Spring 2018

A Study of Neural Networks for the Quantum Many-Body Problem

Liam B. Schramm
Bard College, ls2181@bard.edu

A Study of Neural Networks for the Quantum Many-Body Problem

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
Liam Schramm

Annandale-on-Hudson, New York
May, 2018

Abstract

One of the fundamental problems in analytically approaching the quantum many-body problem is that the amount of information needed to describe a quantum state. As the number of particles in a system grows, the amount of information needed for a full description of the system increases exponentially. A great deal of work then has gone into finding efficient approximate representations of these systems. Among the most popular techniques are Tensor Networks and Quantum Monte Carlo methods. However, one new method with a number of promising theoretical guarantees is the Neural Quantum State. This method is an adaptation of the Restricted Boltzmann machine(RBM). Unlike the traditional RBM, Neural Quantum States act as a feedforward network, calculating a single complex value of the wave function for every spin configuration. We examine this method, and compare its performance to a feedforward network for a similar problem. Another recent application includes the use of neural networks for detecting phase transitions. We examine the claims made about this technique and propose a new method for solving this problem. We report results for both experiments.

Contents

Dedication

I dedicate this senior project to Karuna, for being there every step of the way.

Acknowledgments

I would like to acknowledge the help of my advisors, Paul Cadden-Zimansky and Sven Anderson, for helping me to turn this project into the best work it could be. I would also like to acknowledge Bobby Mills for his debugging help, as well as Karuna Sangam, for their LaTeXpertise.

1
Introduction

This thesis will address the use of artificial neural networks to solve common problems in quantum mechanics. We will give an overview of these problems, as well as common solutions. We will also discuss computational formulations of these methods, and how they relate to the new neural methods.

1.1 A Brief Study of Quantum Mechanics

Quantum dynamics are very different from classical dynamics. The biggest mathematical difference is that in classical mechanics, there is only one possible configuration a system can take at any given time.

If a ball is thrown, it will follow a single physical trajectory through the air until it lands. At each point in time, the ball has one location, one velocity, one energy, and one angular momentum. In quantum mechanics, this is no longer the case. The ball no longer has a well-defined value for any of these physical measures. Instead it has a wave function of a system which encodes information about all the values the ball *could* have. When the system is measured, it could be in any one of a wide range of physical configurations, each with a probability described by the wave function. This function is a single complex-valued function that has a value for every configuration the system could take and carries all the physical information about that

system. Instead of the baseball having a single angular momentum and position, in quantum mechanics the ball's wavefunction would have a complex number for each combination of angular momentum and position. One important detail here is that the wavefunction is not necessarily separable into single-variable components. It might have a high value when the ball is spinning clockwise and is far to the right, and a high value when the ball is spinning counterclockwise and is far to the left, but be very small in other configurations. The wavefunction's value is not simply a product of its effect on individual variables.

1.1.1 Operators

Information is pulled out of the wave function using operators. In classical mechanics, to find a value for momentum, we would take the derivative of the object's position and multiply by mass. However, in quantum mechanics, we do not have a unique position value to work with. Instead, we must change our calculations to instead extract information from the wavefunction. This requires the introduction of ideas from linear algebra.

One important feature of the wavefunction is that solutions can be added together to produce new solutions. These solutions obey the rules of linear operations, so we can say the quantum mechanics in is a sense linear. This means that we can treat the wavefunction as a vector in a vector space that contains all possible wavefunctions for a system. Operators take the role of matrices, mapping wavefunctions in the vector space to other wavefunctions. The eigenvalues of these operators are the physical values that can be observed when applying that operator.

The linear algebraic structure of this field lends itself to a notation that makes vectors and matrices easy to describe. For this reason, Dirac's bra-ket notation is the standard method for describing quantum mechanical systems. [1] The three basic elements of this system are bras, kets, and operators. Kets represent vectors in the vector space of states. So, if the wavefunction $\Psi(x)$ is function from $\mathbb{R}$ to $\mathbb{C}$, then the state corresponding to $\Psi(x)$ is the ket $|\Psi\rangle$. If ket represents the function version of a column vector, then bra takes the place of a row vector. Multiplying

[1] pronounced "brah - keht". The phrase is a pun on "bracket", which everyone still resents Dirac for.

a bra by a ket (written as $\langle\Psi|\Psi\rangle$) gives us the inner product of the two states. For systems with a finite number discrete degrees of freedom (the kind of system we will concern ourselves with), this is the same as a dot product. For continuous systems, it usually takes the form of an integral on the range $\{-\inf, \inf\}$. Operators can act on both bras and kets. An operator O on a ket would be written $|O\Psi\rangle$ or $O|\Psi\rangle$. Moreover, this notation offers us a convenient way of writing the expectation value of an operator on a state. This can be written as $\langle\Psi^*|O|\Psi\rangle$, where $*$ denotes the complex conjugate. Taking the complex conjugate ensures that the value given will be real (assuming the operator follows certain rules which all observable operators do).

Let us take a simple example. Our Ψ looks like the Gaussian function, $G(x)$, and we want to know the expectation (statistically average) value of x. Since this is a bell curve, we should expect the average value to fall exactly in the middle, at 0. But to perform the full calculation, we would take $\langle G^*(x)|x|G(x)\rangle$. x as an operator merely multiplies the function by x, so $x|G(x)\rangle = |x * G(x)\rangle$ Since G is continuous, the inner product will be given by an integral over the real line. Thus, $\int_{-\infty}^{\infty} G^*(x)xG(x)\ dx = \int_{-\infty}^{\infty} x|G|^2(x)\ dx = 0$. The expectation value is 0, just as we would expect.

For our system to behave like a physical system, it first needs constraints on Ψ to restrict it to only physical solutions. The first constraint arises naturally from the way we have defined expectation values. If $\langle\Psi^*|O|\Psi\rangle$ is the expected value of O on Ψ, then $\langle\Psi^*|\Psi\rangle = \langle\Psi^*|1|\Psi\rangle$ is the expected value of 1. Since this should obviously be 1, we know that $\langle\Psi^*|\Psi\rangle = 1$ for all physical wavefunctions Ψ.

The other main constraint comes from the Schrodinger equation: $i\hbar\frac{\partial}{\partial t}\Psi = -\frac{\hbar^2}{2m}\nabla^2\Psi + V\Psi$, This equation works like Newton's laws for quantum mechanics, in that it provides the equations of motion for the system. The intuition for this equation comes from the classical equation for energy. Classically, $E = p^2/2m + U$, where E is energy, p is momentum, and U is potential. In quantum, instead these values are replaced by operators. The left side of the equation is the energy operator, $i\hbar\frac{\partial}{\partial t}$. Momentum is replaced by the momentum operator $p = -i\hbar\nabla$. Lastly, potential energy is replaced with the potential V. The right side of the equation (taken as an

operator on Ψ) is called the Hamiltonian, $\hat{H}$. It turns out that a state is time-independent or stationary only when it is an eigenvector of the energy operator. This gives us the time-independent Schrodinger equation $E\psi = \hat{H}\psi$. [2] Since the energy operator equals the Hamiltonian, this means we can look for eigenstates of the Hamiltonian to use as a basis for our solutions. The vector space this forms is called a Hilbert space, and will be discussed in the next section. All "superpositions", states that are a linear combination of energy eigenstates, evolve in time deterministically. So, the most common way these problems are solved is first by finding the list of energy eigenstates, then building the state from those, and finally finding the time-evolution of the state from the interaction of energy states.

The last constraint deals with continuity and smoothness of the wavefunction. The wavefunction is always continuous. It should also be smooth everywhere, except at points where the potential goes from finite to infinite. Lastly, the wavefunction should be twice differentiable except when the potential is discontinuous.

1.1.2 Hilbert Spaces

For the purposes of physics, a Hilbert space is a vector space with an inner product $\langle \cdot, \cdot \rangle$ and a norm defined by $\|v\| = \sqrt{\langle v, v \rangle}$. Technically, there is also a completeness condition, but this condition holds for all physically realizable systems, so physicists can assume it holds without proof. Hilbert spaces are important to quantum mechanics because they allow physics to use vectors and matrices to represent states and operators. This makes it easy to perform arithmetic operations over these objects, as well as making it easier to show properties like orthogonality between different states. In certain practical situations however, like systems with many particles , Hilbert spaces can be difficult to work with. This is because such a resultant space is not the direct sum of parts as one might naively expect, but the tensor product of these parts. Consider for a moment a two particle system. Each particle has spin $\frac{1}{2}$, meaning that it can have an angular momentum of either $+1/2$ (spin up, or $|\uparrow\rangle$)or $-1/2$ (spin down, or $|\downarrow\rangle$). The system has

[2] Physics convention has it that the capital Ψ is used to represent a time-dependent wave-function, while the lowercase ψ is used to represent the time-independent wavefunction

four possible configurations: $|\uparrow\uparrow\rangle, |\downarrow\uparrow\rangle, |\uparrow\downarrow\rangle$, and $|\downarrow\downarrow\rangle$. These four states form a basis for all the system's possible states. So, it's also possible to create states that are linear combinations of these states. We could define a state $|\Psi\rangle = \frac{1}{\sqrt{2}}|\downarrow\uparrow\rangle + \frac{1}{\sqrt{2}}|\uparrow\downarrow\rangle$. This state would have a wavefunction such that $\Psi(\downarrow\uparrow)$ and $\Psi(\uparrow\downarrow)$ are non-zero, because $|\Psi\rangle$ is not orthogonal to either state, meaning that measuring the spins of the state could produce either the $|\downarrow\uparrow\rangle$ state or the $|\uparrow\downarrow\rangle$ state. However, there is no chance of observing the state in the configuration $|\uparrow\uparrow\rangle$ or $|\downarrow\downarrow\rangle$, so $\Psi(\uparrow\uparrow) = \Psi(\downarrow\downarrow) = 0$. [3]

If we choose to add a third particle to this system, we instead have eight observable configurations. Our basis now consists of the vectors: $|\uparrow\uparrow\uparrow\rangle, |\uparrow\uparrow\downarrow\rangle, |\uparrow\downarrow\uparrow\rangle, |\uparrow\downarrow\downarrow\rangle, |\downarrow\uparrow\uparrow\rangle, |\downarrow\uparrow\downarrow\rangle, |\downarrow\downarrow\uparrow\rangle$, and $|\downarrow\downarrow\downarrow\rangle$. The number of basis vectors needed to span the Hilbert space continues to double with every new particle we add to the system. By the time our system has 45 particles, it would take several terabytes of disk space simply to list the basis vectors.

Given this information, it should not be surprising that even simple problems that involve dealing with these systems are members of difficult to solve complexity classes. Finding the lowest energy state the system can occupy is NP-Hard. Finding how many state vectors are tied for the lowest energy is #P-Hard [55, 5]. And because it's possible to make a Turing machine in a quantum mechanical system, answering non-trivial questions about whether a particular property will hold over the full time evolution of a system is undecidable.

For all these reasons, it is important that we find efficient representations of these quantum states that can be efficiently stored and manipulated. This involves making some kind of approximation, and different approximations hold better in some regimes than others. Matrix product states, for instance, are not capable of describing critical or scale-invariant systems, but are very efficient at describing localized Hamiltonians. Our task, then, is to find an approximation that can be tuned to accurately represent most physical systems with arbitrary accuracy.

[3] We do not normally talk about spins as having a wavefunction, but in order to use a neural network on a system, we need to describe the system in terms of a function instead of a state or vector. Describing spins with a wavefunction is notation adopted from [33, 3]

Before we can delve into the specifics of these methods, however, we need a more formal description of the problem. For the moment, we will restrict our conversation to systems of spin $\frac{1}{2}$ particles, where the Hamiltonian depends exclusively on spin. We can say that the wave function of a such a system is a function from $R^{2^n} \rightarrow C$. We want to approximate this function as closely as we can, using as little memory as we can. Traditionally, this has been done using numerical methods such as tensor networks and the Metropolis-Hastings algorithm

2
Overview of Traditional Approximation Methods

2.1 Tensor Networks

One well known technique for approximating the wave function is the Tensor Network [22, 2]. These techniques have an advantage in that they are able to efficiently represent the geometry of the systems they describe. Tensors are a generalization of matrices to higher dimensions. If we think of a matrix as a square of coefficients, then tensors could represent a cube of coefficients for a tensor of rank n, or an n-dimensional hypercube for a tensor of rank n [1]. Formally, a tensor T is a multi-dimensional array with a number of indices $(a, b, \dots)$ equal to the rank of the tensor. Tensors turn out to be a useful way of representing quantum states of many-body systems, because their exponential size matches the entanglement factor in these systems. However, this exponential size still creates a memory issue. To limit the number of parameters in play, tensor networks restrict the system to only tensors that can be written as a product of a set of tensors of a given size.

Just as tensors function as a high-dimensional analogue of a matrix or vector, it also requires a generalization of matrix or vector multiplication that functions at higher dimension. Two forms

[1] The term rank is used because in linear algebra, dimension is associated with the number of free parameters needed to completely specify a particular object. The space of all 3×3 matrices is 9 dimensional because it takes 9 numbers to specify one particular 3×3 matrix, and rank 2 because it takes two indices to specify the location of any coefficient in the matrix.

of this are the direct product and the tensor contraction. The direct product is a generalization of the Cartesian outer product. If the two input tensors A and B are rank a and b respectively, then the product tensor is rank $a+b$. Each element of $E^{j_1 \dots j_b}_{i_1 \dots i_a} = A_{i_1 \dots i_a} * B^{j_1 \dots j_b}$, where $i_1 \dots i_a$ and $j_1 \dots j_b$ are the indexes for individual elements of each tensor. For instance, the tensor product of the two vectors $v = \begin{bmatrix} a & b & c \end{bmatrix}$ and $w = \begin{bmatrix} d \\ e \\ f \end{bmatrix}$ is $\begin{bmatrix} ad & bd & cd \\ ae & be & ce \\ af & bf & cf \end{bmatrix}$.

The tensor contraction is more similar to an inner product than an outer product, since it includes a summation. Instead of just adding all the indices to the new vector, certain indices are summed over. A matrix multiplication is a good example of this. If we have two matrices A and B, then the product $C^j_i = \sum_k A^k_i * B^j_k$

The bond dimension of the tensor network is related to the size of each tensor in the network and limits the way they can be multiplied together. It is the number of indices that are summed over when performing a tensor contraction. To perfectly represent a system, we would need a bond dimension equal to the size of the system. However, good approximations are possible at low, constant bond dimensions. Luckily, it turns out that almost all physically relevant systems can be expressed in this way.

Tensor networks are written as sets of tensors connected by tensor contractions. This operation is a generalized form matrix multiplication that can be applied to tensors of any rank. Using this operation, we can write some large tensors as a tensor contraction over several small tensors. This formulation is a dramatic scale improvement over the original formalism, as it only uses a polynomial number of coefficients ($O(\text{poly}(n) * \text{poly}(D))$), where n is the number of particles in the system) to represent a given state if the smaller tensors have a fixed size. Even better, the indices used in the tensor contraction turn out to have physical significance. They can be taken to represent the structure of many-body entanglement in the system, with the size of D corresponding to the degree of entanglement in the wave function. The tensors themselves correspond to particle sites.

The most famous version of the Tensor Network is perhaps the Matrix Product State. This is a spacial case that uses rank 2 tensors(matrices) to model a 1D lattice. Each site in the lattice

is represented by a matrix. The interactions between neighboring sites are represented by the bond indices between the matrices, and the open indices correspond to degrees of freedom in the local Hamiltonian. MPSs can represent translationally invariant systems efficiently by imposing the condition that the matrices repeat in a cycle over some finite number of spacial steps. Although they are efficient at representing low energy regions of local, gapped Hamiltonians, MPSs begin to fail when the spectrum of low-energy states is continuous or the Hamiltonian depends on highly nonlocal interactions. In particular, they cannot approximate systems near a critical point, since the correlation length diverges as the system approaches that point.

Correlation length is a measure of how much particles at distant areas are likely to be correlated. In a completely random system, the correlation length would be zero because no particle's position would be correlated with that of any other. However, when a system of spins "freezes" into place at low temperatures, such as with a ferromagnet, correlation length becomes infinite because each particle is pointing in the same direction, and is thus correlated with every other particle. Representing these "frozen" states is not usually very difficult because they have low entropy and are easier to describe conventionally. However, matrix product states can have a hard time describing higher temperature states as they approach the phase transition. States like this might display a combination of frozen regions with areas that have not frozen out yet, and therefore do not work well with this kind of approximation.

2.2 Metropolis-Hastings

While tensor networks attempt to approximate individual states, the Metropolis-Hastings algorithm serves a slightly different function. At its core, Metropolis algorithm is a Monte Carlo bootstrap algorithm. It attempts to reproduce the effect of sampling from another distribution without actually reproducing the target distribution.

The Metropolis Hastings algorithm is, to be more specific, a Random Walk Markov Chain Monte Carlo algorithm. These algorithms model the target probability distribution as a Markov Chain: a directed graph with transition probabilities between each node. Each node represents

a possible state the system can be in. Thus, the graph always has one "active" node that corresponds to the state. This graph is Markovian, meaning that the transition probabilities depend only on the current state.

The distribution is produced by doing a random walk over the Markov chain. However, unlike a normal random walk, the state does not automatically change to the new proposed state. Instead, there is a chance of rejecting each new proposed state based on the *a priori* probability of that state occurring. If the new state is more probable than the old one, the new state is immediately accepted. Otherwise, there is a random chance that the new state will be rejected. The formula for this chance is as follows: $a = \mathrm{P}(x')/\mathrm{P}(x)$, where a is the the probability of acceptance, x is the current state, , x' is the proposed new state, and P() is the probability of a state occurring in the true distribution.

The full algorithm then follows these steps:

1) Select a random state x from the state space

2) for i in range(n)

select an adjacent state x'

set $a = \mathrm{P}(x')/\mathrm{P}(x)$

$u =$ random number in $[0, 1]$

if $u <= a : x_{(t+1)} = x'$

else if $u > a : x_{(t+1)} = x$

This works fine for traditional Bayesian probability distributions, but becomes somewhat trickier when applying this algorithm to physical systems. Here, instead of probabilities, energies are used to calculate the chance of acceptance. For this, the Boltzmann distribution is used. According to this theory, the expected number of particles in a particular single-particle microstate is proportional to $e^{-E/(kT)}$. Thus, $\mathrm{P}(x')/\mathrm{P}(x) = e^{-E'/(kT)}/e^{-E/(kT)} = e^{-(E'-E)/(kT)}$.

For simplicity, we set $k = 1$, so $a = e^{-(E'-E)/(kT)}$, where E is the energy of x, E' is the energy of x', and T is the temperature. [2]

The Metropolis-Hastings algorithm has a number of advantages. Mainly, it is very simple to implement and easy to understand. However, it suffers from convergence issues. Namely, it takes $O(n^3)$ time to converge with the size of the system. This can make the method impractical for large systems. In some cases, it can also take linear time to calculate the difference between the energies of adjacent states, increasing the total time to convergence to $O(n^4)$. Another issue is the correlation of points that are close together in sequence. While over time, the samples drawn from the Metropolis-Hastings algorithm do converge to the actual distribution, samples drawn in short intervals are strongly correlated with each other and do not represent the real distribution. This means that for good results to be achieved, a gap must be included between recordings of the state.

[2]Many MCMC algorithms include a temperature as a tuning parameter for the degree of randomness in a system. This is one of the few situations where the temperature actually refers to a physical temperature. Here, it comes from the use of Maxwell-Boltzmann statistics to calculate the probability of a given state.

3
Ising Model

The high dimensionality of Hilbert spaces affects condensed matter physics significantly more than other fields, because condensed matter physics tends to deal much larger numbers of particles than other fields. For this reason, many of the approximation methods discussed above are first tested on statistical mechanics models. By far the most common such model is the 2D Ising model. This model is sometimes referred to as "the hydrogen atom of statistical mechanics". Like the hydrogen atom in quantum mechanics, it is very simple, yet displays many of the interesting ideas present in more complex models. The model consists of a set of spins in a square 2D lattice. Each spin can be in either an up or down state. The Hamiltonian has two terms. One is a nearest-neighbor interaction. Aligned particles have a contribution to the energy of $-J$, and anti-aligned particles have a contribution of $+J$. Since we will be dealing with the ferromagnetic (self-aligning) version of this model, we will set $J = 1$ The second term comes from the magnetic field. Particles aligned with the magnetic field have a contribution to the energy of $-h$, while particles anti-aligned with the field have a contribution of $+h$. Thus, the complete Hamiltonian can be written as $H = J\sum_{i,j}\sigma_i\sigma_j + h\sum_i \sigma_i$, where σ_i is an operator that measures the spin of the particle at position i. The first summation is done only over adjacent particles.

It should be simple to see that the ground state of this system occurs when all particles are aligned with the magnetic field. It is less trivial, however, to show that this system has a phase

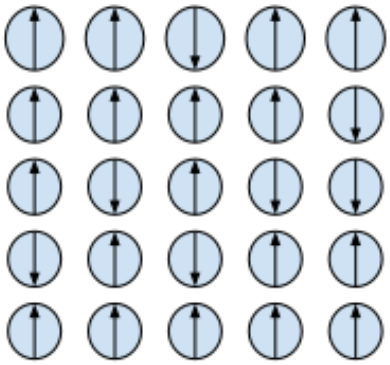

Figure 3.0.1. One configuration of the 2D Ising model

transition. At a certain temperature ($T = 2.269$ when $J = 1$ and $h = 0$), the system suddenly flips from the ground state to being disordered. One common test for numerical methods is their ability to reproduce this phase transition.

4
Neural Networks and Neural Quantum States

4.1 A Brief Neural Network Primer

Neural networks are some of the stranger machine learning methods out there. For one, they've changed rather dramatically over their 60 years of development. The definition of what actually constitutes a neural network has shifted as well, and is sometimes driven more by analogy than by similarity to existing methods. Perceptrons, Hopfield nets, Boltzmann machines, spiking networks, and modern deep architectures have little in common other than the fact that they are graphical methods that include summations. In addition, some classes of neural networks are provably Turing-complete, meaning that there it is always possible to create a neural network that perfectly imitates any given computer program. Thus, we cannot even say that neural networks are a tool for a specific type of problem, since they can demonstrably solve any problem. For this work, we will restrict our focus to two classes of network: Differentiable networks like Multilayer perceptrons and deep architectures, and Boltzmann machines.

Multilayer Perceptrons, or MLPs, were a response to the limitations of the earlier perceptron model. The basic structure of both these models is layers of matrix multiplication, followed by the application of an activation function. Basically, the network receives a set of inputs in the form of a vector of numbers of size n. For each neuron N_i, each value O_j in the output layer is multiplied by a weight w_{ij}, and then added to a bias β_j. The neuron's input weight α_i is

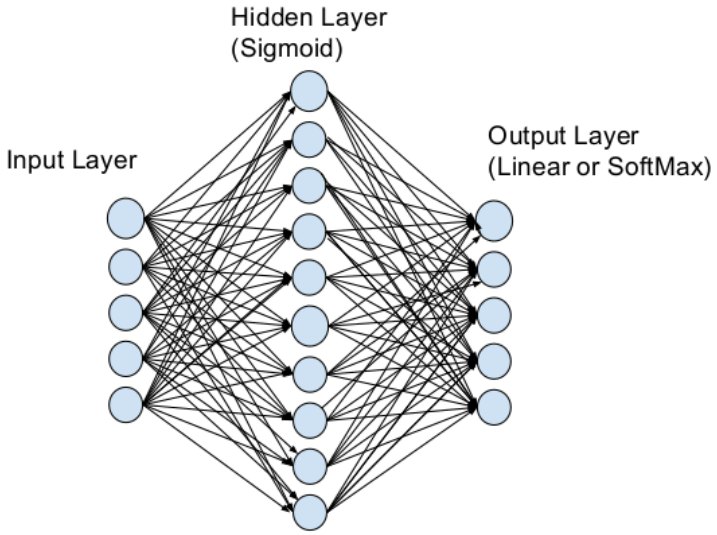

Figure 4.1.1. Multi-layer Perceptron

$\sum_{j=0}^{n}(w_{ij} * O_j + \beta_j)$. The neuron's output β_i is then β_i =activation(α_i). This is supposed to be analogous to the way axons and dendrites work in the brain. Each neuron's potential is the sum of the potentials from the presynaptic neurons. The activation function is what separates these two models of neurons. In perceptrons, the activation function is a simple threshold that represents whether the neuron is firing or not. In MLPs, this activation function is instead a sigmoid, which represents the average firing frequency over time. This is significant because the sigmoid is differentiable, and thus networks made by composing sigmoids and matrix multiplications are differentiable as well. Having the entire network be differentiable means it can be trained by gradient descent. The older perceptron model was restricted to using the Hebbian rule, a biologically inspired learning algorithm which could only train two-layer models, and so could only learn linear functions. Gradient descent however, could train networks of arbitrary depth, allowing them to approximate any function with arbitrary accuracy.

Deep networks function off the same basic structures as MLPs, but use a larger number of narrow layers instead of three very wide ones like MLPs. This is what 'deep' refers too. While MLPs always have three layers, deep networks sometimes have hundreds. The reasoning is that having multiple layers allows the network to learn useful representations of data in greater and

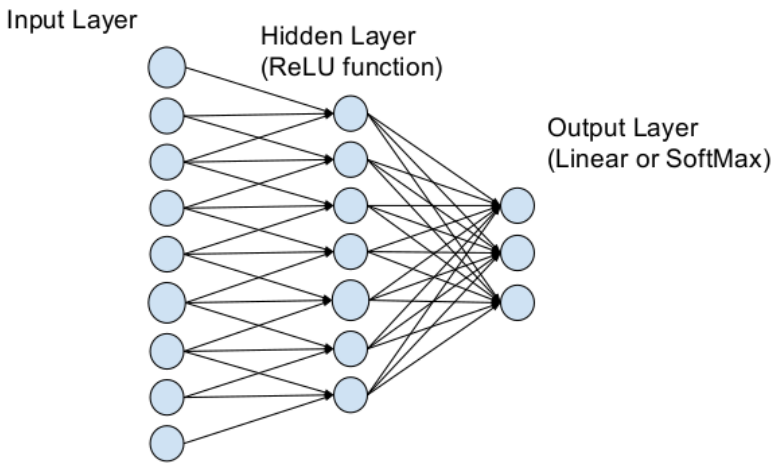

Figure 4.1.2. A simple convolutional network

greater levels of abstraction as data moves up the network. Mathematically, there are several efficiency theorems that show deep networks sometimes require exponentially fewer parameters than shallow networks to achieve the same accuracy. However, training lower-level layers becomes difficult as the network gets deeper. The gradient of a sigmoid is ≤ 1 everywhere, so each chain rule back to a lower layer involves the gradients being multiplied by a number ≤ 1. This makes the gradients become exponentially smaller in lower levels. Most modern deep architectures are built around tricks to minimize this problem. The best known of these tricks is the convolutional network. Convolutional networks are used for systems with shift invariance or similar spatial symmetries, such as image recognition. In these networks, lower layers consist of a small set of neurons with only local connections that is swept (convolved) over the entire input layer, one increment at a time. For images, this is effective because low-level representations such as edge detection are relevant no matter where they are in the image. Higher layers have a similar structure, convolving other neurons over the outputs of the lower level neurons to create more and more complex features. For instance, if the first layer looks for edges, higher levels might learn to recognize corners, T-shapes, and crosses. These networks also usually include pooling or downsampling layers, which select highest output from neurons in a given area and passes it on to the next layer while dropping the rest. This reduces the necessary number of parameters for the network by significantly reducing its size, at the cost of losing precise locational information.

Boltzmann machines are very different from feedforward-type networks. Unlike more traditional neural networks, they don't use deterministic, differentiable activation functions. Instead, activations are probabilistic and binary. The sigmoidal activation function is instead replaced with a binary activation function, with a probability of firing equal to the logistic function of the input. The connections in this kind of network are also bidirectional, unlike the directional edges in traditional neural networks, with only a single visible set of neurons serving as both the input and the output layer. The bidirectional edges permits the definition of an energy function over the whole system. Over time, the network converges to a Boltzmann distribution, where the state vector is determined only by the energy of the system. Typically, this distribution is used for generative purposes. The network is trained so that training vectors have a high probability with respect to the distribution. Sampling this distribution then gives similar vectors to the training vectors. The main issue Boltzmann machines have is that they take exponential time to train if they have more than two layers. On the other hand, Boltzmann machines with only two layers must be exponentially large to approximate certain distributions.

4.2 Neural Quantum States

The first highly successful attempt to use neural networks to solve quantum mechanics problems was Carleo et al.'s Neural Quantum States (NQS) [33, 3]. In a sense, this approach is a response to the failings of other numerical methods for solving quantum mechanics problems. Unlike Quantum Monte Carlo methods, NQS's do not suffer from the sign problem. Unlike Matrix Product States, they do not have limits on their correlation length, so they don't run into difficulties when trying to model critical systems. Instead, the network is a universal function approximator that can approximate any function in the large-N limit. This is the first major advantage of neural networks: they are well studied and have a range of universality proofs.

The NQS is based on the binary-activated Restricted Boltzmann Machine (RBM), a two-layer neural network architecture originally used for learning probability distributions. This architecture is a surprising choice, given that this particular structure is rarely used today.

It first became popular in the mid-2000s as a faster-training version of the Deep Boltzmann Machine. However, it has seen little use in the last 5-10 years, since it has largely been surpassed by other generative models such as generative adversarial networks.

Carleo demonstrates two uses of this architecture. The first is finding the energy eigenstate with the lowest energy, called the ground state. This is an important piece of information, because for most systems, the ground state is by far the most probable configuration to find a system in. For instance, the probability of finding a hydrogen atom at room temperature outside of ground state is on the order of one in 10^{171}. For many systems, approximating only the ground state and first few excited states is sufficient to find a good approximation.

In neural network terms, finding the lowest energy is simple: find the set of weights that minimizes the expected value of the Hamiltonian. The fact that the data can be created automatically using a Markov Chain Monte Carlo (MCMC) method like the Metropolis-Hastings algorithm ensures that this method never runs out of data or overfits. The Hamiltonian also provides the benefit of a built-in loss function with little need for tweaking. However, an additional problem is still present in the size of the configuration space. To evaluate the actual Hamiltonian for the network, it would be necessary to evaluate the network on all 2^N configurations (One configuration for every possible combination of spins). To make this tractable, the authors instead use a MCMC method to sample the configuration space and find an approximate value for the Hamiltonian. With this method, it is possible to find an approximation of the ground state of a system to within a given accuracy, all in $O(\text{poly}(N))$ time, where N is the size of the system.

The second use of these types of networks is for observing the time-evolution of a particular state. To do this, they make all the weights of the network time dependent. This has the effect of making the network's output time-dependent. Instead of attempting to minimize the Hamiltonian, instead the variational residuals are minimized. These are again sampled using a variational Monte Carlo method.

For the ground state problem, Carleo et al. demonstrated that they could produce results of equivalent quality to other approximation methods. They compare the quality of their ground

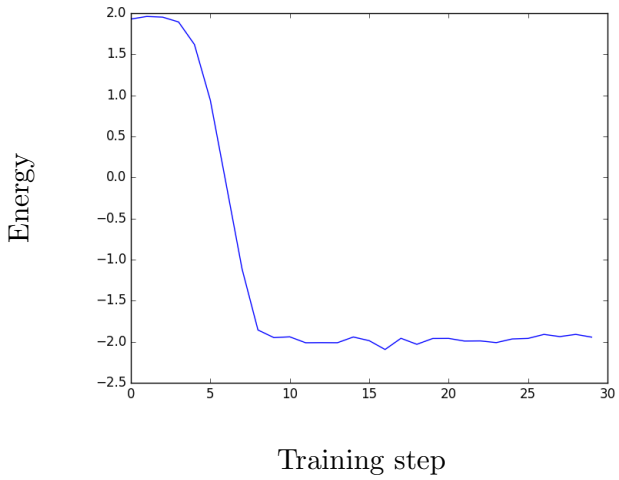

Figure 4.2.1. Energy over time as the Neural Quantum State converges to the ground state

state approximation of a 2D Ising model to the results given by EPS and PEPS, two types of quantum tensor networks. They argue that the number of hidden neurons in the system is analogous to the bond dimension of a tensor network. On this assumption, the performance of the neural quantum state improves much more quickly with high bond dimension than a similar tensor network. An NQS with a bond dimension of 4 was able to achieve a error of 10^{-5}, whereas PEPS was unable to achieve 10^{-3}.

We were able to reproduce the results of this paper for the ground state problem. I based my code on an implementation of the paper we were able to find on github. With this implementation, we were able to test the papers claims. The training algorithm proposed by the authors works, and the network does converge to the correct value. We built a randomized network with a magnetic of of strength $h = 2$ and twice as many hidden neurons as input neurons, and trained the network for 20000 training steps.

Although their results reported by the paper were good, there are a number of reasons to desire a different approach. Firstly, in this paper, the RBM was used for a regression task. The problem was framed as taking a particular system configuration and outputting a single complex value to represent the value of the wavefunction for that configuration. Typically, other network architectures are used for this type of task. Simple feedforward networks can be very effective

for this kind of problem, as can convolutional networks in situations with shift-invariance. The explanation to the choice to use an RBM is tucked away in the appendix of the paper, where the authors state that they only used an RBM because they were unable to get more common architectures to work the for problem. So it seems natural to suspect that other network architectures may produce better results with correct tweaking. If they do not work, it raises the interesting question of why these very well-tested methods fail for what seems to be a very basic task. Is this a problem specific to feedforward architectures? Or does it apply to all soft(not binary)-activated networks? The second reason to consider a switch to a different network architecture is that most of the success of neural networks over the last 10 years have not come from shallow networks like RBMs, but from deep networks like convolutional networks. While RBMs have their uses, they are no longer considered a prime tool in a machine learning toolkit. The authors themselves seem to think that deep networks may be a better choice, as they call out in the future work section. Two interesting questions thus arise: 1) Are deep networks a better tool for treating quantum many-body systems(QMBS) than RBMs? 2) If the failure of deep networks on QBMS is generalized, and not merely a failure of implementation, what is preventing these networks from performing well on this task, when they are known to perform well on so many others like it?

4.3 Deep Networks vs. Shallow Networks

Deep networks have some specific advantages over shallow networks, as well as some specific disadvantages. Montufar et al have shown that deep networks can represent exponentially more complex functions than shallow networks [44, 4]. Similarly, Gao et al have demonstrated an asymptotic difference in performance between Deep Boltzmann Machines(DBMs) and Restricted Boltzmann Machines, assuming that P $\neq$ NP [1010, 10]. However, DBMs are asymptotically more difficult to train. Indeed, the entire reason for the prevalence of RBMs is that they have a polynomial time learning algorithm. For this reason, another deep architecture is likely preferable to using a DBM. In most deep learning applications, the deep network is used because we wish

to make use of a particular symmetry of the training data. In learning on images, we take advantage of semantic symmetries, translations under which the object an image represents does not change. A picture of a car shifted a few pixels to the left is still a picture of a car. The same is true for small rotations, scalings, hue shifts, and small amounts of random noise. These allow us to make networks that are invariant under these kinds changes. Additionally, pictures have a strong aspect of locality. Pixels are much more important to the pixels next to them than ones at the opposite ends of the image. The reason convolutional networks are so successful is that they are good at taking advantage of these kinds of symmetries. Since layers are not fully connected, the lower layers only find local features, and the same local features are extracted at every location.

When designing an architecture for a quantum system, similar concerns apply. For instance, if we are modeling a system with a finite correlation length, we want to prioritize local interactions. This involves modifying the RBM to focus specifically on these kinds of interactions. Additionally, we may want to impose a variety of symmetries on the system, likely including translational and/or rotational invarience. Models like convolutional neural networks come with with these symmetries built in, which could make them significantly more efficient at modeling these types of systems.

4.4 Exploring Feedforward Networks for Modeling States

Our first goal was to test the effectiveness of a feedforward network on a toy problem to test the viability of the technique. Arguably the simplest problem in quantum mechanics is the 1-dimensional infinite square well. The idea is that you have a single particle in a potential well. The potential is zero on the interval $[-a, a]$ and infinite everywhere else. The Hamiltonian is thus defined as $H = \frac{\rho^2}{2m} + V_{well}$. The solution follows fairly simply. If Ψ is non-zero outside of the interval with zero potential, then that point's contribution to the total energy is infinite. Since infinite energy is non-physical, the problem is reduced to solving $H = \frac{\rho^2}{2m}$ with $\Psi(x) = 0$ for all $x > a$ and $x < -a$. Since there is also a continuity condition, all states must have $\Psi(a) =$

$\Psi(-a) = 0$. Under these constraints, the eigenfunctions of the Hamiltonian are $\sin(\frac{n*\pi*x}{2a})$, where n is a positive integer.

For our problem, the neural network should take in a single real value (position) and output a single real value (wave amplitude). Since we will be finding a ground state, the state is time-independent, so the amplitude and phase are separable. This means that we can ignore the phase and only model the amplitude. This would suggest a 1-input, 1-output network with a hidden layer. However, here we encounter a problem. Although the (phaseless) wavefunction is a function from $\mathbb{R}$ to $\mathbb{R}$, the Hamiltonian is not defined for a single value. Since ρ includes a derivative operator, we cannot calculate its value for a single number. We have to modify the network to instead accept and return a vector, and perform a numerical 2nd derivative on the vector. Each element of the input vector can then correspond to one position value, and each element of the output vector will be the amplitude of the wavefunction at that point.

Using this method of approximating the Hamiltonian, we can create a simple feedforward network to approximate the wave function. We trained a 3-layer, densely connected network. The loss function set as $\frac{\langle\Psi|H|\Psi\rangle}{\langle\Psi|\Psi\rangle}$. The reason the expectation value energy $\langle\Psi|H|\Psi\rangle$ was divided by $\langle\Psi|\Psi\rangle$ was to make sure the output value was normalized. Otherwise, the network could reach arbitrarily low loss values simply by making the output vector very small. Dividing by the Ψ's inner product with itself is just an efficient way of preserving the 'norm = 1' condition.

4.5 Experimental Results

Despite its similarities to both the Neural Quantum State architecture, we found that the feed-forward network did not converge to an accurate approximation of the ground state. Instead, the network quickly converged to returning a normalized value of 1 for every position value. This means that at $-a$ and a, the wavefunction is discontinuous, because beyond those values, it is always zero. This produces a much higher energy than would be given by the correct ground state solution.

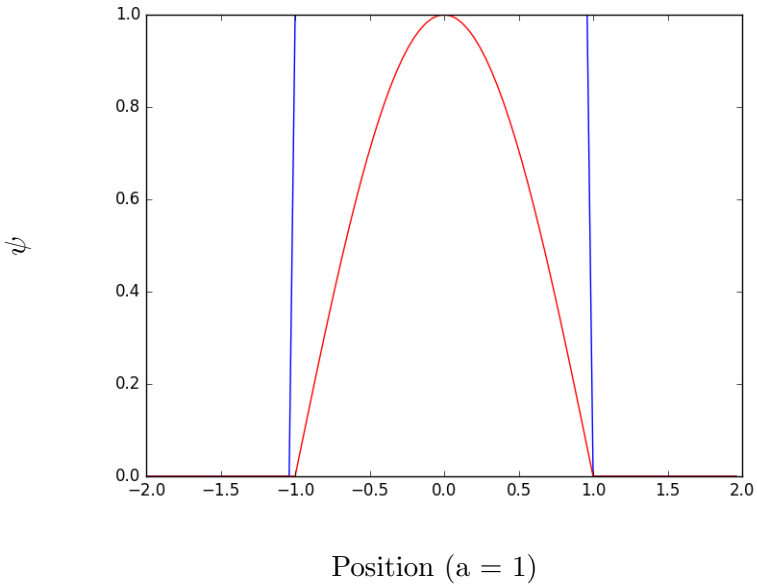

Figure 4.5.1. Blue line: Network's approximation of ψ Red line: Actual value of ψ

Our next step was investigating the reason for this failure. First, we changed the loss function so the network would learn the $\Psi(x) = \sin(x)$ solution directly. This worked exactly as intended, and the network learned the function easily. Testing this solution on the loss function gave the correct value, and the loss for $\Psi(x) = \sin(x)$ was much lower the loss for $\Psi(x) = 1$. This implies that the network got stuck in a local minima, and was unable to find the true global minimum. This is likely because of the very strange loss function this problem employs. Most loss functions for neural networks are very simple. Both Euclidean distance and categorical crossentropy are simple to compute and only involves a few multiplications and summations. Our loss function however, includes a numerical second derivative and a normalization. This could be a problem because complex mathematical operations can produce gradients that are difficult to follow and learn on. Many recent innovations in deep learning, such as ResNets and ReLU neurons, are simply ways of allowing the gradients to reach farther back into the network. So, it might be possible to fix the problem by instead using neurons that have an activation function more amenable to this type of problem. If for instance, neurons with a sin activation function were used, the network could learn to act like a Fourier decomposition. Because for sin, the second

derivative is just the same function multiplied by a constant, it might be possible to find a much simpler term for the Hamiltonian that was more amenable to gradient methods. The implementation of such a method though, is beyond the current scope of this project.

5
Networks for Phase Transitions

Phase transitions occur when there is a discontinuity in one of the properties of a material as a second property is altered. At a certain value of the second property, called the critical point, the properties of the material suddenly shift. Phase transitions are a common area of study, but pose a number of challenges to numerical methods. The conventional way of demarking phase transitions is to use order parameters. Order parameters are physical measures on a system that define phases. For instance, magnetic phases are measured by the average magnetization of a system. Finding phase transitions is frequently difficult when numerical methods are used to calculate the state of a system. Many methods, such as matrix product states, have difficulty describing long range correlations. Since correlation length can become infinite at the critical point, these methods have difficulty describing phase transitions.

It is important to note that the system in discussion for this section, the Ising model, only has a true phase transition when it is infinite size. It cannot have truly discontinous behavior at a finite scale. However, as the system size increases, the divide in the system's characteristics across the critical point becomes sharper, allowing us to approximate the effect of a true phase transition. As a result, the system actually has a transitionary regime between its two phases. This makes trying to detect phase transitions in this slightly ill-defined. However, we trust that

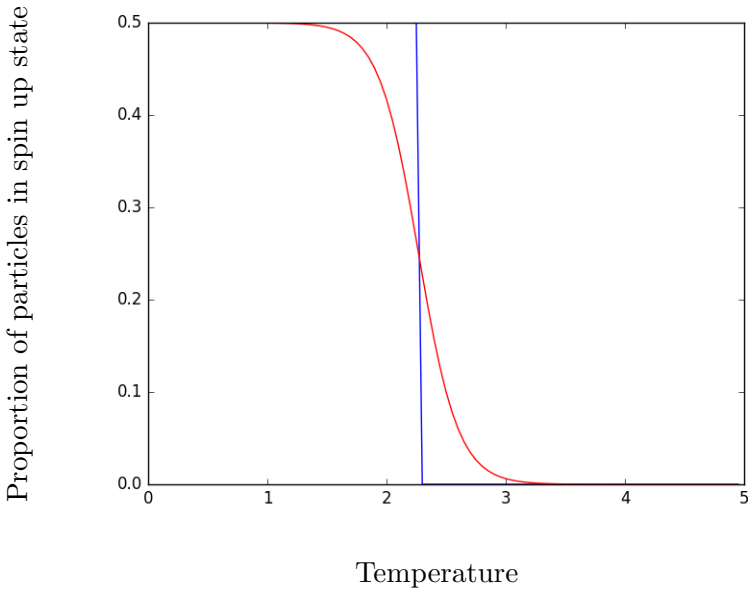

Figure 5.0.1. Blue line: Phase transition for infinite Ising model with true phase transition. Red line: Approximate phase transition for finite Ising model

the reader will agree that it is still possible to recognize characteristics of different regimes that can be observed and classified, albeit in a slightly less rigorous way.

5.1 Detection of Phase Transitions with Neural Networks

In the paper "Detection of Phase Transitions with Neural Networks", Tanaka and Tomiya propose a method for teaching neural networks to recognize these discontinuities [77, 7]. Traditionally, one of the main issues for using learning models in physics has been the availability of labeled data. It's not enough to simply have measurements of the state of a system; the measurements must usually must be labeled with the answers that you want the system to predict. For most problems physicists encounter, this is self-defeating. If you have a method obtaining labels for your measurements, why not just use that? Unlike many kinds of situations where learning is frequently employed, in physics there is no tight time window needed for a response. A hand-calculated result that takes a few hours is no impediment to publishing, whereas product recommendations need to be returned in a fraction of a second. Tanaki and Tomiya's solution to this problem is simple. Instead of using hand-gathered experimental data, generate data for

the 2D Ising model using the Metropolis-Hastings algorithm. Then train a network to predict the temperature used to generate the data.

The paper proposes using a simple convolutional model to detect these phase transitions. This network consists of an 2D convolutional layer and a fully connected layer. Although this is a very simple model, the problem does not demand a very complex architecture. Since particles interact only with their nearest neighbors, we do not need long-distance connections to do learning.

The authors run the Metropolis-Hastings algorithm on a 2D Ising system for a set number of iterations at a temperature T. Each state was recorded along with the temperature range it occurred at. The convolutional network was then trained to predict the temperature range that produced the state. So, for instance, if a particular state occurred when $T = 1.5$, the goal of the network might be to guess that the state occurred in the $1.4 < T < 1.6$ range.

5.2 Replicating Results

First, we present our results for attempting to reproduce the results of the original paper. We found that it was difficult to achieve any degree of accuracy when following the algorithm described by the paper. First we tried to replicate their results using a 10 by 10 lattice and their convolutional network. We ran the Metropolis-Hastings algorithm for 2,000,000 steps for each temperature range. We let the algorithm run for 1,000,000 steps before we began recording data, because the algorithm takes time to converge to the appropriate distribution. These numbers were chosen because they were the upper limit of what the author's computational platform could support. Readers attempting to duplicate results may choose higher values, but as we will explain below, these are sufficient for reasonable results. Before attempting to reproduce the paper's results with the neural network, we first wanted to verify that the data was of good quality. This is especially important for sampling algorithms like Metropolis-Hastings, where points sampled at nearby points in time are correlated with each other. To check that the data had the correct distribution, we plotted the number of particles in the spin up state against the temperature. This number should increase as the temperature increases.

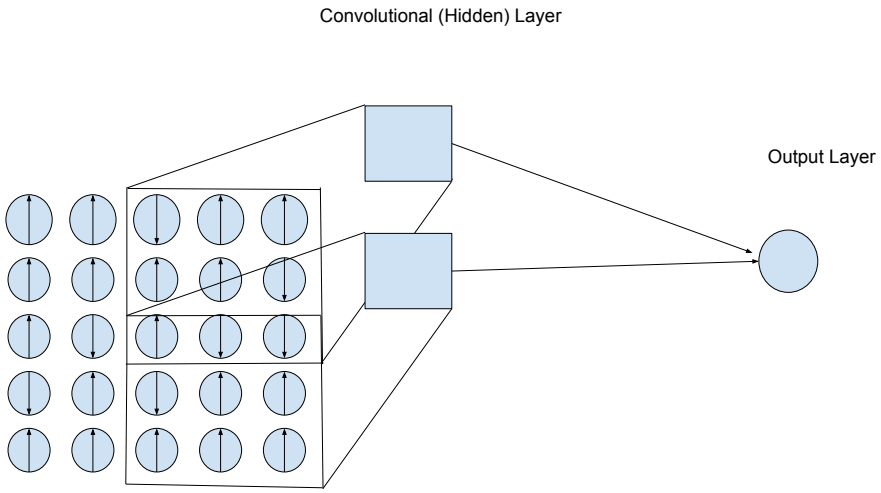

Figure 5.2.1. Convolutional network on 2D Ising model

As we can see, the number of spin up particles increases as the temperature rises. We can also see how this value suddenly increases and peaks at about 2.5. This fits well with the actual critical point value of 2.26, since it is the first temperature value over the critical point.

Since we now had a good assurance of the quality of our data, we began training the learner. We found that using a convolutional network with a categorical output to represent the temperature ranges had very poor accuracy. At its best, it reached an accuracy of 22%. This is clearly nowhere near what we want from a good classifier. In contrast, when training a network to classify the number of particles in the spin-up state, we were able to achieve near-perfect accuracy in a comparable training time. This suggests the inaccuracy may be coming from the data. If it's possible for multiple different temperatures to produce the same state (and it is not only possible but highly likely), than the problem is underdetermined and the learner does not have access to enough information to make a consistently accurate prediction.

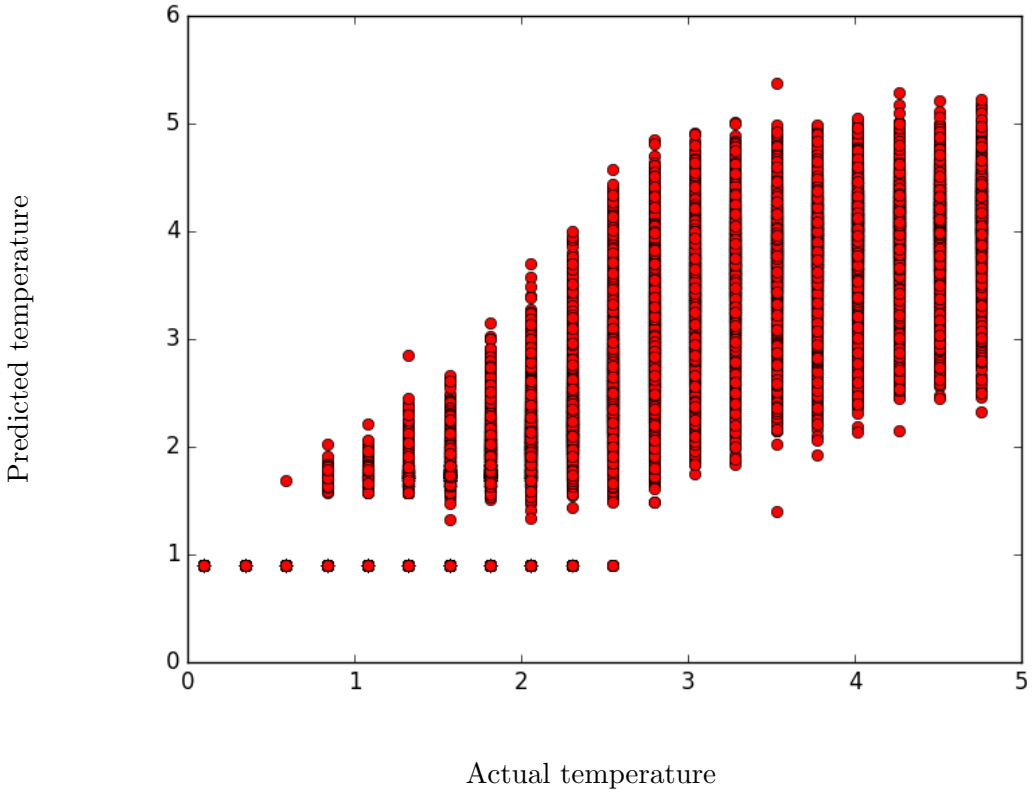

Figure 5.2.2. Predicted temperatures for the convolutional regression model

We also attempted to change the classifier to a more traditional regression classifier. Although there were some accuracy improvements, the fundamental problem remained the same, and it was not possible to get a consistently accurate prediction.

5.3 Methods

We propose an alternative method of finding phase transitions without human intervention. The basic motivation is this: matter within a particular phase has a common set of physical properties. This could make it difficult to guess the temperature simply from the low-level features of the material. However, between different phases, it should be easy to see dramatically different features, even from low-level or local features of the material. Thus, if we adopt the previous paper's use of categorical temperature ranges, we should find that the learner has a hard time discriminating between temperatures within a given phase. However, we would expect a well-

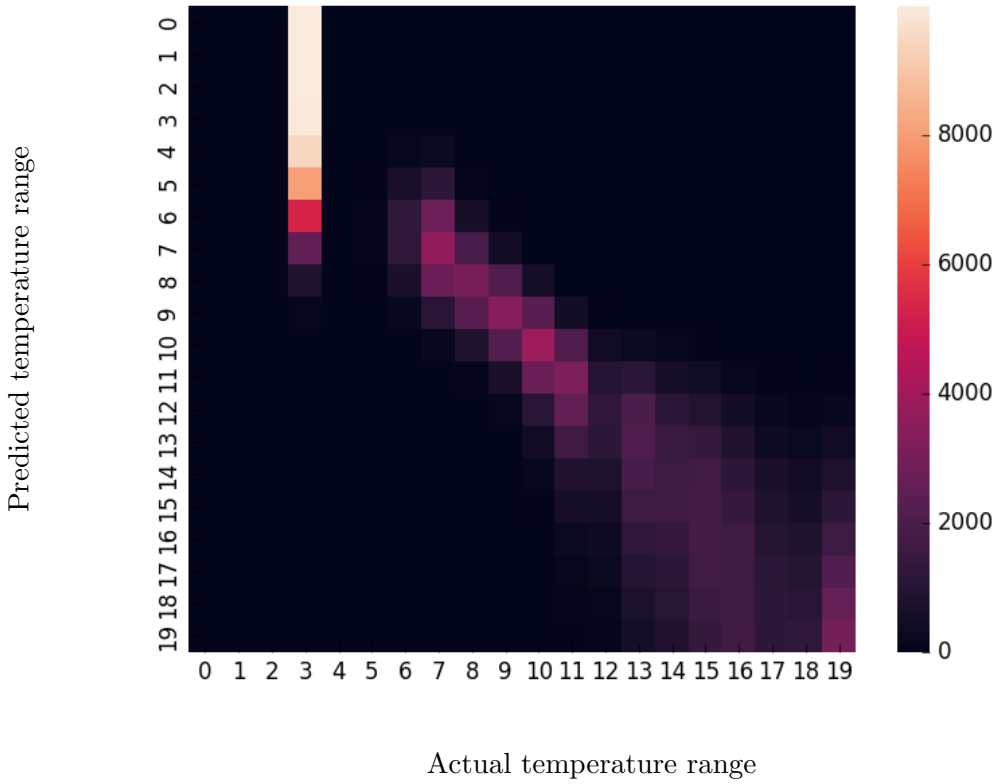

Figure 5.3.1. Confusion matrix for the convolutional regression model (h=0.1)

trained network to be able to quite easily discriminate between matter in different phases. We propose then, using the confusion matrix of a neural network trained to discriminate between temperature ranges.

A confusion matrix is a plot of the predicted class of data against the actual class. Values along the diagonal represent correct predictions, since the predicted and actual classes are the same. Off-diagonal values represent incorrect predictions. We predict that low confusion values between two ranges will suggest presence of a phase transition.

From this image, we can clearly see three distinct regions in the confusion matrix. For low temperature classes (0-5), all the states are indistinguishable from one another, so the network makes the same prediction for all of them. The middle region (6-11) shows a higher accuracy for class prediction, meaning that each of these temperature ranges has fairly distinct physical features from the ranges on either side. This region likely corresponds to the transitional regime

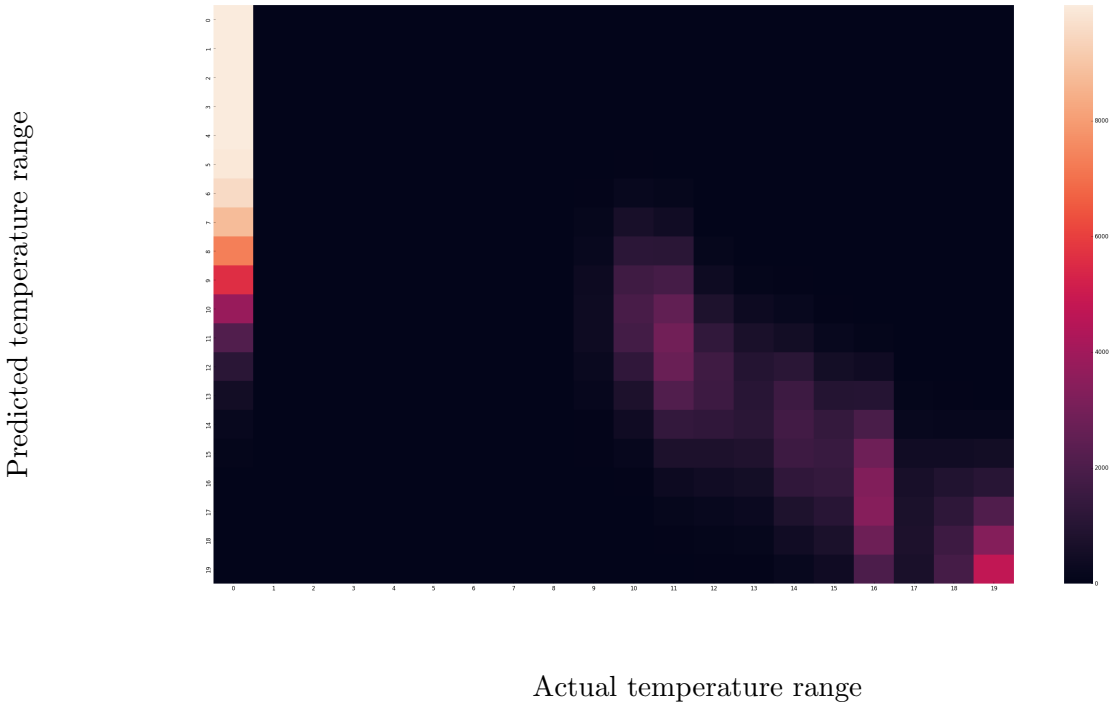

Figure 5.3.2. Confusion matrix for the convolutional regression model (h=2)

between the two more distinct phases, since the physical features are transitioning between those of the phases on either side, which makes it easier for the learner to distinguish between these phases. The high-temperature region(12-19) also has low accuracy, indicating an absence of easily measurable properties to distinguish between these temperatures. However, unlike the low-temperature range, the network predicts a variety of different classes instead of predicting the same class over and over. This tells us that there is significant variation in the states, but that these variations are not strongly correlated with temperature. If they were strongly correlated with temperature, the network could pick up on this correlation to perform accurate classification. But since we know the network is low accuracy in this regime, this indicates the absence of highly predictive physical features.

Increasing the strength of the magnetic field raises the critical temperature, since it takes more energy to break out of the ground state. It is possible to observe this effect by with by comparing the previous confusion matrix to the one in 5.3.2.

The phase transition in this image clearly begins at a much higher temperature, since we only begin to see a steady correlation in the confusion matrix at the 11th temperature range, as opposed to the 5th for the previous setup.

5.4 Discussion

The main success of this method is that it is able to detect phase transitions without prior knowledge of any order parameters of the system. This is its main improvement over the Tomiya paper, which not only made a number of confusing decisions, but also required hand fitting of a regression model using network parameters.

The first strange decision was the choice to use a categorical output instead of a regression. We chose to use a categorical output so we could use an easily readable confusion matrix. The Tomiya paper, however, had no such reasoning, and did not explain their choice. The manner in which the categories are established is even stranger. The categories are constant size increments of inverse temperature, but instead of sampling values from the range of inverse temperatures, values are selected from a discrete set of temperatures, then converted to inverse temperature and re-discretized. This poses two problems. The first is that that some inverse temperature ranges have no temperature values that map to them. For instance, with 10 selectable temperatures with $T_{min} = .5$ and $T_{max} = 4$, there are 4 inverse temperature ranges (indexes 0, 1, 2, and 8) that never see use. 4 different temperatures map to index 3 however. This leads us to a related problem: the amount of data in each category is highly asymmetrical. Because the discretization is linear both in the sampling and the categorization at the end, but non-linear in the middle, different categories receive very different amounts of data. This can be a major problem for a variety of learning algorithms, because it changes the priors for the different classes. If there is twice as many points in category A as category B, a learner is more likely to predict that a unknown datapoint is in category A. This is an issue because if we do not think one class is inherently more likely to appear, it will skew our results.

The paper's results are laid out in a somewhat confusing way. To start with, they do not report the accuracy of their learner. Instead, they argue that the network is able to detect phase transitions because the weight matrix for the last layer has two distinct regions, one corresponding to the region over the critical point, and one for the region below the critical point. This method is unconvincing. For one, red flags are already raised by the paper's nonreporting of accuracy. Secondly, weights for neural networks seldom have simple interpretations, and arguments based on them should be taken with considerable scrutiny unless backed with additional evidence. And finally, without a reported accuracy, it's possible that this weight configuration is simply guessing the same temperature every time, and the transition value simply happens to be at about the half-way point.

Next, to calculate the order parameter, the authors pull values out of the middle of the network and hand-fit them to a tanh function to predict the magnetization. The authors then compare the output of this tanh function to the average magnetization, and demonstrate that the result fits well. This seems to defeat the point of their project. If the intention is to propose a new way of discovering order parameters, then fitting data from the middle of the network to an existing order parameter neither gives us a new way of discovering order parameters for novel systems, nor takes advantage of the training ability of the network to learn the new function. After all, tanh is a supported activation function in Tensorflow (the environment in which the authors were working). Why not just add a tanh layer and train the network to predict the magnetization? Lastly, this approach is somewhat misleading. Attempting to apply this method to a system where the user lacks any knowledge about what order parameter could be causing the system's phase transition could easily result in fitting the wrong parameter, perhaps one that is only correlated with, and using that as the parameter to define the phase.

The primary advantage our method has is that it does not rely on any prior knowledge of the material's order parameters. We could see a phase transition from the confusion matrix without reference to the magnetization at all. This is important because when researchers encounter

new materials, it is not immediately obvious what parameters might be involved with a phase transition. This method helps by circumventing that problem.

One significant downside is that this approach is not as quantitative as other methods. Detection of phases is done by inspection rather than analysis. For this reason, this is likely best used as an aid for quickly discovering and visualizing phase transitions.

5.5 Conclusion

The use of neural networks for representing and analyzing quantum states is a promising new field that has had remarkable success despite its youth. Neural Quantum States offer a powerful alternative to tensor networks and quantum Monte Carlo algorithms that does not suffer from their limitations. In particular, recent work in Neural Quantum States based on deep Boltzmann architectures has offered the promise of more efficient representations and even an exact construction algorithm that runs in linear time [88, 8]. Given the complexity of this problem, obviously this algorithm will not work 100% of the time unless the computational hierarchy collapses. But the exact limits of what types of states can be efficiently approximated using these new methods is largely an open problem.

A number of methods have also sprung up around the use of neural networks for condensed matter problems [1313, 13], [1111, 11]. Models for detecting and quantifying various forms of phase transition could be a powerful new tool for physicists. No matter what form it takes however, it is undoubtable that the rapid advancement of the state of deep learning could have significant consequences for numerical methods in physics.

Bibliography

[1]]cite.szegedy1Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, *Going Deeper with Convolutions*, arXiv preprint **arXiv:1409.4842** (2014).

[2]]cite.biamonte2Jacob Biamonte and Ville Bergholm, *Tensor Networks in a Nutshell*, arXiv preprint **arXiv:1708.00006** (2017).

[3]]cite.carleo13Giuseppe Carleo and Matthias Troyer, *Solving the quantum many-body problem with artificial neural networks*, Science **355** (2017), no. 6325, 602–606.

[4]]cite.montufar2014number4Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio, *On the number of linear regions of deep neural networks*, 2014.

[5]]cite.lucas2014ising5Andrew Lucas, *Ising formulations of many NP problems*, Frontiers in Physics **2** (2014), 5.

[6]]cite.deng2017quantum6Dong-Ling Deng, Xiaopeng Li, and S Das Sarma, *Quantum entanglement in neural network states*, Physical Review X **7** (2017), no. 2, 021021.

[7]]cite.tanaka2017detection7Akinori Tanaka and Akio Tomiya, *Detection of phase transition via convolutional neural networks*, Journal of the Physical Society of Japan **86** (2017), no. 6, 063001.

[8]]cite.carleo2018constructing8Giuseppe Carleo, Yusuke Nomura, and Masatoshi Imada, *Constructing exact representations of quantum many-body systems with deep neural networks*, arXiv preprint arXiv:1802.09558 (2018).

[9]]cite.cai2018approximating9Zi Cai and Jinguo Liu, *Approximating quantum many-body wave functions using artificial neural networks*, Physical Review B **97** (2018), no. 3, 035116.

[10]]cite.gao2017efficient10Xun Gao and Lu-Ming Duan, *Efficient representation of quantum many-body states with deep neural networks*, Nature communications **8** (2017), no. 1, 662.

[11]]cite.van2017learning11Evert PL van Nieuwenburg, Ye-Hua Liu, and Sebastian D Huber, *Learning phase transitions by confusion*, Nature Physics **13** (2017), no. 5, 435.

[12]]cite.schindler2017probing12Frank Schindler, Nicolas Regnault, and Titus Neupert, *Probing many-body localization with neural networks*, Physical Review B **95** (2017), no. 24, 245134.

[13]]cite.yao2017many13Kun Yao, John E Herr, and John Parkhill, *The many-body expansion combined with neural networks*, The Journal of chemical physics **146** (2017), no. 1, 014106.