice of which hidden-to-hidden connections (many) get the chance to be represented by a physical hardware couplers (few).

[3]The figure shows that minimizing correlation, the "minCorr" policy, appears to work best, whereas we had originally stated that maximizing correlation was the better choice. Both are true, in a fashion. Minimizing correlation works the best when we have only a few (less than 50) epochs of training. But as we will see in a later experiment, extending training beyond 50 epochs allows a maximal correlation policy to eventually catch up and surpass a minimum correlation policy.

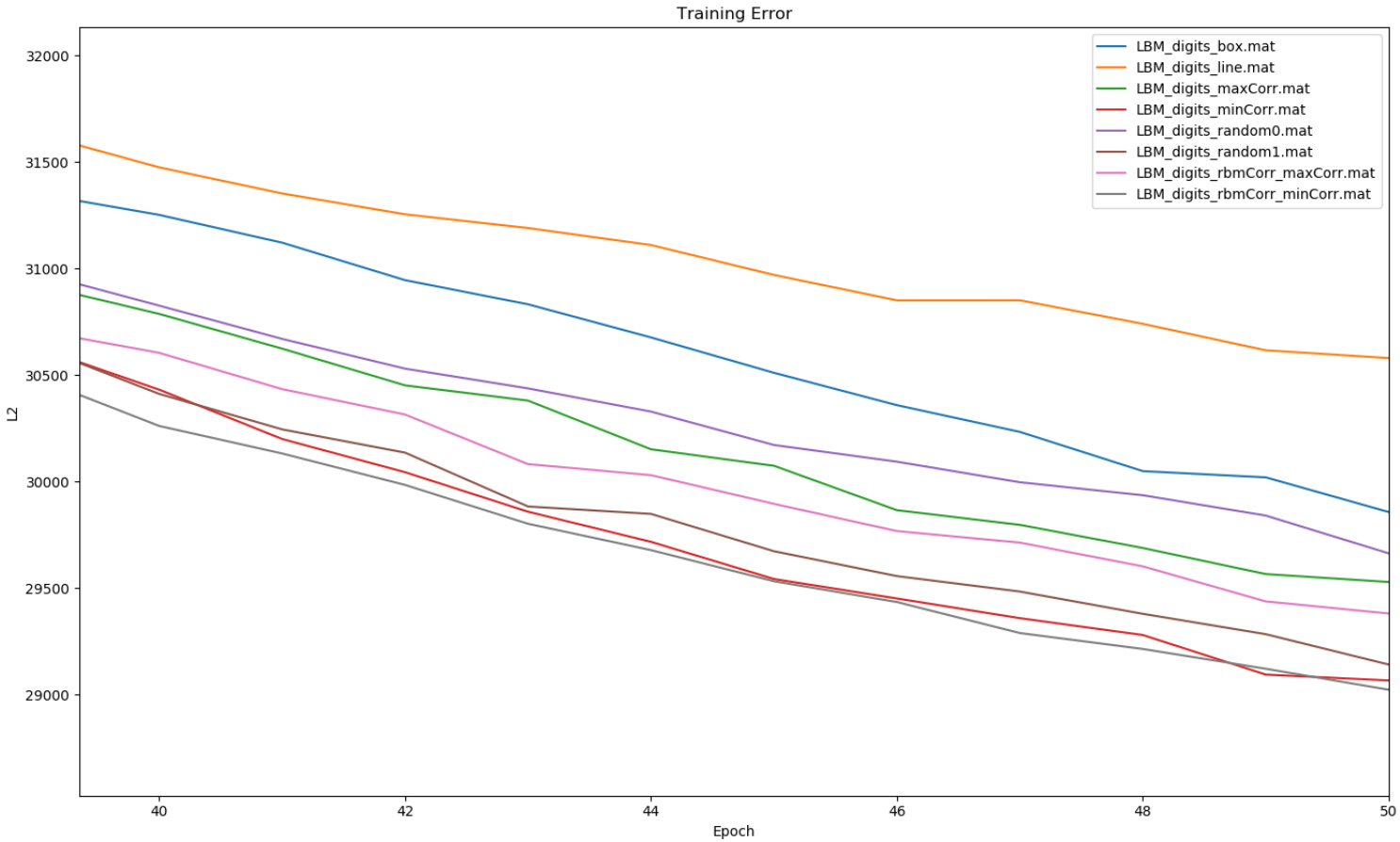


Figure 4.2: Terminal results of training a LBM using a qubit mapping based on varying correlation within chimera cells. We included our previous box and line mappings as a baseline to compare against. As previously noted, random mappings were able to outperform those box and line mappings, and two such random mappings are shown. Results labeled *rbmCorr* are produced via RBM-to-LBM conversion. Otherwise, the results are produced by generating correlations as a LBM (as opposed to a LBM) then remapping qubits.

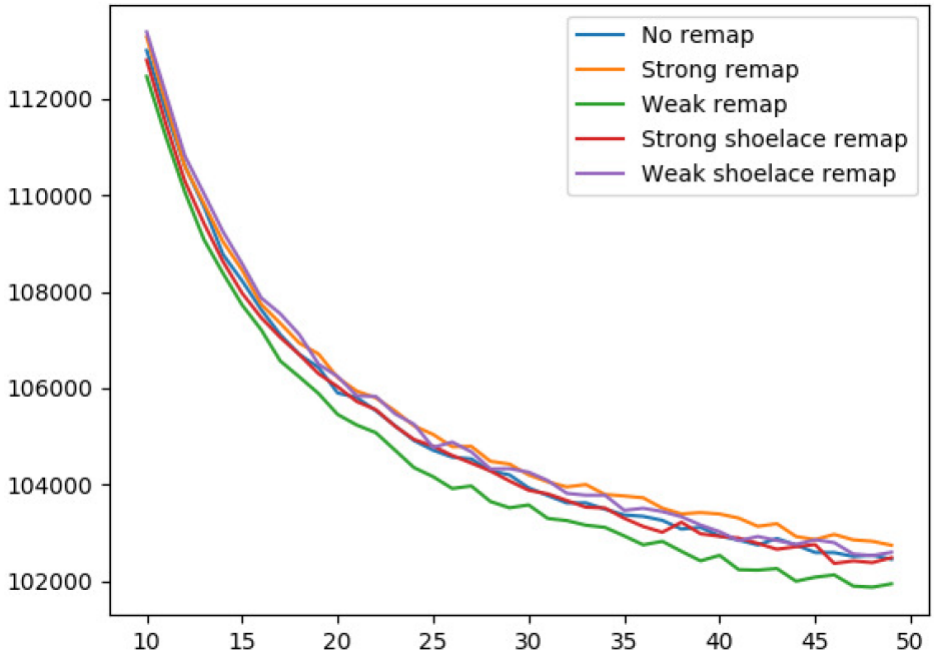


Figure 4.3: A comparison of different policies on a new MoS_2 data set. The y-axis is reconstruction error and the x-axis is training epochs. Our original policy is the "weak" remapping that places uncorrelated qubits together in the same chimera cell. The opposite policy, the "strong" remapping, places highly correlated qubits together. The variant "shoelace" policies achieve the same aims albeit in a slightly different manner. As shown, the results on the new data set align with our initial results on the MNIST digits data set. When trained for less than 50 epochs, our original "weak" policy is the best performer, the opposite "strong" policy the worst, and all other policies (including choosing not to remap at all) are in the middle.

Figure 4.3 contains the results from our experiments on the MoS_2 data set. The BMs were trained for 10 epochs before qubits were remapped according to different policies. Notably, we had a remapping policy that beat out the choice of forgoing remapping altogether. Up until this experiment we had been mostly concerned with how policies such as box mapping, line mapping, random mapping, or weak mapping compared against each other. With this experiment, we decided to also include the sensible baseline experiment of choosing not to remap any qubits and have the logical qubit indices be the same as the hardware indices. The results fortunately suggested that using correlation to guide qubit remapping policies was capable of producing better outcomes for our Boltzmann machines.

We earlier mentioned a variation of our remapping policy and show its results in figure 4.3. We now describe this alternate policy, which we call a "shoelace" policy:

1. Find an available pair of qubits, $\mathbf{x}$ and $\mathbf{y}$, that have the highest correlation coefficient.
2. Place $\mathbf{x}$ and $\mathbf{y}$ in the opposing partitions of an unassigned chimera cell.
3. Find the qubit with the greatest correlation with $\mathbf{x}$ and place it in the partition opposite of $\mathbf{x}$; this qubit is the new $\mathbf{x}$. Do likewise with $\mathbf{y}$.
4. Repeat step 3 until the chimera cell is full.
5. Repeat until all qubits and chimera cells are assigned.

This alternate policy did not perform as well as our original policy, but we created it mainly as a comparison point. It, alongside comparisons to box and line mappings, reinforced our belief that qubit mapping policies do have a significant effect on the eventual training outcome of Boltzmann machines. Although these correlation-based qubit mapping policies were a nice starting point for us, we recognized that our view of the Boltzmann machine connectivity was quite limited.

Chapter 5

Dynamic Qubit Remapping With Entropy

We used correlation to develop our first policies, but our policy objectives were always rather nebulous and ambiguous. We had a vague idea that we wanted to influence correlation within a BM, but we never did develop a good way to objectively measure "how much" correlation was in our network or how it was reduced. At any given time we were only considering isolated pairs of qubits. Correlation coefficients were already producing helpful results, but they could only ever give us insight into the pairwise relationship between qubits. This is when we considered how we could expand our view to include all qubits, or at least groups of qubits. Our examination of correlation between qubits let us to an examination of entropy in BM hidden unit activity.

But beyond considering entropy, we also wanted to adjust our qubit remapping workflow. When conducting our correlation experiments, we would train a RBM for a few epochs, calculate the correlation coefficients between qubits, create a new qubit mapping, and resume training as a LBM. We only offered the BM one chance to reconsider its qubit mapping choice before forcing it to commit to a new mapping for the remainder of training. What we wanted instead was a more dynamic process where a BM would be allowed to reevaluate its qubit mapping choice on multiple occasions before being forced to commit to a final mapping. The plot of correlation coefficients in Figure 4.1 showed us that the first few epochs of training were quite volatile, so picking a final

qubit mapping at such a point[1] might miss better opportunities that arise later. By changing our remapping procedure to allow for multiple reevaluations instead of a single reevaluation, we capture these better opportunities. We now show how we use entropy as a metric to guide our remapping decisions.

5.1 Calculating Entropy

When considering the entropy of hidden unit activity patterns with Boltzmann machines, we found it convenient to divide the hidden units into more digestible subgroups of 4 units. This was done because calculating the entropy of all the hidden units together requires exponential amounts of data - if we have 128 qubits, for example, there are 2^{128} possible activity patterns we might observe, so we have to subsequently gather an appropriately large amount of data to ensure we are covering the space adequately. We instead approximate the overall entropy of all the units by calculating the observed entropy for each chimera cell (composed to 2 4-unit subgroups) and summing over all chimera cells. Another reason we had for creating these groups of 4 was due to Hinton et al.'s work on capsules and routing [41] and ver Steeg, et al.'s work on correlation explanation [55], which group together computational units and manipulate their connectivity according to information theoretic approaches. Hinton's work can be seen as a means of encouraging encodings that are more amenable to human interpretation and ver Steeg's work as a means to better optimize the explanatory power of a network. In particular, the latter work concerns itself with how to create latent variables and group them together using a total correlation metric to achieve better results; broadly speaking, a Boltzmann machine has a very similar concept in its hidden layer, which is a collection of latent variables that tries to explain the data distribution it is trained upon. Though these works are not directly applicable to our endeavors here with Boltzmann machines, they

[1] Recall that we trained a RBM for only 10 epochs before choosing a qubit mapping

nonetheless are conceptually motivating and display some shared principles, and they influenced our decision to start by creating 4-unit subgroups corresponding to partitions within chimera cells.

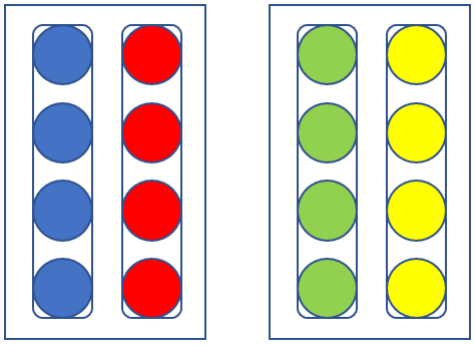
As stated, we calculate entropy on a per-cell basis. Recall that each chimera cell is bipartite in connectivity structure, where each partition is composed of 4 qubits. Supposing that we are given a list of observed hidden unit activities within this chimera cell, it is very straightforward to calculate entropy, defined as:

$$H(X) = \sum_{x \in X} P(x) \log P(x) \tag{5.1}$$

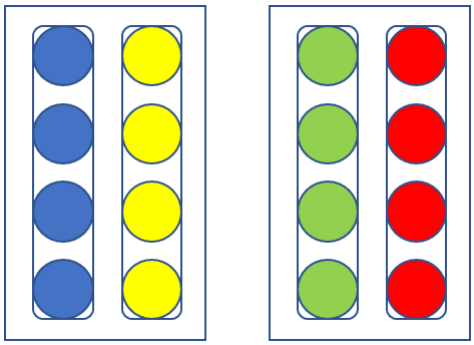
where X would be our distribution of hidden unit states. We obtain $P(x)$ for all $x \in X$ by simply counting the frequency of each activation pattern in the distribution. As for how we obtain the distribution in the first place: we iterate over the data set we train the Boltzmann machine on. After setting the visible units of the Boltzmann machine to an input pattern, we induce a non-deterministic hidden unit activation pattern, which we record. Thus we have one hidden unit encoding for each input pattern[2].

But as we discussed before, using Eq. 5.1 can be troublesome if we have many dimensions in our distribution. Even 128 hidden units is problematic, and future scaling of hardware will make the problem worse. Dividing the hidden unit activation distribution into chunks of 2×4-unit subgroups helps tremendously to mitigate this issue, especially since future scaling most likely involves adding additional cells rather than dramatically expanding the chimera cell itself. With 8 qubits in a chimera cell, we only need to cover 2^8 possible activity patterns, which is entirely manageable. To summarize, for each chimera cell, we filter out all qubits except the ones present within that cell. We then apply Eq. 5.1 to the filtered hidden unit activity patterns to obtain some entropy value. We repeat this for every cell and sum all partial entropy values to obtain a final result.

[2]We could theoretically generate more than one encoding per input pattern due to the probabilistic nature of the Boltzmann machine, but we chose to use only one for convenience's sake.



(a) The hidden units of a Boltzmann machine mapped onto the chimera cells of the D-Wave adiabatic quantum annealer. Here we have a simplified network containing only 2 chimera cells. Note the subgroups, represented by the rounded boxes, are composed of 4 units and that each chimera cell, represented by the angular boxes, contains 2 such subgroups/partitions.



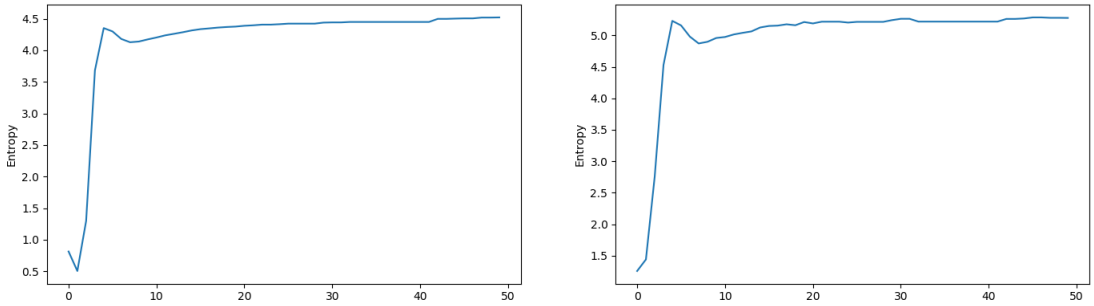
(b) The same network but with subgroups/partitions swapped. The qubits have new neighbors to interact with and new entropy values to calculate.

Figure 5.1: A visual color-coded description of how we remap logical qubits to hardware qubits.

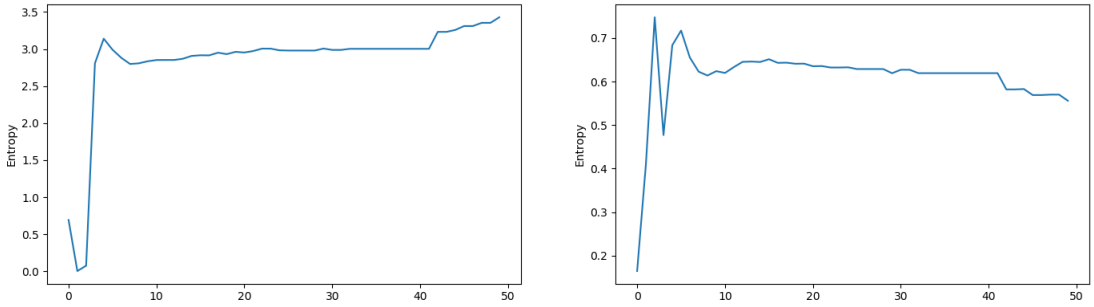
The reason we speak in terms of 4-unit subgroups instead of 8, which might seem natural and fitting for a chimera cell, is that we want to mix and match partitions of chimera cells to manipulate our entropy numbers. See Figure 5.1 for a visual interpretation of the process. In an alternative approach we could try to mix and match individual qubits to create 8-qubit groups, but this generates a factorial number of possibilities to consider. If, instead, we choose 4-unit chimera cell partitions as a sort of quantum, we drastically simplify the accounting we need to perform.

One more simplification we make to this problem is to consider only couplers that exist within a chimera cell. Within a chimera cell there are $4 \times 4 = 16$ couplers owing to the bipartite connectivity between qubits. But the cell also has connections to adjacent cells - each partition

has 4 such connections for a total of 8 per chimera cell. We chose to ignore these adjacent connections to make our mapping efforts easier. However, we will later return to address these adjacent connections and suggest how we can adapt our work to include them and anticipate future topology expansion on new generations of annealing hardware.



(a) Mean and maximum entropy values (y-axis) for chimera cells across training epochs (x-axis)



(b) Minimum and standard deviation values (y-axis) for chimera cells across training epochs (x-axis)

Figure 5.2: Entropy values calculated for a BM trained on MNIST data. Entropy was calculated using the binary state distribution of the 8 qubits within a chimera cell.

Figure 5.2 has the entropy values we calculated for a BM using the method we described. The BM was trained on the MNIST digits data set which naturally has 10 classes of evenly distributed digits, so the entropy of this distribution is around $\log(10) \approx 3.3$. We expect encodings (hidden unit activity) of the digits data to have a somewhat higher entropy value due to variations within a class of digits, and in this case we do have a reasonable match.

With the necessary background information and assumptions we take explained, we now explain the methods we use to generate qubit mappings.

5.2 Dynamic Remapping

Having shown that the way in which we map logical qubits to hardware qubits makes an appreciable difference in LBM results, we now present a method to use our entropy metric to guide our qubit mapping efforts. Our results in Section 5.1 suggested that high entropy in hidden unit activity should be avoided if possible. On the other hand, while low entropy is generally desirable, achieving minimal entropy can lead to negative repercussions. In this section we describe two methods we used to create qubit mappings of varying entropy in hidden unit activity. The first is a simple greedy method, which seems to perform best when we have it find low entropy mappings, and the second is a more complex greedy method that, while producing lower entropy configurations, performs slightly worse.

Common to both methods is a break away from the static method of creating maps. Previously we manually created qubit mappings after considering what domain knowledge about our data set or about the connectivity of our network. Once these maps were set, they could not be altered during the course of training, thus the qubit mapping of the trained network would be exactly the same as the qubit mapping of the initial network. The methods we now use are dynamic in nature: the maps are constantly altered throughout training, and they are altered without any manual input or guidance from an observer or operator who possesses domain knowledge about the data set.

One benefit of this dynamic process is that it opens up an entirely new portion of parameter space for the Boltzmann machine to explore. Previously, with a locked, static qubit mapping, a given qubit was always bound together with the same set of (up to) 6 qubits due to the physical constraints of the D-Wave quantum annealer. Now, however, the ability to remap qubits throughout the training process means that a given logical qubit has many opportunities to interact with qubits it would normally never be able to communicate with.

$w_{0,0}$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$w_{0,N}$
$\cdots$	$\boldsymbol{w_{i,j}}$	$\boldsymbol{w_{i,j+1}}$	$\boldsymbol{\cdots}$	$\boldsymbol{w_{i,j+5}}$	$\cdots$	$w_{i,N}$
$w_{N,0}$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$w_{N,N}$

Figure 5.3: The parameter space qubit i is allowed to explore. Shown is a matrix of weight values. Qubit i's connectivity is limited to merely 6 other qubits, which we conveniently listed as qubits j through $j+1$, so the possible space is quite small. Only the bolded parameters can be changed under a static mapping method, whereas a dynamic mapping method can alter any weight parameter w_{ij} such that $i < j$.

Figure 5.3 shows visually what we describe regarding the expansion of parameter space. If we have N qubits to represent our N hidden units, we potentially have $\mathcal{O}(N^2)$ connections among hidden units, or couplers, that we could tweak and explore. However, the D-Wave quantum annealer can only support 6 connections per qubit. A static mapping method would lock us into choosing which 6 connections out of those $\mathcal{O}(N^2)$ possibilities to use for each qubit and then make us commit to the decision, all prior to any training, feedback, or intermediate results. But using a dynamic method allows us the freedom to abandon poorly performing couplings to seek more promising combinations, greatly enlarging the space we are allowed to explore.

5.3 Greedy Entropy Mapping

As the name implies, the first method we have is a greedy, non-optimal, and simple method. We can variably choose to maximize or minimize entropy; for the rest of the text, we assume we want to minimize entropy since our empirical results show this generates better results overall.

The simple greedy method is straightforward since it was the first attempt we made towards producing some mapping policy guided by entropy. Since we are concerned with fitting quanta of 4-unit subgroups together into configurations that minimize entropy, we simply generate all possible pairs of 4-unit subgroups (recall that each chimera cell can only fit two such groups), calculate the resulting pairing entropy values, and sort them in ascending order according to the calculated values. We then take the lowest entropy pairings and assign them to a chimera cell, skipping pairings if we have already previously mapped an involved subgroup to a chimera cell.

Supposing we have N chimera cells, we then have $2N$ subgroups and $\frac{2N(2N-1)}{2} = \mathcal{O}(N^2)$ possible subgroup pairings. The calculation of entropy for each chimera cell is constant in complexity, and sorting is known to be $\mathcal{O}(M \log M)$. In our case, $M = \frac{2N(2N-1)}{2} \implies \mathcal{O}(N^2 \log N)$. For our annealing hardware, N will generally be quite small. The systems we used have either $N = 144$ or $N = 256$ for 1k or 2k qubits, respectively. Complexity of this remapping operation should not be an overriding concern because it is only executed once per training epoch at most. In the course of our experiments, we only executed it on every fifth training epoch, further minimizing its impact.

With the process explained, it may be easier to see why settling on a 4-unit quantum is more manageable than focusing on individual qubits. With 4-unit subgroups we take advantage of the bipartite connectivity of chimera cells to simplify our calculations. Each chimera cell only has one way to fit two subgroups together, and it makes no difference which partition of the cell is assigned to which subgroup[3]. But if we had chosen to manipulate connectivity on the individual qubit level, the ordering of qubits within a cell becomes an additional layer of concern and the number of possible configurations rapidly grows, as does the complexity of the remapping operation. A single chimera cell has $\binom{8}{4}$ ways to divide its qubits into two partitions in contrast to the one choice when we use 4-qubit subgroups.

While this simplification is very helpful for reducing the complexity of this greedy method, it does not help a brute-force method in any significant way, justifying our attempts to find smarter ways to proceed. If we wanted do an exhaustive search of all possible variable-to-qubit assignments, we would have to sift through $(8N)!$ possibilities for N chimera cells, which is clearly not viable. If we make simplifying assumptions using 4-unit subgroups and disabled inter-cell couplers, we still would not be much better off because we would still be left with an exponential space to explore. Supposing again we have N chimera cells and $2N$ subgroups, we would have $\frac{(2N)!}{N!2^N}$ ways to assign subgroups to cells. Even using $N = 16$ in our small experiments would be too

[3]There are a total of two ways to fit 4-unit subgroups into a chimera cell, but they are equivalent because couplers are symmetric connections between qubits and because we ignore inter-cell couplers.

many possibilities to handle, so we must choose more intelligent way to create mappings. Having shown a simple greedy method, we next describe a greedier variation.

5.4 Greedier Entropy Mapping

The simple greedy mapping method does generate low entropy mappings, but it typically does not find a configuration that produces the lowest possible entropy mapping. In the course of our experiments we wanted to determine how LBMs would perform if given the lowest possible entropy configuration under the constraints we impose, so we developed a recursive greedy algorithm to find even lower entropy mappings[4].

Remap #	Greedy Minimum	Greedier Minimum	Greedier Maximum
0	79.50	79.18	80.54
1	78.15	75.89	81.01
2	72.95	69.21	76.21
3	70.14	66.04	74.64

Figure 5.4: A table of entropy values for a given mapping. We compare our greedy minimal mapping method against our greedier minimum and maximum mapping methods. We see that the greedy minimal method does produce lower entropy values, but it does not find entropy values as low as the greedier minimum. Remapping events occur every 5 epochs.

Figures 5.4 and 5.5 show our greedy method and greedier method compared against each other. We trained a Boltzmann machine with qubit remapping occurring every 5 epochs. Whenever we executed remapping, we would use the mapping produced by the greedy method, but we would also record the mapping results from using either a greedier minimum or global maximum policy for comparison purposes (a what-if scenario). From this we can be relatively assured that the greedier method is working as intended because it produces lower entropy values than our simple greedy method. Also, the greedier maximum policy produces the highest entropy values. As we describe the recursive algorithm to implement this mapping, we will note why this method is called a greedier method and not an optimal method.

[4]The code works such that we can easily find either high or low entropy mappings by changing a flag. We are more concerned with finding minimal entropy mappings, so when we refer to optimal mappings we mean minimal entropy mappings unless otherwise stated.

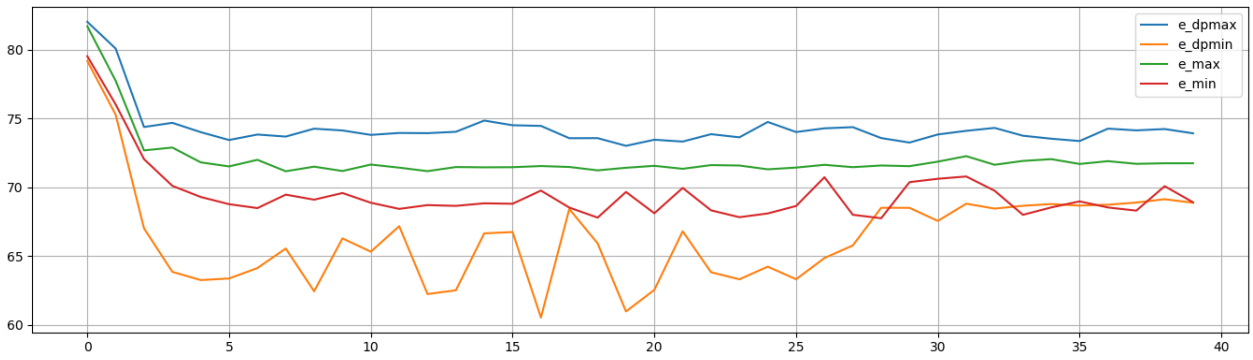


Figure 5.5: Entropy values over time for greedy minimum/maximum (e_max, e_min) and greedier minimum/maximum (e_dpmax, e_dpmin). As with Figure 5.4, the x-axis represents each remapping attempt every 5th epoch and thus covers 200 training epochs total. The policies have significant differences early in training which persist to the end. Of note are the greedy and greedier minimum entropy policies. Although both settle on the same general level of entropy, we found that the greedy minimum policy overall performed better, possibly due to some mapping decisions made early in training.

As mentioned, the algorithm is recursive in one dimension N, the index of the highest-indexed chimera cell we will consider. Let us have a function MinE(n) find the minimal entropy mapping and its corresponding entropy value for if we are considering chimera cells $0 \dots n-1$. We can call the minimum entropy value E(n) and the mapping configuration M(n). It is important to note here that MinE(n) will only use the logical qubits currently mapped to chimera cells $0 \dots n-1$. The qubit mapping of the Boltzmann machine is only changed when we finally finish our calculation of MinE(N).

First we consider the base case where we have the trivial situation of considering only our first-indexed chimera cell, namely MinE(1). There is only one single way to create a mapping, by simply leaving the subgroups where they are. Calculating E(1) and M(1) is trivial as well because our mapping is already determined.

The inductive step is more involved. Let us suppose we are now considering how to calculate MinE(n), assuming we have already calculated MinE($n-1$), E($n-1$), and M($n-1$). In effect what is happening is this: we already have some optimal mapping for chimera cells $0 \dots n-2$ and their associated qubits. Now, we are considering adding the chimera cell $n-1$ and the two

subgroups of qubits associated with it to our overall solution. Let us call those subgroups **a** and **b**. We are confronted with three possible solutions for MinE(n):

1. Accept both **a** and **b** as a pair in the new chimera cell. This is the easiest solution and it is straightforward to update E(n) and M(n). This step is overall $\mathcal{O}(1)$ because we do not need to iterate over anything and we do not execute any expensive functions or searches.

2. Keep **a** in the new chimera cell and force **b** to reside in one of chimera cells $0 \dots n-2$. In this case, we have to run through the mapping produced by M($n-1$) and try replacing each subgroup with **b** and recalculating the entropy associated with the affected chimera cell. We also calculate and store the entropy that results from the replaced subgroup being placed in the new chimera cell at index $n-1$. After running through the entire mapping and trying to replace different subgroups with **b**, we choose the configuration that produced the overall lowest entropy (remembering to account for **a** being paired with a replaced subgroup in chimera cell $n-1$) and store it as a potential solution to E(n), M(n), and MinE(n) accordingly. Recall that calculating entropy is a $\mathcal{O}(1)$ operation and that the mappings contain $\mathcal{O}(n)$ subgroups, so overall this step is also $\mathcal{O}(n)$.

   We note here that this step is why we call this algorithm greedier instead of optimal. Our algorithm assumes that when we force **b** to reside in an older cell $0 \dots n-2$, the subgroup that **b** replaces (call it **d**) automatically gets paired with **a**. This is not necessarily the case. It is possible that **d** can go reside in another older cell $0 \dots n-2$ and produce an overall lower entropy value and that **e**, some low-indexed subgroup, can eventually get paired with **a**.

   Without any loss of generality, we can also force **b** to stay in the chimera cell at index $n-1$ and force **a** to reside in one of chimera cells $0 \dots n-2$.

3. We force **a** and **b** to reside outside of chimera cell $n-1$. This may seem worrying at first because we might have to calculate the entropy of all possible pairings of subgroups again,

driving up the complexity of our calculation of MinE(n). However, all scenarios in this case will degenerate into cases 1 or 2. Let us consider the ways we could proceed.

(a) Suppose the cell $n-1$ contains neither **a** nor **b**. Furthermore, suppose **a** and **b** end up paired together in cell i such that $0 \leq i \leq n-2$. This is equivalent to 1 because we have assumed that inter-cell couplers are disabled. This assumption allows us to consider chimera cells independently of each other, thus their ordering does not matter. We could take cell i and re-index it as cell $n-2$, and vice versa. This degenerates into case 1 and we have no additional work to do.

(b) Suppose cell $n-1$ contains neither **a** nor **b**. Furthermore, suppose **a** and **b** are found in different cells i and j respectively such that $0 \leq i, j \leq n-2$. This is equivalent to 2, again owing to our assumption that inter-cell couplers are disabled, allowing us to re-index chimera cells as we please.

After all this we are presented with possible mappings from cases 1 and 2, recalling that case 3 degenerates into either 1 or 2. From these possibilities we simply pick the lowest entropy mapping and update MinE(n), E(n), and M(n). The most complex operation in all these cases comes from (2), which we showed was $\mathcal{O}(n)$. If we wish to calculate MinE(N), we have calls to MinE(N), MinE($N-1$), ... MinE(1), so overall our complexity is $\mathcal{O}(N^2)$, which is actually slightly better than the $\mathcal{O}(N^2 \log N)$ of the greedy method.

Having established the base case of MinE(1) and shown how to calculate MinE(n) from the results of MinE($n-1$), we can now compute MinE for any value of n where n is the number of chimera cells that compose our Boltzmann machine.

5.5 Optimal Entropy

When we described the greedier method in Section 5.4, we explained why it remained a greedy method and not an optimal method. This problem of matching subgroups to each other in order

to obtain some low entropy value can actually be mapped onto the minimum/maximum weight matching problem. In the minimum weight matching problem, we want to find an independent edge set (a set of edge with no common vertices) such that the sum of weights is minimized. Mapping our subgroup problem to this weight matching problem is easy: the vertices V of the graph are simply our subgroups and the edges E are the resulting entropy values when we pair subgroups together. The result of the weight matching algorithm corresponds to our entropy value.

The weight matching problem is $\mathcal{O}(|V|E^2)$ in complexity, or $\mathcal{O}(N^3)$ for our purposes when considering N chimera cells. Faster algorithms [35] improve the running time to $\mathcal{O}(\sqrt{V}E)$, or equivalently $\mathcal{O}(N^{2.5})$. A randomized algorithm that uses fast matrix multiplication [36] further reduces the running time to $\mathcal{O}(V^{2.376})$.

5.6 Total Correlation

Our use of entropy as a metric to guide qubit remapping policies also led us to considering the use of its cousin, total correlation. Total correlation is defined as:

$$TC(G) = \big(\sum_{g \in G} H(g)\big) - H(G) \tag{5.2}$$

Here, $g \in G$ is a random variable in a group G of random variables. Total correlation measures how much information is shared among the variables. Minimization of total correlation (TC) was an explicit goal in CorEx [55], and since Boltzmann machines can also measure TC using 4-qubit subgroups as each G, we decided to try using it as an alternate metric to determine remapping policies. The only change required for implementation was replacing each $H(G)$ calculation (see Eq. 5.1) with Eq. 5.2. Usage of TC as a metric was not impactful, however, being outperformed by the simpler entropy metric. We nevertheless included TC results with our other findings for completeness.

5.7 Results

To get a better comparison between our different qubit remapping policies, we decided to subject them all to the same number of training epochs on the same data sets. Our previous figures and results were generated from the software simulator provided by D-Wave Systems, so we replicated the experiments on the hardware to confirm our findings held up.

Figure 5.6 plots the reconstruction loss of various BMs trained with different remapping policies. Remapping occurred after epochs 0, 5, 10, and 15, after which there was no more qubit remapping. Training continued until epoch 200. From the results, we conclude that a policy that reduces entropy, but does not minimize it completely, performs best. We also observe that although we remap qubits very early in training, the effects of remapping persist long after dynamic remapping ceases. The top plot of Figure 5.6 shows the overall trend in improvement observed in BMs to verify training behaves as expected while the bottom plot focuses on the terminal performance of the BMs. Of particular note is the amount of training time saved. Directly comparing loss values against each other is not illuminating because we do not know the lower bound on L2 achievable by the BMs; however, what we can do is compare the number of epochs necessary for each qubit remapping policy to match the others. For instance, notice that the loss value obtained by the no-remapping policy at epoch 200 is the value achieved by the greedy entropy policy by epoch 160 - the greedy entropy only took $\frac{3}{4}$ the training time to get the same level of performance. This result is particularly interesting because it is attributable to remapping changes we made very early in training. Our choice of qubit remappings in the first 15 epochs (totalling only four remappings) persisted through the entirety of training and led to a disproportionately large performance gap between policies. No qubits were added or removed and no hyper-parameters were altered, so we can point to a qubit remapping policy as the sole factor driving performance differences.

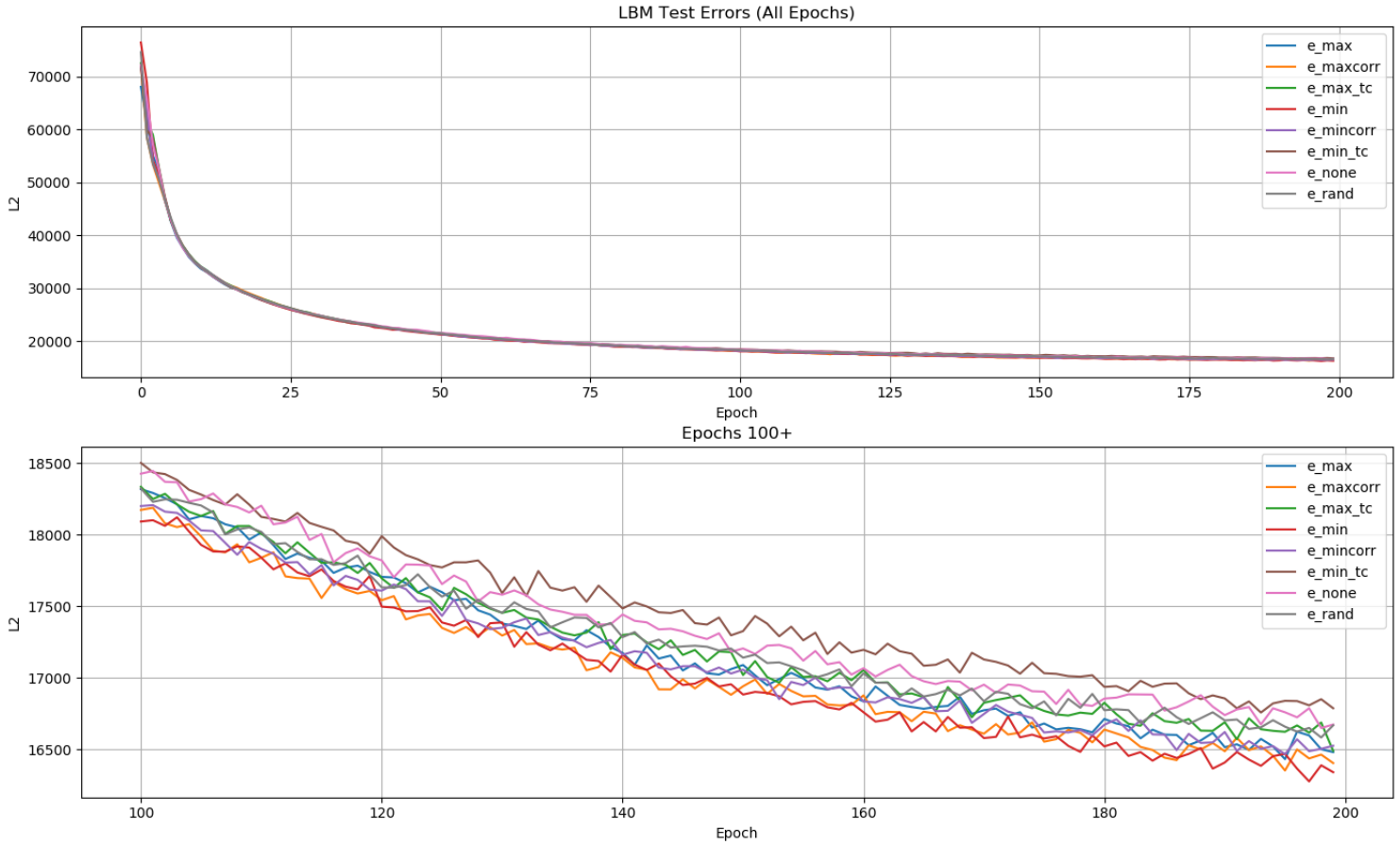


Figure 5.6: Comparison of reconstruction loss (y-axis) resulting from limited qubit remapping policies conducted over a 200-epoch training period (x-axis). The two plots are the same, except the top plot shows all training epochs while the bottom focuses on the latter 100 epochs. The greedy entropy policy performs best overall, achieving a loss value at epoch 160 that a non-remapped BM needs 200 epochs to reach, cutting off 25% of training time.

Policy	Early Rank	Terminal Rank
Greedy Max Entropy	3.435	3.005
Max Correlation	2.882	1.88
Max TC	4.200	4.245
Greedy Min Entropy	0.317	0.675
Min Correlation	1.976	2.290
Min TC	4.365	5.605
No Remapping	5.845	6.282
Random	4.541	4.455

Figure 5.7: Average rank for each policy early (epochs 15-100) or late (epochs 100-200) in training; this BM was trained on the MNIST digits data set using D-Wave's software simulator. "TC" stands for total correlation. The maximum correlation policy initially works better than the minimum correlation policy, consistent with our results in Figure 4.3, but then switches rank in relative performance later in training.

We also found that remapping policies can perform relatively differently early in training. Figure 5.7 shows the average rankings (lower being better) of qubit remapping policies in early and later stages of training. The greedy minimum entropy works best throughout training, but of note are the relative rankings of maximum correlation and minimum correlation policies. Maximum correlation remapping had an average rank of 2.882 early in training versus minimum correlation's average ranking of 1.976. However, by epoch 200 the maximum correlation policy's rank had switched places with the minimum correlation policy's ranking - 1.88 versus 2.290. We felt it important to note this change since the results of Figure 5.6 show maximum correlation outperforming minimum correlation but Figure 4.3 shows the opposite. This discrepancy is resolved by Figure 5.7's split of training into early and late stages.

Based on these results, we performed additional comparisons between remapping policies to see if our results held up across different data sets and when transferred from software to hardware. Whereas the results we just described were gathered from BMs trained on MNIST digit data on D-Wave's software simulator, our next set of results were gathered from BMs trained on MoS_2 data, from running on D-Wave's annealing hardware, or both. We also remapped qubits every 5 epochs throughout the entirety of training rather than just remapping in epochs 0, 5, 10, and 15.

The idea was to increase the influence of remapping policies on final results and hopefully widen any performance gap between policies.

Figure 5.8 shows the results from a BM trained on MoS_2 data running on a software simulator. Overall the trends observed in Figure 5.6 remain the same, where low, but not minimal, entropy gives the best results. The main observation here is that the performance of different policies remains the same across both the MNIST digits and MoS_2 data sets, which strengthens the idea that remapping policies have consistent global and generalized differences. A secondary observation is a noticeable new feature: sudden spikes in error that occur regularly. These performance dips happen every 5 epochs; furthermore, they happen exactly on the epochs when we remap qubits. We expected this result because every remapping action slightly changes the problem we have trained the BM to solve, thus the BM should accordingly perform slightly worse. Recall the discussion of parameter space exploration in Figure 5.3. In performing a remapping action, we potentially remove a parameter the BM has adjusted and trained upon in favor of another parameter which may have received reduced or no training. The looping, interconnected nature of the BM's hidden units means the swapping of parameters will negatively impact short-term performance.

Some questions arise from this secondary observation. How much remapping (in terms of qubits) is actually performed by these policies, and can we attribute their performance differences to this amount? One conjecture we had was that the best performing methods might be changing their qubit maps relatively little, allowing them to train longer on specific parameters and effectively giving themselves more training time. In contrast, we supposed a poorer performing policy might be shuffling qubits around constantly and not allowing any parameters to train for a significant amount of time.

Figure 5.9 gives the amount of subgroup combinations that remain unchanged upon each remapping action. The best performer, the greedier minimum entropy method ("unchanged_dpmin"),

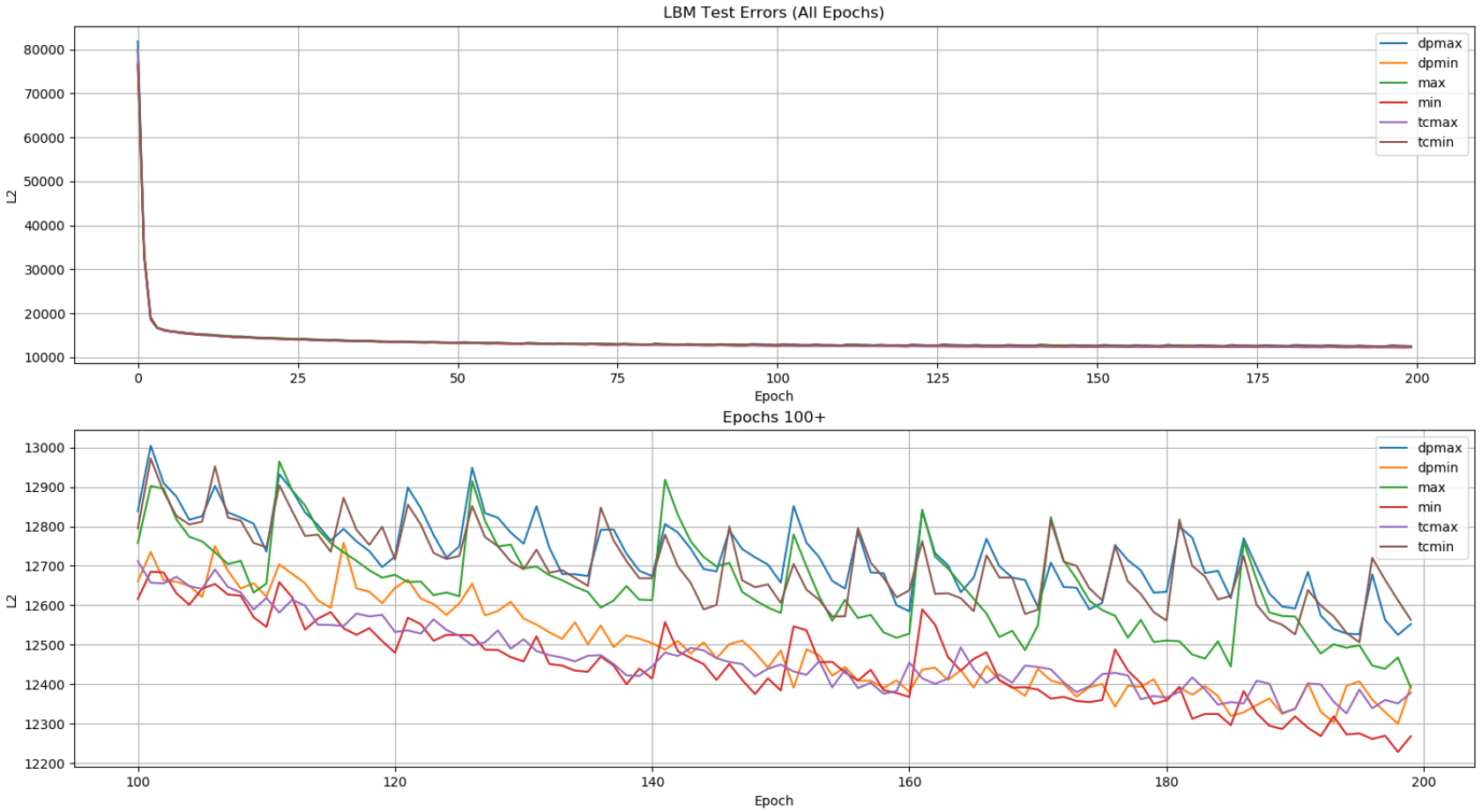


Figure 5.8: Results from a BM trained on MoS_2 data run on a software simulator where qubits are remapped every 5 epochs throughout training. The policy rankings hold steady across data sets, suggesting the usage of entropy as a metric for remapping decisions may be a generally good idea regardless of the input data. The upward spikes of L2 occur every 5 epochs upon remapping.

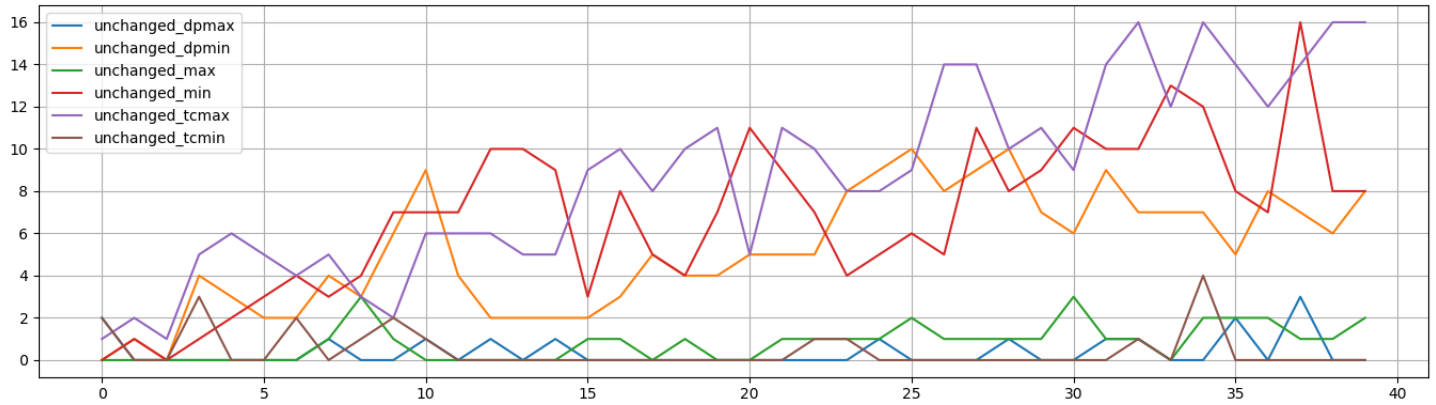


Figure 5.9: The number of chimera cells (pairs of 4-unit subgroups) that remain unchanged (y-axis) across remapping attempts (x-axis). Recall that this experimental setup uses a total of 16 chimera cells and 128 qubits. Overall we do not see much correlation between the amount of change and the performance of a particular policy. The greedier minimum entropy (unchanged_min) had a high number of unchanged cells and was the best performer, but greedier maximum total correlation (unchanged_tc) also changed little and was a poor performer.

tends to change its qubit mapping very little, which at first appeared to support our conjecture. However, we also noticed a poor performer, the greedier maximum total correlation, also changed its qubit mapping infrequently, contradicting our conjecture. Overall we concluded that the "amount" of remapping each policy performs is not the most significant performance factor, and that good or bad performance is due to something more fundamental in choosing smarter connectivity options.

Finally, to address potential concerns that our experiments had all been performed on a software simulator instead of on actual annealing hardware, we trained a Boltzmann machine on MoS_2 data using D-Wave's adiabatic quantum annealer.

Figure 5.10 shows the results gathered from the annealing hardware. We see the same trends as in previous experiments. One noticeable difference is the slightly higher error. This is expected because the hardware has to deal with physical realities such as noise, limited bit precision, and hard limits on parameter ranges. Although these issues are worth discussing in a different context, they are not our focus here. Some constant difference in error numbers aside, our results from the physical hardware align with all of our previous experiments on the software simulator. Overall we have concluded that low entropy, but not minimal entropy, is a good qubit remapping policy that produces consistently better results across data sets in both software and hardware.

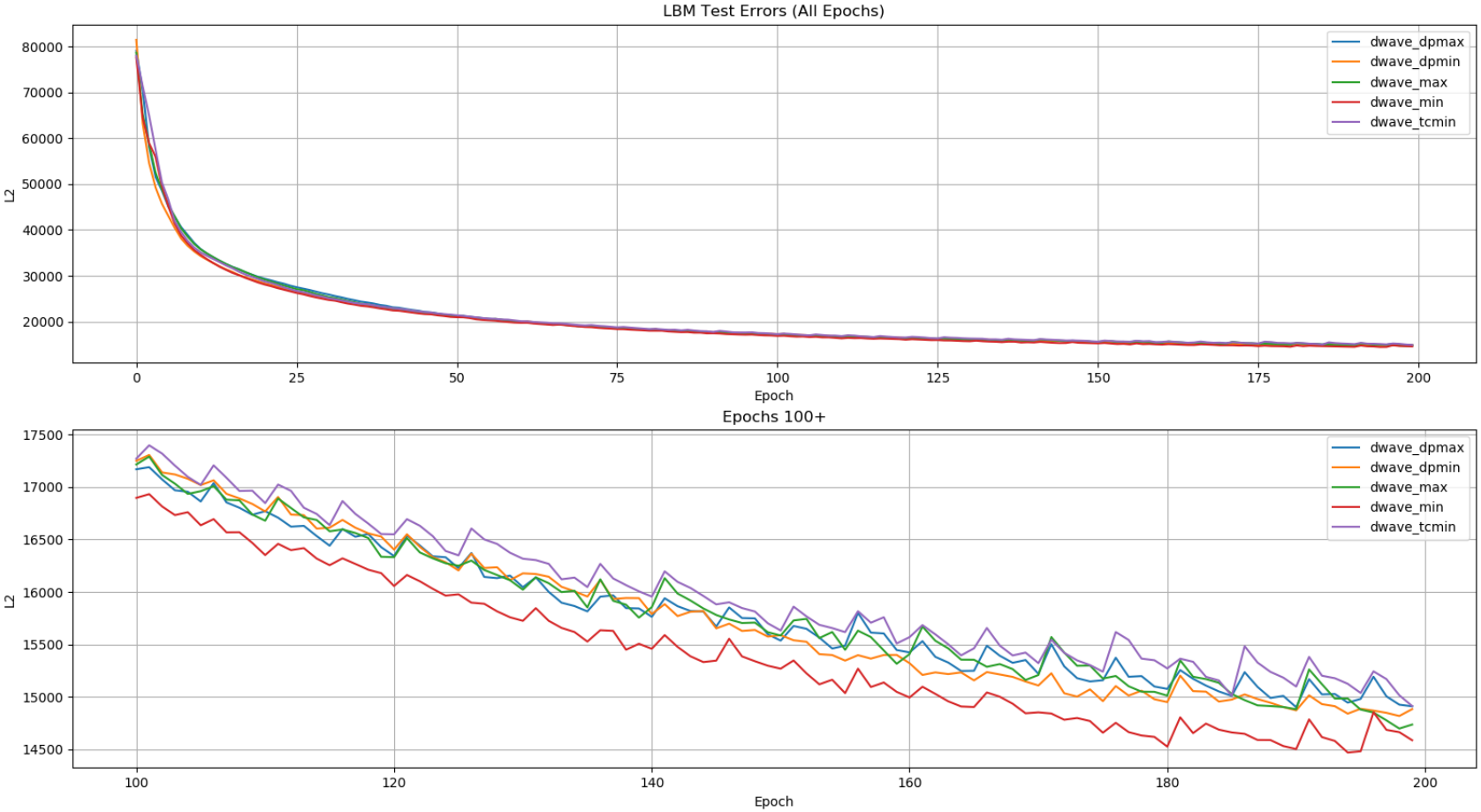


Figure 5.10: The same experimental setup as 5.9 but performed on annealing hardware instead. The trends remain the same even though the error numbers are slightly higher, which was expected.

Chapter 6

Discussion: Balancing Entropy

Our experimental results suggest an organized method of assigning limited Boltzmann machine hidden variables to nodes in a chimera graph improves performance. The LBM concept and variable reassignment policies can both be implemented on an adiabatic quantum annealer or used in a classical setting. The approaches of maximizing correlation and minimizing entropy were initially based on intuition. Post-experiment, we decided to search for a mathematical justification for why our methods were effective, so we settled upon an explanation using hidden information.

A paper by Kamimura et al. [31] examined the entropy of hidden unit activation patterns in autoencoders. Recall that autoencoders produce efficient encodings of data that lose as little information as possible. The authors put forth the idea that information can be categorized as necessary or unnecessary. Necessary information is something intrinsic to the data that we want to capture in our encodings; unnecessary information is random noise or patterns that have no value. The goal of encoding units is to capture necessary information and filter out unnecessary information. As an encoding network trains on input data, the hidden units begin to exhibit low entropy activation patterns. Individual or groups of units begin to specialize as feature detectors that respond specifically to particular data input patterns. Hidden units do not randomly activate in a trained encoding network. If they did behave randomly (high entropy), a network would be

unable to capture any meaningful necessary information about the input data. Echoing similar results to ours, the authors concluded that low entropy, but not minimal entropy, produces better results in encoding networks.

The amount of hidden information captured by a network is defined as the difference in network entropy at the start of training and the network entropy at the end of training. Using the definition of entropy[1] in Eq. 5.1, we say hidden information is:

$$I = H_{start} - H_{end} \tag{6.1}$$

This definition of hidden information can be added to the objective function of an autoencoder, giving the network the dual goals of minimizing residuals and maximizing hidden information. Such a gradient would be defined as:

$$\frac{\partial I}{\partial w_{jk}} = \sum_{s}^{S} \phi_j^s \xi_k^s \tag{6.2}$$

$$\phi_j^s = (\log P_j^s - \sum_r P_r^s \log P_r^s) P_r^s (1 - h_j^s) \tag{6.3}$$

Here, s is a given input pattern from data set S, ξ_k^s is the kth element of input pattern $s \in S$, h_j is the state of hidden unit j. This translates into the weight update rule that includes both hidden information ϕ (with hyper-parameter α) and cross-entropy δ (with hyper-parameter β) measures:

$$\Delta w_{ij} = \sum_{s}^{S} (\alpha \phi_j^s + \beta \delta_j^s) \xi_k^s \tag{6.4}$$

[1]Kamimura calculated and summed the individual entropy of every hidden unit whereas in our work we calculated and summed the joint entropy of chimera cell groupings. The general idea and goal remain the same.

$$\delta_j^s = f'(u_j^s) \sum_i^N w_{ij} \delta_i^s \tag{6.5}$$

where u_j^s is the activity level of hidden unit j (so just $\sum_k^L w_{jk} \xi_k^s$), and

$$\delta_i^s = \xi_i^s - O_i^s \tag{6.6}$$

where O_i^s is the target output of the ith unit of pattern $s \in S$. When using an encoding network, O_i^s is original input data.

Networks that utilized Eq. 6.4 tended to perform better than standard networks. However, the balance of the $\frac{\alpha}{\beta}$ ratio proved to have significant effects on their findings, where too much weighting of α (hidden information and entropy) caused overall results to suffer. We believe this directly relevant to our results because our efforts are largely aligned.

Our formulation of the problem searches for subgroup pairings that produce low entropy. Supposing we have two subgroups L (left) and R (right), we can be in one of three situations. In the first, $R = f(L)$, a minimal entropy arrangement. That is, R is simply a function of L. Then it follows that $P(L, R) = P(R|L)P(L) = P(L)$ and $H(L, R) = H(L)$. However, our procedure is unlikely to find such a pairing, if it even exists, due to the random nature of weight initialization and the distribution of our data.

The opposing extreme situation is one where L and R are independent and produce maximum entropy, or $H(L, R) = H(L) + H(R)$. Our procedure is unlikely to pick this arrangement because we seek low entropy.

We are most likely to find ourselves in the third situation, where we are between the two extremes. For every state vector $L = (l_0, \ldots l_n - 1)$, R can return one of k different state vectors. Then $H(L, R) = H(R|L) + H(L) = \log(k) + H(L)$, where $k = 2^4$ at most in our case where we have 4-unit subgroups. The intuition behind this situation is that our procedure finds an R

that returns a small number of patterns. This has the effect of making our hidden units respond selectively to input.

While our work differs from Kamimura et al.'s, there are enough similarities that the same principles should apply. Although Kamimura used deterministic autoencoders to examine entropy and hidden information, probabilistic Boltzmann machines perform largely the same sort of encoding function - when exposed to input data, a set of stochastic hidden units is activated in response. When applying weight update rules, Kamimura explicitly included a hidden information term whereas we implicitly used it by choosing to include or drop certain hidden-hidden connections based on entropy measures. Both approaches try to minimize entropy in hidden unit encodings and both see improved results.

The examination of entropy in hidden units may not have seen much exploration due to the availability of increased computing power. The Kamimura paper was written before computing power expanded quickly enough to enable usage of other types of neural networks (feed-forward networks trained with back-propagation or CNNs) where improved results were most easily achieved by simply expanding network size. Such an environment made a disciplined investigation of entropy and how to create efficient network topologies cost too much effort for the potential gain in performance. Consequently the authors worked with very small autoencoder networks composed of fewer than 10 units, whereas contemporary networks have orders of magnitude more units and parameters. However, now that we have adiabatic quantum annealers being introduced into the computing environment, we see a sort of return to the conditions Kamimura et al. worked in, where we have to carefully consider how best we should use the limited resources (qubits and connectivity) at our disposal. Annealers exist in an environment where it is relatively difficult to add more qubits, so simply expanding network size is not a reasonable improvement we can make. Annealers are also physically constrained in the number of connections each qubit can support. This makes variable-to-qubit assignments much more meaningful decisions since we can no longer dodge them by simply declaring a network to be fully connected.

We have seen that entropy has a significant effect on our experimental results and that we can manage entropy via the choices we make in our variable-to-qubit assignment decisions. A principle we have observed is that low entropy, but not minimal entropy, is generally desirable. A question that naturally arises is: what level of entropy in our qubit mapping is optimal for LBM performance?

This question has parallels to existing problems in machine learning and statistics. Many learning or optimization problems include a regularization term in their loss function. The regularization term generally acts as a balance against some specific form of misbehavior by the model, be it complexity, overfitting, saturation, or some other offense. So supposing a loss function $V(f(x), y)$ for the output of $f(x)$ where the label is y, and a regularization term $R(f)$ that penalizes some behavior by $f(x)$ itself, we have a loss function $L(x, y)$:

$$L(x_i, y_i) = V(f(x_i), y_i) + \lambda R(f) \tag{6.7}$$

Though the exact form of R varies according to specific needs, determining the proper value of the λ term is typically achieved through an empirical tuning process [54, 12, 60, 22]. In Kamimura's approach, the regularization term R would correspond to the ϕ_j^s term that represents entropy in Equation 6.4, and the λ constant would correspond to the α term in that same equation. The authors searched for a good α value by adjusting it higher and lower, repeating their experiments, then picking the result that performed the best. We believe that we can treat entropy as a regularization term and find the ideal level of entropy that should be associated with good qubit mappings. As it stands, our approach separates entropy from the regular LBM training and weight update process into two steps: we first update network parameters normally, without consideration of entropy, then evaluate the entropy of our mapping and potentially reassign variables to qubits. We see the potential for integrating our entropy measure with the regular update process where we explicitly include a regularization term, representing entropy, into Equation 2.4.

Chapter 7

Conclusion

Our work has contributed several new points of knowledge to the study of Boltzmann machines and to the potential applications of adiabatic quantum annealers. We next summarize and discuss these contributions.

Investigation of LBMs. The discussion of Boltzmann machines has taken place in front of a quantum annealing backdrop. While practical applications of BMs use a restricted connectivity topology to combat tractability issues that arise from using fully connected topologies, quantum annealers allow BMs to use expanded topologies that include hidden-to-hidden connections. We call our implementation of these expanded topologies "limited Boltzmann machines," or LBMs. Our experiments have shown that LBMs offer superior performance to restricted BMs that use only bipartite connectivity between visible and hidden units.

Implementation of LBMs on adiabatic quantum annealers. Our design of the LBM was created with adiabatic quantum annealers in mind. Though adding hidden-to-hidden connectivity back to BMs to create LBMs is conceptually straightforward, realizing LBMs on an annealer is a more involved task because we are faced with a limited number of qubits and a sparse hardware connectivity scheme. We fit LBMs onto our annealer by representing the hidden units using qubits and the intra-layer connections using couplers.

Whereas the expanded connectivity of LBMs would normally lead to intractability issues during training, we leveraged the unique sampling properties of an open-system adiabatic quantum annealer to carry out the learning process. In addition, we made the crucial decision to avoid the graph embedding problem that has stymied other efforts to use adiabatic quantum annealers. Rather than designing an predetermined connectivity topology for our LBMs and trying to make it fit onto a sparse annealer hardware topology, we decided to use the annealer topology "as is" and to manipulate topology through qubit mapping instead.

Creation of qubit mappings to enhance LBM results. In the course of implementing LBMs, we had to decide how to assign BM variables to physical qubits on D-Wave's open system adiabatic quantum annealer. Our initial experiments assigned variables in logical order for simplicity's sake - the first variable to the first qubit, the second variable to the second qubit, and so forth. But during implementation, this process raised a question in our minds - was there a better way to create variable assignments? Any choice of variable-to-qubit assignments/mappings implicitly determines which subset of all possible hidden-to-hidden connections are represented on the hardware, and some subsets surely must perform better than others. Our investigation into this question showed this hypothesis to be correct, that indeed certain subsets, such as those exploiting data locality, do perform better than others.

We started with the usage of correlation to create static qubit mappings. Evidence that the choice of variable-to-qubit assignments makes a difference moved us to ask the natural follow-up question: how do we find variable-to-qubit assignments that perform well? One approach we adopted was to use correlation as a means of creating qubit mappings. We found that attempts to maximize correlation between qubit activity within chimera cells consistently produced better results than the hand-crafted assignments we had used in previous experiments. But we did note a weakness with this approach: correlation only ever considers pairwise interactions between qubits, and focusing on a correlation-based metric could make us lose sight of the larger environment of

possible qubit assignments. To achieve a more encompassing approach, we turned to entropy as a metric for designing qubit mappings.

We discovered that entropy is an effective guide for creating dynamic qubit mappings. We presented results showing that mappings designed around reducing (but not minimizing) entropy performed the best, which appeared consistent with our results showing high correlation worked well[1]. We trained BMs on two different data sets and on two platforms, a software simulator for the annealer and the actual hardware annealer. The results remained consistent across all four iterations of this experiment and show that entropy is an effective metric for deciding how to assign BM variables to hardware qubits.

While this work has already made these several contributions, we believe there are more possibilities we can explore. For instance, future work could delve into the implementation of many-layered LBMs on adiabatic quantum annealers. Our work has covered the implementation of LBMs with one interconnected hidden layer, so it would be of interest to ask how we can address the creation of efficient qubit mappings for many-layered LBMs. We can also try using entropy as a regularization term in our optimization problem, to explicitly include it in our training process. Another possible direction is to determine what amount of entropy is ideal for a data set. While we have shown general performance trends are tied to entropy levels, we have not tried to find the best level of entropy for a given data set. Such an experiment would be a relatively straightforward hyper-parameter optimization process.

We can also consider how to adapt our work to newer generations of hardware, such as the Pegasus architecture announced by D-Wave that features 15 connections per qubit and 5000 qubits [9]. Our work does not rely on a specific number of qubits or couplers, so it can likely be generalized to handle the introduction of new machines. More qubits and couplers would also allow us to build richer, multi-layered networks. Our experiments with Boltzmann machines had

[1] Adopting entropy was our way of expanding upon correlation. Our original problem was the pairwise restriction of correlation, and entropy seemed like a natural multi-variable extension. Low entropy among variables means they have have strong correlation.

assumed a single layer of hidden units due to size constraints, so a larger machine could enable us to experiment with deep Boltzmann machines with multiple layers of hidden units.

Our work on BMs and adiabatic quantum annealers will remain relevant going forward. In contrast to a neural network running in a software environment, it is difficult for us to add more computational units to a BM running on annealing hardware to improve results. Physical limitations and engineering considerations restricted us to just six connections per qubit on the hardware we used. Even if new hardware promises expanded connectivity, it is not likely to fundamentally shift our constraints in the near future, like what had happened with with transistor development. In such a restricted environment, it behooves us to carefully consider how best to utilize the few resources we do have at our disposal.

As stated in Section 1, our goal was to improve data modeling results by developing an entropy-based metric for dynamically reassigning BM variables to qubits. We believe the algorithms we implemented and the experiments we conducted have met this goal, and that the underlying principles will continue being relevant even for future iterations of the hardware we used.

Bibliography

[1] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[2] Steven H Adachi and Maxwell P Henderson. Application of quantum annealing to training of deep neural networks. *arXiv preprint arXiv:1510.06356*, 2015.

[3] Mohammad H Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum boltzmann machine. *Physical Review X*, 8(2):021050, 2018.

[4] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[5] Francisco Barahona. On the computational complexity of ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241, 1982.

[6] Marcello Benedetti, John Realpe-Gómez, Rupak Biswas, and Alejandro Perdomo-Ortiz. Estimation of effective temperatures in quantum annealers for sampling applications: A case study with possible applications in deep learning. *Physical Review A*, 94(2):022308, 2016.

[7] Marcello Benedetti, John Realpe-Gómez, Rupak Biswas, and Alejandro Perdomo-Ortiz. Quantum-assisted learning of graphical models with arbitrary pairwise connectivity. *arXiv preprint arXiv:1609.02542*, 2016.

[8] Sergio Boixo and RD Somma. Necessary condition for the quantum adiabatic approximation. *Physical Review A*, 81(3):032308, 2010.

[9] Kelly Boothby, Paul Bunyk, Jack Raymond, and Aidan Roy. Next-generation topology of d-wave quantum processors. *arXiv preprint arXiv:2003.00133*, 2020.

[10] Max Born and Vladimir Fock. Beweis des adiabatensatzes. *Zeitschrift für Physik*, 51(3-4):165–180, 1928.

[11] Vicky Choi. Minor-embedding in adiabatic quantum computation: Ii. minor-universal graph design. *Quantum Information Processing*, 10(3):343–353, 2011.

[12] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8609–8613. IEEE, 2013.

[13] Misha Denil and Nando De Freitas. Toward the implementation of a quantum rbm. In *NIPS 2011 Deep Learning and Unsupervised Feature Learning Workshop*, 2011.

[14] Guillaume Desjardins, Aaron Courville, Yoshua Bengio, Pascal Vincent, and Olivier Delalleau. Tempered markov chain monte carlo for training of restricted boltzmann machines. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 145–152, 2010.

[15] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE, 2015.

[16] Vincent Dumoulin, Ian J Goodfellow, Aaron C Courville, and Yoshua Bengio. On the challenges of physical implementations of rbms. In *AAAI*, volume 2014, pages 1199–1205, 2014.

[17] Steven K Esser, Paul A Merolla, John V Arthur, Andrew S Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J Berg, Jeffrey L McKinstry, Timothy Melano, Davis R Barch, et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, page 201604850, 2016.

[18] Clément Farabet, Berin Martini, Polina Akselrod, Selçuk Talay, Yann LeCun, and Eugenio Culurciello. Hardware accelerated convolutional neural networks for synthetic vision systems. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 257–260. IEEE, 2010.

[19] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106*, 2000.

[20] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6-7):467–488, 1982.

[21] Asja Fischer and Christian Igel. Training restricted boltzmann machines: An introduction. *Pattern Recognition*, 47(1):25–39, 2014.

[22] Seymour Geisser. The predictive sample reuse meth. Technical report, University of Minnesota, 1973.

[23] Richard Harris, MW Johnson, T Lanting, AJ Berkley, J Johansson, P Bunyk, E Tolkacheva, E Ladizinsky, N Ladizinsky, T Oh, et al. Experimental investigation of an eight-qubit unit cell in a superconducting optimization processor. *Physical Review B*, 82(2):024511, 2010.

[24] Geoffrey Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010.

[25] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[26] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

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

[28] Li Huang and Lei Wang. Accelerated monte carlo simulations with restricted boltzmann machines. *Physical Review B*, 95(3):035105, 2017.

[29] Giacomo Indiveri, Federico Corradi, and Ning Qiao. Neuromorphic architectures for spiking deep neural networks. In *Electron Devices Meeting (IEDM), 2015 IEEE International*, pages 4–2. IEEE, 2015.

[30] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse ising model. *Physical Review E*, 58(5):5355, 1998.

[31] Ryotaro Kamimura and Shohachiro Nakanishi. Hidden information maximization for feature detection and rule discovery. *Network: Computation in Neural Systems*, 6(4):577–602, 1995.

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[33] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998.

[34] Chenchen Liu, Qing Yang, Bonan Yan, Jianlei Yang, Xiaocong Du, Weijie Zhu, Hao Jiang, Qing Wu, Mark Barnell, and Hai Li. A memristor crossbar based computing engine optimized for high speed and accuracy. In *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*, pages 110–115. IEEE, 2016.

[35] Silvio Micali and Vijay V Vazirani. An o (v— v— c— e—) algoithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, pages 17–27. IEEE, 1980.

[36] Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 248–255. IEEE, 2004.

[37] Edwin Pednault, John A Gunnels, Giacomo Nannicini, Lior Horesh, and Robert Wisnieff. Ice: Dynamic ranges in h and j values.

[38] Thomas E Potok, Catherin Schuman, Steven Young, Robert Patton, Federico Spedalieri, Jeremy Liu, Ke-Thia Yao, Garrett Rose, and Gangotree Chakma. A study of complex deep learning networks on high performance, neuromorphic, and quantum computers. *Journal of Emerging Technologies in Computing*, page To appear, 2018.

[39] Robert Raussendorf and Hans J Briegel. A one-way quantum computer. *Physical Review Letters*, 86(22):5188, 2001.

[40] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para.* Cornell Aeronautical Laboratory, 1957.

[41] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3856–3866. Curran Associates, Inc., 2017.

[42] Ruslan Salakhutdinov. Learning deep boltzmann machines using adaptive mcmc. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 943–950, 2010.

[43] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *Artificial Intelligence and Statistics*, pages 448–455, 2009.

[44] Marcelo S Sarandy, L-A Wu, and Daniel A Lidar. Consistency of the adiabatic theorem. *Quantum Information Processing*, 3(6):331–349, 2004.

[45] Sagarvarma Sayyaparaju, Gangotree Chakma, Sherif Amer, and Garrett S Rose. Circuit techniques for online learning of memristive synapses in cmos-memristor neuromorphic systems. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 479–482. ACM, 2017.

[46] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International Conference on Artificial Neural Networks*, pages 92–101. Springer, 2010.

[47] Catherine D. Schuman, Thomas E. Potok, Steven Young, Robert Patton, Gabriel Perdue, Gangotree Chakma, Austin Wyer, and Garrett S. Rose. Neuromorphic computing for temporal scientific data classification. In *NCS '17: Neuromorphic Computing Symposium*, page To appear. ACM, 2018.

[48] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[49] Robert S Smith, Michael J Curtis, and William J Zeng. A practical quantum instruction set architecture. *arXiv preprint arXiv:1608.03355*, 2016.

[50] Rolando D Somma and Sergio Boixo. Spectral gap amplification. *SIAM Journal on Computing*, 42(2):593–610, 2013.

[51] Rolando D Somma, Sergio Boixo, Howard Barnum, and Emanuel Knill. Quantum simulations of classical annealing processes. *Physical review letters*, 101(13):130504, 2008.

[52] Adam M Terwilliger, Gabriel N Perdue, David Isele, Robert M Patton, and Steven R Young. Vertex reconstruction of neutrino interactions using deep learning. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 2275–2281. IEEE, 2017.

[53] The IBM Quantum Experience. http://www.research.ibm.com/quantum.

[54] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[55] Greg Ver Steeg and Aram Galstyan. Discovering structure in high-dimensional data through correlation explanation. In *Advances in Neural Information Processing Systems*, pages 577–585, 2014.

[56] Nathan Wiebe, Ashish Kapoor, and Krysta M Svore. Quantum deep learning. *Quantum Information and Computation*, 16:0541–0587, 2016.

[57] Antonio Jimeno Yepes, Jianbin Tang, and Benjamin Scott Mashford. Improving classification accuracy of feedforward neural networks for spiking neuromorphic chips. *arXiv preprint arXiv:1705.07755*, 2017.

[58] Steven R Young, Derek C Rose, Travis Johnston, William T Heller, Thomas P Karnowski, Thomas E Potok, Robert M Patton, Gabriel Perdue, and Jonathan Miller. Evolving deep networks using hpc. In *Proceedings of the Machine Learning on HPC Environments*, page 7. ACM, 2017.

[59] Steven R Young, Derek C Rose, Thomas P Karnowski, Seung-Hwan Lim, and Robert M Patton. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, page 4. ACM, 2015.

[60] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.