Using Neural Networks to Predict Properties of the Ising Model

Joseph Becque[a], **Conrad Whiteley**[a], **and Massine Yahia**[a]

[a]Department of Physics, McGill University, 3600 Rue Université, Montréal, QC, H3A 2T8, Canada

A paper discussing the use of the Ising model to explore the potential of neural networks in physical research, and to investigate which properties and parameters of the network are most important to achieve greater accuracy in this context. Specifically, datasets containing in the order of 10^6 2D Ising model spin lattices, including their respective analytically calculated energies, were generated using Metropolis Monte Carlo methods. This data was used to train a Convolutional Neural Network. The data set size, the number of hidden layers used, the dimensions of convolutional filters as well as the number of training epochs used were varied to gain an understanding of their impact on accuracy of the model. It was found that a 3 layer convolutional neural network trained on 500,000 training examples was able to achieve an error of less than 1% on predicting a lattice's energy. The parameters that caused the biggest increase in prediction accuracy were the number of training epochs and the number of convolutional layers, which were able to reduce prediction error by a factor of 10. Dataset size and filter dimensions had negligible impacts on accuracy.

Ising model | Machine learning | Neural Networks

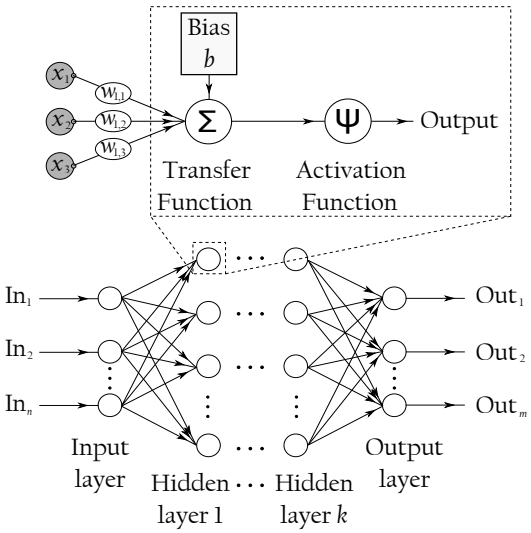

Fig. 1. A diagram of a neural network composed of an input layer, hidden layer(s) and an output layer, and a zoomed in image of the structure of a neuron above.

An artificial neural network is a powerful computational method by which a model is trained to recognize patterns and properties of a given input in order to predict desired output parameters to high accuracies. They were first used in the 1950's to model neurons in the brain, where models such as the perceptron aimed to understand the intelligent organisms' abilities of perception, pattern recognition, memory, and thinking [1]. Since then, computer science and theories of biological learning have developed enormously, and in the 1980s, networks with a hidden layer were developed which used back-propagation methods to adjust the weights of connections between neurons in order to maximize the accuracy achieved [2]. Since then, deep neural networking using multiple hidden layers has developed [3], and now the field of research in the area of machine learning is expanding rapidly. The reason it took so long to become such a dominant area of research since conception is because only recently both the large volumes of training data needed to yield accurate results, and the computer processing power have become accessible [4].

Neural networks have become a powerful tool in the technology, commercial and business sectors, used in such things as web search predictions, pattern and image recognition, text and audio analysis and translation, prediction of equipment failure, and medical diagnostics [5]. However, uses in physics research have been limited; this paper explores whether machine learning presents a suitable method by which to analyze data in a physical context.

1. Machine Learning

Neural Networks.

Artificial Neural Networks are a computational approach to information processing and have the ability to extract patterns and trends that are either too minute or complicated for humans to recognize or too generalized for other computational methods to calculate. Conventional computational methods follow a set of pre-defined instructions, and therefore the user must require the knowledge of exactly how to solve the problem at hand in order to define these instructions. Neural networks on the other hand, are much more adaptive as they have the ability to learn; to change their internal structure based on the information flowing through it.

Fig. 1 shows a diagram of a neural network. It is composed of layers (an input layer, hidden layers and an output layer), each of which is formed by the interconnection of artificial neurons (commonly referred to as 'nodes'). A deep neural

Significance Statement

This paper reports the use of a branch of Machine Learning called Deep Neural Networks to teach a computer to recognize input parameters so that it can predict outputs with a high accuracy.

In this case, the input is a lattice of electron's spin states and the output is the total energy of the lattice due to spin interactions. Data is generated analytically using the Ising model to calculate energies and magnetizations of many lattices, and the computer is then trained to estimate a lattice's energy. With sufficient training data, the algorithm should be able to give good predictions.

network is defined as consisting of multiple hidden layers of neurons. Each neuron in a layer is connected to all neurons of the previous and subsequent layers via a process of weighted connections. The top of Fig. 1 zooms in on a single neuron showing that every connection has a weight and every neuron has a bias. If the sum of the weights and bias is greater than some pre-defined threshold potential then an activation function, Ψ, is 'fired' as the output. This output is sent to all neurons in the subsequent layer and the process is repeated. The transfer function, Σ, is the product of a vector containing the weights, W_i, and the input values, x_i, summed with the bias, b_i. Therefore, y_i, the output of neuron i, with 3 inputs is

$$y_i = \Psi(\Sigma) = \Psi\left([w_{i,1}, w_{i,2}, w_{i,3}] \cdot [x_1, x_2, x_3] + b_i\right) \ . \quad [1]$$

The neurons of a neural network can use a variety of activation functions. For deep neural networks, the rectified linear unit (ReLu) is the most commonly used [6]. It is defined by the function

$$\Psi(\Sigma) = max(0, \Sigma) \ . \quad [2]$$

Neural Network Training.
A neural network's training objective is to minimize a loss function. A loss function is used to describe how well a classification or regression learner performs. For regression, the most used loss functions are the squared error:

$$E = \frac{1}{2}(y - y')^2 \ , \quad [3]$$

and the absolute error

$$E = |y - y'| \ , \quad [4]$$

where y is the learner's prediction and y' is the real value.

To minimize this loss function, the most used method is the backpropagation algorithm [2]. The algorithm contains two phases. The first one is the propagation step: an input vector is given to the network in order to generate an output value. The output value produced and the real output of the input vector are then used to calculate the error on this particular example using a loss function such as Eq. (3) and Eq. (4). The second step then consists of updating the weights of each neuron by a factor proportional to a learning rate and the error given by the loss function. The backpropagation algorithm computes the adjustment applied to each weight W_{ij} using the following gradient descent equation:

$$w_{ij}(t+1) = w_{ij}(t) - \alpha \frac{\partial E}{\partial w_{ij}} \ , \quad [5]$$

where E is the error function, α the learning rate, $w(t+1)$ the updated weight and $w(t)$ the current weight. Various improvements have been added to the gradient descent equation, such as a decaying learning rate, the addition of a momentum term or using a different learning rate for each weight. Two popular optimization algorithms that improve the gradient descent algorithm are ADAM[7] and RMSProp[8], an optimizer that combines a momentum term, rate decay and adapts the learning rate by keeping track of the moving average of past gradients.

Crossvalidation.
In order to estimate a predictive model's performance on unseen data, a technique called crossvalidation is used. The available data is partitioned into two: the training set and the testing set. The proportion of examples placed in the testing set usually varies between 50% and 10%. A training pass over all the training set is called an epoch. The learner is trained for an epoch and then the accuracy on the testing set is calculated using the learned weights. This accuracy gives an estimate of the model's performance. Training is then usually repeated for multiple epochs until the testing accuracy stops improving.

Overfitting.
Like all statistical models, neural networks are prone to overfitting. If a learner has too many parameters relative to the number of training examples, it will tend to "overfit" the training data and have poor performance on new data. For neural networks, the main methods to avoid overfitting are early stopping, regularization and dropout.

Early stopping is the simplest method. It consists in training the neural network continously until the error on a testing set stops decreasing.

Regularization is another method that is efficient for all learners. It consists of adding a penalty that is proportional to the squared or absolute magnitude of the weights. In the case of L2 regularization (also called Tikhonov or ridge regularization), the error function then becomes

$$E_{L2} = E + \lambda ||w||^2 \ , \quad [6]$$

where E is our previously defined error from Eq. (3) and Eq. (4), w the weights of the network and λ is a constant defining the strength of the regularization.

Dropout is a method specific to neural networks. It consists of forcing some of the weights in the network to be 0 during training [9]. This forces the entire network to contribute to the final output instead of relying on a few nodes and is similar to taking the average of multiple networks. The typical implementation is to set the weights in the fully-connected layers to 0 with a probability of 1/2 during training. During prediction or testing, all the weights are used.

Convolutional Neural Networks.
Convolutional neural networks (CNN) are a type of neural networks inspired by the animal visual cortex. They are the most popular tool for visual classification and natural language processing tasks. They were used by Google's AlphaGO, the first AI to beat a human player at Go [10].

Similarly to the cells used to process visual information, the nodes in a CNN are only connected to a small subsection of the input layer, called a receptive field. This allows the network to easily model local phenomenas, such as edges, that can later be combined into representations of increasing complexity in the successive layers.

In terms of matrix operations, it is equivalent to applying a two-dimensional convolution to the input layer using a two-dimensional filter. A filter is applied to every pixel in the input layer to give a feature map which is then used as the input for the next layer. The sharing of weights allows the network to be translationally invariant, as well as reducing the number of weights to optimize.

In order to limit overfitting and reduce computation time, a subsampling operation is often applied after a convolution layer. Max-pooling is often the favored pooling operation. It consists of slicing a matrix into smaller submatrices, taking the highest element of each submatrix and recombining them into a new output matrix. Average-pooling is also sometimes used. Fig. 2 shows an example of a typical CNN, usually consists of several convolutional and pooling layers whose final output is then flattened and fed into a small fully-connected neural network.

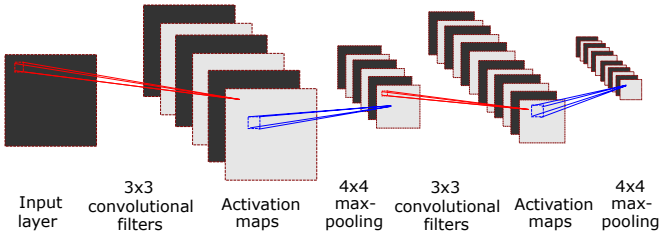

Fig. 2. Diagram of part of a convolutional neural network containing two convolutional and two max-pooling layers. Red: convolutional operations, blue: max-pooling operations.

2. Ising Model

The two dimensional (2D) Ising model is one of few physical statistical models with a solvable analytic solution. This presents an opportunity to solve a physical problem using Machine Learning techniques. It was first invented by Wilhelm Lenz in 1920 and presented to his student Ernst Ising after whom it is named [11]. The model is used to understand ferromagnetic materials by representing them as a lattice of electrons, each with spin-1/2 and therefore a magnetic dipole moment [12]. The electron spins can either align or anti-align such that the individual magnetic fields sum together or subtract to form a macroscopic field in the bulk material [13]. The purpose of the Ising model is to use this set up to approximate bulk properties of the material such as the energy stored in the total field and the magnetization of the material. Other properties such as the specific heat capacity and the critical temperature can also be found.

In the 2D problem, the electrons in the bulk material can be assumed to form a lattice where each electron occupies lattice point, a, with coordinates (i, j), and can either have spin-up, $s_a = 1$, or spin-down, $s_a = -1$ [14]. Each electron in the lattice is then approximated to only interact with it's directly neighboring particles.

As such, when considering a lattice consisting of a single pair of neighboring electrons labeled e_1 and e_2, there are two possible macrostates, each with two corresponding microstates that the system can occupy. For example, one could have the macrostate $s_1 = s_2$, with either microstate {1,1}, or {-1,-1}. Alternatively, one could have the macrostate $s_1 \neq s_2$ with corresponding microstates {1,-1} and {-1,1}.

In a 2D square lattice, like the one modeled in this paper, one particular electron contributes to eight interactions, as shown in Fig. 3. When the lattice becomes large, it must be described statistically because of the large number of possible electron arrangements; as the side length, N, of the lattice increases, the total number of microstates goes with 2^{N^2}. Each possible configuration of Ising space (each permutation of electron spin states) constitutes a microstate, and global

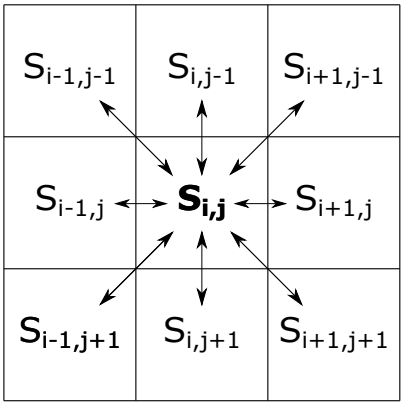

Fig. 3. Diagram of spin interactions. An electron in a 2D lattice has been approximated to interact with all its directly neighboring electrons. Diagonal interactions contribute half as much as horizontal and vertical interactions. The spin, s, of an electron at a position (i, j) in the lattice can be spin up ($s_{i,j} = 1$) or spin down ($s_{i,j} = -1$).

properties of the system such as energy and magnetization constitute macrostates.

The coupling constant, J, is a property of the lattice with units of energy that describes how strongly electrons interact with each other and the manner in which they do. For example, if $J < 0$ the material is anti-ferromagnetic and oppositely aligned spins contribute positively to the energy, whereas if $J > 0$ the material is ferromagnetic and like spins contribute positively to the energy.

Thermodynamic Properties.

The energy of the lattice is found by using the approximation of only neighborly interactions, and by taking the product of neighboring spins and summing over all the lattice points. From this approximation, the Hamiltonian, H, of the lattice is given by

$$H = J \sum_{(a,b)} s_a s_b + \frac{J}{2} \sum_{(a',b')} s_{a'} s_{b'} + h \sum_a s_a \ , \qquad [7]$$

where summing over (a, b) means to sum over all horizontally and vertically neighboring positions in the lattice, and summing over (a', b') means to sum over all diagonally neighboring positions [15].

The first term is the energy contribution due to horizontal and vertical interactions. The second term is the diagonal interactions' contribution to the energy which have been approximated to contribute half as much as the horizontal and vertical interactions because they are further away. The third term is the energy contribution due to an applied magnetic field with magnitude h.

The net magnetization of the system, M, can be calculated from the sum of the electron spins in the lattice,

$$M = \mu \sum_a^N s_a = N\mu < s > \ , \qquad [8]$$

where μ is the magnetic dipole moment, and $< s >$ is the expected spin of particles in the system [16]. From Eq. (8), the expected magnetization per site, $< M >$, can be calculated from

$$< M >= \frac{M}{N} = \mu < s > \ . \qquad [9]$$

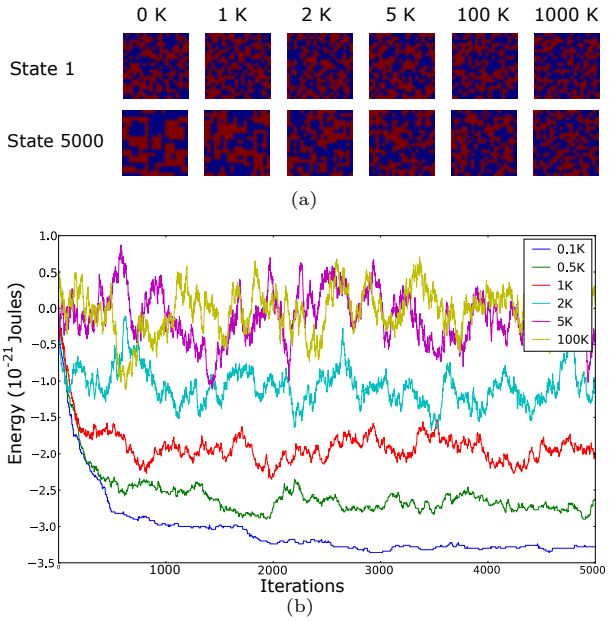

Fig. 4. Time evolution properties of a randomly generated 2D square lattice of electron spins with 20 lattice points along each side, at $J = 10^{-23}$ J, and over 5000 Metropolis Monte Carlo iterations. **(a)** shows the spin states at each lattice point before and after the iterations at different temperatures. **(b)** shows how the energy evolves at different temperatures with time.

Time evolution and temperature dependence.

A critical property of the Ising model is the balance between the thermal energy and the interaction energy, quantified by k_BT and J respectively. This is visualized in Fig. 4 which shows how a lattice evolves at different temperatures.

At high temperatures, k_BT is greater than J, so the electron behavior is dominated by the thermal energy which has the effect of randomizing all the spins in the lattice. As a result, there is no bulk magnetization and the energy fluctuates noisily and does not tend towards an equilibrium.

At lower temperatures, J becomes increasingly significant and clusters of spins with lower energies form. This is because the system is able to descend to lower energy states in which like spins group together. However, thermal energy still causes a few quantum fluctuations leading to random spin flips of electrons which stop the system reaching a ground state where all electrons have the same spin.

At a critical temperature, T_C, there are an equal number of spin up and spin down particles and below this one of either spin up or down becomes more dominant. This breaking of symmetry in the system is characteristic of a phase change which occurs at a the critical temperature.

3. Method

Monte Carlo Data Generation.

In order to train a neural network to identify Ising model properties of a 2D lattice of spins, a large sample of lattice arrangements and their corresponding energies and magnetizations must be generated. This data can then be used for training and testing the network. The number of possible microstates for even a small lattice is huge, so it is important that an accurate way of sampling the microstates of the lattice is established. This paper discusses the use of Monte Carlo (MC) sampling methods which involve observing a fraction of the possible configurations from which the network can be taught. This is an efficient and accurate way of sampling large amounts of data that are representative of the large number of possible microstates available in the Ising model [17].

An obvious MC method to generate data is to randomly generate a lattice and measure it's energy using Eq. (7). This could be repeated in the order of 10^6 times to acquire the desired amount of data. However, this would generate a poor distribution of microstates because high temperature disordered states are much more likely to be generated with this method. This is because there are a larger number of microstates for higher energy, disordered macrostates. As a result, this would lead to a very limited sampling of any low energy states.

Instead, Metropolis MC sampling is used. This involves altering the probability of transitioning from microstate i to microstate f, by considering the ratio of the probabilities of being in their corresponding macrostates. This ratio, $\mathcal{R}$, can be expressed as

$$\mathcal{R} = \frac{N_i}{N_f} = e^{\beta(E_f - E_i)} \ , \qquad [10]$$

where N_i and N_f are the probabilities of being in macrostates with energies E_i and E_f respectively, and β has units of inverse energy and is equal to $1/k_BT$ [18].

To implement this Metropolis MC sampling method a small alteration is made to the current system in order to generate a new system. In the case of the 2D Ising model, the alteration is a random spin flip of a single electron. Metropolis MC allows this spin flip if the energy decreases. However, if the flip causes the energy to increase, it is only accepted with the probability $P(i \to f) = \mathcal{R}$ from Eq. (10) [19]. This is repeated in the order of 10^6 times to collect a sufficiently large sample of data.

A square lattice with side length of 100 was found to be a suitable size for the purpose of Machine Learning. Without being so large that it made energy calculations too slow, it was sufficiently big enough to give a distribution of possible microstates ($\approx 10^{3000}$) which ensures a Boltzmann distribution and that Eq. (10) is accurate. Periodic boundary conditions were implemented such that the contribution to the total energy from the edge points of the lattice was equal to everywhere else.

In a single Metropolis MC run (instance), a random 100×100 lattice of electron spins is generated by creating a two dimensional array filled randomly with values of either 1 or -1. The Hamiltonian is then calculated using equation Eq. (7). Following this, state changes (iterations) are implemented in accordance with the Metropolis MC method described above until the system reaches equilibrium; for a lattice of size 100, each instance discarded the first 10^5 lattices generated to ensure the system reached equilibrium. After this, further state changes and energy calculations are remade for a number of iterations, N_{It}.

In order to take account for any biases introduced by a particular random starting lattice, this was repeated for five instances. Furthermore, the data was collected close to the critical temperature over a range of $T = 0$ to 5 J/k_B K in 100 intervals of 0.05 J/k_B K (note, to remove small numbers from computations, k_B, μ and J were set equal to one). Because a single spin flip in a size 100 lattice does not change the properties that much, only every tenth iteration was saved to

be used for training. Altogether, this means that in a complete data generation process, a total of

$$N_{It} \times 5 \times 100 \times \frac{1}{10} = 50 N_{It} \qquad [11]$$

lattices, with their respective energies, were generated.

Convolutional Neural Network Implementation.

Before being used in the network, all our input vectors which consisted of 100x100 lattices of -1 and 1 spins were first multiplied by 0.5 and added to 0.5 in order to change their range to $[0, 1]$. The energies of our training examples were standardized. First, the mean and standard deviations of a randomly chosen fraction of our training dataset were calculated. Next, the approximate mean was subtracted from each training example, and then each of these values were divided by the calculated standard deviation.

The CNN used for this project consisted of 2 to 5 convolutional "blocks". Each block contained first a convolutional layer then a 2x2 max-pooling operation, and finally a ReLu activation function. The first hidden layer contained 16 filters, the second once 32, the third one 64 and the fourth one 256. The filters used had a size of 3x3. The output of the last convolutional block was then flattened into a vector. This vector had a dropout operation applied to it, before being used as the input for a fully-connected hidden layer of 5 ReLu neurons. Finally, those 5 neurons were connected to a single output neuron. The error was calculated using the following formula:

$$\mathbf{Err} = \frac{|y - y'|}{y'} \ , \qquad [12]$$

where y is the predicted value and y' the real value.

All the weights in the network were chosen randomly from a normal distribution centered at zero and with a standard deviation of 0.1 for filters and 0.5 for neuron connection weights. Biases were initialized to 0. The gradient descent algorithm used was RMSProp with a momentum factor of 0.1 and a decay rate of 0.5.

For crossvalidation, the generated examples were split in equal proportions into the training and testing sets. A smaller subset of 200 examples from the testing set was used to estimate the testing error after each epoch, and the entirety of the testing set was used to calculate the validation error.

Figure 5 shows an example trained network predicting the energy of lattices from its training set. It is quite clear the the predictions tend to follow the real values.

4. Results and Discussion

Using the Metropolis MC sampling method described in section 3, datasets with $N_{It} = 20000$, 50000 and 100000 different lattices were generated (henceforth referred to as the 20k, 50k and 100k datasets). Using Eq. (11), this means 10^6, 2.5×10^6, and 5×10^6 lattices were generated for each dataset respectively. These included the calculated energy for each lattice across 100 different temperatures distributed evenly between 0 and 5 J/k_B K.

These were trained with different parameters using the CNN described in section 3 in order to investigate to what extent their variation effected the accuracy with which the program could predict the energy of any given lattice.

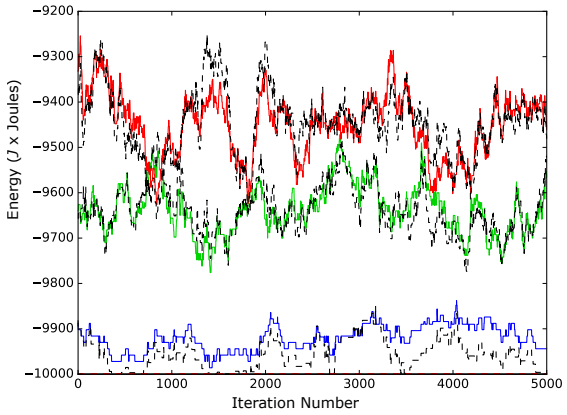

Fig. 5. Example of a trained neural network predicting the energy of a lattice. The model was trained on the 100k dataset for 5 epochs. It contained 2 layers of 5x5 filters. Shown in red, green and blue solid lines are the calculated energies of lattices evolving at 3 different temperatures. The values predicted by the model are shown in dashed black lines.

N_{It}	Epoch	Training Error	Testing Error	Validation Error
20,000	1	0.0902	0.1047	
	2	0.0763	0.0932	0.0508
	3	0.0629	0.0703	
50,000	1	0.1111	0.0618	
	2	0.0879	0.0550	0.0694
	3	0.0856	0.0502	
100,000	1	0.0812	0.0806	
	2	0.0771	0.0843	0.0748
	3	0.0770	0.0858	

Table 1. Training Neural Networks with varying data sizes. Three sets of data with $N_{It} = 20000$, 50000 and 100000 were trained with a Neural Network consisting of 2 hidden layers for a total of 3 epochs.

Accuracy as a function of number of training examples.

It was expected that increasing the number of training examples would increase the accuracy that the model would be able to achieve. In order to establish to what extent the number of training examples used to train the model affected the error, the number of iterations was varied. Table 1 shows the training error, testing error, and validation error for training with the 20k, 50k and 100k datasets calculated using Eq. (12). Each model was trained using 2 hidden layers and for 3 epochs.

The results shown disagree with the expectation that error decreases as dataset size increases, as they find the validation error of each dataset increases with with larger numbers of iterations. To directly compare the accuracy achieved due to the number of training examples, consider the standardized ratio of validation error to N_{It},

$$\frac{\text{Validation Error}}{N_{It}} \cdot \frac{0.5083}{20000}$$

where the factor of 0.5083/20000 standardizes the values to the validation error achieved by 2 layers. This gives 1, 1.831 and 3.398 for the 20k, 50k and 100k datasets respectively, which demonstrates how substantially the validation error increases with N_{It} in this particular case.

Further datasets were trained with a greater number of epochs in order to investigate the behavior of the Neural

N_{It}	Epoch	Training Error	Testing Error	Validation Error
20,000	1	0.1126	0.0526	
	2	0.0656	0.0028	
	3	0.0281	0.0026	
	4	0.0192	0.0031	0.0122
	5	0.0152	0.0028	
	6	0.0128	0.0026	
	7	0.0113	0.0027	
50,000	1	0.0943	0.0637	
	2	0.0628	0.0270	
	3	0.0234	0.0111	
	4	0.0112	0.0085	0.0060
	5	0.0082	0.0077	
	6	0.0067	0.0057	
	7	0.0059	0.0066	
100,000	1	0.0719	0.1345	
	2	0.0476	0.0468	
	3	0.0221	0.0259	
	4	0.0133	0.0199	0.0061
	5	0.0103	0.0175	
	6	0.0087	0.0154	
	7	0.0077	0.0139	

Table 2. The training errors, testing errors, and validation error for N_{It} = 20000, 50000 and 100000 using a Neural Network containing 2 hidden layers which has been trained over 7 epochs.

Network over a longer training time; the 20k, 50k and 100k datasets were trained with 7 epochs for which the data collected from this is shown in Table 2.

Comparing the validation errors achieved shows that after training for seven epochs, the models trained on the 50k and 100k datasets outperformed the model trained on the 20k dataset. We are confident that given enough training time, the accuracy should increase with more data and this is what the accuracies shown in Table 2 seem to indicate. However the accuracy seems to hit an upper bound after a certain amount of training data. The models would have to be run for an even larger number of epochs in order to conclude definitely that this is the case.

The data in 1 also shows that over the three epochs, the testing error on the 20k dataset decreased by 32.9% of its first epoch's accuracy, while the 50k dataset's testing error decreased by 18.8%, and the 100k dataset's testing error increased by 6.5%. This implies that a smaller amount of data requires more training in order to reach its optimum accuracy. Moreover, the increase in the testing error for the 100k dataset indicates that for a data size that large, one epoch is sufficient for the model to identify network parameters that come very close to optimizing the accuracy. It is possible that the model gets stuck in a bad local minimum of the loss function or that the learning rate is too high, but it cannot be determined without training the model on this dataset a second time for more epochs.

Note also that the first three epochs in Table 2 and Table 3 are repeats with the same parameters and yet the training and testing errors for each are vastly different. For example, after 1 epoch the 20k dataset varies by a factor of 2, and after 3 epochs there is a full order of magnitude difference between

Number of layers	Epoch	Training Error	Testing Error	Validation Error
2	1	0.1111	0.0618	
	2	0.0879	0.0550	0.0694
	3	0.0856	0.0502	
3	1	0.0804	0.1326	
	2	0.0418	0.0241	0.0128
	3	0.0140	0.0207	
4	1	0.1003	0.0432	
	2	0.0674	0.0406	0.0869
	3	0.0654	0.0393	
5	1	0.1255	0.0609	
	2	0.0803	0.0544	0.0767
	3	0.0777	0.0498	

Table 3. Training Neural Networks with different numbers of hidden layers. The training error, testing error and validation error are shown for data generated from 50000 Ising model iterations which has been trained for 3 epochs.

the data in each table. This shows that there is significant variation for each measurement using the same parameters which is greater than the variation in accuracy observed when varying the data size. Repeat runs over a large number of epochs with a significantly different number of iterations are needed in order to gain a full understanding of the dependence on the number of training examples. This wasn't possible within this projects time constraints of the project.

Accuracy as a function of number of hidden layers.
In order to investigate the impact of the number of hidden layers used in a Neural Network, the 50k dataset was trained with a range of different numbers of layers. Increasing the number of layers is expected to increase the accuracy of the network because there are more weights and biases used. More parameters means that the network can store more information about the model to understand more complex patterns.

The errors obtained when varying the number of layers from 2 to 5 are shown in Table 3. To compare the validation errors as a function of number of layers,

$$\frac{\text{Validation Error}}{\text{Number of Layers}} \cdot \frac{0.0694}{2}$$

is used, where the coefficient 0.0694/2 standardizes the data to the accuracy achieved by 2 layers. For 2, 3, 4 and 5 layers respectively these values are 1, 0.123, 0.626, and 0.442. This shows that for an Ising model dataset generated with the 50k dataset and trained over 3 epochs, there is no relationship between the validation error and the number of layers.

The reason for this could be because the parameters haven't been optimized after 3 epochs; for all 4 cases tested, the testing error was still decreasing with each epoch when the training stopped. This is likely because using more layers takes longer to train, since to optimize more parameters the learning rate must decrease.

To study this hypothesis, networks were trained with the 20k and 50k datasets using three layers for 7 epochs, as shown in Table 4. Due to time constraints, it was not possible to test greater numbers of layers or the 100k data with more epochs. Note that Table 4 (using 3 layers) is directly comparable to Table 2 (using 2 layers). Comparing the validation errors

N_{It}	Epoch	Training Error	Testing Error	Validation Error
20,000	1	0.0778	0.0585	
	2	0.0417	0.0372	
	3	0.0282	0.0279	
	4	0.0199	0.0174	0.0054
	5	0.0133	0.0096	
	6	0.0095	0.0057	
	7	0.0079	0.0040	
50,000	1	0.0511	0.0271	
	2	0.0118	0.0177	
	3	0.0088	0.0204	
	4	0.0085	0.0277	0.0071
	5	0.0093	0.0253	
	6	0.0077	0.0204	
	7	0.0063	0.0119	

Table 4. Datasets generated with 20000, 50000 and 100000 iterations, and trained using a neural network with 3 hidden layers over 7 epochs.

Filter dimensions	Epoch	Training Error	Testing Error	Validation error
3×3	1	0.0754	0.0274	
	2	0.0249	0.0119	
	3	0.0160	0.0077	0.0110
	4	0.0146	0.0078	
	5	0.0135	0.0068	
5×5	1	0.0887	0.0578	
	2	0.0461	0.0254	
	3	0.0230	0.0093	0.0055
	4	0.0134	0.0064	
	5	0.0110	0.0065	
7×7	1	0.1399	0.0934	
	2	0.0735	0.0893	
	3	0.0469	0.0290	0.0147
	4	0.0220	0.0150	
	5	0.0128	0.0218	

Table 5. Training Neural Networks with different filter sizes. A dataset generated using 100000 iterations was trained with 2 hidden layers for 5 epochs with various filter dimensions.

achieved using 2 layers and 3 layers shows that adding an extra layer decreases the validation error by a factor of 10, such that the error was less than 1%. This is a significant improvement and demonstrates that the Ising model is highly sensitive to the number of hidden layers.

Accuracy as a function of filter size.
The size of the filters in the convolutional layers was increased from 3x3 to 5x5 pixels, and then again to 7x7 pixels in order to see if this would increase the prediction accuracy of our model. The dataset used was the one containing 100k iterations and the neural network contained 2 convolutional layers. The models were all trained for 5 epochs. Note that a 3x3 convolutional layer contains 9 parameters per filter, a 5x5 layer contains 25, and a 7x7 layer contains 49 parameters. As shown in Table 5, all three layer sizes errors are around 1%, with no clear correlation between filter size and accuracy. This could simply be a characteristic of the data that is being analyzed. Since the energy at a site only depends on the relationships between a spin and its immediate neighbouts, a 3x3 filter should be enough to estimate energy. Notice that for the 5x5 and 7x7 filters, the testing error in the last epoch increases. This might be due to overfitting and too many parameters for our number of training examples. It is recommendable to avoid bigger filter sizes or to try and combine dropout with another regularization method like L2 when using larger filters since the number of parameters increases drastically.

5. Conclusion

It has been shown that with the right parameters (3 hidden layers and a 3×3 filter), using 0.5 million training examples generated by the 20k dataset was sufficient to predict properties of the Ising model with an error of less than 1%.

Further analysis found that an increase to 2.5 million examples was not enough to observe a significant decrease in error. It is expected that to be able to obtain more accurate predictions, the dataset size must be increased much further beyond this, and most importantly be run for a greater number of epochs than time constraints allowed in this project.

It has also been shown that the error obtained by the neural network is highly sensitive to the number of hidden layers used, such that increasing from 2 to 3 layers reduces the error by a factor of 10. Conversely, changing the dimensions of filters was found to have no impact on the accuracy achieved. In order to further investigate these parameters, repeat runs over longer training periods would provide more insight into these relationships.

In summary, our results have demonstrated that physics presents a realistic domain where machine learning could be a useful tool.

6. Acknowledgments

We would like to thank Dr. Hong Guo for his supervision, guidance and discussions concerning the theory, computation and analysis which made this work possible. We would also like to thank Dr. Sabrina Leslie for her oversight of this project.

7. Appendix and Commentary

All programming was done in Python (version 3.5.2), with SciPy (version 0.18), Numpy (version 1.12) and TensorFlow (version 12.1) libraries, and ran on an Intel Core i7 processor. All the code used can be found at https://github.com/bbbxyz/phys449

8. References

1. Rosenblatt F (1958) The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* pp. 65–386.
2. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. *Nature* 323:533 – 536.
3. Panchal, Gaurang; Ganatra AKYPPD (2011) Behaviour analysis of multilayer perceptrons with multiple hidden neurons and hidden layers. *Journal of Computer Theory and Engineering* 3:332.
4. Goodfellow I, Bengio Y, Courville A (2016) *Deep Learning.* (MIT Press). http://www.deeplearningbook.org.
5. Geoffrey Hinton, Li Deng DY (2012) Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29:82–97.
6. Lecun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521:436–444.
7. Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

8. Tieleman T, Hinton G (2012) Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4(2).
9. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15:1929–1958.
10. David Silver DH (2016) Alphago: Mastering the ancient game of go with machine learning. *Google Research Blog*.
11. BRUSH SG (1967) History of the lenz-ising model. *Rev. Mod. Phys.* 39:883–893.
12. Griffiths D (2013) *Introduction to Electrodynamics*, Always learning. (Pearson), 4 edition, p. 378.
13. Chikazumi S (2009) *Physics of Ferromagnetism*, International Series of Monographs on Physics. (OUP Oxford), p. 129.
14. Newman M, Barkema G (1999) *Monte Carlo Methods in Statistical Physics*. (Clarendon Press).
15. Lahiri A (2002) *Statistical Mechanics: An Elementary Outline*. (Universities Press), p. 241.
16. Niss M (2005) History of the lenz-ising model 1920–1950: From ferromagnetic to cooperative phenomena. *Archive for History of Exact Sciences* 59(3):267–318.
17. Portman N, Tamblyn I (2016) Sampling algorithms for validation of supervised learning models for Ising-like systems. *ArXiv e-prints*.
18. Schroeder D (2013) *An Introduction to Thermal Physics*. (Pearson Education, Limited), p. 229.
19. Suzuki S, Inoue J, Chakrabarti B (2012) *Quantum Ising Phases and Transitions in Transverse Ising Models*, Lecture Notes in Physics. (Springer Berlin Heidelberg), p. 49.