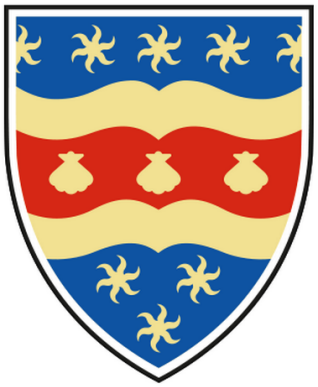

Phase Transitions in the Ising Model: from Monte Carlo Simulations to Automised Classification using Machine Learning

Bachelor Degree Project

Author: Ruby Pearce-Casey
Supervisor: Dr. Vincent Drach

Centre for Mathematical Sciences
Plymouth University
May 4, 2022

Abstract

The Ising Model is a particular example of a thermodynamic system, and it is the model we use to study phase transitions. It is one of few exactly solvable models where we can compute its thermodynamic quantities and interpret them. We introduce the fundamental concepts of statistical mechanics in section 2, and in section 3, we introduce the Ising Model. We will solve the model in $1D$ and present Onsager's[1] solution to the model in $2D$. In section 4, we will introduce a modified Monte Carlo[12] method, namely the Metropolis[2] algorithm, to simulate the Ising Model in $2D$. Using this simulation, we can numerically simulate the magnetisation and the specific heat. We show that the system undergoes a drastic change of macroscopic quantities at a particular temperature, called a phase transition. Not only will we introduce the Ising Model and methods we can use to simulate the model to identify a phase transition, but we will introduce machine learning[6] in section 5. We will see that a Neural Network, which is a machine learning architecture, can identify phase transitions in a variety of spin systems and we will present the result in section 7. The goal of this project is to replicate the results of an exisiting paper published in Nature, *Machine Learning Phases of Matter* [5].

[1]Lars Onsager 1903-1976. Physical chemist and theoretical physicist
[2]Nicolas Metropolis 1915-1999. Physicist

Contents

1 Introduction

The Ising model is a mathematical model of ferromagnetism in statistical mechanics and was first introduced by Wilhelm Lenz[3] in 1920. The model in its entirety arises from expressing such a system's Hamiltonian in two parts: one as the energy contribution from particle interactions and the second energy contribution from constraints on the system, such as an external electric field. The field of statistical mechanics is a mathematical framework which allows us to summarise microscopic dynamics of a system to the description of its macroscopic properties[9]. Furthermore, statistical mechanics offers us a formalism in which we can derive the macroscopic properties of the Ising model, such as its magnetisation, through the model's Partition Function. This is possible using the Boltzmann distribution which is fundamental in statistical mechanics.

Lenz gave the proposed model to Ernst Ising[4] in 1922. It wasn't until 1925 that Ising published a summarisation of the calculation of the free energy for the one-dimensional model with an external electric field which demonstrated that the model in one dimension did not exhibit a phase transition for any temperature. Regarded as an oversimplification, it was proposed that the model had no practical applications, leading to Ising claiming that the model did not exhibit a phase transition for higher dimensions. Unfortunately, Ising abandoned the model and the field of physics altgother. As the knowledge of ferromagnetism developed, Peierls[5] contradicted Ising's claim in 1936[20] and stated that the Ising model in two- and three-dimensions does indeed exhibit a phase transition. The big breakthrough, however, came in 1942 where Lars Onsager published his analytical solutions to the two-dimensional Ising model in which the Ising model became the first exactly solvable model that enabled us to observe phase transitions analytically. In the first part of this project report, we introduce the Ising model and derive its important macroscopic properties. We solve the model in one dimension and show that the one-dimensional model does not exhibit a phase transition. We merely state Onsager's analytical solution which describes how the Ising model in two dimensions does indeed exhibit a phase transition.

In the following section, we introduce a Markov Chain Monte Carlo (MCMC) method to simulate the Ising model in two dimensions. We implement the Metropolis-Hastings[6] algorithm which was first developed after World War II when Metropolis and his team of mathematicians were exploring the physics behind fission and fusion for use in thermonuclear weapons. In the Ising model, the distribution of spins depend on the temperature and follow the Boltzmann distribution. To identify the temperature at which a phase transition occurs, macroscopic properties can be determined as a function of the reduced temperature to see where singularities in the derivatives of the free energy occur. However, the problem is that the Boltzmann distribution is not uniform and is weighted towards lower energies. Ideally, there would be a way to pick the spin configurations that correspond more often to the more frequently occuring states. That was precisely the contribution from physicist Metropolis and his team of mathematicians, though without explaining the statistics behind the method. That's where the name Metropolis-Hastings arises, acknowledging the statistician W.K.Hastings' contribution in 1970 where the theory behind the method was explained and generalised. The Metropolis-Hastings algorithm is also used in various disciplines: from numerical integration to simulated annealing.

In the final section, we introduce the concept of machine learning, considered to be a subset of artificial intelligence. Machine learning is used across many areas, not just mathematics and physics. We will focus on supervised learning consisting of a regression problem in the aim to predict the order parameter of the Ising model. Following *Machine Learning Phases of Matter*, an Artificial Neural Network can be implemented to correctly identify the magnetisation of the Ising model as the relevant order parameter.

The aim of this project is to replicate the results from the Nature paper which uses a Neural Network to predict the order parameter of the Ising model. By first introducing the Ising model and deriving its macroscopic properties from the knowledge of the Partition function, we can simulate the model

[3]Wilhelm Lenz 1888-1957. Physicist

[4]Ernst Ising 1900-1998. Physicict

[5]Rudolf Peierls 1907-1995. Physicist

[6]Nicholas Metropolis proposed the algorithm for symmetric distributions. In 1970, W.K.Hastings extended the algorithm to the more general case.

using the Metropolis-Hastings algorithm. Using the spin configurations and magnetisation from the simulations, we can feed this to our Neural Network to give us the desired output.

2 Fundamental Results of Statistical Mechanics

The Boltzmann distribution is a fundamental result in statistical mechanics and thermodynamics. The Boltzmann distribution is a probability distribution that gives the probability that a system will be in a certain state as a function of the energy of that state and the temperature of the system. For example, if we are studying a car engine, the burning petrol inside the cylinder of the engine is the *system*; the piston, exhaust system, radiator and air outside form the *surroundings* of the system. Moreover, the *state* of that system would be the knowledge of measurable quantites at a given time, such as the pressure, temperature or volume within the cylinder. We must first introduce important terminology, such as a system and its state, which we will do through an example throughout this section.

2.1 Systems

To define the term "system", we will introduce and use the following example which we will build upon in the following sections. Consider a small system, S, which is a system that can exchange energy, such as heat, with its surroundings. Let S be in contact with an infinitely large system, L, of temperature, T, which forms an *isolated* system, namely $S + L$. *An isolated system is one which does not exchange energy with its surroundings.* The energy of a system is given by the sum of the kinetic energy and potential energy of each particle within the system. We will denote the energy of the isolated system as ϵ_0. The large system and small system will eventually reach *thermal equilibrium*, where the large system, L, is big enough such that the temperature of L remains unchanged. For example, if we were to throw an ice cube into the ocean then the ice cube will not affect the temperature of the ocean since it is much larger in comparison to the ice cube. Heat can be exchanged between the small and large systems, so, if the small system has energy, ϵ_i, then the large system will have energy $\epsilon_0 - \epsilon_i$. We can visualise this in Fig. 1:

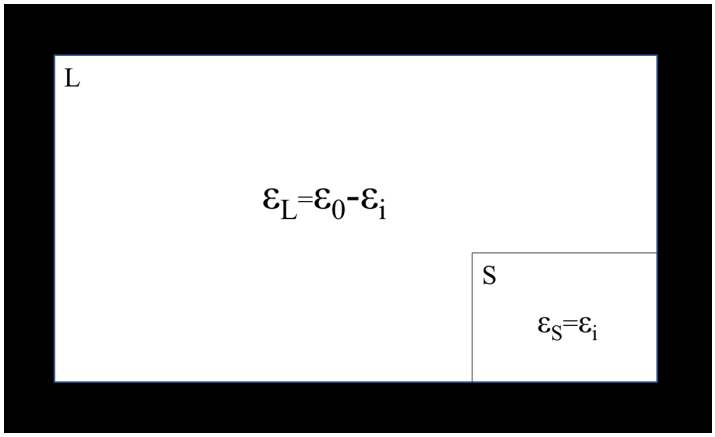

Figure 1: *Small system of energy ϵ_i forming an isolated system $S + L$ of energy ϵ_0.*

2.2 Microstate

A microstate is one possible arrangement of the distribution of energy of all particles within our isolated system[14]. For example, consider a gas (such as water vapour) in a box. A microstate of the gas would be the knowledge of the position and velocity of each particle. If our gas has 10^9 particles within the box, then we have a much larger number of microstates to consider, since each particle can be in more than one state. In terms of our isolated system, $S + L$, if we specify the microstate of S, then the probability of the system, $S + L$, being in a certain state depends on the number of microstates of the large system, L, with energy $\epsilon_0 - \epsilon_i$. We will let Ω_L and Ω_S denote the number of microstates of the large system and small system, respectively. Since the small system and large system are independent, the total number of microstates is given by $\Omega_{TOTAL} = \Omega_S\Omega_L$. Suppose we

are interested in one specific microstate of S, which we will call the i-th microstate. We ask ourselves, what is the probability, p_i, of finding the system in that specific microstate?

The probability, p_i, will be proportional to the number of microstates, Ω_{TOTAL}, of the whole system $S + L$. Since the number of possible microstates in the small system, Ω_S, is *small* in comparison to Ω_L, we can neglect Ω_S and assume $\Omega_{TOTAL} \sim \Omega_L$. We postulate that the total number of microstates in the system corresponds to the number of microstates of L such as:

$$\Omega_{TOTAL} = \Omega_L(\epsilon_0 - \epsilon_i). \tag{1}$$

2.3 Entropy

Now that we have defined a *microstate*, we must introduce an important concept called the *entropy*. The entropy is given by[14]:

$$S = k_B \ln(\Omega), \tag{2}$$

where k_B is defined as the Boltzmann constant ($\approx 1.38 \times 10^{-23} JK^{-1}$) and Ω denotes the total number of microstates available to a system. In our example, the total system is $S + L$. If we are interested in the entropy of the large system, L, then using the definition above, we can say that the entropy of the large system is $S_L = k_B \ln(\Omega_L)$. We may wish to know why the entropy in eq. 2 is formulated this way. It is clear to see that the number of microstates available to our total system will be exponentially large and we want the entropy to be extensive, meaning that the entropy will change if the number of microstates change. So, we must take the logarithm. The entropy of the total system is given by the sum of the entropy of the sub-systems and the total number of possible microstates is given by the product of the microstates of the two systems. However, this is not extensive. In order to get additivity in the entropy, we recall that $\ln(AB) = \ln(A) + \ln(B)$ for arbitrary A and B. By taking the logarithm, the entropy is extensive.

The derivative of the entropy, S, with respect to the energy, ϵ, is an important result in thermodynamics. From the second principle of thermodynamics, it can be shown that $\frac{\partial S}{\partial \epsilon}$ is just the inverse temperature, given by:

$$\frac{\partial S}{\partial \epsilon} = \frac{1}{T}, \tag{3}$$

where T is the temperature of the system we are interested in. This is an important result which we will require in our derivation of the Boltzmann distribution.

2.4 Derivation of the Boltzmann Distribution

We are now equipped with the important and neccessary terminology and definitions to derive the Boltzmann distribution. From our example that we have used so far, we know that the system we are interested in is $S+L$ and that the number of microstates in this system is given by eq. 1. We previously stated that the probability of finding the system, $S + L$, in a specific microstate is proportional to the total number of microstates of the system, Ω_{TOTAL}. We will let S_L denote the entropy of the large system, L. Using the relationship between the entropy of the large system, S_L, and the number of microstates of the large system, Ω_L, we can write:

$$p_i \propto \Omega_L(\epsilon_0 - \epsilon_i) = e^{\frac{S_L(\epsilon_0 - \epsilon_i)}{k_B}}. \tag{4}$$

Since we suppose L is infinitely large, it must be that $\epsilon_i \ll \epsilon_0$. Therefore, we can Taylor expand S_L about $S_L(\epsilon_0)$ to obtain:

$$S_L(\epsilon_0 - \epsilon_i) = S_L(\epsilon_0) - \epsilon_i(\frac{\partial S_L}{\partial \epsilon})_V + \ldots, \tag{5}$$

where V denotes the volume which we keep constant. Any higher order terms are insignificant, so, we keep only the lowest terms in ϵ_i. Since we know from section 2.3 that the derivative of the entropy, S, with respect to the energy, ϵ, is the inverse of the temperature, then we must have:

$$p_i \propto e^{(\frac{S_L(\epsilon_0)}{k_B} - \frac{\epsilon_i}{k_B T})} \propto e^{(\frac{-\epsilon_i}{k_B T})}. \tag{6}$$

We note that $S_L(\epsilon_0)$ is just a constant, independent of the microstate we are interested in. Hence, if we call the constant of proportionality, $\frac{1}{Z}$, our result for the probability, p_i, becomes:

$$p_i = \frac{1}{Z} e^{-\frac{\epsilon_i}{k_B T}}, \tag{7}$$

which is the Boltzmann distribution. If we say that the probability that the system is in *some* microstate is equal to 1: $\sum_j p_j = 1$, then we can write the normalisation constant of the Boltzmann distribution as:

$$Z = \sum_{i \in states} e^{\frac{-\epsilon_i}{k_B T}}, \tag{8}$$

where ϵ_i is the energy of the i-th microstate, k_B is the Boltzmann constant and T is the temperature in which we sum over all possible microstates.

The importance of the Boltzmann distribution should not be over-looked. It lies at the heart of thermodynamics and statistical mechanics. As suggested in the title of this project, we are interested in such a concept called a *phase transition*. A phase transition is simply where the macroscopic properties of matter or a material drastically change phase as a consequence of variable macroscopic quantities such as the temperature or volume. We see this in everyday life with water at different temperatures. At $0°C$, water changes state and freezes to ice which expands and takes over a larger volume than the water. Melting the ice causes the ice to change state again to water. If we heat the water to $100°C$, we will see that the water boils and changes state to water vapour which we know is a gas that can fill a larger volume in a room than the water. These are examples of phase transitions which we will now discuss more in depth for an important, and widely used, model called the *Ising Model*.

3 The Ising Model

In this section, we will introduce the Ising Model. We will discuss the idea of a Hamiltonian after which we will then solve the model in 1 dimension and present the solution.

3.1 The Hamiltonian of the Ising Model

The Ising model is a discrete mathematical description of particles, where each particle is fixed to a lattice configuration of a finite number of sites. Each site can be in one of two states. Consider a set, Λ, of lattice sites, each with a set of adjacent sites forming a D-dimensional lattice. For each lattice site, $i \in \Lambda$, there is a discrete variable σ, where $\sigma_i \in \{-1, 1\}$ represents the site's spin. We say that $\sigma_i = +1$ represents spin "up" and $\sigma_i = -1$ represents spin "down". A spin configuration, σ, of the system is given by $\sigma = \{\sigma_i | i \in \Lambda, \sigma_i \in \{-1, 1\}\}$. The representation of a $2D$ system of spins is shown in Fig. 2:

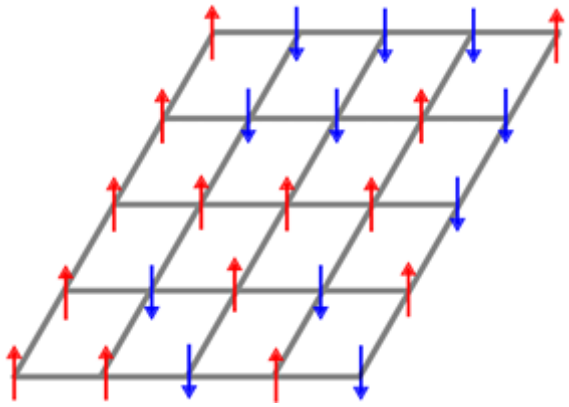

Figure 2: *A 2-dimensional lattice configuration of size $N = 5$ with arrows indicating the direction of spin. Red arrows pointing up represent spin "up" and blue arrows pointing down represent spin "down".*

In a regular, square lattice of D dimensions, each site has 2^D nearest neighbours. We will impose periodic boundary conditions so that each site has the same number of neighbours. For the example where $D = 2$, the lattice is of size, $N \times N$, with N spin sites in each direction, so, there are 2^{N^2} different configurations of sites.

Now, we want to know the total energy of the whole spin system. In the previous section, we denoted the energy of a system as ϵ. In the Ising model, the energy of the spin system is referred to as the *Hamiltonian*, so from now on, we will denote the energy of our spin system as H. The Hamiltonian, H, is composed of the energy from interactions between spin sites and the energy arising from spin interactions with the electric field, if the system is in the presence of an external electric field. We can write the Hamiltonian as:

$$H(\sigma) = -J \sum_{<i,j>} \sigma_i \sigma_j - h \sum_i \sigma_i, \tag{9}$$

where J is the interaction constant with units of energy and h is the external electric field term. We will assume that the coupling constant $J > 0$. In the first summation of eq. 9, we sum over pairs of adjacent spins where every pair is counted once. We assume that each site only interacts with the sites directly adjacent to it, which we have denoted by $< i, j >$, meaning that the sites i and j are nearest neighbours. Consider two neighbouring spin sites to be σ_i and σ_j. If the two spins are aligned $\pm$ and $\pm$, respectively, then we have a negative contribution to the energy. If, however, the spins are anti-aligned $\pm$ and $\mp$, respectively, then we only receive a positive contribution to the energy. To heighten our discussion of the coupling constant, J, further, the size of J tells us how strongly neighbouring spin sites are coupled and the sign of J shows whether neighbouring spin sites prefer to align/anti-align. We must note that the lowest energy state occurs when all spin sites are aligned,

meaning the model favours alignment and can exhibit a spontaneous magnetisation. In the second summation of eq. 9, we consider only one spin and its interaction with the electric field. The size of h tells us how strong the field is and the sign of h tells us whether spin "up" or spin "down" is preferred.

3.2 Magnetisation

An important property of the Ising model which we need to introduce is the magnetisation. The magnetisation, M, is one of the macroscopic properties of the Ising model which describes the average spin state on each particle. The magnetisation per spin for a system of lattice size $N \times N$ spins is:

$$M(h,T) = \frac{1}{N^2}\sum_i \sigma_i, \tag{10}$$

where σ_i is the spin on site i. When a system's spin states align such that the magnetisation is on a macroscopic scale, the system is said to be ferromagnetic [8]. It can be shown that the magnetisation is related to the partition function defined in eq. 8 via:

$$M(h,T) = -k_B T \frac{\partial \ln(Z)}{\partial h}, \tag{11}$$

where h is the external electric field.

3.3 Free Energy

For a system at constant temperature, we get the thermodynamic potential, F, from the first law of thermodynamics called the free energy. The *First Law of Thermodynamics* states that the change in *internal energy* of a system is given by the difference between the *heat* added to the system and the *work done* by the system, $\Delta U = Q - W$. The free energy is the difference between the energy needed to create a system and the energy available from the system's surroundings, $F = U - TS$, where T is the absolute temperature and S is the entropy. It can be shown that F is related to the partition function as:

$$F(h,T) = -k_B T \ln Z. \tag{12}$$

3.4 Specific Heat

Another important property, called the specific heat $C(h,T)$, is also related to the partition function, Z, via:

$$C(h,T) = \frac{\partial^2 F}{\partial T^2} = \frac{1}{k_B}\frac{\partial^2 \ln(Z)}{\partial \beta^2}, \tag{13}$$

where F is the free energy of the spin system, T is the temperature and $\beta = \frac{1}{k_B T}$. We will soon simulate the specific heat, $C(h,T)$, against the reduced temperature, $T_r = \frac{J}{k_B T}$, to see what effect the temperature has on this property concerning a certain phenomenon of the model about a critical temperature.

3.5 The Ising Model in One Dimension

We need to discuss the importance of periodic boundary conditions for the one-dimensional model which we will use to write the Hamiltonian and the Partition function. In order to solve the model, we will go through the transfer matrix method to solve the partition function in one dimension. Lastly, we can use our solution to the partition function to determine the average magnetisation and the specific heat via deriving the free energy and we will find that no phase transition can exist in the one-dimensional Ising model.

3.5.1 Boundary Conditions

We have previously introduced the Hamiltonian of the Ising model in eq. 9. However, in order to solve the model in one dimension, we choose to impose periodic boundary conditions. Consider a lattice configuration in one dimension consisting of N spin sites. Each site is labelled with integer i and we assume periodic boundary conditions, namely $\sigma_1 = \sigma_{N+1}$. We can think of this as connecting the two end points of a line to form a circle, seen in Fig. 3:

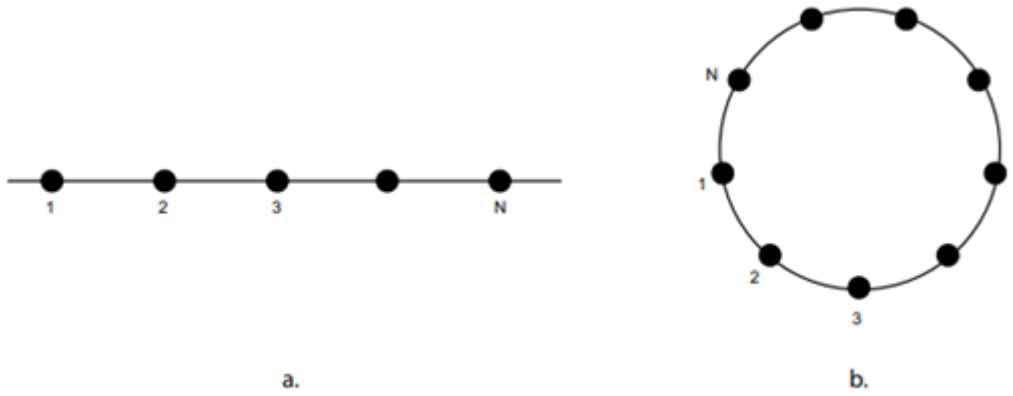

Figure 3: *Periodic boundary conditions imposed on the Ising model in one dimension. Black dots represent spin sites and black lines represent interaction. (a) lattice configuration in one dimension, (b) one-dimensional lattice configuration with periodic boundary conditions.*

We can only consider the system as closed when we impose periodic boundary conditions i.e no terms in the endpoints.

3.5.2 The Partition function

The Partition Function for the Ising model in one dimension follows from eq. 8 as:

$$Z_N = \sum_{\sigma} e^{k\sum_i^N \sigma_i\sigma_{i+1} + \alpha\sum_i^N \sigma_i}, \tag{14}$$

where we sum over all possible spin configurations, with $k = \beta J$ and $\alpha = \beta h$ for J the coupling constant and h the external electric field term.

3.6 The Solution in 1D

We want to write the partition function in a form such that we can derive the magnetisation and the specific heat from the free energy. In order to do this, we will have to start by rewriting the partition function so that we can drop all summations from eq. 8. We can do this by expanding the sum in the Boltzmann weight [23] such as:

$$Z_N = \sum_{\sigma} e^{[k(\sigma_1\sigma_2 + \sigma_2\sigma_3 + \dots) + \alpha(\sigma_1 + \sigma_2 + \dots)]}, \tag{15}$$

and split the α terms:

$$Z_N = \sum_{\sigma} e^{(k\sigma_1\sigma_2 + k\sigma_2\sigma_3 + \dots + \frac{\alpha}{2}\sigma_1 + \frac{\alpha}{2}\sigma_1 + \frac{\alpha}{2}\sigma_2 + \frac{\alpha}{2}\sigma_2 + \dots)}, \tag{16}$$

and we want to express Z_N as a product for just the nearest neighbour interactions, using the periodic boundary condition for the one-dimensional model in the second term:

$$Z_N = \sum_{\sigma} e^{[k\sigma_1\sigma_2 + \frac{\alpha}{2}(\sigma_1 + \sigma_2)]} \dots e^{[k\sigma_N\sigma_1 + \frac{\alpha}{2}(\sigma_N + \sigma_1)]}. \tag{17}$$

We define the function $T(\sigma,\ \sigma^*)$ such that:

$$T(\sigma,\ \sigma^*) = e^{[k\sigma\sigma^* + \frac{\alpha}{2}(\sigma+\sigma^*)]}. \tag{18}$$

Substituting eq. 18 into eq. 17 gives the partition function with the summation over the possible spin state of each spin, σ, to be:

$$Z_N = \sum_{\sigma_1}\sum_{\sigma_2}\cdots\sum_{\sigma_N} T(\sigma_1,\ \sigma_2)T(\sigma_2,\ \sigma_3)\ldots T(\sigma_N,\ \sigma_1). \tag{19}$$

Consider T to be a 2×2 matrix with elements representing all possible spin configurations of $T(\sigma,\ \sigma^*)$ such that T is given by:

$$T = \begin{pmatrix} T(+,+) & T(+,-) \\ T(-,+) & T(-,-) \end{pmatrix} = \begin{pmatrix} e^{k+\alpha} & e^{-k} \\ e^{-k} & e^{k-\alpha} \end{pmatrix}. \tag{20}$$

Notice that each factor $T\left(\sigma_i\sigma_{i+1}\right)T\left(\sigma_{i+2}\sigma_{i+3}\right)$ of the partition function can be regarded as the matrix product of T. We consider our partition function:

$$Z_N = \sum_{\sigma_1}\sum_{\sigma_2}\cdots\sum_{\sigma_N} T(\sigma_1,\sigma_2)\ldots T(\sigma_N,\sigma_1). \tag{21}$$

By the definition of a matrix product, we can define this as the matrix T^N up from the summation of σ_1 as:

$$Z_N = \sum_{\sigma_1} T(\sigma_1,\sigma_1)^N, \tag{22}$$

and from the definition of the trace of a matrix, eq. 22 gives us the partition function, dependent only on the interaction constants, in the form:

$$Z_N = Tr(T^N). \tag{23}$$

If T is a real and symmetric matrix, then it can be diagonalised via:

$$T = ADA^{-1}, \tag{24}$$

where D is a diagonal matrix with eigenvalues λ_1 and λ_2 of T in its diagonal such that:

$$D = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} = \begin{pmatrix} -2\sinh(2\beta J) & 0 \\ 0 & e^{\beta J}\cosh(\beta h) - \sinh(2\beta J) \end{pmatrix}, \tag{25}$$

and A is the matrix containing the eigenvectors: $\vec{x_1}$ and $\vec{x_2}$ of T such that:

$$A = (\vec{x_1}, \vec{x_2}). \tag{26}$$

The same matrix, A, that diagonalises T can also diagonalise T^N by:

$$T^N = (ADA^T)\cdot(ADA^T)\cdot\cdots\cdot(ADA^T) = AD^NA^T, \tag{27}$$

and using the property of the trace of a matrix, we can write eq. 27 as:

$$Tr(AD^NA^T) = Tr(D^N). \tag{28}$$

Since the trace of a matrix is unchanged after diagonalisation, eq. 28 can be written as the summation of the eigenvalues of T. Then, we have that the partition function can be written as:

$$Z_N = Tr(D^N) = \lambda_1^N + \lambda_2^N. \tag{29}$$

3.6.1 Free Energy

We need to be able to derive the free energy in order to determine the magnetisation and the specific heat for the one-dimensional solution. From eq. 12, we can write the free energy in terms of the partition function:

$$-(k_BT)^{-1}F = \log Z_N = \log(\lambda_1^N + \lambda_2^N) = \log[\lambda_1^N(1 + \frac{\lambda_2^N}{\lambda_1^N})]. \tag{30}$$

Since the free energy, F, is related to the size of the system, N, we need a quantity in the limit that is not proportional to N [3], hence:

$$-(k_BT)^{-1}N^{-1}F = \log\lambda_1 + N^{-1}\log[1 + (\frac{\lambda_2}{\lambda_1})^N], \tag{31}$$

where we assume $\lambda_1 > \lambda_2$. From this assumption, we can show that the last term on the RHS will vanish by taking the limit as $N \to \infty$ such as:

$$\lim_{N\to\infty}(N^{-1}\log[1 + (\frac{\lambda_2}{\lambda_1})^N]) = 0, \tag{32}$$

giving the free energy per lattice site to be:

$$F = -k_BT\log(2\sinh(2\beta J)). \tag{33}$$

3.6.2 Magnetisation and Specific Heat

Our last step is to derive the magnetisation and the specific heat from our derivation of the free energy. Returning to our matrix T, we will calculate the eigenvalues of T by:

$$0 = \det(T - \lambda) = \begin{vmatrix} e^{k+\alpha} - \lambda & e^{-k} \\ e^{-k} & e^{k-\alpha} - \lambda \end{vmatrix} \tag{34}$$

$$= (e^{k+\alpha} - \lambda)(e^{k-\alpha} - \lambda) - e^{-2k} \tag{35}$$

$$= \lambda^2 - \lambda(e^{k+\alpha} + e^{k-\alpha}) + e^{2k} - e^{-2k}. \tag{36}$$

Giving the eigenvalues as:

$$\lambda_\pm = \frac{1}{2}e^{-2k}(e^{3k+\alpha} + e^{3k-\alpha} \pm (e^{2(3k+\alpha)} + e^{2(3k-\alpha)} - 2e^{6k} + 4e^{2k})^{\frac{1}{2}}). \tag{37}$$

Using hyperbolic functions, we get the simplified, final form of the eigenvalues to be:

$$\lambda_\pm = e^k\cosh\alpha \pm (e^{2k}\sinh^2\alpha + e^{-2k})^{\frac{1}{2}}. \tag{38}$$

We get the free energy per lattice site as a function of h and the temperature, T, as:

$$F(h,T) = -k_BT[e^k\cosh h + (e^{2k}\sinh^2 h + e^{-2k})^{\frac{1}{2}}], \tag{39}$$

and substituting eq. 39 into eq. 11, we get the magnetisation as a function of h and the temperature, T, to be:

$$M(h,T) = \frac{e^k\sinh(\beta h)}{(e^{2k}\sinh^2(\beta h) + e^{-2k})^{\frac{1}{2}}}. \tag{40}$$

We notice that M is an analytical function for real h and positive T, meaning that no phase transitions take place in the one-dimensional model for positive temperatures as we suspected. A plot of $M(h,T)$ is shown in Fig. 4 for a $1D$ system of $N = 100$ spins for the external electric field $h = -0.1$, $h = 0$ and $h = 0.1$, respectively, and the interaction constant, $J = 1$:

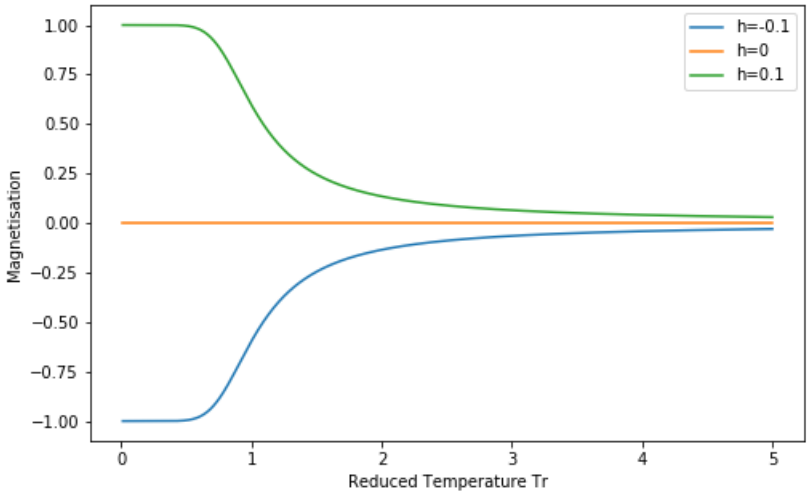

Figure 4: *Analytical solution for the magnetisation, $M(h,T)$, based on a 1D system of $N = 100$ spins with $J = 1$. Three plots are shown. The magnetisation for $h = -0.1$ is shown in blue. The orange horizontal plot shows that the magnetisation is zero for $h = 0$, M(h=0,T)=0. Finally, the magnetisation for $h = 0.1$ is shown in green.*

Since the energy of the system can be written as:

$$H(h,T) = -\frac{\partial}{\partial\beta}\log Z_N, \tag{41}$$

we can write the specific heat as the partial derivative of eq. 41 with respect to the temperature, T, by:

$$C(h,T) = \frac{\partial H}{\partial T} = k\beta^2\frac{\partial^2}{\partial\beta^2}\log Z_N. \tag{42}$$

3.6.3 Special Case

We can also consider the solution to the one-dimensional Ising model in the special case in the absence of the external electric field term: $h = 0$. We know from eq. 23 that the partition function takes the form:

$$Z_N = Tr(T^N), \tag{43}$$

which we can write as:

$$Z_N = e^{N\beta J}[(1+e^{-2\beta J})^N + (1-e^{-2\beta J})^N], \tag{44}$$

$$Z_N = (e^{\beta J}+e^{-\beta J})^N + (e^{\beta J}-e^{-\beta J})^N. \tag{45}$$

Using the hyperbolic functions, we can write the partition function as:

$$Z_N = (2\cosh(\beta J))^N + (2\sinh(\beta J))^N, \tag{46}$$

and by the definition of the tangential hyperbolic function, we get the partition function to be:

$$Z_N = (2\cosh(\beta J))^N[1 + (\tanh(\beta J))^N]. \tag{47}$$

If we take the limit: $N \to \infty$, we can eliminate the second term and calculate the free energy to be:

$$\frac{F(T)}{N} = -\frac{1}{N\beta}\log Z_N = -\frac{1}{\beta}\log(2\cosh(\beta J)), \tag{48}$$

and the energy of the system to be:

$$\frac{H}{N} = -\frac{1}{N}\frac{\partial}{\partial\beta}\log(Z_N) = -\tanh(J\beta). \tag{49}$$

From eq. 49 we can determine the specific heat as:

$$\frac{C(T)}{Nk_B} = \frac{1}{Nk_B}\frac{\partial H}{\partial T} = \frac{\beta^2}{N}\frac{\partial^2}{\partial\beta^2}\log Z_N = (\beta J)^2\,\mathrm{sech}^2(\beta J) = (\frac{J}{k_BT})^2\,\mathrm{sech}^2(\frac{J}{k_BT}). \tag{50}$$

A plot of the specific heat for $h = 0$, $C(h = 0, T)$, for a $1D$ spin system of $N = 100$ spins with $J = 1$ is shown in Fig. 5:

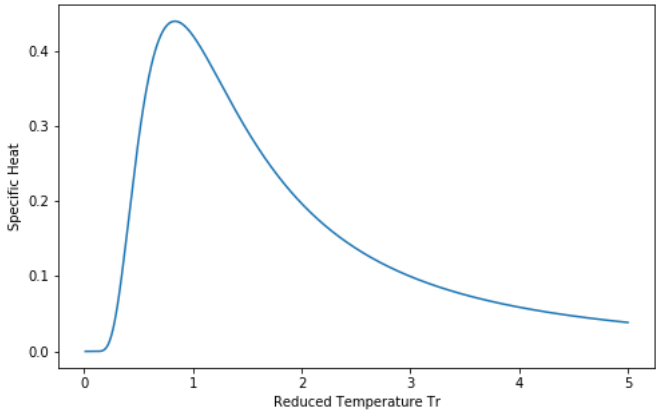

Figure 5: *Analytical solution for the specific heat, $C(h = 0, T)$, with $h = 0$ and $J = 1$ based on a system of $N = 100$ spins.*

We notice that the specific heat is a smooth function at $T \in [0\,,\,\infty)$ since no phase transition occurs in the one-dimensional Ising model.

3.7 Solution in 2D

The exact solution of the two-dimensional Ising model represents one of the landmarks in theoretical physics. It was originally derived by Lars Onsager in 1942 and published in Physical Review[19] in 1944. Onsager's paper revolutionalised the study of phase transitions and what we now call critical phenomena[17]. To appreciate as to why it is regarded as one of the landmarks of theoretical physics, it is worth remembering that, for a long time, it remained the first and only (mathematically rigorously) exactly solvable model exhibiting phase transition. Its discovery completely changed the course of developments in statistical mechanics.

It must be said that Onsager's method is highly non-trivial. His derivation is relatively difficult to follow step-by-step and his plan is quite obscure. A simplified version of Onsager's derivation, as given by Kaufman[7][10], has been discussed in various books, however, we shall merely quote the result.

Onsager showed that the canonical partition function $Z(h = 0, T)$ in the limit as $N \to \infty$ is given by:

$$\lim_{N\to\infty}\ln Z(h = 0, T) = \ln(2\cosh(2\beta J)) + \frac{1}{2\pi}\int_0^\pi \ln\frac{1}{2}(1 + \sqrt{1 - \kappa^2\sin^2(\phi)})d\phi, \tag{51}$$

where $\kappa = 2\sinh(2\beta J)/\cosh^2(2\beta J)$. The critical temperature, at which a phase transition occurs in the $2D$ model, is given by:

$$\tanh(\frac{2J}{k_BT_c}) = \frac{1}{\sqrt{2}} \implies \frac{k_BT_c}{J} = 2.269185, \tag{52}$$

[7]Bruria Kaufman, 1918-2010. Theoretical physicist

where T_c denotes the critical temperature. Onsager famously announced the following expression for the spontaneous magnetisation, M, in $2D$ for a square lattice, at the region $T_c < T$, at two different conferences in 1948, though without proof [16]:

$$M(h = 0, T) = \sinh(2\beta J)^{-\frac{1}{4}}, \tag{53}$$

where J is the interaction energy between spins. This is true only for $T_c < T$, otherwise the magnetisation is zero, $M(h = 0, T) = 0$, for $T < T_c$. An important remark is the exponent $-\frac{1}{4}$ which describes the property of the phase transition which we can see occuring in Fig. 6:

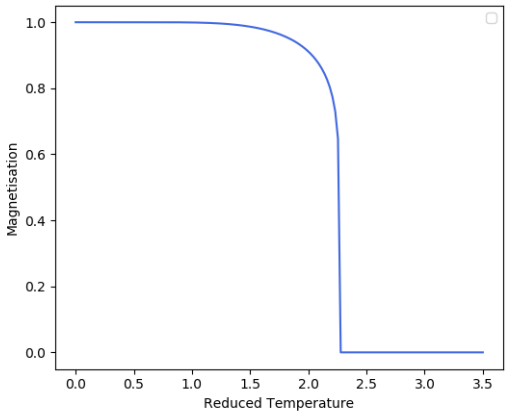

Figure 6: *Onsager's solution for the magnetisation, $M(h = 0, T)$, plotted against the reduced temperature, T_r, for the $2D$ Ising model for a lattice of size $L = 10 \times 10$ with $h = 0$ and $J = 1$.*

From here, it follows that the specific heat, $C(h = 0, T)$, is defined by:

$$C(h = 0, T) = k_B \beta^2 \frac{\partial^2}{\partial \beta^2} [\ln Z(h = 0, T)], \tag{54}$$

which diverges logarithmically as $T \to T_c$ as shown in Fig. 7:

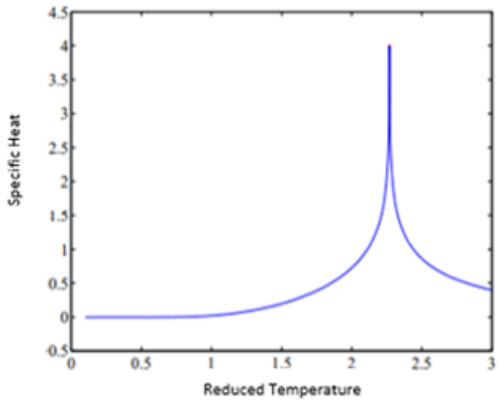

Figure 7: *Onsager's solution for the specific heat, $C(h = 0, T)$, plotted against the reduced temperature, T_r, for the $2D$ Ising model for a 10×10 lattice with $h = 0$ and $J = 1$.*

3.8 Phase Transitions

The Ising model is a model that can exhibit phase transitions. The model is exactly solvable in $2D$ and exhibits a phase transition between a paramagnetic and ferromagnetic phase. The model is used as

a theoretical laboratory since decades to understand phase transitions. In general, phase transitions appear when there are singularities in the derivatives of the free energy, F. Using a Monte Carlo simulation in section 4, we will see that the model indeed exhibits a phase transition in 2D.

4 Simulating the Model

In this section, we will introduce the idea of using a Monte Carlo approach to simulate the Ising model in 2-dimensions. We will explain the importance of the Metropolis-Hastings algorithm. This will enable us to study two thermodynamic properties, namely the magnetisation and the specific heat, against the reduced temperature, $T_r = \frac{J}{k_B T}$, which will allow us to generate a sequence of samples that are distributed according to a probability distribution. This sequence is labelled with an index called the Monte Carlo time. Finally, we will use the analytical result from Onsager to show that our implementation of the Metropolis-Hastings algorithm is correct.

4.1 The Metropolis-Hastings Algorithm

The Metropolis [15] algorithm is a Monte Carlo method. In our case, ***each sample describes a possible state of the system at equilibrium at a fixed temperature*** where the configuration is distributed according to the Boltzmann distribution. The time we refer to, in this case, is fictitious.

Consider a two-dimensional square lattice of spins as shown in a previous section. Let N be the number of spins in each direction. The Monte-Carlo approach to the Ising model is based on the Metropolis algorithm as follows:

1. A random configuration of spins of size $N \times N$ is intialised.
2. A spin is chosen at random and is flipped (meaning if its spin state was previously $+1$ it will now be -1 and vice versa).
3. The change in energy of the spin, ΔH, is calculated.
4. If the change in energy, $\Delta H \leq 0$, then the flip is accepted.
5. Otherwise, if $\Delta H > 0$, the spin is flipped if and only if $e^{-\beta H} > r$ where r is a random number on the interval $[0, 1]$ and $\beta = \frac{1}{k_B T}$.
6. This process is repeated until every spin in the configuration has been flipped.
7. An updated configuration is returned and the MC time is increased by 1.

The Metropolis algorithm shuffles through all possible spins of the lattice system, and ensures that the system occupies a given state with the Boltzmann probability. The algorithm generates samples that are distributed according to the Boltzmann weight. The Metropolis algorithm is an example of a Markov Chain that we will discuss in this section. To show the validity of the algorithm, we will use the concept of Markov chains.

4.2 Markov Chains

We will focus on a class of *discrete-time* stochastic processes. A discrete-time stochastic process is a sequence of random variables: $\mathbf{X} = \{ X_n : n \in \mathbb{N} \}$. We will denote the state space of $\mathbf{X}$ by S which we will assume to be discrete (but otherwise general). We will start by looking at some of the basic structures of Markov Chains.

4.2.1 Markov Property

A discrete-time stochastic process, $\mathbf{X}$, is said to be a Markov Chain if it satisfies the following. For any $s, i_0, \ldots, i_{n-1} \in S$ and for all $n \geq 1$:

$$P(X_n = s | X_0 = i_0, \ldots, X_{n-1} = i_{n-1}) = P(X_n = s | X_{n-1} = i_{n-1}). \tag{55}$$

In words, we say that the distribution of X_n given the entire past of the process depends only on the immediate past. For example, we may think of a particle moving around in the state space as time goes on to form a random sample path. The Markov property states that the distribution of where the

particle goes next depends only on where it is at that time, not on where it has been. This property is a reasonable assumption for many (though certainly not all) real-world processes.

There is one further assumption we will make about the process, $\mathbf{X}$, in addition to the Markov property. The Markov property states that the distribution of X_n given X_{n-1} is independent of $X_0, \ldots, X_{n-2}$, however, that does not comprehend the possibility that this conditional distribution could depend on the time index, n. In general, it could be, for example, that the distribution of where we go next given that we are in state, s, at time $n = 1$ is *different* from the distribution of where we go next given that we are in the same state, s, at time $n = 10$. Processes where this can happen are called time-inhomogeneous. In the rest of this section, we will assume that the process we are interested in is *time-homogenous*. That is, every time we are in state, s, the distribution of where we go next is the same. Mathematically, for any $i, j \in S$ and for all $n, m, k \geq 0$, the time-homogeneity property states:

$$P(X_{n+m} = j | X_m = i) = P(X_{n+m+k} = j | X_{m+k} = i). \tag{56}$$

The above conditional probabilities are the *n-step transition probabilties* of the chain because they are the probabilities of where we will be after n time steps from where we are now. Of basic importance are the 1-step transition probabilities:

$$P(X_n = j | X_{n-1} = i) = P(X_1 = j | X_0 = i), \tag{57}$$

which are time-independent and we denote the transition probabilities by p_{ij}. These transition probabilities can be put into matrix form called the transition probability matrix of a Markov Chain.

4.2.2 Transition Probability Matrix

We will denote the transition probability matrix by $\mathbf{P}$ with the (i, j)-th element being p_{ij}. Since $\mathbf{P}$ is a probability matrix, we have:

$$\sum_{j \in S} p_{ij} = \sum_{j \in S} P(X_1 = j | X_0 = i) = 1. \tag{58}$$

Thus, each row of $\mathbf{P}$ is a probability distribution over S. For this reason, we say that $\mathbf{P}$ is a stochastic matrix. It is clear that the 1-step transition probabilities determine the n-step transition probabilities, for any n, which we discuss in the following important properties [7].

4.2.3 Properties of the Transition Matrix

Let $p_{ij}(n)$ denote the n-step transition probabilities:

$$p_{ij}(n) = P(X_n = j | X_0 = i), \tag{59}$$

and let $\mathbf{P}(n)$ denote the n-step transition probability matrix whose (i, j)-th entry is $p_{ij}(n)$. Then, for all $n, m \geq 0$:

$$\mathbf{P}(m + n) = \mathbf{P}(n)\mathbf{P}(m), \tag{60}$$

which is equivalent to:

$$p_{ij}(m + n) = \sum_{k \in S} p_{ik}(m) p_{kj}(n). \tag{61}$$

In words, we have: the probability of going from i to j in $(m + n)$ steps is the sum over all k of the probability of going from i to k in m steps, then from k to j in n steps. We previously mentioned

that the 1-step transition probabilities determine the n-step transition probabilities. By repeated application of 60, we have:

$$\mathbf{P}(n) = \mathbf{P}(n-1)\mathbf{P}(1) \tag{62}$$
$$= \mathbf{P}(n-2)\mathbf{P}(1)^2 \tag{63}$$
$$\vdots \tag{64}$$
$$= \mathbf{P}(1)\mathbf{P}(1)^{n-1} \tag{65}$$
$$= \mathbf{P}(1)^n. \tag{66}$$

However, $\mathbf{P}(1)$ is just $\mathbf{P}$, the 1-step transition probability matrix. We conclude that $\mathbf{P}(n)$ can be determined by raising the transition matrix, $\mathbf{P}$ to the n-th power. Now, we will introduce some important properties of Markov Chains.

4.2.4 Irreducibility

A Markov Chain is said to be irreducible if, for all pairs of states $i, j \in S$, there exists some $m \in \mathbb{Z}$ such that $(p^m)_{ij} > 0$. What this means is that there are no two regions in the state space that are not connected and that it is possible to eventually reach any state from any other state. This property plays an important role in determining the long-time behaviour of the Markov Chain.

4.2.5 Stationary Distribution

Consider a Markov Chain over a finite state space, S. The transition probabilities are represented by the transition probability matrix $\mathbf{P}$. A stationary distribution for $\mathbf{P}$ is a row vector, denoted by $\vec{\pi}$, which satisfies the following properties:

1. $0 \leq \pi_i \leq 1$;
2. $\sum_i \pi_i = 1$;
3. $\mathbf{P}\vec{\pi} = \vec{\pi}$, or alternatively, $\pi_j = \sum_i \pi_i p_{ij}$ since the (i,j)-th element of $\mathbf{P}$ is p_{ij} with $\sum_j p_{ij} = 1$.

From the definition and the properties given above, we can see that $\vec{\pi}$ is an eigenvector of $\mathbf{P}$ with eigenvalue 1. It is normalised such that the sum of its entries are equal to one. If a Markov Chain starts in a stationary distribution, it will stay in that stationary distribution for all time, t. Note that a given Markov Chain may have more than one stationary distribution. For instance, if the transition probabilities are given by the identity transformation, then any distribution will be stationary.

4.2.6 Detailed Balance

We are interested in pairs, $\mathbf{P}$ and $\vec{\pi}$, for which the transition matrix, $\mathbf{P}$, has stationary distribution $\vec{\pi}$. The stationarity condition can be written:

$$\sum_{i\in\Omega} \pi_i p_{ij} = \pi_j = \sum_{i\in\Omega} \pi_j p_{ji}, \tag{67}$$

for all $j \in \Omega$. The first equality in eq. 67 is the definition of stationarity, $\mathbf{P}\vec{\pi} = \vec{\pi}$ as in bulletpoint 3 in 4.2.5. The second equality multiplies π_j by $\sum_{i\in\Omega} p_{ji}$ which equals 1. Fixing j and subtracting $\pi_i p_{ii}$ from both sides of eq. 67 yields:

$$\sum_{i:i\neq j} \pi_i p_{ij} = \sum_{i:i\neq j} \pi_j p_{ji}. \tag{68}$$

Eq. 68 may be interpreted as a balance condition. The left side shows the probability flowing into j from other states. The right side shows the probability flowing out of j to other states. In equilibrium, those two flows are equal, or balanced. A sufficient condition for eq. 68 is that:

$$\pi_i p_{ij} = \pi_j p_{ji}, \tag{69}$$

for all states $i, j \in \Omega$ and for the given distribution $\vec{\pi}$. Eq. 69 is the *detailed balance* equation[4]. The above states that the probability of transitioning from state i to state j is the same as the probability of transitioning from j to i. Hence, the chain running backwards is indistinguishable from the chain running forwards and so, we say that the system is in statistical equilibrium. For the case that $i = j$, eq. 69 carrys no information. It is easily seen that a transition probability that satisfies 69 with respect to the given distribution will leave that distribution stationary, since:

$$\sum_i \pi_i p_{ij} = \sum_i \pi_j p_{ji} = \pi_j \sum_i p_{ji} = \pi_j, \tag{70}$$

for the given distribution $\vec{\pi}$. A Markov Chain that respects detailed balance is said to be *reversible*.

4.3 Returning to the Metropolis Algorithm

In statistical mechanics, we often want to calculate an expectation value of a quantity for a dynamic system that is in thermal equilibrium at temperature, T. This is exactly what we want to do for the $2D$ Ising model using the Metropolis algorithm, but instead to simulate a phase transition at temperature, T. We know that the Metropolis algorithm is an example of a Markov Chain and we saw that a Markov Chain demands a stationary distribution. What we want to be able to show is that the Metropolis algorithm is appropriate for simulating the Ising model. To do this, we will show that the Boltzmann distribution is indeed the stationary distribution of the Metropolis algorithm.
In the Ising model, spin states are weighted by a probability dependent on the Boltzmann factor:

$$e^{-\beta H_i}, \tag{71}$$

where H_i is the energy of the system and $\beta = \frac{1}{k_B T}$. A normalised energy function for each energy state can be written:

$$P_B(H_i) = \frac{1}{Z} e^{-\beta H_i}, \tag{72}$$

where Z is known as the Partition Function of which we saw in section 2.
The first thing we noted about a Markov Chain is that it calls for a transition matrix, $\mathbf{P}$. Any matrix of this type over the state space of the Ising model would be too large to work with, especially for our simulation of the model, so we choose to pick a spin, say in initial state i, at random from our randomly initialised configuration of spins. We then flip that spin, say to state j, and calculate the change in energy, ΔH. What this means is that we start with a system in a particular energy state H_i. Each flip involves a random choice to move the system into another energy state or not. We use the transition probability, p_{ij}, to move the system from state H_i to H_j. With reversed indices, p_{ji} moves the system from H_j back to H_i. Since p_{ij} is a probability, we require that $\sum_j p_{ij} = 1$.
We **choose** transition probabilities such that they satisfy:

$$\frac{p_{ij}}{p_{ji}} = \frac{P_B(H_j)}{P_B(H_i)} = \frac{\frac{e^{-\beta H_j}}{Z}}{\frac{e^{-\beta H_i}}{Z}} = e^{-\beta(H_j - H_i)}, \tag{73}$$

where P_B is defined in eq. 72 via the Boltzmann factor. Notice the similarity between this condition above and that of eq. 69. We are choosing transition probabilities of a Markov Chain that will give us a stationary distribution consistent with Boltzmann probabilities.
If we start in energy state H_i, the distribution of the new energy states, H_j, after we make a transition using the transition probability p_{ij} is given by p_{ji}. Let H_i have probability distribution $\pi(H_i)$. Using eq. 68, the probability distribution of H_j after a transition is given by:

$$\pi(H_j) = \sum_i p_{ij} \pi(H_i). \tag{74}$$

What if the probability that the state is in H_i is equal to the Boltzmann distribution $\pi(H_i) = P_B(H_i)$? Using eq. 73, we must have:

$$\pi(H_j) = \sum_i p_{ij} \pi(H_i) = \sum_i p_{ij} P_B(H_i). \tag{75}$$

Then, by detailed balance, we have that:

$$\pi(H_j) = \sum_i p_{ji} P_B(H_j) = P_B(H_j) \sum_i p_{ji}. \tag{76}$$

Since we stated that $\sum_i p_{ij} = \sum_j p_{ji} = 1$, this simplifies to:

$$P_B(H_j) \sum_i p_{ji} = P_B(H_j). \tag{77}$$

Thus, if the probability of being in state i satisfies the Boltzmann distribution then so will the probability of being in state j. We have just shown that the Boltzmann distribution *is* a **stationary distribution** of the iterative Metropolis algorithm as long as the transition probabilities satisfy eq. 73. This is because a distribution satisfying detailed balance is a stationary distribution.

Hence, we conclude that the Metropolis algorithm is appropriate to simulate the Ising model, in which we will now discuss the simulation of two macroscopic properties, namely the magnetisation and the specific heat.

4.4 Simulating the Model

We have seen that the Metropolis algorithm is appropriate for simulating the $2D$ Ising Model since it allows us to set up a Markov Chain which is guaranteed to converge, after a sufficient amount of time (the equilibrium time) has elapsed, according to the Boltzmann distibution by repeatedly applying the Markov Process [21]. Once the system has been simulated, the aim is to investigate observables of the model by calculating various macroscopic properties to determine whether or not the system undegoes a phase transition. In particular, we want to see how the magnetisation and specific heat of a $2D$ square-lattice depend on the temperature of the system.

4.4.1 Macroscopic Properties

The average magentisation per unit spin is given by eq. 10 as:

$$M(h,T) = \frac{1}{N^2} \sum_i \sigma_i, \tag{78}$$

where σ_i is the spin on site i. We would expect that the order parameter should be close to zero for large temperatures where the system is disordered, while it should be close to 1, in absolute value, for small temperatures. Statistical mechanics tells us that these phases are separated by a phase transition, at which the value of the magnetisation departs from zero where it has a discontinuous derivative. The magnetisation can be used to show that a phase transition has occured and to approximate the critical temperature, T_c. By plotting the average magnetisation after equilibirum has been reached against a series of values of the reduced temperature, T_r, we expect a phase transition to be observed at T_c. The magnetisation should initially be $+1$ or -1 since it is ferromagnetic, but after T_c, it should be zero, since it becomes paramagnetic.

We will also recall from eq. 13, that the specific heat is given by:

$$C(h,T) = \frac{\partial^2 F}{\partial T^2} = \frac{1}{k_B} \frac{\partial^2 \ln(Z)}{\partial \beta^2}, \tag{79}$$

where F is the free energy of the spin system, T is the temperature and $\beta = \frac{1}{k_B T}$. Since the specific heat is a second derivative of the free energy, when plotted against the reduced temperature, T_r, it should exhibit a discontinuity at the critical temperature, T_c, which would indicate that a phase transition that has occured is of second order.

In terms of implementing the Metropolis algorithm, the code consists of a series of simple functions determining the configuration of spins, ΔH for a spin site, flipping the spin and sweeping through the configuration, picking spin sites at random. After the system is successfully updated, the macroscopic properties need to be calculated. Functions were defined to calculate the energy per spin, which can be

used to calculate the specific heat, and the average magnetisation per spin. The code was written such that the system was brought to equilibrium with these properties recorded and plotted as a function of the reduced temperature, T_r, accordingly. A large number of iterations (≈ 5000) were required to produce acceptable results. This method can be very computationally-expensive, which is why more efficient programming techniques were required.

The Metropolis algorithm was successfully used to produce a plot of the two macroscopic properties: the magnetisation and the specific heat, against the reduced temperature. The simulations on a $L = 10 \times 10$ lattice for the magnetisation and for lattice sizes $L = 10, 30, 60$ for the specific heat are plotted against the analytical result, obatined for $N \to \infty$ described in section 3.7 to show the accuracy of the simulation as shown in Fig. 8:

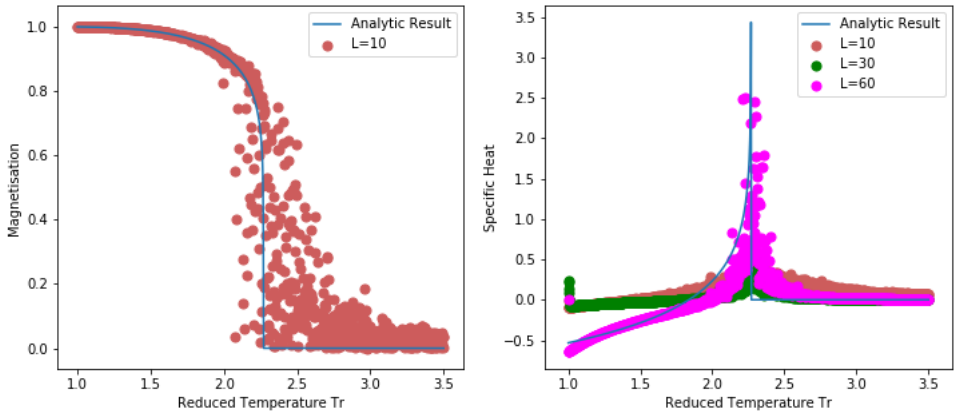

Figure 8: *Monte Carlo simulation of the average magnetisation, $M(h = 0, T)$, pictured left, and the specific heat, $C(h = 0, T)$, pictured right against the reduced temperature, T_r, with 5000 iterations of the Metropolis algorithm, both with $h = 0$ and $J = 1$. Simulations are plotted against the analytical solution given by Onsager. This system was represented by a lattice size $L = 10$ for the simulation of the average magnetisation. Lattice sizes $L = 10, 30, 60$ are pictured for the simulation of the specific heat.*

We see that the simulation of the magnetisation is initially $+1$ and as the temperature increases, the simulated magnetisation converges to 0. Between these phases, the simulated magnetisation suddenly drops around a critical temperature, $2.0 < T_c < 2.5$, indicating that a phase transition has occured, at which the magnetisation becomes 0 after this transition and is said to be paramagnetic. This compares well with the theoretical value from Onsager which we know to be $T_c \approx 2.269185$. In the plot of the specific heat, a discontinuity close to the critical temperature, T_c, is seen to occur, and is immediately 0 as the temperature is increased. This indicates that a phase transition has occured and is of second order. Since the Monte Carlo simulation compares well to the analytical result, we can conclude that the $2D$ square-lattice of spin sites *is* a disordered/ferromagnetic system since $M(h = 0, T) \to 0$ as $T \to T_c$.

4.4.2 Configuration

We can use the Metropolis algorithm to sweep through the configuration of spins and update the spins appropriately. Starting with a random initial configuration, we can then plot the instantaneous configurations as the system coarsens to its equilibrium state as shown in Fig. 9:

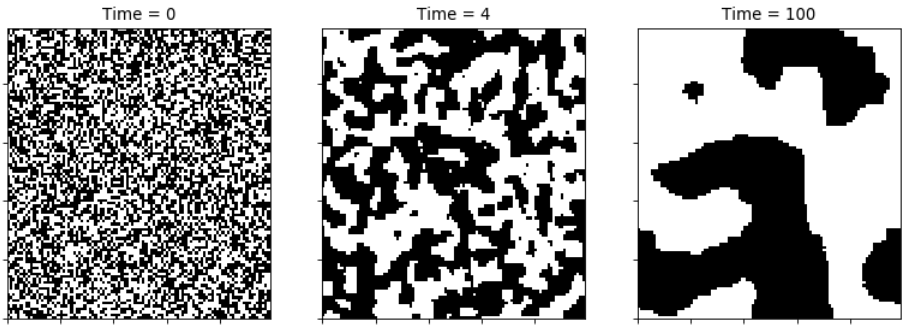

Figure 9: *The initially randomised configuration (left) compared with updated configurations at time=4 (center) and at time=100 (right). The system was represented by a* 16×16 *lattice at a reduced temperature of* $T_r = \frac{J}{k_B}$ *with* $J = 1$, $h = 0$ *and* 5000 *iterations of the Metropolis algorithm. White dots represent 'spin up' and black dots represent 'spin down'.*

Since $T_r = \frac{J}{k_B} < T_c$, this system should be ferromagnetic, which is clear from both of the updated configurations as the spins are beginning to form domains. By inspection, it is also evident that the periodic boundary conditions are correctly implemented. However, the configuration is initially in a disordered state (pictured left). When this happens, large domains of opposite spins (pictured center and right) will form. These separate domains cannot be easily aligned by flipping individual spins, and, as a result, the Markov Chain gets 'trapped' in this part of the state space for a long time, failing to access the more favourable set of states where most of the spins form a single aligned-domain. This is computationally-expensive, so it is wise to choose an initial state which *is* paramagnetic and in an ordered state.

5 An Introduction to Machine Learning

As stated in the introduction, the goal of this project is to reproduce the results of the Nature paper, [5]. In order to do so, we need to use a machine learning architecture, called a Neural Network, and teach that Neural Network how to calculate the order parameter of the $2D$ Ising model. However, before we start to implement our machine learning architecture and interconnect it with the Ising model, we need to understand the various types of machine learning, their concepts and an important method of optimisation that can be used.

5.1 Unsupervised and Supervised Machine Learning

Machine learning is a sub-field of artificial intelligence that gives computers the ability to learn without being explicitly programmed[13]. It is broadly defined as the capability of a machine to imitate intelligent human behaviour and is used to perform complex tasks in a way that is similar to how we solve problems. It becomes important when problems cannot be solved by means of typical approaches. In order to explain what machine learning can do, we will discuss the two major categories of machine learning.
Machine learning implementations are classified into two major categories, depending on the nature of the learning system. We will start by introducing supervised learning. *Supervised learning* is when an algorithm learns from example data and an associated target which can consist of numerical values or string labels, such as classes, in order to later predict the correct outcome when posed with new data. This approach is, of course, similar to the way we learn under the supervision of a teacher. *Unsupervised learning*, on the other hand, is when an algorithm learns from plain examples without any associated target, leaving the algorithm to determine any data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features which may represent a class or even a new series of uncorrelated values.

Another categorisation of machine learning tasks arises when we consider the desired output of a machine-learning system. Example categories of tasks are classification, regression and clustering. Since we want to be able to calculate the order parameter of the Ising model, the desired output of our system will be numerical values of the magnetisation for the spin configurations. Such a task where the predicted output is a continuous, numerical value is an example of a *regression* problem. Regression problems are usually tackled in a *supervised* way.
Thus, for our example of phase transitions in the Ising model, we need a *supervised* machine learning architecture in order to predict numerical values which will be the magnetisation for each spin configuration. Before discussing the machine learning architecture we will use and our learning problem, it is neccessary to first understand a machine learning problem in the context of supervised learning.

5.2 The Learning Problem

We will introduce a learning problem with a regression task as an example. Consider a dataset consisting of d items, each item having a corresponding label. There are also d labels. We want to be able to predict the label for each individual item within the dataset. This is a type of supervised machine learning; we give an example dataset and we also give the target, which are the labels for each item within the dataset. Our machine learning algorithm can learn from the example dataset and use that to make predictions on any new dataset we feed to it. Let $\vec{X}$ denote our input to the algorithm which is of the form:

$$\vec{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}, \tag{80}$$

where the items are $x_{i=1...d}$. Each item is given a known label, $\vec{Y}$, such as:

$$\vec{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \end{bmatrix}, \tag{81}$$

for individual labels $y_{i=1...d}$. The dataset formed by the input and the labels is given by (x_i, y_i). In order to be able to train the algorithm on example data and, later, test the algorithm on new data, we need to split the dataset into training and testing. For example, we could split the dataset by 80% of the data for training and 20% for testing.
Our learning problem is to create a function, say f, which predicts the label, y_i, for each input item x_i such that $f(x_i) \approx y_i$ for all i in the training set. This is the learning process at which the algorithm trains and learns from the training set. Once the algorithm is trained, we can make predictions to test to algorithm. The testing set can be used to test the reliability of predictions on new data that has not been encountered in the learning process. There is such a system which excels in regression tasks called an *Artificial Neural Network*. ANNs are one way to build such an approximate function by optimising its weights and biases.

5.3 Artificial Neural Networks

Artificial Neural Networks, or ANNs, have been inspired by the human brain. *An Artificial Neural Network is a collection of neurons that take an input and, in conjection with information from other neurons, develop an output without specific programmed rules*[2]. We may want to ask how such a network can give us our output without specific rules? The amazing thing about an Artificial Neural Network is that it is a type of supervised machine learning which means that we don't have to program it to learn explicitly: it learns all by itself. Essentially, Artificial Neural Networks solve problems through trial and error. In order for the Artificial Neural Network to learn, we must provide it with a dataset that it can train from. However, how can an Artificial Neural Network train itself just from data we have provided?
We will abbreviate Artificial Neural Network with ANN. An ANN is composed of a series of *neurons* which are organised in layers. A typical layout of an ANN is illustrated in Fig. 10:

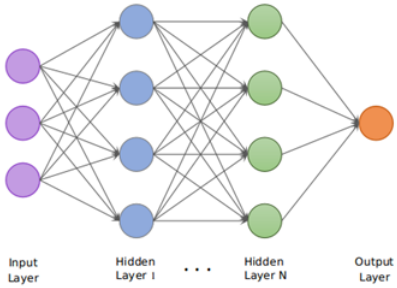

Figure 10: *A simple set-up of an ANN with an input layer, N hidden layers and an output layer.*

Within each neuron is a set of inputs, weights and a bias value. An example of an neuron can be visualised in Fig. 11:

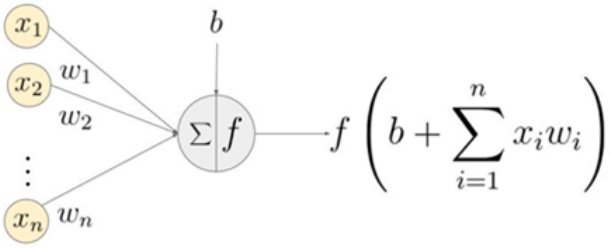

Figure 11: *An example of a neuron showing the input, $x_1 \dots x_n$, their corresponding weights, $w_1 \dots w_n$, a bias, b, and the activation function, f, applied to the weighted sum of the inputs.*

The ANN has neurons within the *input layer* which take the input signals and pass them to the next layer [18]. As an input enters the neuron in the next layer, it gets multiplied by a weight value and the resulting output is either observed, or passed to the next layer in the ANN. The weights are often contained within the *hidden layers*, which is where the training process occurs.
Weights and bias are both learnable parameters inside the network. The network will randomise the weight before the learning initially begins. As training continues, both parameters are adjusted towards the desired values and the correct output. The two parameters differ in the extent on their influence upon the input data. Put simply, a *bias* represents how far off the predictions are from the intended output and it makes up the difference between the network's output and its intended output. For example, a low bias would suggest that the network is making more assumptions about the form of the output, whereas a high bias would suggest the opposite. *Weights*, on the contrary, can be thought of as the strength of connection since a weight affects the amount of influence a change in the input will have on the output. A low weight value will have no change on the input, and, alternatively, a larger weight value will more significantly change the output. Additionally, a parameter called the learning rate determines how the weights are adjusted; whether they are adjusted by small amounts or by larger amounts.
To get a greater understanding of how the weights affect the output and how the training process works, it is helpful to imagine the theoretical ANN pictured in Fig. 10. The ANN contains a series of N hidden layers which apply transformations, in the form of a function called the *activation function*, to the input data. It is within the neurons of the hidden layers that the weights are applied. For example, a single neuron may take the input data and multiply it by an assigned weight value, which we know is initially random. This neuron may then add a bias value before passing this input to the next layer. The final layer of the ANN is known as the *output layer*. The output layer often tunes the inputs from the hidden layers to produce the desired output. The final result will be a model built from that data which is able to make predictions on future data.
To train the ANN, the first thing to do is to *initialise* the parameters. We have already said that the weights are randomly initialised and this is important in order to break the *symmetry* of the network.

This prevents all neurons in a layer from learning the same thing. Usually, the weights in each layer are generated using a normal distribution of zero mean and a variance of $\frac{1}{d}$, where d represents the number of inputs. The biases are initialised to zero. After the parameters are initialised, we proceed, iteratively, by following an *optimisation* algorithm that will try to minimise the difference between the actual output and the output predicted by the network. The most used algorithm to train Artificial Neural Networks is *Gradient Descent*.

6 The Gradient Descent Method

We want to generalise our understanding of a single-layer ANN to a multi-layer ANN. This is often called a *multi-layer perceptron* which we will denote as MLP. Firstly, we need to understand the architecture of a MLP which will enable us to describe the MLP in a mathematical way. Since we will consider a classification problem, the classic MLP can be described by:

- A linear function containing the input values.
- A Sigmoid function as the activation function (although this is arbitrary, we could also choose the tanh function).
- A loss function to compute the overall error of the neural network.
- A procedure to adjust the weights of the network (we will not cover this here).

6.1 Linear Function

The linear function is the function which contains our input values, set of weights and a bias value. Our input values will be a vector input, $\vec{x}$, and so a generic input will be of the form:

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = (x_d) \in \mathbb{R}^d, \tag{82}$$

where we have d input values within $\vec{x}$ and so $\vec{x} \in \mathbb{R}^d$. Next, we need to introduce the weights which will be a multi-dimensional array of values. We will denote the weights within a matrix of m rows, for the m number of neurons, and d columns, for d input values, where a generic weight matrix is of the form:

$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1d} \\ w_{21} & w_{22} & \dots & w_{2d} \\ \dots & \dots & \dots & \dots \\ w_{m1} & w_{m2} & \dots & w_{md} \end{bmatrix} = (w_{md}) \in \mathbb{R}^{m \times d}, \tag{83}$$

where (w_{md}) denotes all elements of W up to md which is in real coordinate space of $m \times d$ dimensions. Now we have this notation, we will look at a simple example of a network with:

- 3 input values
- 1 hidden layer with 2 neurons

The input vector, $\vec{x}$, for this example would be:

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \tag{84}$$

and since we have 3 input values connecting to 2 hidden neurons, we require 2×3 weights of the form:

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}. \tag{85}$$

The output of the linear function is given by the multiplication between $\vec{x}$ and W. This is given by:

$$\vec{z} = W \cdot \vec{x} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{86}$$

$$\vec{z} = W \cdot \vec{x} = \begin{bmatrix} x_1 w_{11} & +x_2 w_{12} & +x_3 w_{13} \\ x_1 w_{21} & +x_2 w_{22} & +x_3 w_{23} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}, \tag{87}$$

which we can also write as:

$$z_m = F(x_d, w_{md}) = b + \sum_m x_d w_{md}, \tag{88}$$

where z_m is the output of the linear function with z having m rows and F is a function of each element of $\vec{x}$ and each element of W. The d index indicates the rows in $\vec{x}$ and the columns in W whereas the index m denotes the number of rows in both W and z. Note that we have added a bias term, b.

6.2 Sigmoid function

Each element of the vector, $\vec{z}$, becomes an input for the sigmoid function, σ. The sigmoid function is the activation function of our network, although we could have chosen a different activation function. A nice property of sigmoid functions is that they are "mostly linear". We will denote the output from the activation function as $a(m)$, a being for activation, where:

$$a(m) = \sigma(z_m) = \frac{1}{1 + e^{-z_m}}. \tag{89}$$

The output, $\sigma(z_m)$, is another m-dimensional vector $\vec{a}$ with one entry for each neuron in the hidden layer such as:

$$\vec{a} = \begin{bmatrix} a(1) \\ a(2) \end{bmatrix}. \tag{90}$$

6.3 Loss function

The loss function is the measure of performance of a neural network. Conventionally, the loss function refers to the measure of error for a single training set and we will define this function generically as the mean squared error, MSE by:

$$LF = \sum_k (a_k - y_k)^2, \tag{91}$$

where k denotes the index for the input-output pair, a_k is the predicted output for training example k, and y_k is the expected output for training example k.

6.4 Training Process

We now have everything we need to be able to calculate the gradient but first, we need to understand the process of training the network. The training phase can be divided into two parts: a forward propagation phase, where our inputs and weights are fed into the network to compute predictions and the error (this is typically know as feed-forward propagation), and a backward propagation phase, where the error derivatives are calculated and the weights of the network are updated. To understand this, we will look at a simple example. Consider a network with 2 input units, 3 hidden layers and 1 output unit. The forward propagation phase connects the steps that we've defined so far: the linear function and the sigmoid function. Let the linear function be, λ, and the sigmoid function be σ, then our simple network can be described by:

$$\hat{y} = \sigma^{(2)}(\lambda^{(2)}(\sigma^{(1)}(\lambda^{(1)}(x_d, w_{md})))), \tag{92}$$

where $\hat{y}$ is the prediction. Notice that this is just a composition of functions, so, if we were to add another hidden layer, then our network becomes:

$$\hat{y} = \sigma^{(3)}(\lambda^{(3)}(\sigma^{(2)}(\lambda^{(2)}(\sigma^{(1)}(\lambda^{(1)}(x_d, w_{md})))))). \tag{93}$$

Now, we want to back propagate. In order to get a desired output, our network searches for a set of weights which minimise the error and, in our network, it will do this via the gradient descent algorithm. We now have the tools we require to calculate the gradient.

Our goal is to capture how the error changes as we change the weights of the network by a small amount. Recall our loss function:

$$LF = \sum_k (a_k - y_k)^2. \tag{94}$$

In order to calculate the gradient efficiently, without confusing ourselves, we will introduce a new notation. We will let L denote the last function in the network. For example, $a^{(L)}$ denotes the last sigmoid function at the output layer, $a^{(L-1)}$ denotes the previous sigmoid function at the hidden layer, and $x^{(L-2)}$ denotes the items in the input layer (which are the only items in that layer). To calculate our gradient, we must understand the following steps:

1. The error, E, depends on the value of the sigmoid function, a.
2. The value of the sigmoid function, a, depends on the value of the linear function, z.
3. The value of the linear function, z, depends on the values of the weights, w,

and so, we can trace a change of dependence on the weights.

It is common for a neural network to have more than one neuron. Since we will deal with many different layers, we will need to index them appropriately. Let j denote the units in the output layer, k denote the units in the hidden layer, and i to denote the units in the input layer. We will also use this notation for the weights.

Recall that the learning rate parameter was important in how the weights are adjusted. Let α denote the learning rate parameter. The error changes as the weights change via:

$$\delta w_{jk}^{(L)} = \alpha \frac{\partial E}{\partial w_{jk}^{(L)}}. \tag{95}$$

We want to find the derivative of the error with respect to the weights and to do this, we will have to use the multivariate chain rule since we are dealing with composite functions. This is calculated via:

$$\alpha \frac{\partial E}{\partial w_{jk}^{(L)}} = \frac{\partial E_i}{\partial a_j^{(L)}} \times \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \times \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}}, \tag{96}$$

where $\frac{\partial E_i}{\partial a_j^{(L)}}$ is the derivative of the error with respect to the sigmoid activation, $\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}$ denotes the derivative of the sigmoid activation with repesct to the linear function, and $\frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}}$ denotes the derivative of the linear function with respect to the weight. To make this clear, we will differentiate each part separately, starting with the outermost derivative. The derivative of the error with respect to the sigmoid activation function is:

$$\frac{\partial E_i}{\partial a_j^{(L)}} = a_j^{(L)} - y. \tag{97}$$

Now, the derivative of the sigmoid activation function with respect to the linear function is:

$$\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} = a_j^{(L)}(1 - a_j^{(L)}), \tag{98}$$

and finally, the derivative of the linear function with respect to the weights is:

$$\frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} = a_k^{(L-1)}. \tag{99}$$

Substituting 97, 98 and 99 into 96, respectively, gives:

$$\alpha \frac{\partial E}{\partial w_{jk}^{(L)}} = a_j^{(L)} - y \times a_j^{(L)}(1 - a_j^{(L)}) \times a_k^{(L-1)}. \tag{100}$$

We have found how the error changes as the weight connecting the hidden layer and the output layer, $w^{(L)}$, is changed. But now, we want to know how the error changes as the weight connecting the hidden layer and the input layer, $w^{(L-1)}$, is adjusted. To compute the derivatives of the error with respect to the weights, W, in $(L-1)$, we can apply the chain rule once more to obtain:

$$\alpha \frac{\partial E}{\partial w_{ki}^{(L-1)}} = \left(\sum_j \frac{\partial E}{\partial a_j^{(L)}} \times \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \times \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \right) \times \frac{\partial a_k^{(L-1)}}{\partial z_k^{(L-1)}} \times \frac{\partial z_k^{(L-1)}}{\partial w_k^{(L-1)}}. \tag{101}$$

Replacing 101 with the derivatives gives:

$$\alpha \frac{\partial E}{\partial w_{ki}^{(L-1)}} = a_j^{(L)} - y \times a_j^{(L)}(1 - a_j^{(L)}) \times w_{jk}^{(L)} \times a_k^{(L-1)}(1 - a_k^{(L-1)}) \times x_i^{(L-1)}. \tag{102}$$

That's the derivative of the error with respect to the weights for our MLP. Considering the new index notation, we have to write the derivative of the error with respect to the bias, b. There are two ways we could do this. We could treat the bias, b, as another feature and just simply add it to the weight matrix, w, or we could compute the derivative separately which is shown below as:

$$\frac{\partial E}{\partial b_j^{(L)}} = \frac{\partial E}{\partial a_j^{(L)}} \times \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \times \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}}. \tag{103}$$

We have already calculated the values for the first 2 derivatives, so we just need to calculate $\frac{\partial z_j^{(L)}}{\partial b_j^{(L)}}$. Keep in mind that the gradient of z does not depend on the bias, b, since b is a constant which we add at the end. Thus, 103 reduces to:

$$\frac{\partial E}{\partial b_j^{(L)}} = a_j^{(L)} - y \times a_j^{(L)}(1 - a_j^{(L)}). \tag{104}$$

Now, all we need is the derivative of the error with respect to the bias at layer, $(L-1)$, which is given by:

$$\frac{\partial E}{\partial b^{(L-1)}} = \frac{\partial E}{\partial a_j^{(L)}} \times \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \times \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \times \frac{\partial a_k^{(L-1)}}{\partial z_k^{(L-1)}} \times \frac{\partial z_k^{(L-1)}}{\partial b_k^{(L-1)}}. \tag{105}$$

And finally, by replacing 105 with the values of the derivatives, we obatin the derivative of the error with respect to the bias as:

$$\frac{\partial E}{\partial b^{(L-1)}} = a_j^{(L)} - y \times a_j^{(L)}(1 - a_j^{(L)}) \times w_{jk}^{(L)} \times a_k^{(L-1)}(1 - a_k^{(L-1)}), \tag{106}$$

and that is it, we have our derivatives of the error with respect to the weight and bias. Every time we train a neural network using the gradient descent, we need to compute the derivatives for all the weight and biases as we have generally shown above. The hardest part of our calculation was ensuring we were working with the correct indices. The next step would be to use the derivatives we have calculated to update the weight and bias values, however, we will end here.

7 Machine Learning and the Ising Model

In the final section, we study the application of machine learning methods on the $2D$ Ising model to predict the critical temperature in which a phase transition occurs. Using the architecture described in section 5 allows for the magnetisation to be learned as the relevant order parameter consistent with the results from the paper pubished in Nature [5].

7.1 Our Learning Problem

We finally have everything we need to tackle our learning problem which is to show that the magnetisation is the relevant order parameter for a phase transition to occur. This learning problem is approached in a supervised way using an Artificial Neural Network. Before introducing the structure and set-up of our network, it is wise to first study the learning problem, from input to output.
Our input to the model will be the spin configurations of the $2D$ Ising model, denoted by $\vec{X}$, such that:

$$\vec{X} = = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \tag{107}$$

where we have n spin configurations within $\vec{X}$. We label each spin configuration, x_i, with its corresponding magnetisation which we will denote y_i, where y_i are the labels within $\vec{Y}$. The reason for this is because we want the network to be able to predict the magnetisation of a spin configuration. An advantage we have is that we know, already, how to compute the magnetisation simply by hand from eq. 10 and so we know a way to check how good the network performs. We denote the dataset formed by the input and labels, or the spin configurations and the corresponding magnetisation values, by $(\vec{X}, \vec{Y})$. We can then feed this dataset to the model for training and testing. The output from our network will be a set of values of the magnetisation for the spin configurations witin the test dataset over a range of reduced temperatures, T_r. If the network can predict the magnetisation to a high accuracy, we expect to see an output similar to what is shown in Fig. 6, showing the plot of the magnetisation which becomes zero at a temperature that we identify as the critical temperature, indicating that a phase transition has occured.

7.2 Tensorflow Environment

Following *Machine Learning Phases of Matter*, a Neural Network with a single hidden layer is used to train the model. The Neural Network is implemented using Tensorflow. Tensorflow is a library created specifically for machine learning and artifical intelligence models[1]. Implementing these methods from scratch would be unnecessarily difficult. Environments such as Tensorflow are created by the best computer scientists in the domain with better optimisations and structure. Thus, it is more beneficial to learn these libraries and utilise them for our own purposes.

7.3 Generate Training and Testing Data

Before we can set up our neural network and begin the training and testing, we need to generate the data to feed to the network. Our input to the network will be the spin configurations, $\vec{X}$, of the $2D$ Ising model. The spin configurations are generated by sampling from the Boltzmann distribution over a range of reduced temperatures ($T_r = 1.0 \ldots 3.5$) with a Markov Chain Monte Carlo algorithm, setting $h = 0$ and $J = 1$, and we did this is in section 4 using the Metropolis algorithm. The network will have a better performance with more data to train with so we generate 5000 spin configurations for the network. Moreover, the data is stored with an additional label which is the magnetisation of each configuration, $\vec{Y}$, in order for the network to learn the order parameter of the phase transition.
Since we want to train and later test our model, we need to split our dataset $(\vec{X}, \vec{Y})$ such that the network is tested on new data which it has not encountered during the training process. This is

done using a *train-test split* procedure, a procedure in which is used to estimate the performance of a machine learning model when it is used to make predictions on data not used to train the model. Since we have a large dataset with 5000 inputs, it is appropriate to use this procedure. We will choose to split $(\vec{X}, \vec{Y})$ by 90% for training and 10% for testing. However, we have to be careful! We have generated our spin configurations over a *range* of reduced temperature points, so we cannot just take the first 90% of configurations for training since the network will only be tested at the higher range of temperature points for which we know the phase transition would have already occured. Our network is implemented using Python, where there exists a function, called train-test split, for the train-test procedure that splits the input in an ordered way such that both training and testing will include data across the entire range of the input. This also means that the order of the training and testing datasets will be the same for each training epoch the network undergoes.
For completeness, we will generate the dataset $(\vec{X}, \vec{Y})$ for lattices sizes $L = 10, 20, 30, 40, 60$ to train and test the network for a range of lattice sizes, separately.

7.4 ANN Structure

In *Machine Learning Phases of Matter*, a fully-connected convolutional neural network, CNN, is used to classify data in a dataset from temperatures above and below T_c. A CNN is a class of ANN, more commonly applied to analyse visual imagery [22]. Since we are not dealing with images, we use a Feed-Forward Neural Network for simplicity. This means that data can only move forward in the network, from the input neurons to the hidden layer and finally to the output neuron. That being said, our neural network is relatively simple. We have one input neuron, connected to one hidden layer consisting of 100 neurons which is connected to one output neuron. The structure of the neural network is shown below in Fig. 12:

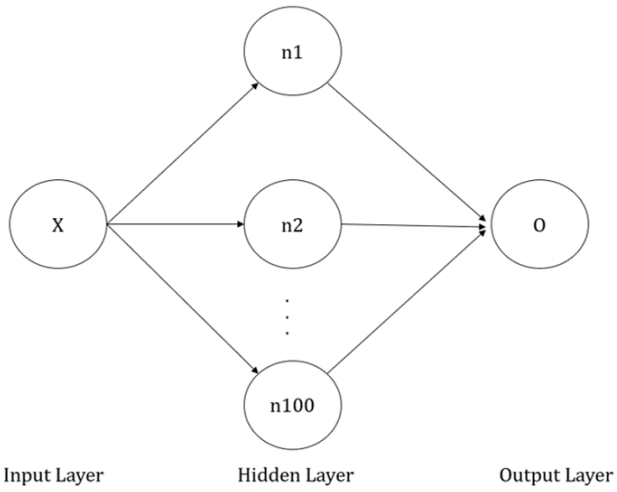

Figure 12: *The structure of our Artificial Neural Network. The ANN is shown with one input neuron, one hidden layer with* 100 *neurons and one output layer.*

The spin configurations, $x_{i=1...n}$, are fed to the network through the input neuron along with their corresponding labels, $y_{i=1...n}$, the value of the magnetisation. The inputs are connected to the 100 neurons in the hidden layer where every input has exactly one connection to each neuron in the hidden layer. The final layer contains all the possible outputs of the network. It is within the hidden layer that the training occurs. The operations performed by the neurons are pretty simple, but it is easier to first visualise this in Fig. 13 before explaining:

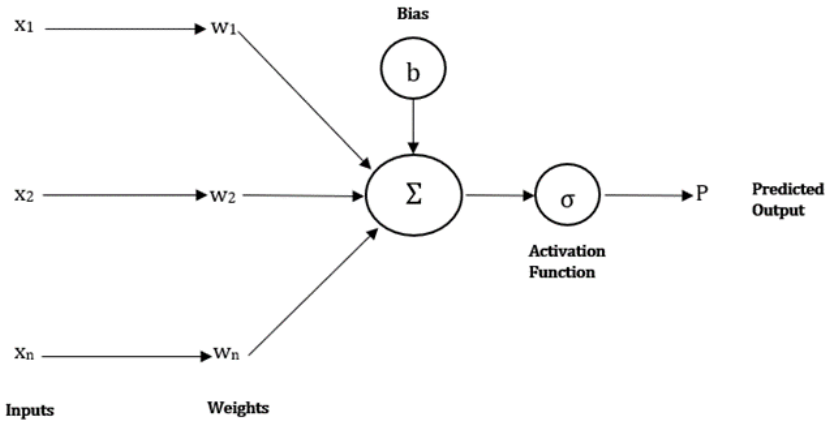

Figure 13: *An example of a neuron shown taking input values, $x_{i=1...n}$, mulitplying by weight values, $w_{i=1...n}$ and adding a bias value, b, before passing to the Sigmoid activation function to be processed as a predicted output, P.*

First, it adds up the number of neurons that it is connected to from the previous layer. In Fig. 13 there are n inputs, the spin configurations x_i, coming to the neuron, so there are n inputs connected to our neuron. These inputs are multiplied by the weights, $w_{i=1...n}$, which determines the connection between the two neurons. Each connection between neurons has its own weight value which is the only value that is modified during the learning process. Moreover, a bias value is added to the total value calculated. The bias value is not a value coming from a specific neuron and it is initialised to zero before the learning starts. This first operation can be written more mathematically via:

$$z = \vec{X} \cdot W + b, \tag{108}$$

where $\vec{X}$ is our input of spin configurations, $x_{i=1...n}$, W is a weight matrix and b is a bias value with z being the value obtained from this operation. Since we are dealing with matrix mulitplications here, we have to ensure that dimensions are consistent. This means that the number of rows of our weight matrix must equal the number of inputs, and the number of columns must be equal to the number of neurons in the next layer. Since we have n inputs and 100 neurons in the hidden layer then $W \in \mathbb{R}^{n\times 100}$. Similarly, the number of bias values should be consistent with the number of neurons in the next layer. After this matrix mulitplication and with the bias value added, the neuron will apply the sigmoid function from eq. 89 as the activation function to the obtained value z. This activation function serves to turn the calculated value z to a number between 0 and 1 before passing this value to the next layer. We will denote this value by P for the predicted value. This is all the operations within a neuron before passing P to the next layer.
P is an important value for calculating the loss function (commonly called the cost) of the network which is used to indicate how well the parameters of the network, the weights and bias, perform in giving the desired output. These are the parameters we wish to optimise. The loss function of our network, LF, is given by the mean-squared error:

$$LF = \frac{1}{n}\sum_{i=1}^{n}(y_i - P_i)^2, \tag{109}$$

where n is the number of inputs, y is the true label (magnetisation of each configuration) for each input and P is the predicted label (predicted magnetisation). We expect the loss function to decrease throughout the learning process as the network becomes more familiar with the data and better at predicting the desired output.
Since this is an optimisation problem, in order to optimise the weights and bias of our network, we need an optimiser. We have seen in Section 6 that the Gradient Descent Method is a good optimsation method for supervised regression tasks. Our neural network uses Adam optimisation[11] which is an optimisation algorithm for stocastic gradient descent. We will not discuss how Adam optimisation is

different from classical gradient descent but just state that it is a variation of our gradient descent method. The weights and bias are optimised in a way such that P is closer to the desired output from the neuron.
There is one final parameter which controls how the neural network learns called the learning rate. This value determines the speed at which the network learns. More specifically, the learning rate determines how the weights should be modified: in little steps or bigger steps. We will set our learning rate to a small number, 0.001, which allows our network to modify the weights little-by-little for a more accurate output.
The last neuron is the output neuron which contains all the possible outputs which is the magnetisation of the spin configurations x_i.
This whole process is first trained on the training dataset, $(\vec{X}, \vec{Y})_{train}$, and then tested on the test dataset, $(\vec{X}, \vec{Y})_{test}$. $(\vec{X}, \vec{Y})_{train}$ is used for the learning process of our network using 5000 training epochs such that the network can recognise patterns within the data and learn how to predict the magnetisation for each input, $(x_i)_{train}$. Once the training is finished, the network is tested on $(\vec{X}, \vec{Y})_{test}$ which is new data for the network. The network can no longer rely on recognising definite patterns, however, it has learned to predict the magnetisation within the learning process and so is able the predict the magnetisation for each $(x_i)_{test}$ which can be compared to the true value $(y_i)_{test}$.

7.5 Results from Training the Network

With the dataset that we have generated using the Metropolis algorithm, and after splitting this dataset into training and testing, we can test our Neural Network with the hope that it can predict the magnetisation as the relevant order parameter, consistent with the results from *Machine Learning Phases of Matter*. From this, we can explore whether ANNs, in general, are sufficient to make predictions on such a model as the Ising model.
The output of our ANN is plotted as a function of the temperature for lattice sizes, $L = 10, 20, 30, 40, 60$ with error bars showing one standard deviation of the output, shown in Fig. 14:

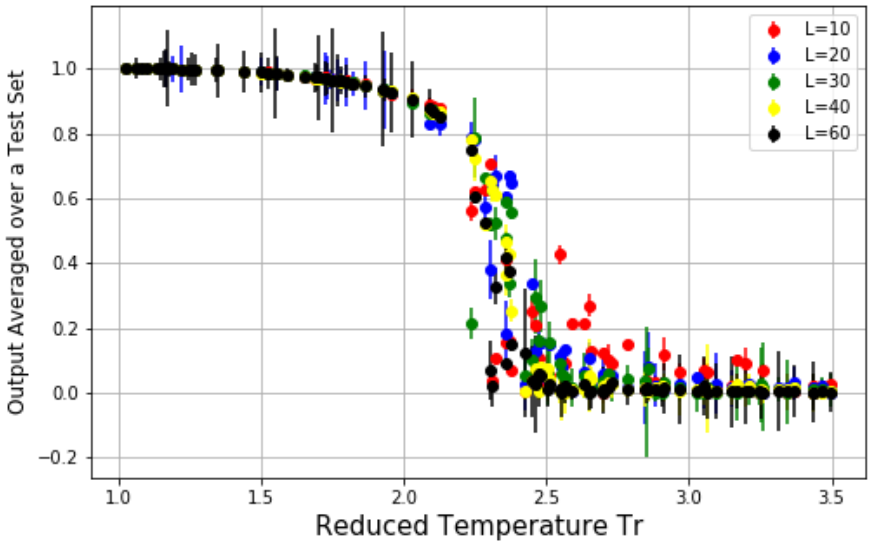

Figure 14: *Output of the ANN averaged of a test set plotted as a function of the reduced temperature, T_r, for linear system sizes $L = 10, 20, 30, 40, 60$ with error bars showing one standard deviation of the output.*

We can see that the neural network is able to learn the order parameter of the phase transition with high accuracy. In other words, the neural network is able to distinguish the two phases of the model: before the phase transition and after the phase transition. Since we know that the magneti-

sation with zero external electric field, $M(h = 0, T)$, of the Ising model is zero for $T_c < T$, it is clear from Fig. 14 that the neural network is able to correctly identify the magnetisation as the relevant order parameter.
We can investigate the performance of our model in many ways but the most common is to plot a loss curve to show how the loss function changes as the network trains using the training dataset and also when testing the network on a different dataset. Using a loss curve, we can visualise how the loss changes for both the training and testing data as the neural network undergoes the learning process. A loss curve gives us a snapshot of the training process and the direction in which the network learns. For us to visualise how the network learns, we can plot the training loss for system sizes $L = 10, 20, 30, 40, 60$, as shown in Fig. 15:

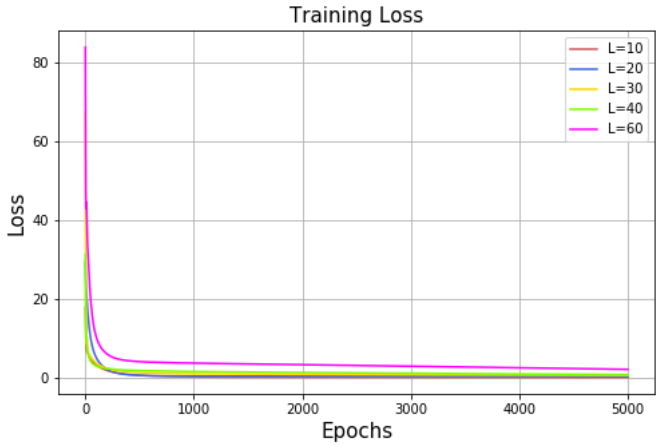

Figure 15: *Training loss of the ANN plotted against the number of training epochs for system sizes* $L = 10, 20, 30, 40, 60$*, respectively.*

It is clear from the loss curve during training that the loss function decreases rapidly as a function of the number of training epochs. This is a good sign, indicating that the network is excelling at learning how to predict the magnetisation of the spin configurations during the learning process. However, to see how good our network is at these predictions, we need to test the network on the testing dataset. To give us more insight to the performance of the network, we can plot the validation loss, the loss curve for the testing dataset, as a function of the number of epochs in Fig. 16:

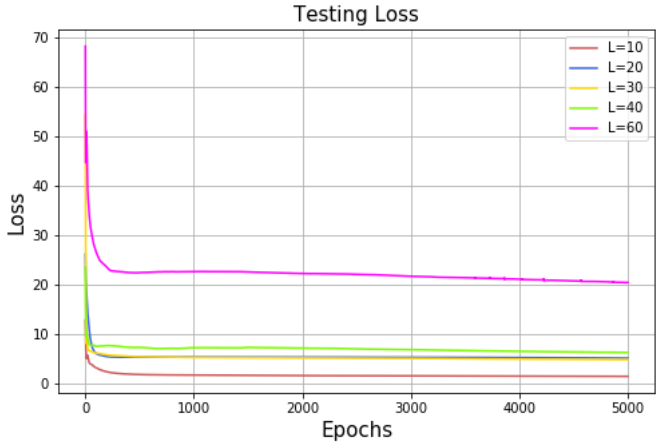

Figure 16: *Validation loss of the ANN plotted against the number of epochs for system sizes* $L = 10, 20, 30, 40, 60$*, respectively.*

We see that the loss curve decreases for both the training and validation set. Both curves indicate that the network has a good learning rate, however, the network clearly performs better on the training set, data that it has learned from, than on the validation set, completely new data for the network, mostly for the larger lattice sizes $L = 40, 60$. This is a slight case of *overfitting* showing that the model struggles to generalise to new data. In overfitting, the model tries to learn too many patterns within the training data, so much that it almost memorises this dataset. When faced with a new dataset, the model cannot determine the same patterns or details and often gives wrong predictions. This is a common machine learning problem.
We want to have a **Good Fit Model**; a model that suitably learns the training dataset and generalises well to the old out dataset. To do this, we must try to avoid overfitting. The two ways we could do this is by:

1. Reduce overfitting by training the network on more examples.

2. Reduce overfitting by changing the complexity of the network.

A benefit of neural networks is that their performance continues to improve as they are given larger datasets. A larger dataset would be time-consuming, computationally-expensive and would cause an eventual plateau in the network learning since the network does not have an infinite capacity. This would not be the most efficient way of improving the performance of our network. Instead, we should change the complexity of our model, in terms of neurons, layers and network parameters such as the weights and bias. We can reduce the complexity of the model to reduce overfitting by:

1. Changing the network structure (the number of weights).

2. Changing the network parameters (the value of the weights).

We have already implemented the second point by penalising the weights by $L2$ regularisation, so instead, we change the number of the weights to prevent overfitting. For example, the structure of the network could be tuned by searching for the optimable number of layers and neurons. This would tune the number of weights to a size appropriate for the network which would reduce or remove the problem of overfitting.
Although we have these methods to reduce overfitting in mind, it's worth noting that our model does perform well, even with a small amount of overfitting, correctly identifying the magnetisation of the $2D$ Ising model as the relevant order parameter, consistent with the results from [5].

7.6 Distribution of Results

To further understand the output of our Artificial Neural Network and show how well the network performs, it is a good idea to show how the output is distributed. As we have already said, we can compute the magnetisation of a spin configuration by hand and compare this value to the predicted value of the ANN. It would be interesting to investigate how the distribution of the predicted magnetisation from our ANN compares to the distribution given by data sampled from the Metropolis algorithm.
In order to gather the data, we generate 1000 spin configurations for a lattice size $L = 10 \times 10$ using the Metropolis algorithm. We do this for three different fixed reduced temperatures; a small reduced temperature $T_r = 1.5 < T_c$, a reduced temperature close to the critical temperature $T_r = 2.2 \approx T_c$, and for a large reduced temperature $T_r = 2.9 > T_c$. This allows us to explore how the distribution of the data changes for different values of the reduced temperature.
Similarly, we do the same for the output of our ANN. We generate spin configurations for lattice size $L = 10 \times 10$ for the same fixed reduced temperature values and feed this to the network for training. The ANN is then tested and the output of the network is saved in order to make a plot of the distribution of the output. The output of the ANN and the magnetisation values from the spin configurations generated by the Metropolis algorithm are visualised using a histogram for each individual reduced temperature value for a lattice of size $L = 10 \times 10$, shown in Fig. 17:

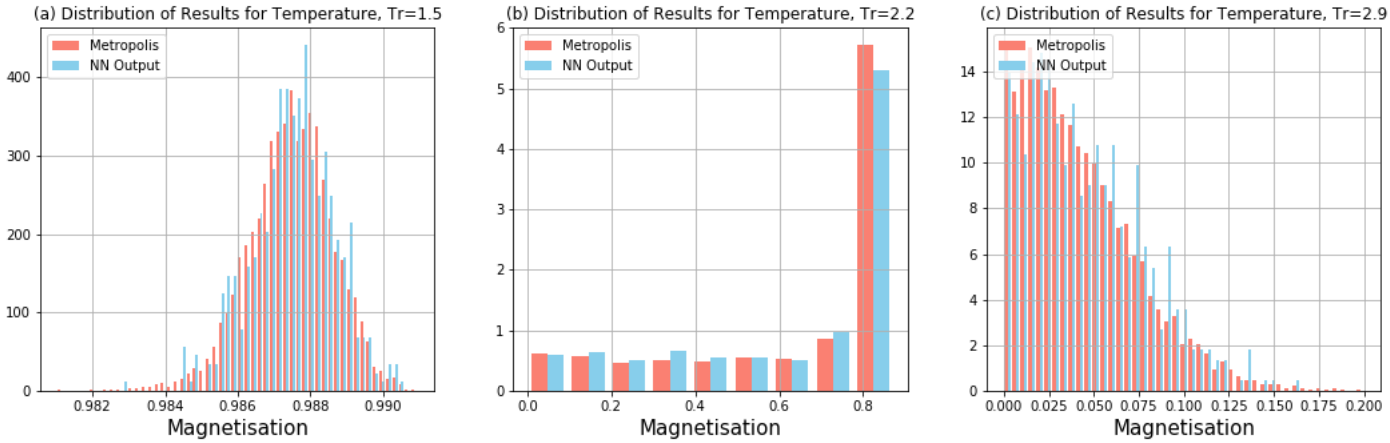

Figure 17: *The comparison of distributions from the output of the ANN, represented by blue bars, and from the Metropolis algorihtm, represented by pink bars, for a lattice size* $L = 10$ *with* $h = 0$ *and* $J = 1$*. The* x*-axis shows the magnetisation value and the* y*-axis shows the frequency of data within each bar. (a) shows both distributions of the magnetisation for a small temperature,* $T_r = 1.5$*. (b) shows the distributions for a temperature close to the critical temperature,* $T_r = 2.2$*. Finally, (c) illustrates the distribution of the magnetisation for a large temperature,* $T_r = 2.9$*.*

It is clear from Fig. 17 that the distribution of the magnetisation for each reduced temperature value, T_r, is very similar for both the output from the ANN and the data generated by the Metropolis algorithm. Both distributions for each reduced temperature value show what we expect to see. We expect the magnetisation to be close to 1, $M(h = 0, T) \sim 1$, for a small temperature which is shown in (a) with the frequency of data within this plot being relatively dense. Around the critical temperature, $T_c \approx 2.269185$, the magnetisation goes to zero, $M(h = 0, T) \to 0$, in which (b) doesn't show a concrete pattern in the distributions. For large temperatures, we know that the magnetisation is 0, $M(h = 0, T) = 0$, which we can evidently see in (c), since the distributions are positively skewed towards 0.
Hence, it seems that the ANN is not only able to reproduce the mean value of the magnetisation but also the actual distribution of the data. We can confidently say that our ANN performs to a high accuracy in determining the magnetisation as the order parameter.

8 Final Remarks

This project has covered a vast range of subjects, from thermodynamics to Monte Carlo simulations and machine learning, at which I have learned and applied many notions on mathematics, physics and machine learning. In section 2, we introduced important and fundamental concepts of statistical mechanics which enabled us to derive the Boltzmann distribution. Following this was section 3 where we introduced the Ising model and its macroscopic properties. We solved the Ising model in one dimension by deriving the Partition function, Z_n, allowing us to compute the free energy, magnetisation and the specific heat. From the free energy, we concluded that the one-dimensional Ising model does not demonstrate a phase transition due to the magnetisation being an analytical function. However, the two-dimensional model does exhibit a phase transition at a critical temperature, T_c where we stated Onsager's solution in section 3.7.

To follow, we introduced a Markov Chain Monte Carlo method, namely the Metropolis-Hastings algorithm, to simulate the $2D$ Ising model by plotting the macroscopic properties as a function of the reduced temperature, T_r. The significance of the MCMC simulations is that they show the two phases of the model, a ferromagnetic and a paramagnetic phase, thus deducing that the $2D$ Ising model undergoes a phase transition.

In section 5, we introduced machine learning and categorised our learning problem as a supervised, regression task. This lead to an important discussion of Neural Networks and an important method of optimisation called the Gradient Descent Method. The data sampled using the Metropolis-Hastings algorithm was used as an input for our neural network in section 7 where our results from the network did, indeed, agree with *Machine Learning Phases of Matter* in identifying the magnetisation of the $2D$ Ising model as the order parameter.

That being said, there is indeed potential work and further exploration of the title of this project which can be done. Similar to the set-up of *Machine Learning Phases of Matter* published in Nature, our ANN could be trained on the square Ising model in $2D$ as shown in Section 7.5 and then tested on triangular Ising configurations in order to see whether our ANN works for the triangular case. This is an example of transfer learning. This would mean that our ANN would store the knowledge gained while predicting the magnetisation of the square-lattice configurations and apply or *transfer* this knowledge to a different but related problem: the triangular-lattice configurations. If I was to continue this project, the goal would be to investigate if transfer learning works for the Ising model with square and triangular configurations. To conclude, I would like to thank my supervisor, Dr. Vincent Drach, for his guidance, support and advice throughout this project.

References

[1] Martín Abadi. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

[2] James A Anderson. *An introduction to neural networks*. MIT press, 1995.

[3] R. J. Baxter. *Exactly Solved Models in Statistical Mechanics*, pages 5–63.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[5] Juan Carrasquilla and Roger G Melko. Machine learning phases of matter. *Nature Physics*, 13(5):431–434, 2017.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[7] Olle Häggström. Finite markov chains and algorithmic applications. 2002.

[8] Jin Hanmin and Terunobu Miyazaki. *The Physics of Ferromagnetism*, volume 158. 01 2012.

[9] Kerson Huang. *Statistical Mechanics*. John Wiley & Sons, 2 edition, 1987.

[10] Bruria Kaufman. Crystal statistics. ii. partition function evaluated by spinor analysis. *Phys. Rev.*, 76:1232–1243, Oct 1949.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[12] Dirk P. Kroese, Tim Brereton, Thomas Taimre, and Zdravko I. Botev. Why the monte carlo method is so important today. *WIREs Computational Statistics*, 6(6):386–392, 2014.

[13] Miroslav Kubat. *An Introduction to Machine Learning*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[14] Lev Davidovich Landau and Evgenii Mikhailovich Lifshitz. *Statistical Physics: Volume 5*, volume 5. Elsevier, 2013.

[15] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087–1092, 1953.

[16] Elliott W. Montroll, Renfrey B. Potts, and John C. Ward. Correlations and Spontaneous Magnetization of the Two-Dimensional Ising Model. *Journal of Mathematical Physics*, 4(2):308–322, February 1963.

[17] Ole G. Mouritsen. Introduction to phase transitions and critical phenomena. by h. eugene stanley, oxford university press, oxford, 1987. *International Journal of Quantum Chemistry*, 35(4):583–584, 1989.

[18] Michael A. Nielsen. Neural networks and deep learning, 2018.

[19] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65:117–149, Feb 1944.

[20] R. Peierls. On ising's model of ferromagnetism. *Mathematical Proceedings of the Cambridge Philosophical Society*, 32(3):477–481, 1936.

[21] Christian P. Robert. The metropolis-hastings algorithm, 2015.

[22] M.V. Valueva, N.N. Nagornov, P.A. Lyakhov, G.V. Valuev, and N.I. Chervyakov. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation*, 177:232–243, 2020.

[23] Wenlong Wang, Rogelio Díaz Méndez, and Raudys Capdevila. Solving the one-dimensional ising chain via mathematical induction: An intuitive approach to the transfer matrix, 07 2019.