Bachelor thesis

Pattern recognition in the 2D-Ising model with neural networks

Mustererkennung im 2D Isingmodell mit neuronalen Netzen

Author: Lukas Holzbeck

Supervisor: Dr. Karin Everschor-Sitte

Second corrector: Prof. Dr. Friederike Schmid

Group: TWIST

vorgelegt dem Fachbereich Physik, Mathematik und Informatik (FB08) der Johannes Gutenberg-Universität Mainz am 22. Juli 2019

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Mainz, den 22. Juli 2019 [Unterschrift]

Lukas Holzbeck
Gruppe: TWIST
Johannes Gutenberg-Universität D-55099 Mainz
lholzbec@students.uni-mainz.de

Abstract

We studied the behavior of magnetization of a 2D Ising model around the critical temperature where it changes its phase from disorder to ordered and vise versa. Since this critical temperature is strongly depending on the coupling constant, we simulated different coupling parameters for different regions of the lattice in order to test recognition methods.

First we use calculated the critical temperature by identifying the ordered and disordered phases with different temperatures. We studied the dependence of the critical temperature with the coupling parameters. We wanted to study systems where the lattice is divided in regions with different coupling constants. Then we trained a neural network on the data to achieve a learning after which the Deep Neural Network was able to recognize the different regions in the data. The recognition was based on the magnetization profile after a certain step interval in the Monte Carlo simulation of the 2D Ising model.

We were able to show that the neural networks used were able to identify the different coupling constants in the data resulting in a clear distinction between the regions.

Contents

1 Introduction

Physicists need methods to find underlying structures in data obtained from experiments and simulations.Neural networks are prominent tools for picture recognition and analysis of large data sets.They provide new prospects for data analysis in physics and other sciences. Neural networks have been used with success to solve various problems in fields like Astrology [1], image recognition [2], fluid dynamics [3]and for Ising spin systems [4]. It was shown [5] that it is possible to train networks on recognizing phase transition curves from data. In summary it is possible to train neural networks to find pattern representing physical properties in data without lengthy calculations nevertheless the training process of the neural network takes time. This Bachelor thesis is about pattern recognition of data with thermal fluctuations from the 2D-Ising model for a square lattice using deep neural networks. The 2D Ising model is a simplified system that can approximate interactions in spin systems. The 2D Ising model consists of spins on a square lattice which are either plus or minus one. The system will start disordered and will order when introduced to a infinite heat bath below the ordering temperature. The simulation used in this work was a Monte Carlo simulation with replica exchange. The transition point of the 2D Ising model was determined. The Deep Neural Network that was used in this thesis was a feedforward neural network. The magnetization curve for high and low temperatures was also simulated and displayed. It was possible to detect square substructures, with a different coupling constant J, in the square lattice after training the neural network.

The thesis is structured as follows. In Sec.II we will discuss the theory behind the 2D Ising model including exact solutions for the square lattice. We will also introduce the Monte Carlo algorithm with replica exchange and the basic principles behind neural networks. In Sec.III.1 we will discuss the simulation results using the Monte Carlo algorithm with replica exchange for the 2D Ising model and the obtained magnetization curve. In Sec.III.2 we will discuss the training and the results of the trained Deep Neural Network.

2 Theory

2.1 2D-Ising model

In this thesis we consider the Ising model in two dimensions. The Ising model was introduced in 1920 by Wilhelm Lenz to describe ferromagnets in a simplified model. It was first solved for the one dimensional case by E.Ising [6]. The Ising system consists of a square lattice with a specific size N = LxL. On each lattice point lies a point particle with a single integer value which is either +1 or -1 representing the spin at that site. σ is the representation of the spin value at each lattice point.

Integer values are a good representation of spins in some physical materials like spin glasses. Spin glasses have a disordered and a ordered phase for different temperatures1. When the system is at a high temperatures the system is not ordered. When such a system is introduced to a infinite heat reservoir with $T < T_c$, the system will reach the thermal equilibrium after a certain amount of time. When the system reaches the equilibrium all of its spins will be aligned in the same direction. This is due to the coupling between the spins and the reduction of the thermal noise. The interactions between the spins can be described in first order as interactions between the nearest neighbors on the lattice. The Hamiltonian for the Ising model is:

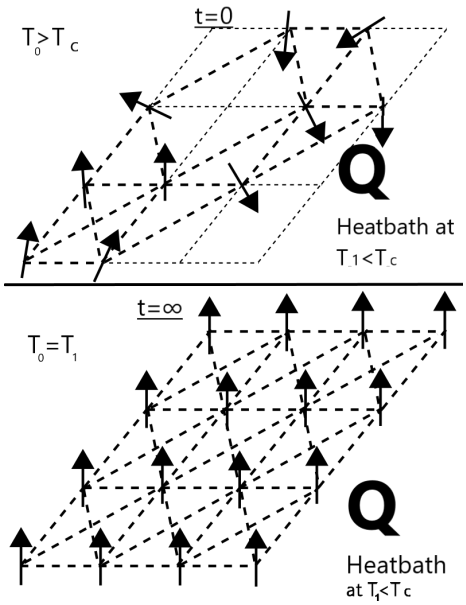

Figure 1: Spin glasses for different temperatures. Image changed from Ref.[7]

The electrons in spin glasses behave for low temperatures like point particles with a spin-half pointing orthogonal out of the lattice plane.

$$H = -J \sum_{i,j} \sigma_i \sigma_j - \mu \sum_j h_j \sigma_j \tag{1}$$

In our case we can neglect the second term because we set the magnetic field $h = 0$ which results in a simplified Hamiltonian. μ is the magnetic moment of

the particles, J the coupling constant between the spins and σ_i the value for the spin at the position i and σ_j the value for the spin sitting on site j.

$$H = -J \sum_{i,j} \sigma_i \sigma_j \tag{2}$$

The Hamiltonian for the Ising model is given by the sum over all pairs of adjacent spins. The energy of the model follows subsequently as:

$$E = \frac{1}{2} \sum_{i,j} H_{i,j} = -\frac{1}{2} J \sum_{i}^{N} \sum_{j,nn} \sigma_i \sigma_j \tag{3}$$

J is the coupling constant that represents the strength of the coupling between the spins on the lattice. The second sum sums over the four nearest neighbors of the spin at the site i. A interaction with a positive coupling constant J is called ferromagnetic. Interactions with a coupling constant smaller zero are called antiferromagnetic and coupling constants of zero lead to no interaction. The energy in the system will tend towards equilibration for low disturbances. For no disturbing factors the systems will reach the ferromagnetic or antiferromagnetic state.

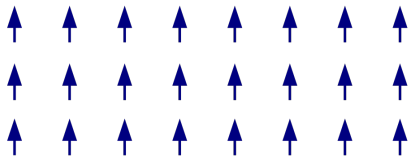

Figure 2: Ferromagnetic ordering for $J > 0$. Image reproduced from Ref.[8]

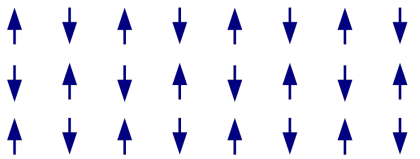

Figure 3: Antiferromagnetic ordering for $J < 0$. Image reproduced from Ref.[9]

In this thesis we will only consider ferromagnetic systems which means we only allow positive values for the coupling constant J. The total magnetization of the system can be determined by calculating the sum over all the spins in the system and dividing by the total number of spins. The formula for the energy of the system is :

$$M = \frac{1}{N} (\sum_{i=1}^{N} \sigma_i) \tag{4}$$

M is the magnetization of the system and N the total number of spins in the system. σ_i is the spin at the position i.

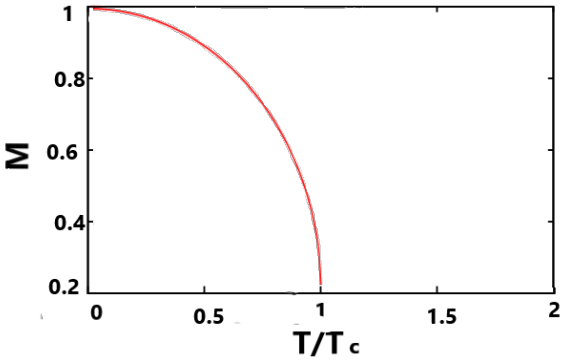

Figure 4: Theoretical curve for the modulus of the magnetization of the 2D Ising model. Image reproduced from Ref. [10]

The model can be used to model physical spin systems because spin glass systems can be reduced to first order interactions between nearest spins on a square lattice with spins in positive or negative direction orthogonal to the lattice plane.

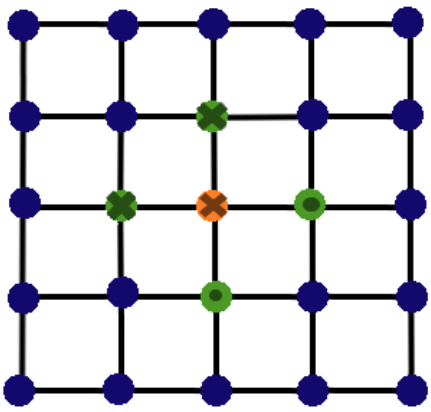

Figure 5: Next neighbor spins in green which interact with the center spin in orange. Interactions between the blue spin and the red spin are neglected in the Ising model due to only considering nearest-neighbor interactions

By considering disturbing forces the system gets more complicated and the problem has no longer a known solution. Local spins can flip when the system sits in a heat bath at a finite temperature (real systems). Flips of a local spin can be caused by thermal fluctuations. This is considered in the flip probability:

$$p_{\text{flip}} = e^{-\frac{\Delta E}{k_B T}} \tag{5}$$

p_{flip} describes the probability of a spin to flip because of thermal fluctuations in the system because we assumed the system to be canonical. ΔE describes

the energy difference between the original state and the energy for the same configuration but with a flipped spin in the position i. k_B is the Boltzmann constant and T the temperature in Kelvin. The exponential factor resembles the Boltzmann distribution which describes the system when it reached thermal equilibrium with the heat bath at the temperature T. Therefor the flipping probability depicts the chance of random fluctuation of spin due to thermal noise for temperature T > 0K.

For the 2D-Ising model we chose a square lattice with a total number of N = L x L spins. When the lattice is finite there will be finite size effects in the simulation. To reduce the finite size effect of smaller systems, one can use periodic boundary conditions. A visual explanation for the periodic boundary condition can be found in (6).

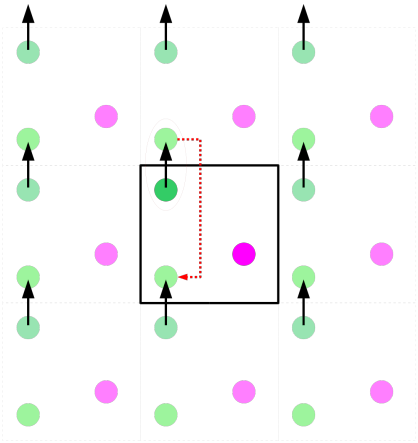

Figure 6: Shows the interaction for a lattice with periodic boundary condition(black box) with its periodic continuation. Image reproduced from Ref.[11]

Sloving the 2D Ising model lead to a single second order phase transition at the critical temperature T_c which was shown by Onsager [12]. When we just look at the two dimensional Ising model, we obtain a second order phase transition at a critical Temperature T_c. This critical temperature depends on the strength of the coupling constant J and on finite size effects when considering a finite square lattice. For temperatures below the critical temperature T_c the system is relaxed which means almost all of the spins point in +1 or -1 direction(15). For temperatures much higher than T_c, the spins in the system fluctuate mostly random without any alignment(17).

2.1.1 Exact Solution for the 2D-Ising model and Finite Size Effects

The 2D-Ising model was first solved by Onsager in 1944 for the infinite square lattice which means we know the value for the critical temperature for the case of the infinite lattice [12].

Since we have a canonical ensemble (N,V,T constant) one can start with the free energy of the system.

$$F = -\beta \ln(Z) \tag{6}$$

The partition function Z is the sum over all spin configurations.

$$Z = \sum e^{-N\beta H} \tag{7}$$

With the Hamiltonian $H = -J\sum_{i,j}\sigma_i\sigma_j$. N is here the total number of spins in the system and β the Boltzmann constant. The Onsager solution for a square lattice Ising model with periodic boundary condition was calculated for the limiting case of $N \to \infty$.

$$F(\beta) = -\frac{\ln(2)}{2} - \ln\left(\cosh(2\beta J)\right) - \frac{1}{2\pi}\int_0^\pi \ln\left(1+\sqrt{1-K^2\cos^2(\theta)}\right)d\theta \tag{8}$$

$$K = \frac{2\sinh(2\beta J}{\cosh^2(2\beta J)} \tag{9}$$

K is a factor resulting from the derrivation of the free energy formula. With K=1 (knowing that there is only one phase transition) the critical temperature can be obtained by solving the equation for β.

$$\beta_{\text{ONS}} = \frac{1}{2J}\left(\ln(1+\sqrt{2})\right) \approx \frac{0.4407}{J} \tag{10}$$

[12] N is here the total number of spins in the system. The results for the finite size model were calculated by I.M. Karandashev [13]. He calculated the free energy of the system with the Kasteleyn-Fisher algorithm. This algorithm is based one the calculation of the determinant of the transition matrix representing the finite 2D square Ising model. Calculations in the paper[14] led to the approximation for the free energy for large values of β(asymptotical behavior of the free energy).

$$F \approx -2\beta J\left(1-\frac{1}{\sqrt{N}}\right) \tag{11}$$

The correct values were calculated only for lattices sizes of L=25, L=50 and bigger square lattices. The approximation for β was derived by fitting the calculated results for the different lattice sizes([13]page 3-5).

$$\beta_c \approx \frac{\beta_{\text{ONS}}}{J}\left(1+\frac{5}{4\sqrt{N}}\right) \tag{12}$$

2.2 Simulation

2.2.1 Monte Carlo Simulation

A Monte Carlo Simulation is a simulation that relies on random picked numbers that are then checked for specified interaction criteria. With the random behavior of the Monte Carlo method it is possible to approximate complicated behavior of dynamical systems. At the beginning of the algorithm stands a random input which can then be checked for certain behavior. This process is repeated multiple times to ensure that every possible point that can be picked was picked and checked multiple times by the algorithm criteria. This results in a good representation of a time evolving system. It consists of a random number generator and a threshold value n. At the beginning there is a random point picked, which than can be fed into a following algorithm. After the following algorithm is finished, the changes will be saved and the loop will start again until it ran n times.

2.2.2 Metropolis Algorithm

The advanced Monte Carlo algorithm that was used for the simulation was a Metropolis Algorithm. The algorithm was inspired by the work from Jacques Kotze ([15]).

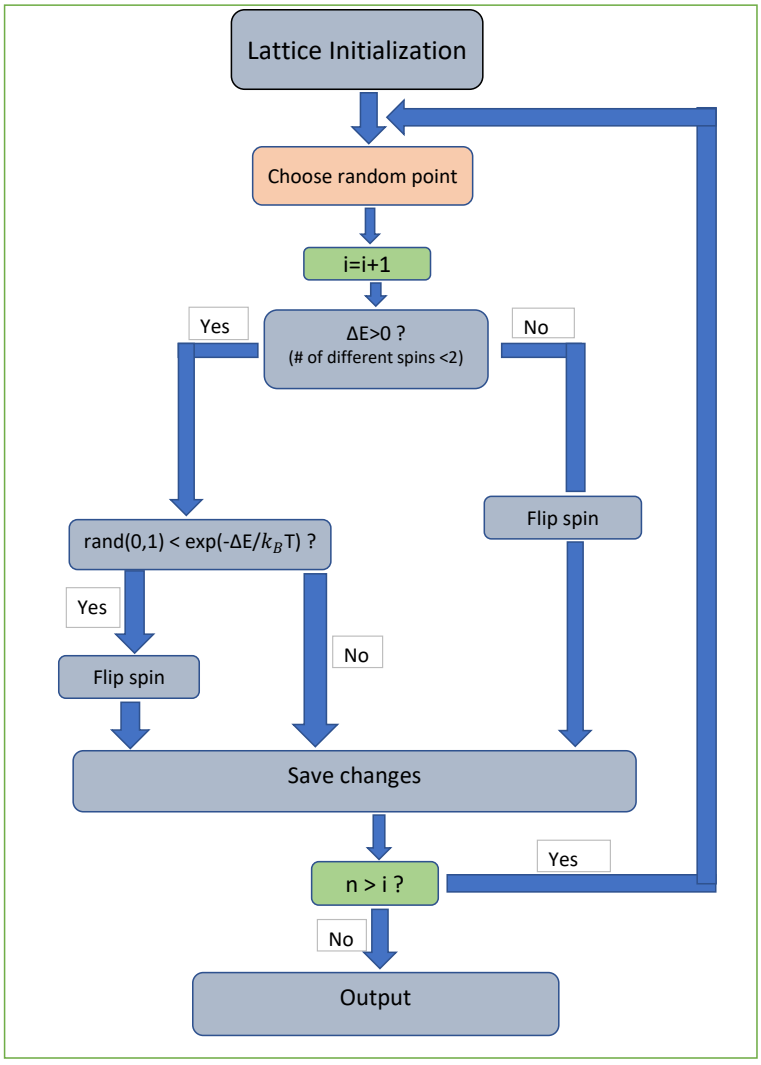

Figure 7: Flowchart of the Metropolis Algorithm

The Metropolis algorithm receives an input which is in our case a square

lattice with random polarization directions orthogonal to the lattice planes resembled by the values +1 and -1 and the temperature T for a infinite thermal reservoir around the lattice. A point on the lattice will get randomly picked. The energy difference between the spin at the site and the flipped spin at the same site is calculated. When the energy difference ΔE is smaller or equal to one, which means the flipped state results in a energetically more stable state, the spin gets flipped. When the energy is positive then spin gets flipped when a random number from 0 to 1 (random probability) is lower than the Boltzmann distribution for the energy difference. This resembles the influence of thermal noice caused by the heat bath. These changes are then saved and other random lattice points are picked and checked until the total number of steps is reached and the final lattice is printed out. A good random number generator and a large step number n are required to ensure that every lattice-point was checked multiple times during the Metropolis algorithm. For systems that rely purely on random variables, the Metropolis Algorithm is a good approximation if the number of Monte Carlo steps n converges towards infinity. This argument is based on the central limit theory which states that a large number of independent random variables n form a normal distribution when summed and normalized even if the initial variables are not normally distributed. This means that in limit of large numbers of variables the results will tend towards a normal distribution of the correct result. The size of the lattice has to be high enough to resemble a periodic lattice without border effects like clustering on the borders or substructures. Nevertheless every point in the large lattice has to be reached multiple times for the system to reach its equilibrium state(n needs to be high enough). Under these circumstances the Metropolis Algorithm leads to accurate results for systems like the 2D-Ising model (if the temperature is under T_c) to a relaxed equilibrium state.

2.2.3 Replica exchange

Most systems take plenty computational time to reach the equilibrium state. This is often caused by the existance of metastable states. Metastable states can be described as a local minimum in the energy landscape of the system. If this minimum is low enough the probability for the system to travel over the metastable energy barrier will be very small thus leading to longer computation times to reach the equilibrium state.

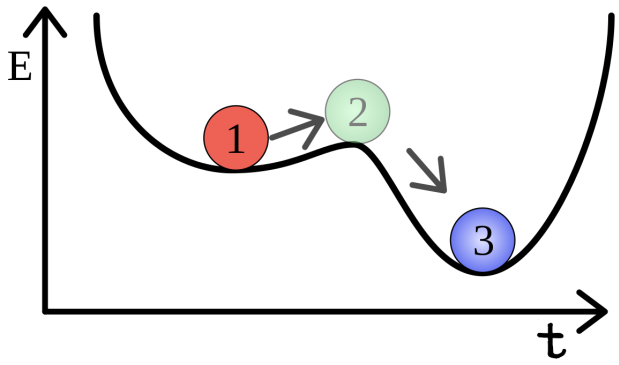

Figure 8: Picture of a system with one metastable state. Image reproduced from Ref. [16]

The key factor in the lifetime of metastable states is, in case of the Ising Model, the temperature. Low temperatures will result in lower numbers of thermally flipped spins so the system can reach metastable states more often. With larger thermal fluctuations, the energy of the system will fluctuate more frequently. The energy barriers are crossed at an increasing rate when the temperature increases. Metastable states are caused by random areas with imbalanced numbers of spins with the same polarization. This happens randomly because of the initialization of the lattice or the random picks in the algorithm steps.

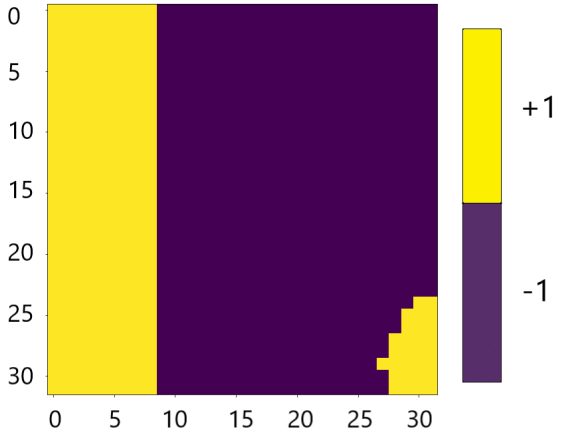

Figure 9: Picture of simulation data in a metastable state at $T = \frac{0.2}{k_B}$. The temperature of the heat bath is so low that the disturbance by thermal noise can be neglected thus the lifetime τ of such a state would be $\tau \to \infty$

The focus of Replica Exchange is to calculate the transition probability of Boltzmann distributed systems for neighboring temperatures. Meaning one calculates the uncertainty for which the system can occur with the same energy

but a heat bath at a different temperature. When the temperatures are close enough to each other, the probability density functions meaning the Boltzmann distributions of both neighboring systems overlap. The overlap can be seen as a probability of the system to exist in a neighboring thermal bath with a different temperature.

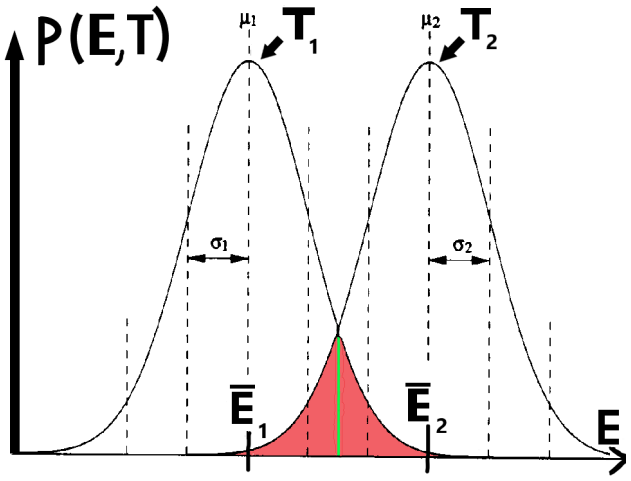

Figure 10: Overlap probability densities for two neighboring temperatures. Image edited from Ref. [17]

The green line in the graphic shows the point of equal probability for a energy to happen in systems under temperatures T_1 and T_2. For a energy state measured in system T_1 that lies right of the green intersection line, the probability of the energy state is more likely to happen with a heat bath at the different temperature T_2. When this case occurs in the simulation the neighboring systems heat baths are interchanged. This results in the system previously having the temperature bath at T_1 now having the temperature T_2. So when replica exchange is introduced, one needs a system with different temperatures running simultaneously through the algorithm. After a certain number of steps in the algorithm(can be seen as a time-step) the transition probability between neighboring systems will be calculated and the interchange of temperatures accepted or neglected. If the transition is accepted, the temperatures for both systems will be interchanged. This is checked alternately for all odd and even replicas to prevent systems from traveling over more than one temperature at once. The criterion for a replica exchange in a system that depends only on the previous state with no memory of its past(Markov Chain like) model can be calculated with the Metropolis-Hastings criterion.

$$p(i \to j) = \min\left(1, \exp\left(\frac{E_i - E_j}{\frac{1}{k_B T_i} - \frac{1}{k_B T_j}}\right)\right) \tag{13}$$

The equation is taken from ([18]). E_i is the energy in the system i. E_j is the energy in the neighboring system j. T_i and T_j are the temperatures of the

systems i and j. The formula is derived from two Boltzmann factors.

$$\frac{e^{-\frac{E_j}{k_B T_i}-\frac{E_i}{k_B T_j}}}{e^{-\frac{E_i}{k_B T_i}-\frac{E_j}{k_B T_j}}} = \exp\left(\frac{E_i - E_j}{\frac{1}{k_B T_i}-\frac{1}{k_B T_j}}\right) \tag{14}$$

This criterion ensures that the overlap between the probability density functions is sufficiently high to ensure an equal or higher probability for the state to exist in the neighboring temperature. To obtain good results for the whole energy landscape, the overlap and the number of total replica exchange steps has to be high enough to allow every replica multiple to reach every temperature multiple times.

2.3 Neural networks

Neural networks like those used in this thesis are based on the neural connections in the human brain. The human brain consists of neurons and synapses. Neurons have inputs and outputs, which are synapses(axons and dendrites) that connect the neuron cells to one another and can propagate input signals over multiple neurons.

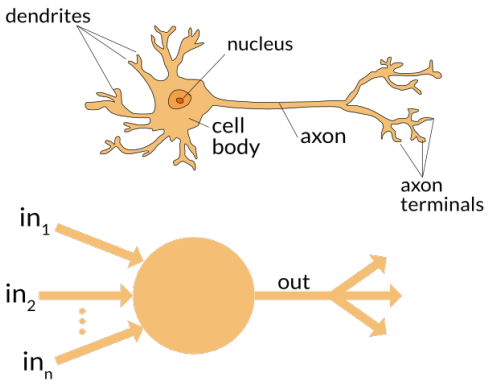

Figure 11: Human neuron and axon connection in the human brain(upper picture) and simplified neural connections in the human brain(lower picture). Image reproduced from Ref.[19]

In the human brain, the inputs(currents) get propagated with dendrites to the cellular body. If the combined current from all the dendrites reaches a specific threshold, the neuron cell will fire a current that is then sent through the axon and reaches the connected neurons.

In a simplified picture we can reduce the cellular body to a single node now called the neuron. This single node can be implemented on a computer with a function that receives inputs and calculates an output. When implemented on a computer the connection strength between the neurons can be described by a single weighted input channel. The threshold at which the neuron fires a signal is when implemented on a computer normally represented by a nonlinear function with a large gradient. This function is an approximation of the influence of the propagation of currents in the human brain. It describes the strength of the

output current for the neuron. In nature, the activation function is normally a step function which means either the neuron is firing or not.

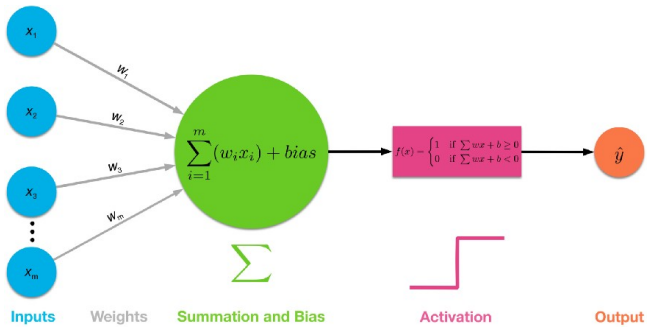

Figure 12: A neural network with all steps. Image was taken from Ref.[20]

The picture shows a neural network with many inputs(blue) and the accompanying weights w that determine the contribution strength to the neuron. In the next step the weighted inputs will be summed up and put into a activation function f(x). The result of the activation function is the output y.

2.3.1 Deep Neural Networks

Deep Neural Networks are in principle similar to normal neural networks. The difference between both lies in the size. Deep Neural Networks consist of one input layer, one output layer and multiple hidden layers which all consist out of multiple neurons. Hidden layers are layers without any output or direct data input. This means they are acting like a black box because there is no direct information on what their effect on the input data is. Another difference to normal Neural networks is the number of neurons per layer. Simple neural networks consist of a couple of neurons while large DNN consist of hundreds or thousands of neurons per layer. For many years Deep Neural Networks were not able to be trained because the training of the weights in the network took too long. This only changed in recent times due to the progress in cheaper and faster computational power from GPUs and rentable power from computer clusters.

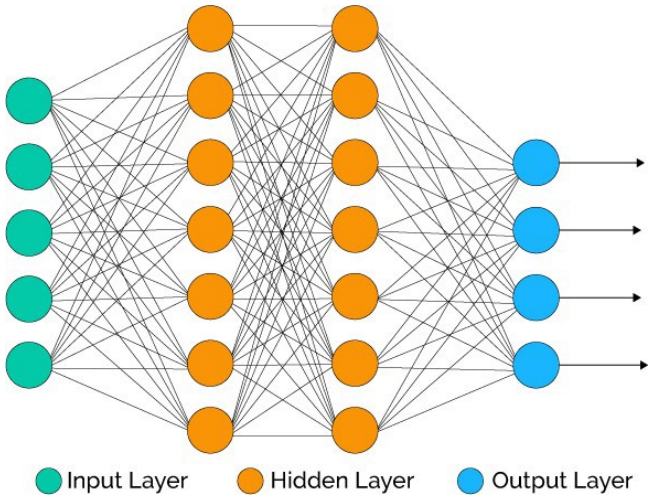

Figure 13: Example of a Deep Neural Network(DNN) with 2 hidden layers. Image reproduced from Ref.[21]

In a simple DNN every neuron in one layer is connected to every neuron in the following layer. Each neuron has different weights between its connections. When the neurons are only connected to a few surrounding neurons the layer is called convolutional. Convolutional layers are used to reduce inputs to the important parameters. A convolutional layer acts like a progressive filter that can filter unnecessary information. This results in less training time for the network but also to some data loss. When considering pattern recognition with image reconstruction the input layer has normally the same amount of neurons than the output layer. When the input got filtered the dimension of the input layer can be higher than the dimension of the output layer. There are many more types of Neural Networks at this time but these two are the only Networks that will be discussed in this thesis.

2.3.2 Supervised and Unsupervised Learning

To train a neural network one needs feed data into the network to train the weights in the network(the weights are randomly changed for each test). When a changed weight results in a better recognition its value will be updated. If a changed weight does not improve the recognition it will not be updated. Learning can be done in two different ways which all have advantages and disadvantages. The first way is to feed input data and labels which represent the output data into the system. The better the labels fit to the output of the network the more the weights are changed towards this values. This training form with labels is called supervised learning. For the unsupervised learning the network is only given the input data without any labels. The network has to find similarities in the given data and has to learn what the important pattern of the system are.

2.3.3 Training a neural network

Both techniques, the supervised and the unsupervised learning suffer from various flaws. In Supervised learning the output of data has to be initially known. So there has to be a large amount of data with a known underlying structure that represents the pattern one wants to find in data.

In Supervised learning one is able to train pattern for known structures in materials very efficiently but the network fails when it is confronted with data that differs from the known structure. For unsupervised learning the system needs much more training data than networks which use supervised learning. The reason for that is obvious because the unsupervised system has to find underlying structures in the data without any previous information about them. For just one known underlying structure the system will produce similar results and will fail for new underlying structures. Unsupervised learning can also lead to the recognition of unwanted pattern in the data. So Unsupervised Learning also requires a large variety of different input data to possibly find a universal underlying structure.

We can conclude that the Supervised Learning needs the information of the underlying structure for all input data and has often trouble to recognize related structures in the test data. The Unsupervised Learning needs much more input data with different substructures but is able to recognize related structures often better than supervised systems. This assumes that the pattern that was learned by the network is universal to all of the data that the network will be tested with afterwards. Training data also has to pass the unsupervised system much more often(when the training data is limited) which can lead to results specific to the training data.

2.3.4 Overfitting

Input data is normally split initially into a training and a testing set. This is done, because when testing the training results of a network, it is useful to check the results with a second data set. When the second data set was never introduced to the network, the testing results will filter out any possible learning of the training data itself. This can include the order of the data or pattern that were unique to the first data set. The network can also train the data set itself which means the network will gives the same results for multiple different systems. This training of the data and not the pattern in the whole data is called overfitting. Overfitting happens mostly in very large networks or when very little data is used for multiple times for training the network.

3 Results

Our results are structured in three parts. The first part includes the Monte Carlo Simulation and the results for the critical temperature using the magnetization curve. In the second part we trained a neural network to find a fixed squared patch of a different coupling constant in the lattice. In the last part we allowed the squared patch to exist everywhere in the lattice and trained the neural network again.

3.1 Monte Carlo Simulation

3.1.1 Building the finite system

When using computers for simulations one needs to use finite systems to allow calculating the simulation with Monte Carlo methods. In our case we chose smaller systems to allow faster computation of the simulation. The Monte Carlo simulation was programmed with python(31 and following pages). Replica exchange was included for the sampling of magnetization data around the critical temperature. The size of the square lattice, for which we simulated the 2D Ising model, was for the observation of the magnetization curve a N=16x16 lattice and for the training data for the neural network the size was N=32x32. This allowed us to simulate enough data to sample the magnetization curve and to train the neural networks in the limited time. The small system sizes were not a big problem because we know the correction formula for finite sized systems.

3.1.2 Randomness of the Simulation

A pseudo random number generator was used to simulate the randomness in the Monte Carlo simulation. The used number generator was, for the randomly picked lattice site which produced integer values from 0 to L, random.randint(0,L) and for the flipping probability it was the function random.random() (7). The used number generators are only pseudorandom, which means they are created by the computer with a function instead of a purely random number. We can see in the following 2D histogram that the deviations to a true random number generator are not visible in the data.

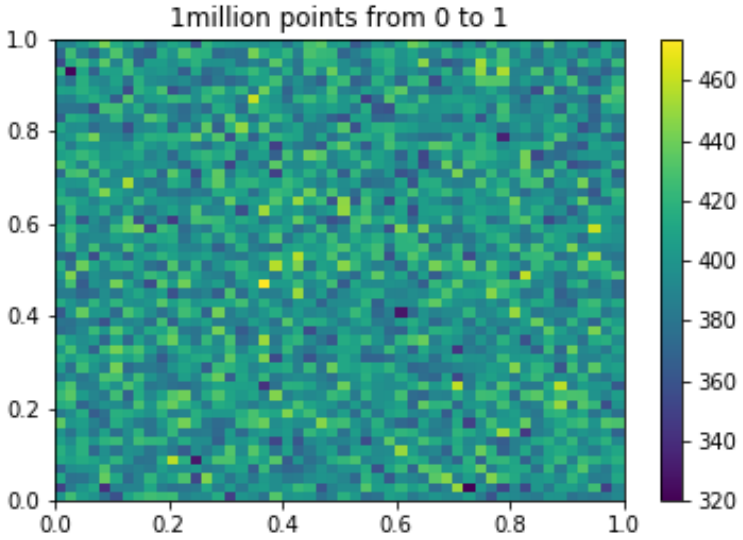

Figure 14: 2D histogramm of the data from 2 random number generators(random.random() with 50x50 bins

We can see that there are no points that did not get picked during the testing for every point and that all points got picked similarly often and we can not see any pattern in the data by eye. We can assume a sufficient randomness for the pseudo random number generator in our Monte Carlo simulation.

3.1.3 Results for different temperatures of the simulation

The 2D Ising model has a single second order phase transition. In order to introduce pattern recognition with neural networks we need to find the different pattern that can happen in the simulation. In the 2D Ising model there are three different cases that are useful to consider. The temperature can be much lower than the critical temperature, around the critical temperature and above the critical temperature. The following pictures show the three cases for a 32x32 lattice with picutres after 10000,50000 and 500000 Monte Carlo steps for a coupling constant of $J = 1$. The simulation was run without replica exchange.

The critical temperature for the system is $T_{c,L=32} \approx \frac{2.18}{k_B}(18)$.

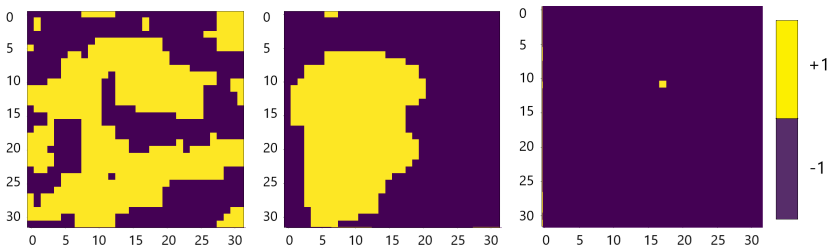

Figure 15: Result for low temperatures$T = \frac{1.1}{k_B} < \frac{2.18}{k_B} = T_{c,L=32}$

Temperatures under the critical temperature lead to an ordered state and clear clustering. The system is dominated by the next neighbor interactions. The total magnetization is close to one because the spins are almost uniform.

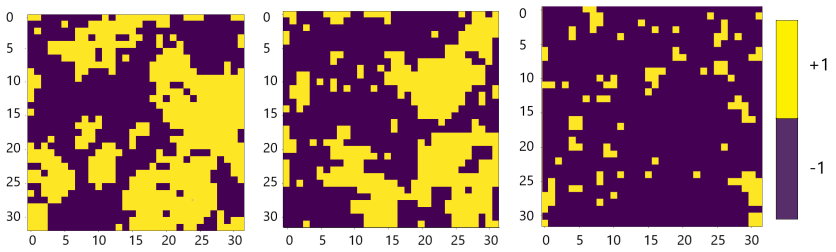

Figure 16: Result for temperatures at T_c. $T = \frac{2.18}{k_B} \approx \frac{2.184}{k_B} = T_{c,L=32}$

A Temperature around the critical temperature leads to a semi clustered case. Which means it shows the transition case between the ordered and unordered case. The total magnetization is close to one half.

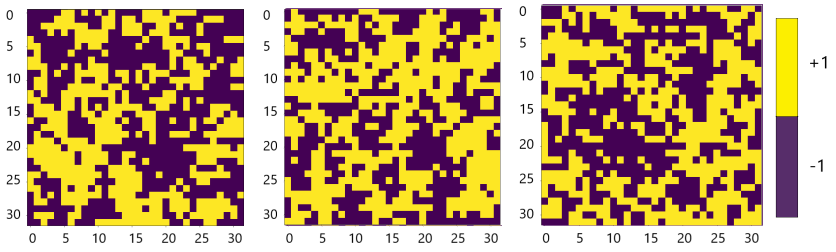

Figure 17: Result for low temperatures$T = \frac{4.0}{k_B} > \frac{2.18}{k_B} = T_{c,L=32}$

Higher temperatures than T_c cause thermal fluctuations to dominate the fluctuations in the system.There might still be some areas with local polarization but they are mostly a cause of the random fluctuations. Finite size effects also cause in a magnetization different to the theoretical prediction of zero for the infinite square lattice. This is caused by the periodic boundary conditions.

3.1.4 Phase transition with parallel tempering

To analyze the behavior around the phase transition, one can calculate the magnetization of the relaxed systems at different temperatures. To obtain the results for our case (32x32 lattice), the approximated formula(12) for the critical temperature was used. This leads to the approximated results for both used lattice sizes.

$$\beta_{c,L=16} \approx \frac{0.4751}{J} \tag{15}$$

$$\beta_{c,L=32} \approx \frac{0.4579}{J} \tag{16}$$

This can be easily converted into temperatures since $\beta = \frac{1}{k_B T}$.

$$T_{c,L=16} \approx \frac{2.105J}{k_B} \tag{17}$$

and

$$T_{c,L=32} \approx \frac{2.183J}{k_B} \tag{18}$$

For the limiting case the temperature converges towards $T_{c,N\to\infty} \approx 2.268 k_B J$.

$$M = \frac{1}{4nm} \sum_{n,m} (\sum_{nn} \sigma_{n,m}\sigma_{nn}) \tag{19}$$

The formula for the magnetization curve was modified by taking the close pattern in the data into consideration. This allows better sampling around the critical temperature because large pattern of different polarization do not disturb the value of the magnetization as much as in the normal magnetization formula(4). This modified formula for the magnetization does not only take single spins into consideration, but also the local magnetization of the system, which depends on the next neighbors. The result is then normalized to obtain magnetization values from zero to one. This magnetization draws a better picture for semi clustered cases that happen around the critical point.

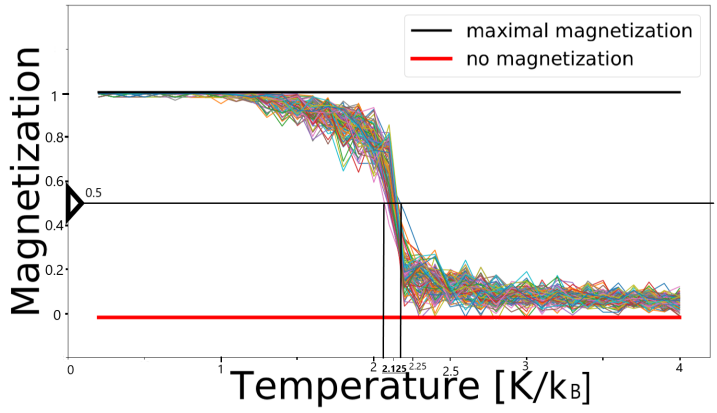

Figure 18: Obtained transition curve after averaging over 100 runs with parallel tempering with replica exchange for a 16x16 Matrix with J=1. The black line in the middle describes the point of the critical temperature in y direction and the two black lines at the two neighboring temperatures represent the lowest and highest slope in the data set

The simulation was run for 15 million steps with a replica exchange after every 20000 steps. The replica exchange is used for the even and odd systems alternatively. This was done to prevent random walks over more than one temperature step at a time. After the system reached thermal equilibrium (at 10 million Monte Carlo steps) the systems for the different temperatures are saved every 50000 steps.

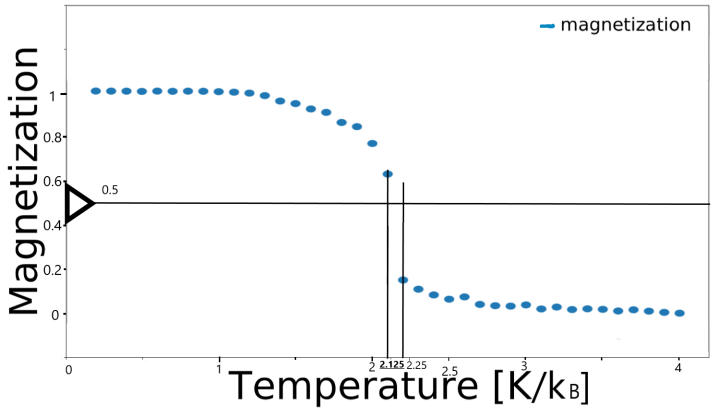

Figure 19: Avereged values for the transition curve(100 values per temperature) with tempering and replica exchange for a 16x16 Matrix with J=1. The intersections of the black lines represent the minimal and maximal value of the critical temperature for the surrounding averages.

The temperature was chosen to be from $0.2k_BT$ up to $4.0k_BT$ in steps of $0.1k_BT$. The critical temperature could not be measured precisely but one has

already information on the interval the critical temperature lies which is:

$$\frac{2.083}{k_B} < T_{c,L=16} < \frac{2.20}{k_B} \tag{20}$$

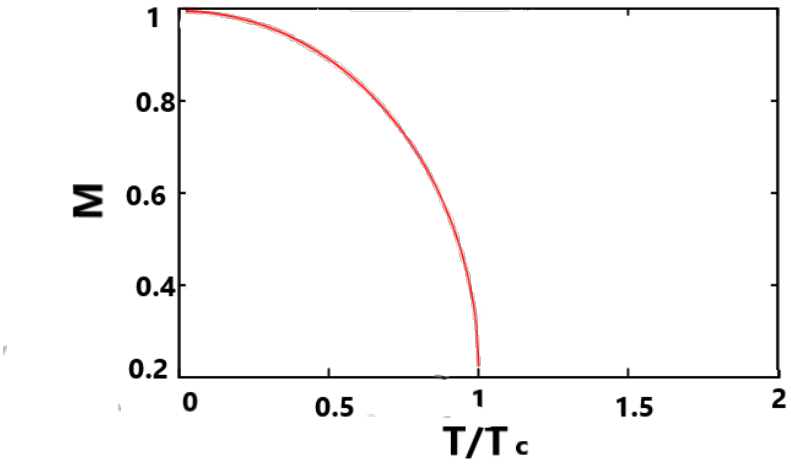

Figure 20: Theoretical curve for the magnetization in the infinite ising model. Image reproduced from Ref.[10]

In the infinite lattice the total magnetization is zero at temperatures above the critical temperature and between zero and one for smaller temperatures. We can improve our estimate of the critical temperature by fitting the theoretical curve to the data.

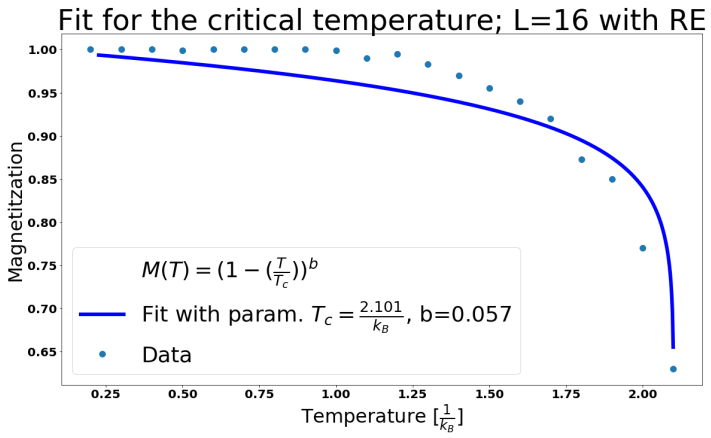

Figure 21: Result for the critical temperature for the finite system. The fitting function was taken in a simplified form from [22]

T is the temperature value from the data, T_c the temperature estimate for the critical temperature and b the critical exponent. After fitting the function

on the data, we obtained the results for the critical temperature. The critical temperature for the system was $T_c = \frac{2.101}{k_B} \pm 6.210^{-7}$. An the value for b was $b = 0.057 \pm 3.610^{-6}$ So the critical temperature lies in the interval we obtained from the averaged data which was $\frac{2.083}{k_B} < T_{c,L=16} < \frac{2.20}{k_B}$. The deviation towards the theoretical value for the finite system was 0.19%(17). We can conclude that we were able to find the critical temperature of the finite system and we were able to confirm the approximation formula for the critical temperature of finite square lattice 2D Ising systems.

3.1.5 Data aquisition

To obtain the training and testing data used to train the network, the Monte Carlo simulation was run without replica exchange. This was done to ensure reasonable computation times and a sufficient amount of data. The system was relaxed for fifty thousand steps and then a picture was taken.

3.2 Pattern recognition

To analyze the data from the simulation it was necessary to ensure that the input data was prepared and one was able to understand what the network actually learns.

3.2.1 Building the neural network

The pattern recognition was done with four dense layers. The input and output layer consisted of 1024 neurons(one per pixel on the 32x32 lattice). So the output layer allowed a complete reconstruction of the lattice. The hidden layers consisted of 4096(first hidden layer) and 1024 (second hidden layer) neurons. The activation functions that were used were sigmoid and tanh functions. Sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{21}$$

Tanh function in exponential notation:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{22}$$

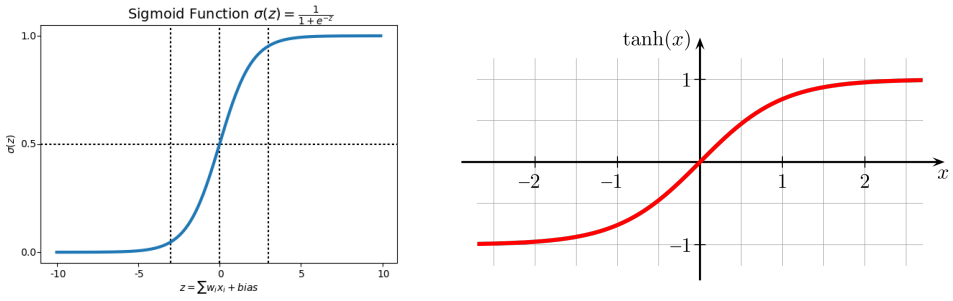

Figure 22: Sigmoid and tanh activation functions[23],[20]

The tanh activation function was needed to obtain outputs for the whole range of input vallues (-1 and +1). So the tanh is used in the first hidden dense layer and in the output layer. The sigmoid function weights the input with weights from zero to one. This was used in the second dense layer to obtain output that are non integer numbers (-1,+1).Without the sigmoid activation function the results of training showed a recognition but failed to create a visible pattern.

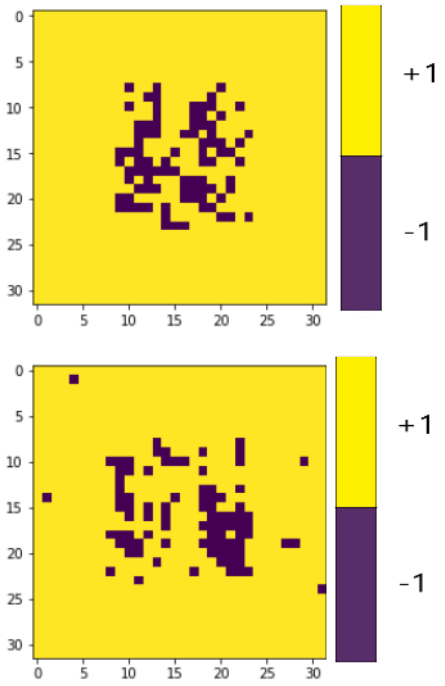

Figure 23: Training result(upper picture) for a neural network with three tanh activation functions

The integer values in the picture are caused by the training without a sigmoid function. The labels which were integer values were accepted as good fitting and the allowed +1 -1 results from the tanh caused the initial testing results to be best for integer values.The integer values were assumed by the network because the labels suggested just values of -1 and +1. When a sigmoid function is added to the neural network, the results show a larger gradient and produce a clearer picture of the patch(uncertainties from the recognition can be shown). So we used for the final neural networks a sigmoid function in one of the hidden layers to ensure non integer results for the patch recognition.

3.2.2 Data and label preparation

The data was obtained by the Monte Carlo simulation without tempering(to reduce the run time). To increase the amount of training data, the pictures were inverted and the spin polarization of the particles was flipped. This doubled the amount of our data while retaining the same physical properties and the same information.

This does not lead to defective data, because it is unimportant to which polarization the lattice relaxed to and both pictures represent valid physical states. The dominant relaxation polarization for the lattice is only determined by chance and by the random starting conditions in the lattice. So this simple process helps to double our training data without loosing information about the system but to add further information. The information added information is that it is equivalent for the physical system to which polarization the equilibrates to. The labels for the pictures were chosen as pictures with the information of the underlying structure. The dimension of the output data was equivalent to the dimension of the training pictures to keep as much information about the system as possible. The underlying structure/patch was in the first case a cube with a different coupling constant of $J_{\text{inner}} = 2$ of size 16x16 compared to the rest of the lattice $J_{\text{outer}} = 1$.

The labels were chosen to represent the position of the patch with the different coupling constant in the lattice. The outer part of the lattice was chosen to be the same polarization as the majority polarization of the associated picture. The inner part was chosen to be the opposite spin for every point on the inner patch.

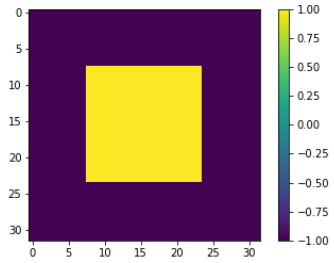

Figure 24: Label for the data with the outer part being predominantly negative polarized

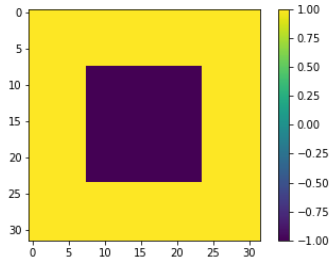

Figure 25: Label for the data with the outer part being predominantly positive polarized

The so prepared data has to be randomized to prevent any order in the data which might result in unwanted pattern a neural network could learn. So in our work we used the list.sort() command from python to randomize the order

in which the data is presented to the network. To achieve unbiased results we split the data in two different sets. The first set contained 90% of the data and was used to train the network. The second data set was never introduced to the network. This enabled us to test the results of the pattern recognition from the trained neural network. When the training and testing results diverge, we know that we did not learn the data correctly. Which means the network learned data that was explicit for the training data but not the testing data. This preparation process was not changed during the training of the neural networks because it achieved proper results.

Other possible label preparations that could have lead to similar or better results might have been to set the values at the position of the patch to 0 (random fluctuations). This can be useful for training data at temperatures between both critical temperatures, where we have one ordered and one unordered phase. One could also try and choose the label as the coupling constants at each point on the lattice and see if the results are still correct.

3.2.3 Patch recognition

The temperature for which the data was simulated was $T = \frac{3.2}{k_B}$. The coupling constants in the system were $J_{\text{outer}} = 1$ and $J_{\text{inner}} = 2$. This results in the critical temperatures of(using equation (12)):

$$T_{c;J=1} = \frac{2.183}{k_B} \tag{23}$$

for the outer part. The critical temperature for the patch was :

$$T_{c;J=2} = \frac{4.366}{k_B} \tag{24}$$

This means that the outer part will be ordered and the inner part will be unordered.

The training of the network was done with 7594 pictures for the fixed box. The size of the testing set was 844. The network was trained for 50 epochs which means the data was presented to the network repeatedly. The following pictures show the results of the learning process after the 50 epochs. The data is shown on the left and the output from the neural network on the right.

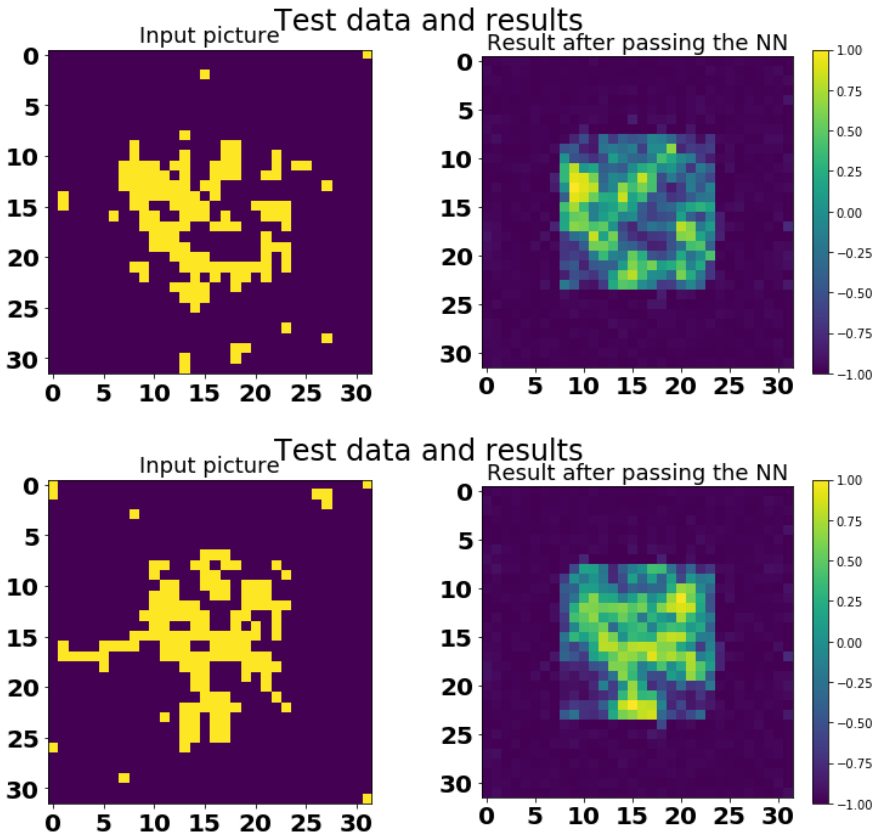

Figure 26: First result for a box with different coupling constants $J_{\text{inner}} = 2$ and $J_{\text{outer}} = 1$ at a temperature of $T = \frac{3.2}{k_B}$

We are able to see a clear picture of the box while still seeing the influence of the random clustering inside the box. We can say that we were able to recognize the box properly without loosing much information about the middle part.

3.2.4 Dynamical patch

In the next step we moved the data to resemble the patch placed randomly in the lattice. This was done to test if it is also possible to detect the noisy signal everywhere in the lattice.

The boundary condition allows movement of the patch without the loss of information, since the lattice borders are connected to the periodic continuation of the lattice (this can be seen in figure (6)).

This random position of the patch is a more realistic behavior when compared to data from real physical systems. The input data from the previous learning was taken and the patch was moved by a function everywhere in the lattice. This was permitted because we implemented periodic boundary condition in the Monte Carlo simulation. The patch was randomly moved in x and y direction. The range for moving the box was chosen to be $\mathrm{x}, \mathrm{y} = \pm 8$ because higher values would lead to a overlapping patch because of the boundary condi-

tions(6). Moving more than ±8 will not lead to useful results. The argument for this is that for a real measurement this would never happen, since the overlap is just a result of the periodic boundary conditions and would normally not happen in real systems. Including overlap data, would only result in the network having to learn the periodic boundary condition too. With this movement of the patch, we increased the original data by a factor of 16x16 which lead to over 1 million pictures for training and testing. The training was done with 500k pictures out of possible 1.2 Million. This was done because the RAM of the computer was a limiting factor. Another reason why we just used 500k pictures was that the previous network was able to learn the pattern for the testing sets with a good prediction on a similar total amount of data given into the network. The training was done for one epoch which already lasted 3 hours. The testing was done with 500 unused pictures from the 1 million set.

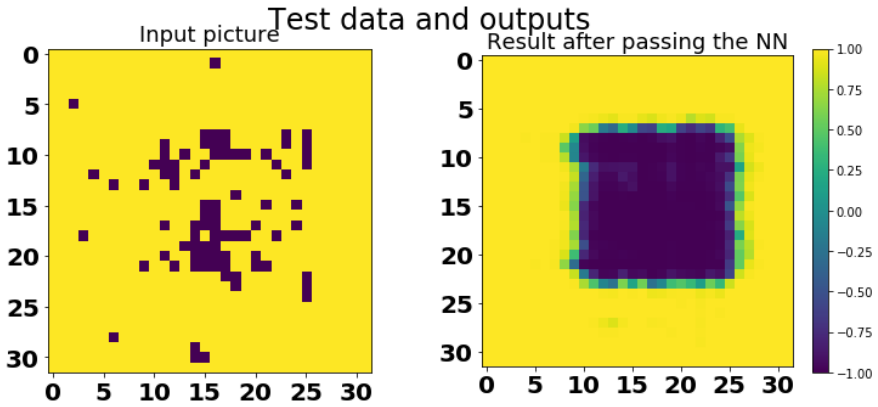

Figure 27: Test result for a changing box position for the network

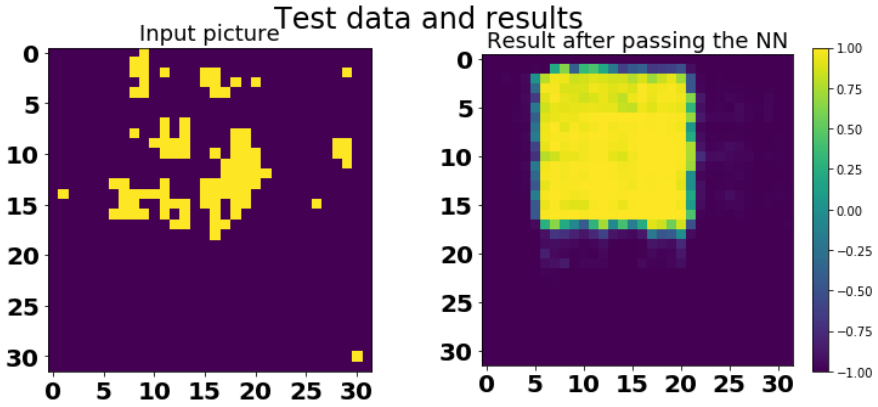

Figure 28: Test result for a changing box position for the network

The fitting results all show a clear patch in the lattice. sub pattern and edges are compared to the static patch less visible but still exist. The patch itself is

shown with very little uncertainty and a border region with polarization of zero is visible. So we can conclude that it was possible to train the network on a patch in the lattice without known position. Since this change in the position is a increase in the complexity of the problem, it is logical that the system was harder to train. This can be seen in the following picture.

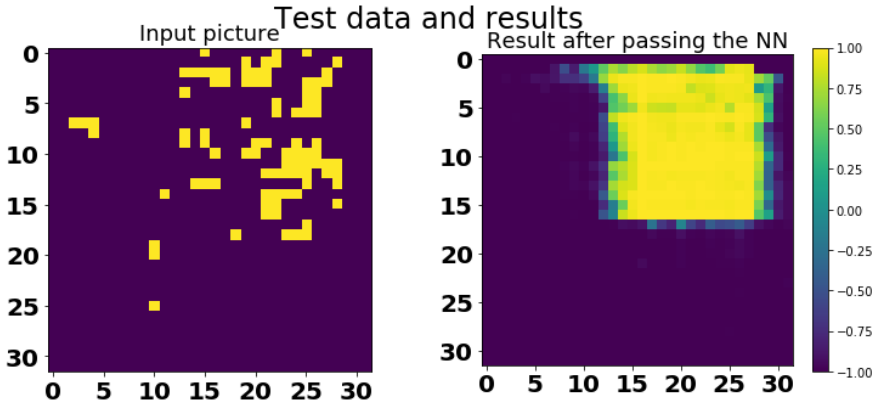

Figure 29: Difficulties in some bordering cases

When the patch position approaches the border of the lattice the recognition lacks accuracy. This can have many different reasons. First of all was the sampled size for each position in the lattice only about half of what is was during the first training. This limits the information about that specific site to the system. Maybe much longer testing for a similar number of epochs compared to the first training would have reduced those uncertainties even more. But even in those border cases we were able to identify the pattern in the data very accurate.

We can conclude that we were able to train a network to recognize a patch at all possible positions in the lattice. There is still way to improve this recognition by adding more neurons or layers to the system and thus increasing the complexity the system can learn. Compared to the first testing we lost some information of the system including the local clustering in the inner patch and a little bit of the accuracy at the borders. With additional testing time and more data the results might be improvable.

4 Conclusion

In this thesis we have investigated the phase transition of the 2D Ising model with a Monte Carlo simulation. We then used Deep Neural Networks to find pattern in the simulation data.

To simulate the 2D Ising model, on a finite square lattice, we programmed a Monte Carlo simulation based on the Metropolis algorithm with replica exchange using Python. The critical temperature for the 2D Ising model was obtained with a fit of the magnetization curve using data from our Monte Carlo simulation where replica exchange was included to improve the sampling. The critical temperature for the finite (N=16x16) system was $T_c = \frac{2.101}{k_B}$ which has a deviation of 0.19% towards the theoretical value from the formula for the finite size square lattice(12). This means that we were able to verify our simulation because it led to correct results for the position of the second order phase transition.

After ensuring that our simulation led to valid results we rescaled the lattice to a (32x32) square lattice to increase the resolution. The neural networks that were trained were Deep Neural Networks with two hidden dense layers using a sigmoid and two tanh functions as activation functions. The neural networks were trained supervised with labels representing the position and form of the patch with the different coupling constant. The trained networks were able to recognize patches with a coupling constant of $J_{\text{inner}} = 2$ compared to the other parts of the lattice which had a coupling constant of $J_{\text{outer}} = 1$. This was done first for a centralized patch and resulted in a correct recognition of the inner patch.

A second network, using the same structure as before, was trained to recognize a patch of same size for different positions on the square lattice. Both neural networks were able to identify the pattern in the data successfully. We can conclude that we were able to validate the theoretical formula for the critical temperature for finite size systems. We were also able to show that Deep Neural Networks can be used for the recognition of specific patches with different physical parameters, in our case the coupling constant.

Nevertheless there is still more work to do to examine the limits of pattern recognition for large temperature interval and different patch sizes. Finally it seems as if neural networks can be a useful addition for specific physical problems that have large data sets and uniform pattern.

5 Appendix

5.1 Code

5.1.1 Simulation

```
from matplotlib.figure import Figure

import numpy as np
#import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from matplotlib.figure import Figure
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
import numpy as np
#import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from matplotlib.figure import Figure
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
```

Figure 30: Import functions for the simulation

```
def Matrix():
    import random
    import numpy as np
    import scipy
   # import time
    import matplotlib.pyplot as plt
#construct a 2d lattice with x and y variables
    x1=[]
    xf=[]
    x2=[]
    print("Please set the length of the lattice :")  #need to set a queston for the size of the lattice LxL
    L = float(input())
    j=i=n=0
    while j<L:
        if random.randint(0,1)<1: # randomly choose if the spin is +1 or -1
            x1.append(-1)
            i=i+1
        else:
            x1.append(1)
            i=i+1
        if i==L:
            x2.append(x1)
            x1=[]
            i=0
            j=j+1
            continue
    xf=np.matrix(x2)
    return xf
```

Figure 31: Matrix initialization

```
def energy(xm200,J):
    i=0
    j=0
    a=i+1
    b=i-1
    c=j+1
    d=j-1
    if a==len(xm200):
        a=0
    if b==0:
        b=(len(xm200)-1)
    if c==len(xm200):
        c=0
    if d==0:
        d=(len(xm200)-1)
    if c==0:
        c=(len(xm200)-1)
    Htot=0
    while j<len(xm200): # calculating the energy of the system by summing up over all the nn
        #then running over the whole grid and summing up the energy
        Htot=Htot-J*(xm200[i,j]*(xm200[i,c]+xm200[i,d]+xm200[b,j]+xm200[a,j]))
        i=i+1
        if i==len(xm200):
            j=j+1
            if j==len(xm200):
                return Htot
            i=0
            continue
    return Htot
```

Figure 32: Energy calculation

```
86 def createFolder(T):
87     import os
88     import datetime
89     now = datetime.datetime.now()
90     try: #try to create the new folder and check if there is already one with that name
91         if not os.path.exists("F:/Lukas/Bachelorarbeit/27.3.19/"+str(now.day)+" "+str(now.month)+" "+str(now.year)+"    "+str
92             os.makedirs("F:/Lukas/Bachelorarbeit/27.3.19/"+str(now.day)+" "+str(now.month)+" "+str(now.year)+"    "+str(now.h
93             Folder="F:/Lukas/Bachelorarbeit/27.3.19/"+str(now.day)+" "+str(now.month)+" "+str(now.year)+"    "+str(now.hour)+
94     except OSError: #print the error if the directory is already there
95         print ('Error: Creating directory. ' +  str(now.day)+"."+str(now.month)+"."+str(now.year)+"    "+str(now.hour)+"_"+st
96         Folder=0
97     return Folder
```

Figure 33: Creating of the folders for the pictures

```
86 def createFolder(T):
87     import os
88     import datetime
89     now = datetime.datetime.now()
90     try: #try to create the new folder and check if there is already one with that name
91         if not os.path.exists("F:/Lukas/Bachelorarbeit/27.3.19/"+str(now.day)+" "+str(now.month)+" "+str(now.year)+"    "+str
92             os.makedirs("F:/Lukas/Bachelorarbeit/27.3.19/"+str(now.day)+" "+str(now.month)+" "+str(now.year)+"    "+str(now.h
93             Folder="F:/Lukas/Bachelorarbeit/27.3.19/"+str(now.day)+" "+str(now.month)+" "+str(now.year)+"    "+str(now.hour)+
94     except OSError: #print the error if the directory is already there
95         print ('Error: Creating directory. ' +  str(now.day)+"."+str(now.month)+"."+str(now.year)+"    "+str(now.hour)+"_"+st
96         Folder=0
97     return Folder
```

Figure 34: Creating of the folders for the pictures

```
def mcshort1(Tmin,Tmax,Tstep,N,NN,xm300,T,Folder,Htot,J,n,Hnew,Hold,change,nNn):
    Zahl=0
    jj=0
    i=0
    while n>-1:
        Zahl=Zahl+1
        import random

        i=random.randint(0,len(xm300[jj])-1)
        j=random.randint(0,len(xm300[jj])-1)
        r=random.uniform(0, 1)
        if 7<i<24 or 7<j<24: #if the outer part is chosen
            J=-1
        if -1<i<8 or -1<j<8 or 23<i<33 or 23<j<33: #check if the inner box is chosen
            J=-2
        k=i
        l=j
        a=k+1
        b=k-1
        c=l+1
        d=l-1
        if a==len(xm300[jj]):
            a=0
        if b==0:
            b=(len(xm300[jj])-1)
        if c==len(xm300[jj]):
            c=0
        if d==0:
            d=(len(xm300[jj])-1)
        if c==0:
            c=(len(xm300[jj])-1)
        xn=xm300[jj][k,l]
        T[jj]=round(T[jj],1)
        if n%NN==0 and NN!=-1 : #every how many steps it should save
            figure = Figure(figsize=(19.2,10.8)) #create the pictures if we reach a step we want to sav
            canvas = FigureCanvas(figure)
            axes = figure.add_subplot(1,1,1)
            Plot= axes.imshow(xm300[jj])
            figure.colorbar(Plot,values=[-2,2],ticks=[10,-10],label=[-1,1])#,labelsize="18")
            axes.plot([7.5,23.5],[7.5,7.5],c="red",markersize=12)
            axes.plot([7.5,7.5],[7.5,23.5],c="red",markersize=12)
            axes.plot([7.5,23.5],[23.5,23.5],c="red",markersize=12)
            axes.plot([23.5,23.5],[7.5,23.5],c="red",markersize=12)
            figure.savefig(str(Folder[jj])+"/"+str(n+nNn)+".png")#,dpi=1)#,dpi=0.2) #save the picture
            figure.savefig(str(Folder[jj])+"/"+str(n+nNn)+".png")#,dpi=1)#,dpi=0.2) #save the picture
            np.savetxt(str(Folder[jj])+"/"+str(n+nNn)+".txt",xm300[jj])#save also as a txt

        Hold[jj]=-J*(xm300[jj][k,l]*(xm300[jj][k,c]+xm300[jj][k,d]+xm300[jj][b,l]+xm300[jj][a,l]))
        #calculate the old energy of the point to compare both after flipping the spin
        xm300[jj][k,l]=-1*xm300[jj][i,j]
        #flip the spin
        Hnew[jj]=-J*(xm300[jj][k,l]*(xm300[jj][k,c]+xm300[jj][k,d]+xm300[jj][b,l]+xm300[jj][a,l]))
        #calculate the new energy of the point with its neighbors
        if Hnew[jj]<Hold[jj] or Hnew[jj]==Hold[jj]: #energetic more stable so save the change
            change.append([jj,i,j,xn,xm300[jj][i,j],Hnew[jj]-Hold[jj],T[jj]])
            if n==N:
                if jj==len(T)-1:
                    break
                else:
                    print(T[jj],"Temperature finished")#number of steps reached so going on to the next temp
                    n=0
                    jj=jj+1
            else:
                n=n
        if Hold[jj]>Hnew[jj]: #criterion for thermally flipped spin if it is energetically not more stable
            if r<np.exp(1*(Hnew[jj]-Hold[jj])/(T[jj])): #r is a random number from 0 to 1(Metropolis cirterion)
                change.append([jj,i,j,xn,xm300[jj][i,j],Hnew[jj]-Hold[jj],T[jj]])
                Htot[jj]=Htot[jj]-Hold[jj]+Hnew[jj]
                if n==N:
                    if jj==len(T)-1:
                        break
                    else:
                        n=0
                        jj=jj+1
                else:
                    n=n
            else:
                xm300[jj][i,j]=xn #if the criterion is not fullfilled the old value will be chosen

                if n==N or n>N:
                    if jj==len(T)-1:
                        break
                    else:
                        n=0
                        jj=jj+1
                else:
                    n=n
        n=n+1

    return xm300#,Htot
```

Figure 35: Monte Carlo algorithm

```
def mcpptf(Number=200):#tmin tmax.......    #Number stands for the times you repeat your MCsim
    nNn=0
    change=[]
    #xm3001=[]
    xm200=Matrix1(32)
    xer=[]
    print("Choose the min temperature for your run (larger than 0.1K:")#because the number generater has only a accuracy
    #of 18 digits
    Tmin = float(input())
    print("Choose the max temperature for your run (larger than 0.1K:")#because the number generater has only a accuracy of
    Tmax = float(input())
    print("Choose the temperature step size for your run (about 0.1K:")
    Tstep = float(input())
    print("Choose J")#because the number generater has only a accuracy of 18 digits
    J = -1*float(input())
    T=np.linspace(Tmin,Tmax,round(((Tmax-Tmin-Tstep)/Tstep),3)+2)
    import matplotlib.pyplot as plt
    xm300 = [xm200[0:len(xm200)]*1 for x in range(len(T))]
    print("Please set the total number of picks :")
    N = int(input())
    print("Please set Number of picks after which you want to take a picture (if 0 no pictures) :")
    NN = int(input())
    if NN==0:
        Folder=0
        NN=-1
    Zahl=0
    Hnew=[0]*len(T)
    Hold=[0]*len(T)
    Htot=[0]*len(T)
    n=0
    r=0
    Hnew1=0
    Hold1=0
    n1=0
    r1=0
    change=[]
    jj=0
    Htot=[0]*len(T)
    Fold=0
    Folder=[]
    while jj<len(T):
        Fold=createFolder(T[jj],J, N, NN, Tmin, Tmax,Tstep)
        Folder.append(Fold)
        jj=jj+1
    jj=0
    xsxs=[]
    xsx=[]
    jk=0
    xsx=[]
    jk=0
    ppr=0
    while jk<Number:
        nNn=N*jk
        ppr=jk%2
        xm300=mcshort1(Tmin,Tmax,Tstep,N,NN,xm300,T,Folder,Htot,J,n,Hnew,Hold,change,nNn)
        #monte carlo algorithm creates the matrixes which are saved
        ii=0
        xenergy=[]
        while ii<len(xm300):
            xenergy.append(energy(xm300[ii],J))#
            ii=ii+1



        pp=ppr+1
        xmnew=xm300*1 #only every second point is chosen for replica exchange to prevent a walk over every temp
        while pp>len(xenergy[ppr::2]):        #parallel tempering loop which checks the different temperatures of the system
            #and compares them after metropolis hastings criterion
            xnew=xm300[ppr::2][pp]*1
            if np.exp((xenergy[ppr::2][pp]-xenergy[pp*2+ppr-1])*(1/T[2*pp+ppr-1]-1/T[ppr::2][pp]))>1:
                print(np.exp((xenergy[ppr::2][pp]-xenergy[pp*2+ppr-1])*(1/T[2*pp+ppr-1]-1/T[ppr::2][pp])),T[2*pp+ppr-1],
                      T[ppr::2][pp],"PT?",xenergy[ppr::2][pp-1],"i",xenergy[pp*2+ppr-1],"j")
                xm300[ppr::2][pp]=xm300[pp*2+ppr-1] #flip the systems while letting the temperature list unchanged
                xm300[pp*2+ppr-1]=xmnew[ppr::2][pp]
                print(energy(xmnew[pp*2+ppr-1],J),xenergy[ppr::2][pp],"energies")
                print(energy(xm300[pp*2+ppr-1],J),energy(xmnew[ppr::2][pp],J),"energyfinals",energy(xm300[pp*2+ppr-1],J))
            pp=pp+1
        jkj=0
        jk=jk+1
```

Figure 36: Metropolis algorithm with replica exchange

5.1.2 DNN

```
import tensorflow
import numpy as np
I=50000
II=[]
Igood=[]
Ik=1
while Ik<464:
    II.append((np.loadtxt("F:/Lukas/Bachelorarbeit/27.3.19/data tot pattern j1,2 50000/6 6 2019   18_55 3.2K/"#folde
                          +str(I*Ik)+".txt"),mp))#mp is the minus outside plus inside label
    Igood.append((-1*np.loadtxt("F:/Lukas/Bachelorarbeit/27.3.19/data tot pattern j1,2 50000/6 6 2019   18_55 3.2K/"
                                +str(I*Ik)+".txt"),pm))#pm is the plus outside minus inside label

    Ik=Ik+1
    if Ik%1000==0:
        print(I)
print(len(II))
print(len(II))

#F:/Lukas/Bachelorarbeit/27.3.19/data tot pattern j1,2 50000/6 6 2019   18_55 3.2K/50000.txt
#F:\Lukas\Bachelorarbeit\27.3.19\data tot pattern j1,2 50000\6 6 2019   18_55 3.2K
```

```
463
```

Figure 37: Import of the training and testing data with labels(the folders were presorted to only have the outside plus or the minus orientation)

```
#
import random
random.shuffle(II)
random.shuffle(Igood)
i=0

train=[]
test=[]
ii=0
while ii<len(II):
    while ii<len(II)-0.1*len(II):
        train.append(II[ii])
        train.append(Igood[ii])
        ii=ii+1
    test.append(II[ii])
    test.append(Igood[ii])
    ii=ii+1
random.shuffle(test)
random.shuffle(train)
```

Figure 38: Shuffle of the data with the labels to randomize the positions and prevent the network from learning just the changes in the input data(pm and mp cases)

```
testd=np.array(testd)
trainlaa=np.array(trainla)
traind=np.array(traind)
Idata=[]
Idatal=[]
Itest=[]
Itestl=[]
I=0
i=0
II=0
print(len(traind))
while i<len(traind):
    I=0
    while I<16:
        II=0
        while II<16:
            Idata.append(move(I-8,II-8,traind[i]))
            Idatal.append(move(I-8,II-8,trainl[i]))
            II=II+1
        I=I+1
    if i%200==0:
        print(i)
    i=i+1
I=0
i=0
II=0
print(len(traind))
while i<len(testd):
    I=0
    while I<16:
        II=0
        while II<16:
            Itest.append(move(I-8,II-8,testd[i]))
            Itestl.append(move(I-8,II-8,testl[i]))
            II=II+1
        I=I+1
    if i%200==0:
        print(i)
    i=i+1
print(len(Idata))
```

Figure 39: Creation of the data for a box with random position in the lattice

```
import keras
import tensorflow as tf
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(32,32)), #input layer
    keras.layers.Dense(32*32*4, activation=tf.nn.tanh),#first hidden layer
    keras.layers.Dense(32*32, activation=tf.nn.sigmoid),#second hidden layer
    keras.layers.Dense(1024, activation=tf.nn.tanh) #output layer
])
model.compile(optimizer='adam',
              loss='mse',
              metrics=['accuracy'])

model.fit(Idata2,
          a[0:561152], epochs=10)
test_loss, test_acc = model.evaluate(Itest, Itest1)
acc.append(test_acc)
print('Test accuracy:', test_acc)
predictions = model.predict(Itest)
predictions[0]
```

Figure 40: Building of the layers for the final neural network,fitting on the training data and testing with unlabeled data

```
1 model.fit(traind, trainlaa, epochs=30)
```

```
Epoch 1/30
4384/4384 [==============================] - 42s 10ms/step - loss: 0.2129 - acc: 0.0000e+00
Epoch 2/30
4384/4384 [==============================] - 42s 10ms/step - loss: 0.2107 - acc: 0.0000e+00
Epoch 3/30
4384/4384 [==============================] - 43s 10ms/step - loss: 0.2088 - acc: 0.0000e+00
Epoch 4/30
4384/4384 [==============================] - 42s 10ms/step - loss: 0.2061 - acc: 0.0000e+00
Epoch 5/30
4384/4384 [==============================] - 43s 10ms/step - loss: 0.2046 - acc: 0.0000e+00
Epoch 6/30
4384/4384 [==============================] - 43s 10ms/step - loss: 0.2029 - acc: 0.0000e+00
Epoch 7/30
4384/4384 [==============================] - 47s 11ms/step - loss: 0.2012 - acc: 0.0000e+00
Epoch 8/30
4384/4384 [==============================] - 43s 10ms/step - loss: 0.1993 - acc: 0.0000e+00
Epoch 9/30
4384/4384 [==============================] - 43s 10ms/step - loss: 0.1983 - acc: 0.0000e+00
```

Figure 41: training for the neural network(training for the fixed box)

5.2 Training results

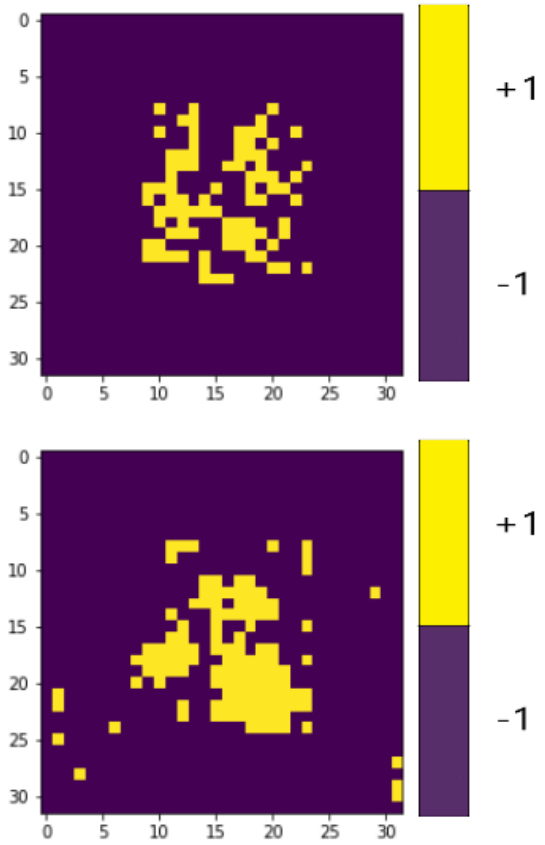

Figure 42: Training result(upper picture) for a neural network with three tanh activation functions

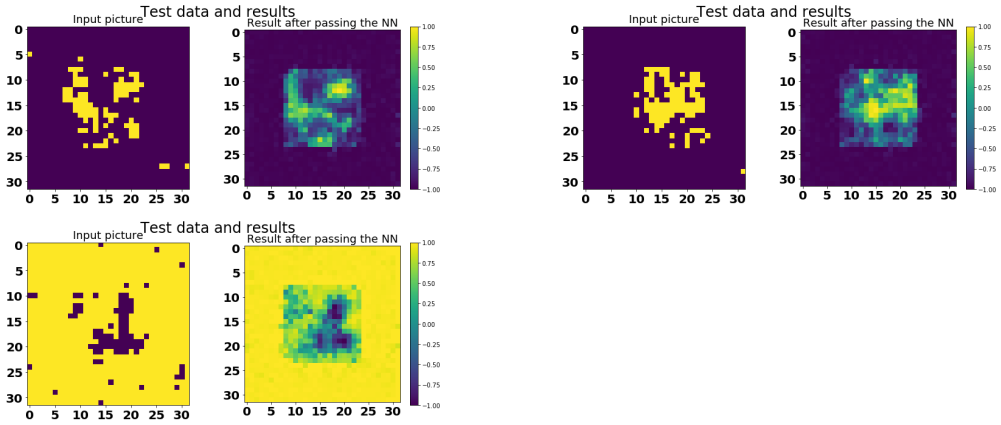

Figure 43: More training results for the first network with data on the left and the training result on the right

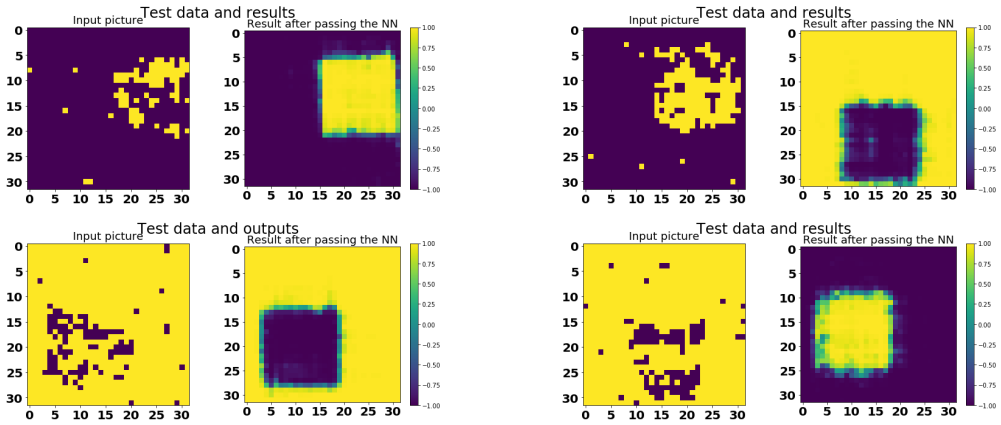

Figure 44: More training results for the second network with data on the left and the training result on the right

References

[1] A. Miguel San Martin Phillip Alvelda. *Neural Network Star Pattern Recognition for spacecraft attitude determination and control.*

[2] Geoffrey E. Hinton Alex Krizhevsky Ilya Sutskever. *ImageNet Classification with Deep Convolutional Neural Networks.* URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[3] William Watkins Anuj Karpatne. *Physics-guided Neural Networks PGNN.* URL: https://arxiv.org/pdf/1710.11431.pdf.

[4] K. Schulten E. Domany J.L. van Hemmen. *Models of Neural Networks.* 1991.

[5] Andre Manoe Marylou Gabrié. *Entropy and mutual information in models of deep neural networks.*

[6] Evan Shorman Alexandra Garraud et al. *Beitrag zur Theorie des Ferromagnetismus.* 1925. URL: Z.%20Phys.,%2031%20(1):%20253%E2%80%93258,%20https://www.researchgate.net/figure/Theoretical-fit-of-Co-rich-Co-Pt-saturation-magnetization-behavior-versus-temperature_fig5_260435097.

[7] Zureks. *Spin glasses.* URL: https://commons.wikimedia.org/w/index.php?curid=10198960.

[8] Michael Schmid. *Ferromagnetic ordering in the strict sense.* URL: https://commons.wikimedia.org/w/index.php?curid=1445910.

[9] Michael Schmid. *Antiferromagnetic ordering.* URL: https://commons.wikimedia.org/w/index.php?curid=1445942.

[10] Hendrik Weimer. *Spontaneous magnetization of the Ising model.* URL: https://commons.wikimedia.org/wiki/File:Spontaneous_magnetization_of_the_Ising_model.svg.

[11] Grimlock. *Periodic boundary conditions.* URL: https://en.wikipedia.org/wiki/Periodic_boundary_conditions.

[12] Onsager L. *Crystal statistics. I. A two-dimensional model with an order-disorder transition.* 1952. URL: Phys.Rev.%2065.808.1952..

[13] I.M. Karandashev M. Yu. Malsagov and B.V. Kryzhanovsky. *The Analytical Expressions for a Finite-Size 2D Ising Model.* URL: https://arxiv.org/ftp/arxiv/papers/1706/1706.02541.pdf.

[14] Zecchina R. Martin O.C. Monasson R. *Statistical mechanics methods and phase transitions in optimization problems.* 1952. URL: Theoretical%20Computer%20Science.%20265(1-2).pp.3-67.,%202001.

[15] Jacques Kotze. *Introduction to Monte Carlo methods for an Ising Model of a Ferromagnet.* URL: url%20=%20%7Bhttps://arxiv.org/pdf/0803.0217.pdf%7D,.

[16] Georg Wiora. *Metastable.* URL: https://en.wikipedia.org/wiki/File:Meta-stability.svg.

[17] Aldo Campana. *GFMER.* URL: https://www.gfmer.ch/Books/Reproductive_health/Basic_Fig3.html.

[18] P99am. *Parallel Tempering*. URL: https://en.wikipedia.org/wiki/Parallel_tempering.

[19] Unknown. *Perceptron*. URL: https://appliedgo.net/media/perceptron/neuron.png.

[20] Nahua Kang. *Multi-Layer Neural Networks with Sigmoid Function— Deep Learning for Rookies (2)*. URL: https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f.

[21] Rahul Bhatia. *When not to use Neural Networks*. URL: https://cdn-images-1.medium.com/max/1200/1*DW0Ccmj1hZ0OvSXi7Kz5MQ.jpeg.

[22] Evan Shorman Alexandra Garraud et al. *Influence of temperature on the magnetic properties of electroplated Co-rich Co–Pt thick films*. URL: https://www.researchgate.net/figure/Theoretical-fit-of-Co-rich-Co-Pt-saturation-magnetization-behavior-versus-temperature_fig5_260435097.

[23] Geek3. *Hyperbolic Tangent function plot*. URL: https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f.

6 Zusammenfassung

Diese Bachelorarbeit behandelt die Simulation des 2D Isingmodells mit Hilfe einer Monte Carlo Simulation, sowie von Mustererkennung zu Simulationsdaten des 2D Isingmodells mit tiefen Neuronalen Netzwerken(Deep Neural Networks). Im ersten Schritt dieser Arbeit wurde das 2D Isingmodell eingeführt und deren Lösungen erörtert. Im nachfolgenden Schritt wurde eine Lösung für beschränkte und unendliche Systemgrößen des 2D Isingmodells eingeführt. Dannach wurde mit Hilfe einer, mit Python erstellten, Monte Carlo Simulation mit Parallel Tempering(PT), die Magnetisierung des Systems zu verschiedenen Temperaturen gemessen und die Simulationsdaten aufgetragen, gemittelt und gefittet. Die Systemgröße betrug hierbei 16x16 Spins. So konnte zum einen die Lösung der Formel für begrenzte Isingsysteme bestätigt werden und zum anderen unsere Simulation geprüft, sowie die dazu gehörige kritische Temperatur bestimmt werden. Diese betrug für das beobachtete System $T_c = \frac{2.101}{k_B}$, was einer Abweichung von 0.19% zu dem Theoriewert entspricht.
Nun konnten wir mit Hilfe des Wissens über die kritische Temperatur Daten zu unterschiedlichen Temperaturen aufnehmen und angemessen einordnen. Dies konnte nun ohne Parallel Tempering gemacht werden, da die kritische Temperatur bereits bekannt war. Durch das Wegfallen von Parallel Tempering in der Simulation, konnte die Laufzeit erheblich verbessert werden, und ein größeres 32x32 System für die Simulation zur Datenaufnahme verwendet werden. Das Isinggitter, welches zum Training von Musttererkennung verwendet wurde, besteht aus zwei unterschiedlichen Bereichen mit unterschiedlichen Kopplungsstärken, von welchen die kritische Temperatur des Systems abhängt. Nach Datenaufnahme konnten wir unser Netzwerk auf einen zentralen Kasten mit unterschiedlicher Kopplungskonstante trainieren und erreichten hierbei gute Ergebnisse. Die Fläche des Kastens konnte deutlich besser aufgelöst werden und die Abgrenzung zwischen beiden Kopplungskonstanten wurde deutlich klarer sichtbar, als es in den Daten der Fall war. In einem weiteren Schritt erhoben wir Simulationsdaten zu einem Kasten, welcher sich nicht mehr zentral im Gitter befand, sondern frei innerhalb des Gitters platziert wurde. Nachdem auch diese Daten in einem Neuronalen Netzwerk gelernt wurden, kamen wir zu dem Ergebnis, dass eine gute Mustererkennung weiterhin möglich ist. Obwohl die Mustererkennung weiterhin deutlich sichtbar war und auch die neuronalen Netzwerke, die gleichen Ausmaße besaßen, kam es im Vergleich zu der ersten Mustererkennung zu schlechterer Erkennung des Kastens. Es kam im Vergleich zur ersten Erkennung zu einem Informationsverlust in der Darstellung der Ergebnisse, was mit der erhöhten komplexität der Datenstruktur zu erklärt werden könnte.
Zusammenfassend kann man sagen, dass es uns möglich war, das 2D Isingmodell genauer zu untersuchen und aufgestelte Gleichungen zu überprüfen, des Weiteren gelang es uns zu zeigen, dass Neuronale Netzwerke bei physikalischen Systemen wie dem 2D Isingmodell für Mustererkennung eine künftige und nützliche Rolle spielen kann.

7 Danksagung

Ich bedanke mich bei allen Personen die mich bei meiner Arbeit unterstützt haben. Besonders bedanken möchte ich mich bei Dr. Karin Everschor Sitte und bei Timo Pulch für ihre Betreueung und die Unterstützung bei Fragen und Unklarheiten. Bedanken möchte ich mich zusätzlich auch bei Benjamin McKeever und Davi Rodrigues für ihre Hilfe bei thematischen Fragen, sowie an alle Korrekturleser. Ein abschließender Dank geht auch an meine Familie, die mich im Verlauf der Arbeit tatkräftig unterstützt hat.