universität
wien

Machine Learning the Ising Model

LP Praktikum Computational Statistical Mechanics

Philip Blair

philip.blair@gmx.de

Institute of Physics
University of Vienna

Supervisor:
Prof. Dr. Christoph Dellago

August 30, 2021

Contents

1 Introduction **1**

1.1 The Ising Model 1

1.2 Monte Carlo Algorithm: Implementation 3

2 Machine Learning: Neuronal Networks **6**

2.1 Backround 6

2.1.1 General Idea 6

2.1.2 Notation and Architecture 7

2.1.3 Training and Loss 8

2.1.4 Optimizers and Flux 8

2.2 Learning the Phases of the Ising Model 10

2.2.1 Results 10

2.3 Learning the Temperatures of the Ising Model 12

2.3.1 Approach 12

2.3.2 Results 13

3 Conclusion **19**

3.1 Results 19

3.2 Future Focus 20

3.3 Unsolved Problems 20

Bibliography **21**

Chapter 1

Introduction

The goal of this project is to get a first impression of Machine Learning (ML) in the context of physical processes. Inspired by Melko and Carrasquilla [1], the Ising Model will be used as a simple example and try to predict some well-known features of the model. To provide a comprehensive picture of the development of the project, the introduction will contain information about how Ising configurations are generated by Monte Carlo algorithms and the basic idea of a neuronal network.

1.1 The Ising Model

The Ising model can be regarded as the textbook model of statistical mechanics. Originally used to study ferromagnetism, its applications are now much larger than initially intended. Its physical properties are well understood and it is therefore excellent for testing new techniques for physical problem-solving. Because it is based on simple assumptions, it is also easy to implement with modern programming languages. The focus of this project will be the two-dimensional Ising model on a square lattice. Each site s_i of the lattice has either value $s_i = 1$ or $s_i = -1$, corresponding to an upward or downward spin. The energy of the system is given by the Hamiltonian H, which will only include nearest neighbor interactions. It is given as

$$H = -J \sum_{\langle i,j \rangle} s_i s_j \tag{1.1}$$

In our case, the interaction strength J will be set to $J = 1$, because it will not have an impact on the ML algorithm used later. The composition of the individual upward and downward spins is called a *configuration*. Given a total energy E, the probability to find the system in a certain configuration is related to the Boltzmann-weight

$$p \propto e^{\frac{-E}{k_B T}} \tag{1.2}$$

and is therefore depending on the temperature of the system. Now what makes this model particularly interesting is that in two dimensions or more, the system

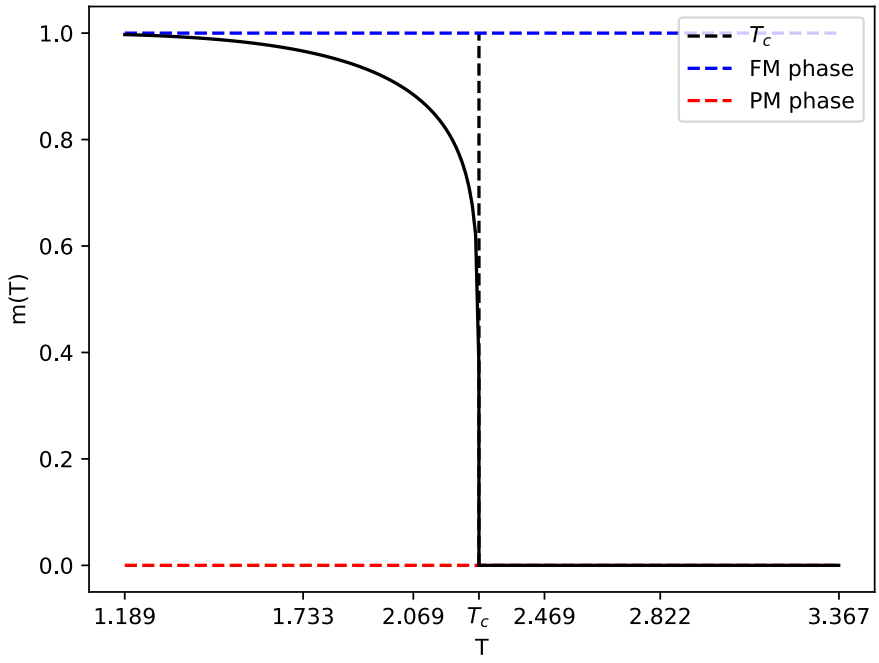


Figure 1.1: The phase transition of the two dimensional Ising model occurs at the critical Temperature T_c. This is valid in the ferromagnetic limit, where the lattice size $L \to \infty$.

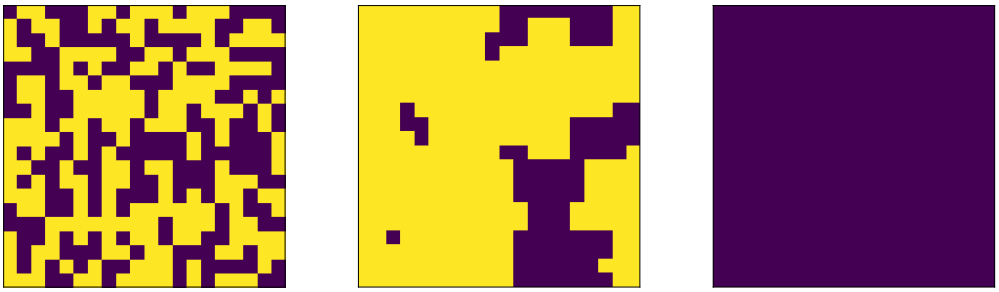
can exist in two distinct phases, a ferromagnetic (FM) phase and a paramagnetic (PM) phase. The FM phase is an ordered state of aligned spins, while the PM phase is disordered. For low temperatures, the system will be in the FM phase. As the temperature increases, the system undergoes a phase transition at a critical temperature T_c, with

$$T_c = \frac{2J}{k_B \ln\left(1+\sqrt{2}\right)} \tag{1.3}$$

so for $k_B = J = 1$, the critical temperature has a value of $T_c = 2.26$. Above this point, the system will be in the PM phase. Quantitative measurement of configurations of the Ising model is the so-called magnetization m with

$$m = \left| \frac{1}{N} \sum_i^N s_i \right| \tag{1.4}$$

the mean value of all the spins in the system. Plotting the magnetization against the temperature of the system leads to figure 1.1



(a) $N = 0$ (b) $N = 5$ (c) $N = 100$

Figure 1.2: Applying the Monte Carlo algorithm will lead to an equilibrium distribution. In this example, the input temperature was set as $T = 0.1$ and the configuration reached a complete ferromagnetic phase where all spins are aligned.

1.2 Monte Carlo Algorithm: Implementation

Generating configurations of the Ising model can be realized by using a Monte Carlo algorithm. By applying the algorithm repeatedly on an initially random configuration, an equilibrium distribution can be obtained for any desired temperature T. The result will be a final configuration that has the same properties as the theoretical model. The algorithm works as follows:
First, a lattice with random spin orientations is created. Then, the following steps will be repeated until an equilibrium distribution is reached:

- Chose a spin s_i on the lattice
- Calculate the local Energy E_+ of the spin according to the Hamiltonian $H_{ij} = \sum_j^N s_i s_j$
- Calculate the Energy E_- for a changed sign in s_i
- Calculate the difference $\Delta E = E_+ - E_-$.
- Calculate the Boltzmann-weight: $p = e^{-\frac{\Delta E}{k_B T}}$
- Create a random number r between 1 and 0. If $p > r$, then change the sign of s_i, else continue.

One loop of the algorithm is called a Monte Carlo step. Applying a Monte Carlo step to every spin on the lattice is called a Monte Carlo sweep. In Figure 1.4, we can see one example of a random starting configuration that completes several Monte Carlo sweeps.

An equilibrium distribution is reached when the magnetization of the configuration fluctuates around a fixed value. The closer the temperature is around T_c, the

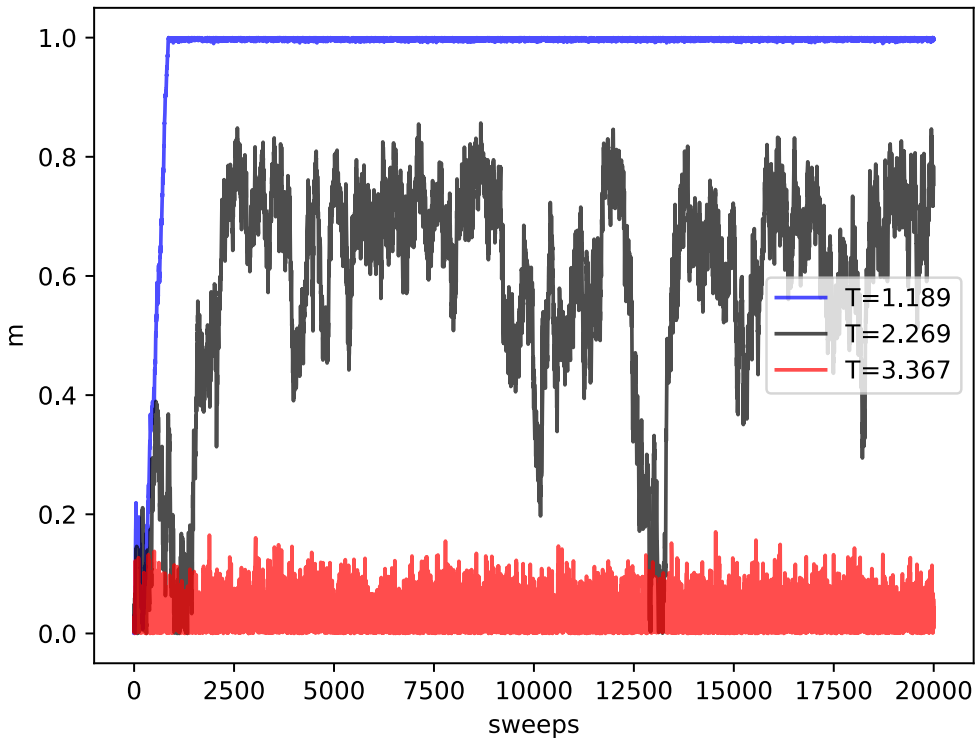
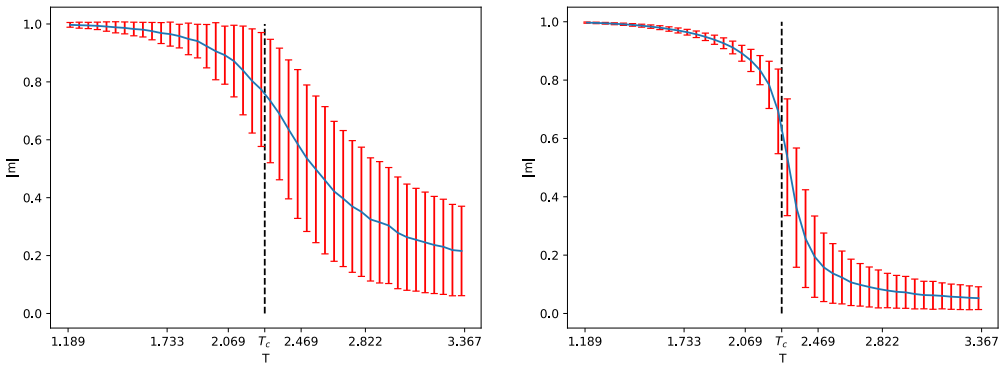


Figure 1.3: Spreading of the magnetization m for different Temperatures T and sweeping-runs N for a lattice size $L = 60$



(a) $L = 10$ (b) $L = 40$

Figure 1.4: Magetization obtained form 2000 configurations for each temperature T for different lattice sizes. The standard deviation is large around the critical temperature. This fluctuation gets smaller for bigger lattice sizes

bigger the fluctuations. Generating different configurations for the same number of sweeps will thus lead to spreading magnetizations, see figure 1.3. This effect must be considered because many configurations are needed to train a Machine Learning algorithm. Note that this is a feature of small lattice sizes L. For bigger lattice sizes, the fluctuations are smaller, as can be seen in figure 1.4.

For the following chapter we will look at different temperature sets T_{set} that is used to generate Ising model configurations. First, a small set with $T_{set} = \{1.189, 1.733, 2.069, 2.269, 2.278, 2.469, 2.822, 3.367\}$ is used to predict the Ising model phases. To predict the temperatures, a larger set with 44 different equidistant temperatures ranging from 1.2 to 3.35 is used. A total of 2000 configurations for each temperature in those sets were generated. These configurations serve as the data sets that are going to be used to train the ML algorithm. By performing symmetry operations on the configurations, e.g. inversion, it is possible to expand the data set even further. To compare the effect of finite lattice sizes, the same procedure was done for lattice sizes $L = 10$, $L = 20$ and $L = 40$.

Chapter 2

Machine Learning: Neuronal Networks

The general idea behind Machine Learning is to develop algorithms that can make predictions for new data on the basis of historic data. There are different approaches to do this automatically. Neuronal networks are one realization of a machine learning algorithm and will be used in this project. They are especially suited for tasks that include complex data sets while only allowing a few possible solutions as output. As we will see, a neuronal network is able to both predict the phase and the temperature of the Ising model configurations. To provide an appropriate insight into the method of neuronal networks, a short introduction to its most important parts will be given.

2.1 Backround

2.1.1 General Idea

For a basic understanding of a neuronal network, it is useful to keep in mind that it is basically a multivariate regression of a large number of parameters. The analogy to a biological neuronal network is grounded on the fact that during a training process, certain connections in the network will be weighted differently than others, just like the learning process of our brain will treat some neuronal connections with a higher priority than others. Each of those weights is a parameter that is changed repeatedly until a sufficiently good prediction is possible. Those changes happen if the network is trained, meaning it receives input data and adjusts its parameters according to a wanted output. In this project, the main goal will be to solve classification problems, such as predicting the phase or the initial temperature of the Ising model. For learning the basics of neuronal networks, the introduction for machine learning by P. Mehta et al. [2] was used and serves as a guideline in the following sections.

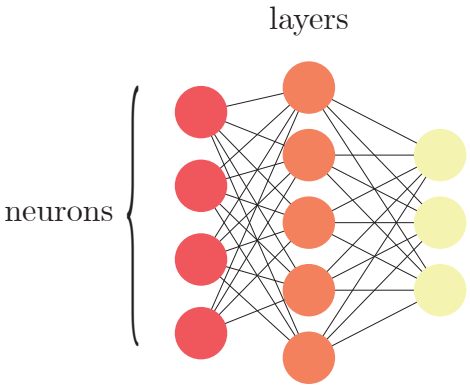


Figure 2.1: A neuronal network is structured in different layers. The input layer receives the initial data, while the output layer contains the final prediction of the network. Layers in between are called hidden layers. Each layer contains neurons that build up the connections in the network

2.1.2 Notation and Architecture

Figure 2.1 illustrates the structure of a neuronal network. It is composed of neurons and connected via a network of layers. Each arbitrary neuron n_i holds a value, the so-called *activation* a_i, which is a number $\in [0, 1]$. The first layer is the input of the network and receives the encoded information of the system of interest. In this project, the spins of the square lattice will be the input for the network. Each spin will be encoded in an activation, by assigning a spin value of 1 to the activation 1 and a spin value of -1 to an activation 0. The last layer of the network is the output layer, which represents the prediction of the system. By definition, every neuronal network has at least one input and one output layer. Layers in between are called hidden layers. The activations between layers are connected via individual weights and biases. After the input layer, the activation a_i^l of an arbitrary neuron $n_i \in [1, I]$ in layer $l \in [1, L]$ is calculated with a weighted sum z_i, followed by a nonlinear transformation $\sigma(z_i)$, also called a classifier. The weighted sum usually has the form

$$z_i^l = \sum_j w_{i,j}^l a_j^{l-1} + b_i^l \tag{2.1}$$

$w_{i,j}$ being the weight between neuron i in layer l and a connected activation a_j of a neuron in the previous layer $l - 1$. b_i is a bias that recenters the weighted sum. We can write all the weights and biases in their own vectors with

$$w = \begin{pmatrix} w_{1,1}^1 \\ w_{2,1}^1 \\ \vdots \\ w_{I,J}^L \end{pmatrix}, b = \begin{pmatrix} b_1^1 \\ b_2^1 \\ \vdots \\ b_I^L \end{pmatrix} \tag{2.2}$$

In the case of a classification problem, the activation a_i needs to satisfy $a_i \in [0, 1]$. For that, the classifier $\sigma(z_i)$ is needed. The most used classifier is the sigmoid function

$$\sigma(z_i) = \frac{1}{1 + e^{-z_i}} \tag{2.3}$$

that maps the weighted sum z_i to the desired interval. Note that the bias b_i in (2.1) then serves as a threshold value, that can suppress the activation of a neuron. In total, the activation of a neuron is therefore given with $a_i = \sigma(z_i)$. There are different versions of classifiers, but the sigmoid function is especially handy because it can be differentiated, which is needed for certain training methods. With the connection between neurons according to eq. 2.1, it is obvious that the output layer is fully connected with all previous neurons. Therefore, changing the weights w and biases b results in a changed output. The process of changing w and b such that the mean output of the network matches a predefined label as best as possible is called *training*.

2.1.3 Training and Loss

The neurons in the output layer with length L are at the end of the neuronal network. Their values serve as the overall prediction $p(w, b) = (p_1, p_2, ..., p_L)$ of the algorithm and depend on the weights and biases of the connected network. The better a prediction is, the closer it is to a desired and fixed distribution $q = (q_1, q_2, ...q_L)$, also called label. To quantify the difference between p and q, a suitable function must be constructed. Speaking in terms of ML, these functions are called loss functions. There are many possibilities to compare p and q. In our case, the crossentropy C with

$$C(q, p) = \sum_i q_i \ln p_i \tag{2.4}$$

serves as a measure of difference. Since $C(p, q)$ is minimized for $p = q$, the training of a neuronal network is a optimization problem. The goal is to minimize the value of C by adjusting w and b such that p is approaching a real distribution q provided by historic data. To make this more clear, it is useful to write $C(q, p(w, b)) = C(q, \boldsymbol{\theta})$ with $\boldsymbol{\theta} = (w, b)$.

2.1.4 Optimizers and Flux

To make sure that the weights and biases are changed systematically, new algorithms are needed. They are called optimizers because they improve the overall prediction of the network by minimizing the loss function. There are again different variations of optimizers, changing in complexity and efficiency. Due to the high dimensionality of $\boldsymbol{\theta}$, considering computation times is always needed. The result is a variety of specialized optimizers for various types of problems.

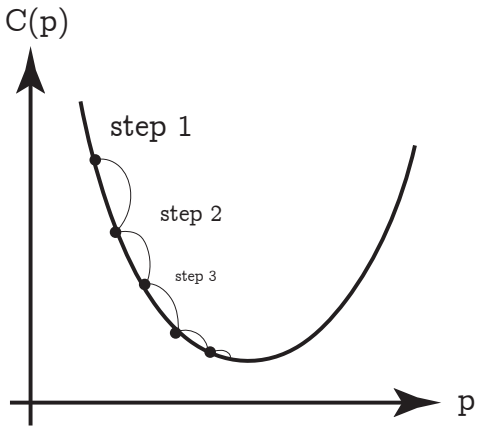


Figure 2.2: Basic concept of the gradient descent. The algorithm chooses to change the parameters of the network such that the loss function $C(p(\boldsymbol{\theta}))$ is minimized

The basic concept can be understood by looking at the simplest optimizer, the method of gradient descend (GD). It works by starting with arbitrary weights $\boldsymbol{\theta_0} = (w_0, b_0)$ and changing them in discrete steps. The size of a step is defined by the gradient of the loss function with

$$\epsilon_i = \alpha_i \nabla C(q, \boldsymbol{\theta}) \tag{2.5}$$

and changes the networks parameters with $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \epsilon_i$, where ∇C is the gradient of the loss function. The *learning rate* α_i determines the magnitude of the change and can depend on the number of steps taken. If the learning rate is too high, there is a chance of never reaching a minimum, while a too small learning rate can result in too long computation times. If calibrated correctly, the algorithm will approach a local minimum of the loss function with each change in $\boldsymbol{\theta}$, as illustrated in figure 2.2.
In this project, a more sophisticated algorithm was used. It is called the ADAM optimizer (adaptive moment estimation) [3] and also considers performance and memory storage. It has to be noted that there was no comparison of different optimizers and their performance, because the used optimizer was sufficiently fast and accurate enough for the used data set.

2.2 Learning the Phases of the Ising Model

The first implementation of a neuronal network is done according to Melko and Carrasquilla [1], where the goal is to predict the phases of individual configurations of the Ising model. The basis will be the square lattice ferromagnetic Ising model with the Hamiltonian 1.1 as mentioned before. The neuronal network is composed of an input layer with L^2 neurons that takes the value of each spin as an activation, a hidden layer with 20 neurons and a output layer with 2 neurons, representing the FM and PM phase. This leads to a prediction $p = (p_1, p_2)$ with $p_1 + p_2 = 1$. Therefore, the prediction can be interpreted as a probability that a input configuration is in a certain phase. This chosen architecture is not necessarily the best approach, but changing the number of hidden neurons has almost no effect on the prediction. A quantitative analysis of different network structures was left blank, but should be considered for a deeper discussion.
Following the paper, only configurations created with temperatures above or below the critical point T_c are used to train the network. After the training, a new data set with no restriction to certain temperatures is passed to the network. It is now expected that for high and low temperatures, the network should accurately predict the phases of the model. As the temperature approaches T_c, the distinction of the two phases becomes increasingly harder, leading to a point of maximal uncertainty right at the critical point. As a result, the point at which the network has a $p = (0.5, 0.5)$ prediction for both states marks the critical temperature T_c.

2.2.1 Results

For this network, the temperature set
$T_{set} = \{1.189, 1.733, 2.069, 2.269, 2.278, 2.469, 2.822, 3.367\}$ was used to create input data. In the next step, for each temperature the average output of the network was calculated, leading to Figure 2.3. The result shows how the probability to categorize a phase changes with respect to the temperature. This change reflects the behavior of the Ising model phase transition, which occurs at the critical point. Using the point of maximum uncertainty at $p = (0.5, 0.5)$, the critical point was estimated to be at approximately $\tilde{T}_c = 2.26 \pm 0.01$. Therefore, the network has successfully predicted the critical point and the phases of the testing data set. As expected, bigger lattice sizes lead to sharper transitions. In the thermodynamic limit of $L \to \infty$, the transition should approach the behavior shown in fig. 1.1. It is noted that this method is successful at determining the critical temperature for different coupling constants J, as well as different lattices such as a triangle lattice. This generalization marks the power of neuronal networks, as Melko and Carrasquilla pointed out [1].

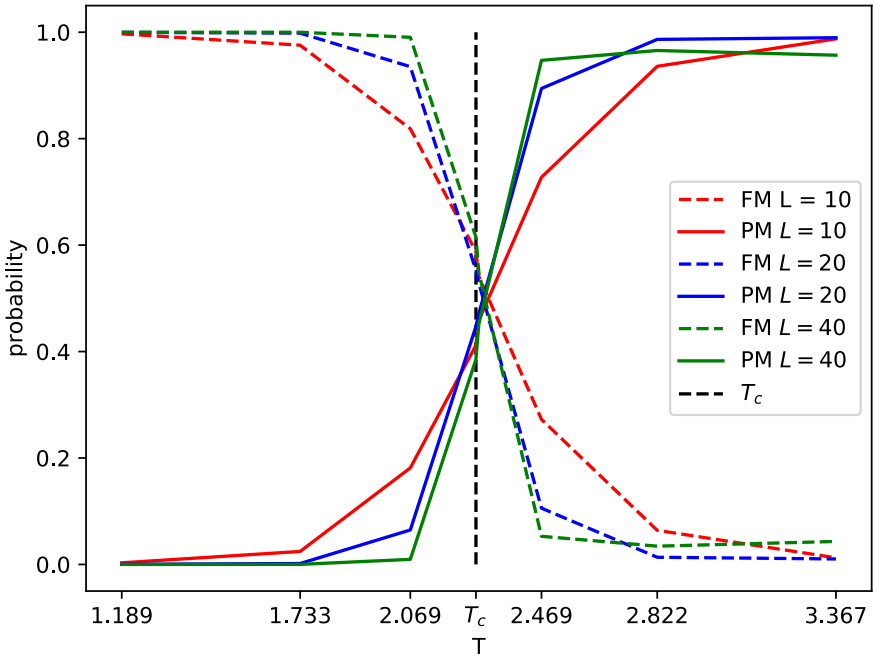


Figure 2.3: Predicted probability to be in the paramagentic (PM) or ferromagegentic (FM) phase of the Ising Model for lattice sizes $L = 10$,$L = 20$ and $L = 40$. The point of maximal confusion is directly linked with the critical point.

2.3 Learning the Temperatures of the Ising Model

As a new approach, the neuronal network will be changed, such that a prediction of temperatures of given configurations is possible. Given a configuration X, the network will predict the temperature $T \in T_{set}$ that was used in the Monte Carlo algorithm to create X. The focus of the discussion will be on the quality of the predictions and their relation to the magnetization of Ising configurations.

2.3.1 Approach

There are two possibilities to predict the temperatures with a neuronal network. As a straightforward approach, the network could only contain one neuron in its output layer. The activation of this neuron would be labeled according to the temperature of the configuration that served as the input. The network would therefore be no classification problem anymore. While this architecture seemed promising at first, it faced some serious problems.

First, the network was not able to benefit from the so-called *hot-encoding* method that is part of the library that was used to create the neuronal network called Flux [4]. Hot-encoding means that each output activation is labeled in a binary fashion: they are either hot if equal to 1 or cold if equal to 0. This method is not possible if there is only one output neuron with a continuous activation. As a consequence, computation times were not acceptable, even for lattice sizes with $L = 10$.
Furthermore, the single output can only make a statement about a unique temperature, but not about the probability to be in a certain temperature range. To overcome this, it would be needed to predict the temperatures of various configurations and calculating the mean and spread of the results. This conflicts with the first issue of high computation times.

These problems can be solved by using the second approach, namely mapping each possible input temperature to a corresponding output neuron. In this way, the network can be efficiently trained by labeling with one-hot encoding. On top of that, each neuron gives information about the probability of a temperature.

One negative aspect is the resulting discrete resolution of predictable temperatures. If higher resolutions of the predictions are desired, then the input must be adjusted accordingly.
For this problem, a new temperature set $T_{set} = \{1.2, 1, 25, \ldots, 3.35\}$ with $\#T_{set} = 44$ was defined and used to create new input configurations. In this way, the computational cost was shifted from the neuronal network towards the Monte Carlo algorithm. Although generating this many configurations takes some time, the overall computation time was still much faster than with a single output neuron.

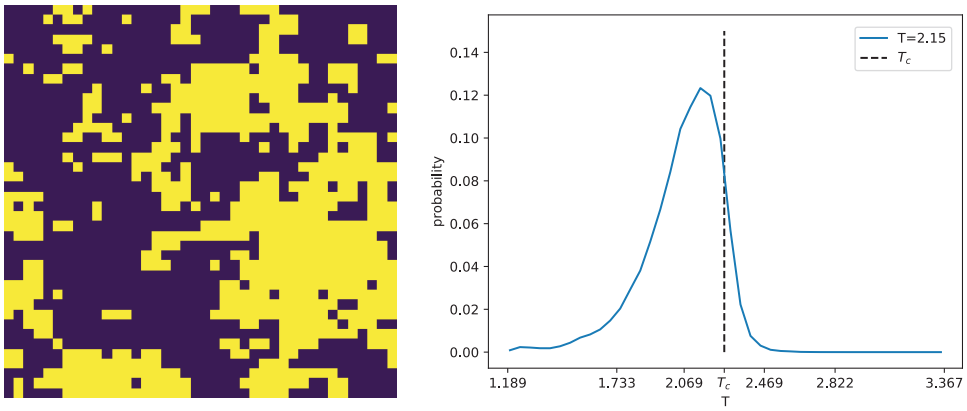


Figure 2.4: The input configuration (left) was created with an initial temperature $T = 2.1$. The neuronal network takes the configuration as an input and generates the discrete density distribution (right) as an output. In this example, the network has a median around the initial temperature, but also considers other temperatures as suitable, though not with the same probability.

As in the categorization problem before, the sum of the output activations can be interpreted as a probability with $p = p_1 + p_2 + \cdots + p_{44} = 1$. This normalization leads to a prediction that represents a *discrete probability density*. In figure 2.4 a typical output for a given input configuration is shown.

It is expected that on average, the median $\bar{p}$ of the distribution p matches the temperature $T \in T_{set}$ that was used to generate the configuration. Also, the shape of p contains information about the uncertainty of the prediction, meaning that the bigger the standard deviation $\sigma(p)$ of the distribution, the smaller the certainty for a specific temperature. With that information, a discussion about the decision-making of the network was motivated and will be of importance below.
To minimize the computational duration for generating configurations, testing data was inferred from the training data by applying symmetry transformations, including rotation, inversion and mirroring. In this way, unique lattices that correspond to a new and unique input vector were created without excessive use of the Monte Carlo algorithm. This is especially handy for large lattice sizes.

2.3.2 Results

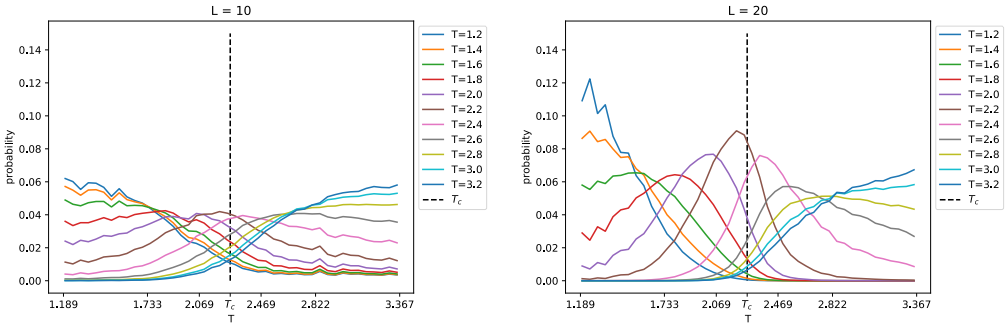
The final network uses L^2 input neurons, one hidden layer with 20 neurons and 44 output neurons, reflecting the number of elements in the set of temperatures used to create the input configurations, T_{set}. Again, a detailed comparison of different architectures should be added for future discussions. Repeating the example in

figure 2.4 many times for configurations with the same initial temperature and averaging their resulting probability distributions leads to a single curve in figure 2.7.

Comparing different lattice sizes L shows that $\sigma(p)$ is getting smaller for growing L. This is similar to the sharpening transition in the Ising phase in section 2.2 for bigger lattices and is also based on the fact that finite-size effects become smaller and smaller.
In addition, note that $\sigma(p(T))$ is getting smaller for $L = 40$ near T_c. This feature is particularly interesting because it was not expected that the network makes better predictions around the highly unstable critical point than for high and low temperatures. To explore this feature, it is needed to understand the basis on which the neuronal network makes a prediction.

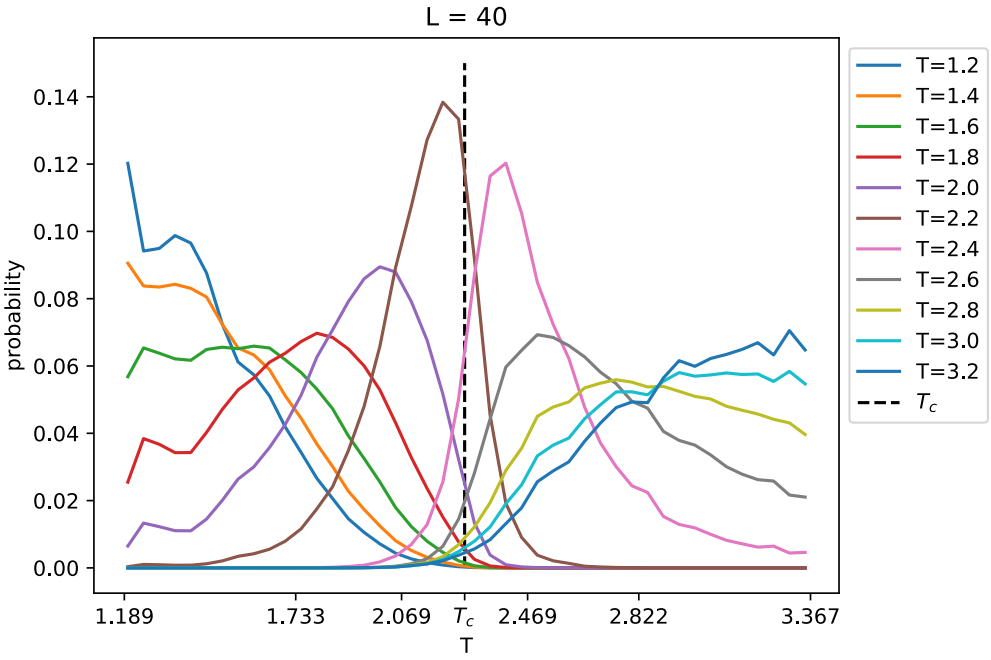
How the network decides

Let us assume the network is indeed intelligent and just uses the average value of all spins, the magnetization m (1.4), as an indicator for temperature. If this is the case, then the predicted temperatures of configurations should be nothing more than a prediction of their magnetization. Plotting the median $\bar{p}$ of each probability distribution p against the magnetization of the configurations is resulting in figure 2.6. This scatter plot follows the same statistics as the input configurations in figure 1.4. Both the spread and the change in the magnetization are exactly matched. This can be inferred as an indicator for the initial claim because predicting the temperature is the same as predicting the magnetization. Since we now know that the prediction of the network is based on the magnetization, we can also discuss why the predictions for $L = 40$ get better near the critical point.



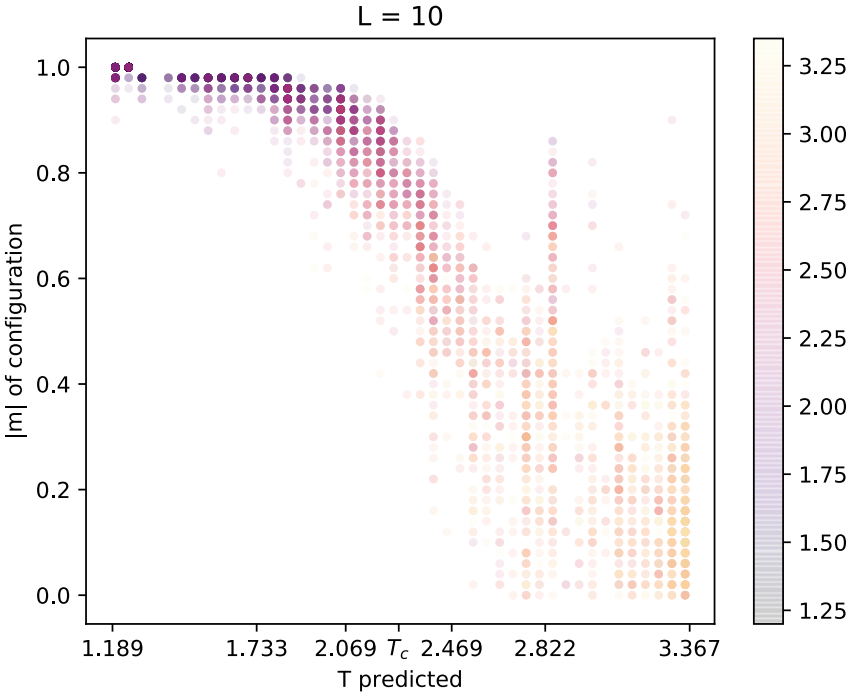
(a) $L = 10$

(b) $L = 20$

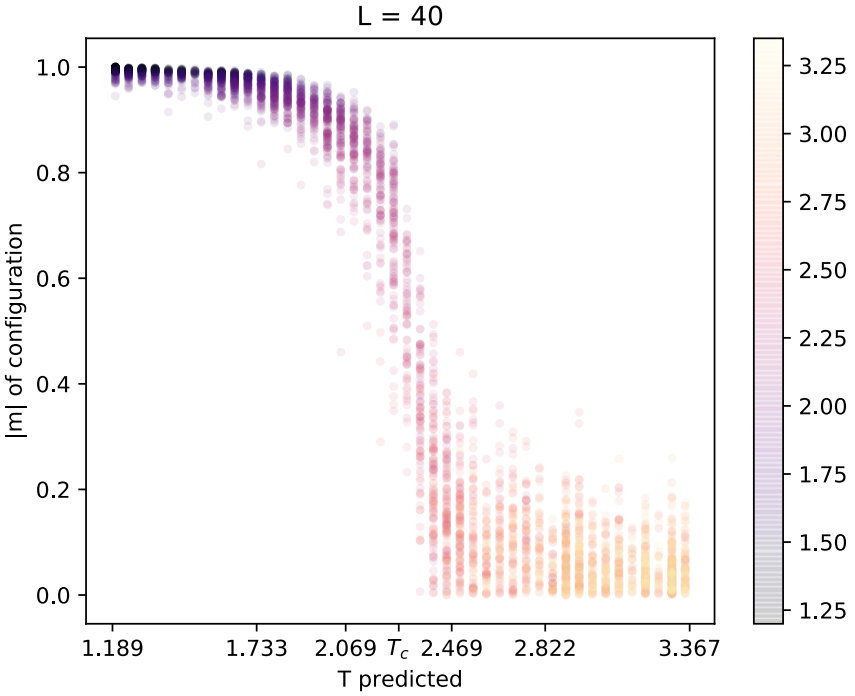


(c) $L = 40$

Figure 2.5: Each curve represents the average output of the neuronal network for configurations with different temperatures.



(a) $L = 10$



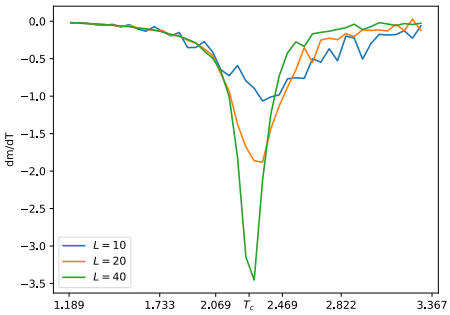
(b) $L = 40$

Figure 2.6: Each configuration is plotted according to its magnetization against the predicted temperature. The color of the resulting point indicates the temperature that was used for generating the configuration.

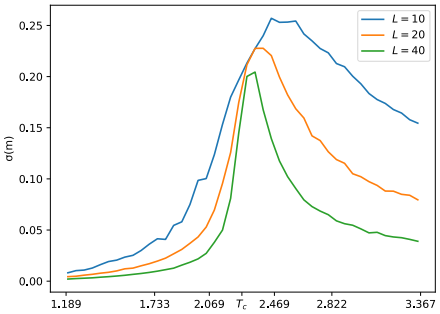
Effect of the magnetization on the prediction

We can already see the reason for better predictions near the critical point by looking at figure 1.4. There are two features of the magnetization that can influence the prediction quality: First, the change of the magnetization with respect to the temperature, $\frac{dm}{dT}$ is almost zero for high and low temperatures but increases rapidly near T_c. Therefore it is easier for the network to recognize distinct temperatures near T_c than in other regions. Second, the standard deviation of the magnetization $\sigma(m)$ is growing near T_c. The result are lower quality predictions since the input of the network is highly unstable. This effect is even higher for small lattice sizes and works inversely to the first effect, which improves the prediction.

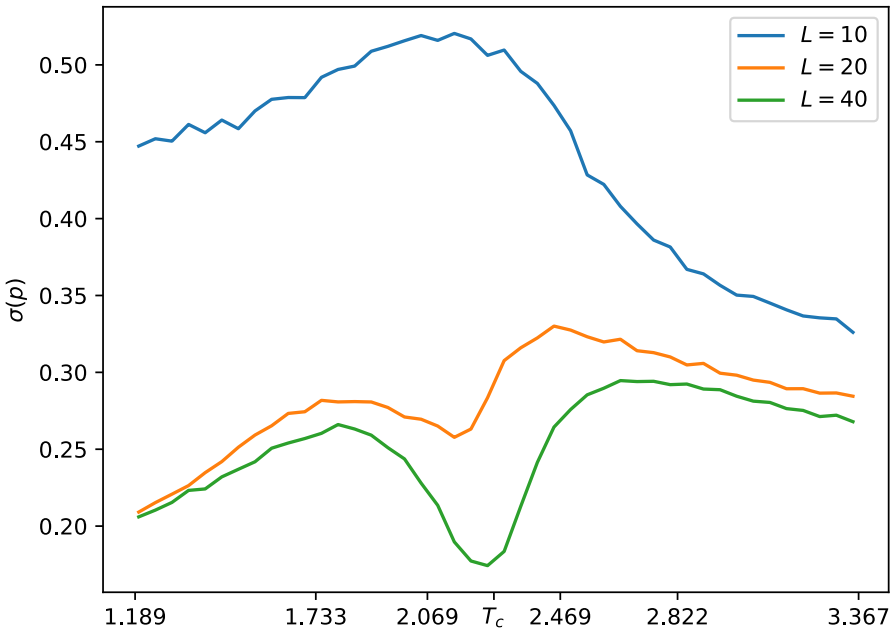
To get a better feeling for the two effects, $\frac{dm}{dT}$ and $\sigma(m)$ were calculated numerically and plotted in figure 2.7a and figure 2.7b for different lattice sizes. The total effect on the prediction is plotted in figure 2.7c. While a small L has both a high $\sigma(m)$ and a low $\frac{dm}{dT}$ near T_c, the exactly opposite is true for bigger lattice sizes. This results in a better prediction quality near T_c for bigger lattice sizes.



(a) $\frac{dm}{dT}$



(b) $\sigma(m)$



(c) $\sigma(p(T))$

Figure 2.7: Each curve represents the average output of the neuronal network for configurations with different temperatures.

Chapter 3

Conclusion

It was shown that Machine Learning Algorithms such as neuronal networks can play an important role in understanding the physical properties of a system. In our case, the fundamental processes of the Ising model were already known and the implementation of the neuronal network was just for testing and learning the handling of such algorithms. With modern libraries like TensorFlow [5] and Flux it is possible to implement these methods in a straightforward way without knowing all the details. This makes it even more important to keep track of the underlying mechanisms that lead to a prediction.

3.1 Results

It was successful to reproduce the results of the Melko paper [1]. Furthermore, it was possible to predict the critical temperature of the Ising model with a lattice size between 10 and 40 and ensembles of $N = 2000$ configurations as $\tilde{T}_c = 2.26 \pm 0.01$. As expected, the predictions got better for bigger lattice sizes.

In a new approach, it was possible to predict the temperatures of configurations of the Ising model. The output of the neuronal network was changed to a discrete number, corresponding to different temperatures. This has the benefit of fast computation times due to the hot-batch encoding. The output of the neuronal network can be interpreted as a probability distribution with a median and a standard deviation. Predictions are made according to the magnetization, which Figure 2.6 suggests. The prediction quality is depending on the input configurations and especially on their magnetization m and is a result of two main effects that are inverse to each other: The more m deviates from the ensemble average, the bigger the standard deviation $\sigma(p)$ of the prediction. Therefore, $\sigma(p)$ is especially high near T_c. In contrast, the rapidly changing average magnetization near T_c allows the network to distinguish easier between different temperatures. For bigger lattice sizes, the fluctuation of m decreases while the derivative dm/dT increases, leading to higher quality predictions near T_c.

3.2 Future Focus

As a first approach, the architecture of the neuronal network was not changed and compared to different approaches. The number of hidden layers could have been changed, as well as the number of neurons inside the network. It remains a method of trial and error which kind of structure results in good predictions. This open question should be reviewed in future works. In more general problems where there is no system variable like the magnetization, altering the network can lead to dramatic changes in the prediction.

The assumption the neuronal network decides according to the magnetization was inferred from Figure 2.6. This claim should face some more tests. One could create artificial configurations only according to the magnetization and compare the predictions. Suppose we make a copy of a configuration C called C' with magnetization m_C. We now shuffle the spins in configuration C' in a random fashion. After that, C' is an artificial configuration because it may have the same magnetization m_C, but its energy may be altered. If we now compare the prediction for both C and C' and they are almost identical, no matter what configuration C was in the first place, then the network highly relies on the magnetization.

3.3 Unsolved Problems

Using the method of a discrete set of neurons to predict temperatures was a workaround due to slow computation times. Even though this method works fine and even has some benefits like a probability distribution as an output, it would be interesting to compare its predictions to a single output neuron. This could not be achieved and is an unsolved problem.

Bibliography

[1] M. R. Carrasquilla, J., "Machine learning phases of matter," 2017.

[2] P. Mehta, M. Bukov, C.-H. Wang, A. G. R. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, "A high-bias, low-variance introduction to machine learning for physicists," 2019.

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[4] M. Innes, "Flux: Elegant machine learning with julia," *Journal of Open Source Software*, 2018.

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/