universität innsbruck
Fakultät für Mathematik,
Informatik und Physik

Bachelor Thesis

Generative modeling of spin systems

Florian Fürrutter

October 15, 2022

Supervisor: Ass.-Prof. Dr. Mathias S. Scheurer

Institute for Theoretical Physics

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.

Ich erkläre mich mit der Archivierung der vorliegenden Bachelorarbeit einverstanden.

Abstract

Einheiten sollten immer aufrecht (roman) gesetzt werden $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{C}, \mathbb{R}$. Am [20]

Contents

1. Introduction

In the last decade neural networks became the most relevant machine learning technique. Advancements in hardware performance allowed larger networks to be trained. Neural networks have an enormous field of applications due to their excellent generalization properties. Ongoing research, especially in generative models, showed that it is possible to generate indistinguishable fake images from training datasets consisting of real world photos.

f [20] [15] [12]

2. Neural networks

The most famous machine learning technique, neural networks, is a numerical method capable of modeling an enormous spectrum of problems. In 1943 the subject started with the creation of the first computational model of a neuronal network [13].

Neuronal networks are an arrangement of many basic units, namely the neurons. Initially, the artificial neurons were biological motivated and designed to resemble the behaviour of biological found neurons. A single neuron receives signals either from other neurons or external inputs. Within the cell body the inputs $\mathbf{x}$ get combined and the neuron outputs a corresponding signal y to next node in the network. The modern translation into a mathematical model is shown in Figure 2.1. First, the inputs get multiplied individually with corresponding weight training parameters $\mathbf{w}$. These weights are the influence strength of the individual inputs. The weighted inputs are summed up and an additional training parameter called the bias b is added. With this we are able to model any affine transformation. For real problems this will hardly be sufficient, therefore we apply the so-called activation function f on the summation result. The key point of the activation function is the nonlinearity. Recall, nonlinear system often behave chaotic. This is a reason why we are able to utilize neuronal networks as universal approximators [6, 24]. The latter is an important point, these networks are very powerful approximators but never exact.

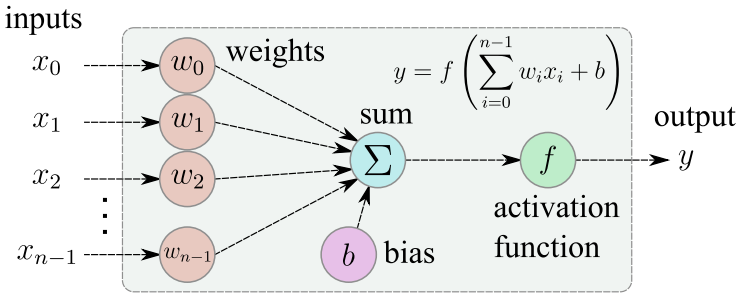


Figure 2.1.: Mathematical model of a single neuron. Given an input vector $\mathbf{x}$ with size n the neuron outputs a scalar y.

Within a network the activation function to choose is arbitrary. At the network output one has to consider the model purpose in order to utilize the correct activation function. The choice is depending on the output variables themselves, e.g. for object classification they will be probabilities and for estimating the temperature of same state the output is continuous and positive. The activation function ideally resembles this behaviour. Common activation functions are presented in Figure 2.2. The rectified linear unit (ReLU) Equation 2.1 is the non-negative identity and an example for the mentioned temperature estimation. LeakyReLu Equation 2.2 consists of two differently sloped lines and is commonly used within a network. When the problem demands a binary classification

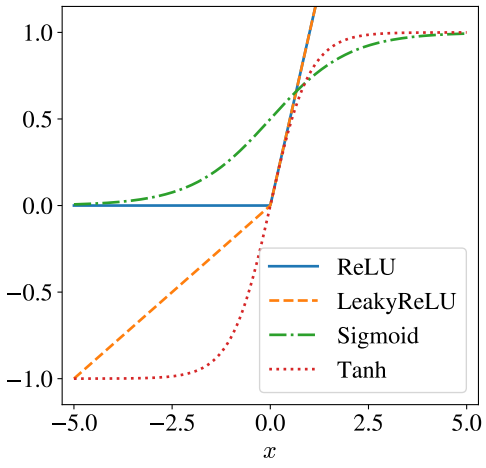


Figure 2.2.: Graphs of activation functions. For LeakyReLU $\alpha = 0.2$ is used.

Common activation functions:

$$f_1(x) = \max(0, x) \quad \dots \text{ ReLU} \tag{2.1}$$

$$f_2(x) = \begin{cases} x: & x \geq 0 \\ \alpha x: & x < 0 \end{cases} \quad \dots \text{ LeakyReLU} \tag{2.2}$$

$$f_3(x) = \frac{1}{1 + \exp(-x)} \quad \dots \text{ Sigmoid} \tag{2.3}$$

$$f_4(x) = \tanh(x) \quad \dots \text{ Tanh} \tag{2.4}$$

then the sigmoid function Equation 2.3 is used at the output. The reason is that logistic regression tells us the probability for binary draws has a sigmoid dependency. In chapter 3 we discuss the Ising model with output variables ± 1. Hence, we use the hyperbolic tangent Equation 2.4 as the output activation function to directly model the output.

2.1. Dense layer

A neural network can contain numerous neurons. Normally, networks are abstracted in terms of layers. Each layer is a collection of neurons arranged in some specific way. Furthermore, a single layer has a number of defined inputs and outputs. Model architectures specify the type of utilized layers and how to connect them.

The simplest layer is the dense-connected layer where a bunch of neurons are placed in parallel. They share the same inputs $\mathbf{x}$ that are connected to every neuron, with its own corresponding weights and bias. Every neuron has one output y. Hence, the dense layer has the same number of outputs $\mathbf{y}$ as the number of neurons in the layer.

Definition 2.1.1 (Dense connected layer)**.** *Consider a layer with n inputs $\mathbf{x} \in \mathbb{R}^n$ and m parallel neurons with outputs $\mathbf{y} \in \mathbb{R}^m$. We call the layer with the operation*

$$\mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{2.5}$$

a dense connected layer. Here $\mathbf{W} \in Mat_{m,n}(\mathbb{R})$ is the weight matrix, $\mathbf{b} \in \mathbb{R}^m$ the bias vector and f an activation function that is applied element-wise.

An arrangement of layers in series is called a straight feedforward configuration. Like a chain, starting from the model input, every layer is only connected to the next one until the model output is reached. We present an example of such a Deep-Neural-Network in Figure 2.3. A hidden layer is a layer that is not model input nor output. The word deep refers to the fact that hidden layers exist in the network.

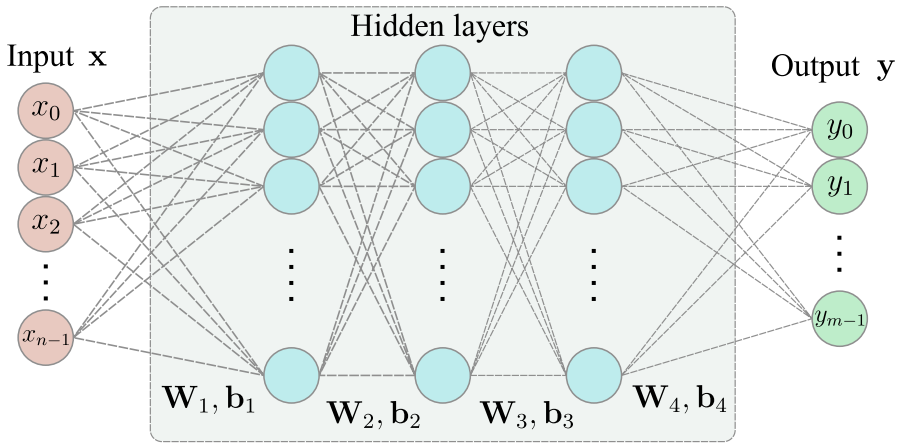


Figure 2.3.: Example of a Deep-Neural-Network (DNN). A straight feedforward configuration of three hidden layers, an input **x** and output **y**. Each connection step has its own weights **W** and biases **b**. The neuron count per layer is an arbitrary number depending on the architecture.

A nice property of these networks is the proven universal function approximation capability, summarized in the *universal approximation theorem* [6, 24]. The general statement is that a dense network with one hidden layer is sufficient to approximate every continuous function with a compact domain to any accuracy. No prediction about the number of neurons in the hidden layer or how to optimize the weights (training parameters) is made. The problem remains that training neuronal networks is still not trivial.

In practice networks only consisting of dense layers suffer from some additional issues that limit the accuracy. For the following points it is helpful to imagine images of a chair with the goal to train a network that tells us were the chair is located in the image.

- **Not considering locality:** In the sense that every neuron is connected to all the inputs disregarding their position. Shuffling the model input (e.g. an image) for the whole data set only changes the location of the weights in **W** and not their value. Clearly, for us a shuffled image of a chair will look more like random noise than a chair.
- **Not translation invariant:** A chair remains a chair, regardless where in the image it is located. The dense network has different weights for different inputs. Thus, a translation in the input also changes the corresponding weights.
- **Overfitting:** A well known problem of neural networks where the accuracy on the training set is high, but performance on test data is poor. Especially too large/deep dense networks with too many training parameters suffer from this problem.

2.2. Convolution layer

Motivated by visual cortex cells and the translation invariance problem of dense layers the convolutional layer emerged [3, 11]. The key idea is to consider a layer of only a few

neurons called the kernel or filter. The inputs of these neurons are only connected to local layer inputs. Moving the kernel over the whole input (image or tensor) allows for translation invariant local feature extraction. Mathematical we call such an operation convolution. One kernel produces one output (image or tensor). Normally one uses n kernels per convolutional layer to extract multiple features. The number n has multiple names: filter-size, features or output channels.

Definition 2.2.1 (2D Convolutional layer)**.** *Consider an equilateral kernel size k (for simpler notation $k_x = k_y = k$) and a filter-size of n. The layer input tensor has the height h, width w and c channels. With $i = 1, \dots, n$ the feature index: kernel $\mathbf{K}^{(i)} \in Mat_k(\mathbb{R}^c)$ and $\mathbf{Y}^{(i)} \in Mat_{h',w'}(\mathbb{R})$. Further, the input $\mathbf{X} \in Mat_{h,w}(\mathbb{R}^c)$ and bias $\mathbf{b} \in \mathbb{R}^n$. We call a layer with the operation*

$$\mathbf{Y}^{(i)} = f(\mathbf{K}^{(i)} * \mathbf{X} + b_i \cdot \mathbb{1}_{h',w'}) \tag{2.6}$$

a convolutional layer with the concatenation of all $\mathbf{Y}^{(i)}$ as the layer output. Here $\mathbb{1}$ is the matrix of ones, f an activation function and $*$ is the discrete convolution operation

$$\left(\mathbf{K}^{(i)} * \mathbf{X}\right)_{j,l} = \sum_{\alpha,\beta=0}^{k-1} \mathbf{k}^{(i)}_{\alpha,\beta} \cdot \mathbf{x}_{j+\alpha-d,\, l+\beta-d} \quad \textit{with} \quad d = \begin{cases} (k-1)/2 & : k \textit{ odd} \\ k/2 - 1 & : k \textit{ even} \end{cases}. \tag{2.7}$$

Note, elements of $\mathbf{K}^{(i)}, \mathbf{X}$ are vectors $\mathbf{x}, \mathbf{k} \in \mathbb{R}^c$. The range of j, l and the output dimensions h', w' depend on the stride s and padding p (e.g. zeros, periodic, nearest, ...).

The stride s indicates how many pixels the kernel is moved for every evaluation of Equation 2.7 and the input $\mathbf{X}$ is padded with p pixels. We can calculate the output dimensions of such a 2D convolutional layer with

$$(h', w', c') = \left(\left\lfloor \frac{h - k + 2p}{s} + 1 \right\rfloor, \left\lfloor \frac{w - k + 2p}{s} + 1 \right\rfloor, n \right), \tag{2.8}$$

where $\lfloor \cdot \rfloor$ is the floor operator. Furthermore, the convolution operator can be written as a matrix vector multiplication. For this we have to construct a sparse matrix $\mathbf{G}$ with circular double blocked matrices and flatten out the matrix $\mathbf{X}$ into a vector. The construction is not hard but tedious. Hence, we do not write it explicitly here. Just note that matrix multiplication can be easily optimized and parallelized.

For one convolutional layer we have the training parameters of the kernel $\mathbf{K} \in \mathbb{R}^{n \times k_x \times k_y \times c}$ and the bias $\mathbf{b} \in \mathbb{R}^n$. The parameters stay the same when we change the input dimensions height h and width w. Usually one combines convolutional and dense layers in a straight feedforward model architecture for classification tasks.

2.3. Training

We train feedforward neuronal networks with the backpropagation algorithm, first applied to neuronal nets in 1974 [22]. Backpropagation is used for efficiently minimizing

the expectancy value of a loss/cost/energy function $\mathcal{L}$. One is likely familiar with the mean squared error from linear regression. This is an example of extremizing a defined loss function. In our case the networks have numerous training parameters. Therefore, the optimization landscape of $\mathcal{L}$ possesses a high complexity with many local minima. The optimization is based on the gradient of the loss function with respect to the training parameters. Backpropagation describes the method to calculate the gradients of every layer consecutively from the back. Hence, derivatives given by the chain rule are evaluated only once and the algorithm removes this redundancy from direct gradient calculation.

In this section we regard the neuronal network as a black-box $\mathrm{Net}(\mathbf{x})$ with some input $\mathbf{x}$. The models weights $\mathbf{W}$ are all training parameters (weights and biases combined). The goal is to iteratively minimize the loss function (e.g. mean squared error) evaluated with the training data while varying the model weights

$$\mathcal{L} = \mathcal{L}\left(\mathrm{Net}(\mathbf{x}), \mathbf{y}\right) \quad \text{with} \quad \{\mathbf{x}, \mathbf{y}\} \text{ the training set.} \tag{2.9}$$

The training set consists of inputs $\mathbf{x}$ and corresponding outputs $\mathbf{y}$ (labels). We consider the gradient of $\mathcal{L}$ in the weight space and evaluate $\boldsymbol{\nabla}_{\mathbf{W}}\mathcal{L}$ while considering the training set as constant. How to update the weights according to the gradient is specified by the used optimizer algorithm. A simple example is the stochastic gradient descend optimizer.

Definition 2.3.1 (Stochastic gradient descend, SGD)**.** *One iterative update of the model weights* $\mathbf{W}$ *is done with*

$$\mathbf{W}' = \mathbf{W} - \lambda \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{\nabla}_{\mathbf{W}} \mathcal{L}(\mathrm{Net}(\mathbf{x}_i), \mathbf{y}_i) \quad \textit{where} \quad (\mathbf{x}_i, \mathbf{y}_i) \in \text{Train subset.} \tag{2.10}$$

Here n *is called the batch size and* λ *is the learning rate hyperparameter. Further, the train subset is drawn consecutively from the whole train set.*

The stochastic word refers to the average over the drawn subsets. A whole pass of the train set is called epoch. Choosing the batch size n is a trade-off between how many weight updates are done per epoch and better global minima convergence. Furthermore, less updates per epoch are more efficient because writing weights is expensive. In Figure 2.4 we illustrate the iterative updating process with convergence into a local minimum.

The SGD can be improved to insure faster convergence and local minima escape capabilities. There exist many such optimizer algorithms, each with their pro and cons. An often used choice is the Adam (Adaptive Moment Estimation) optimizer [10].

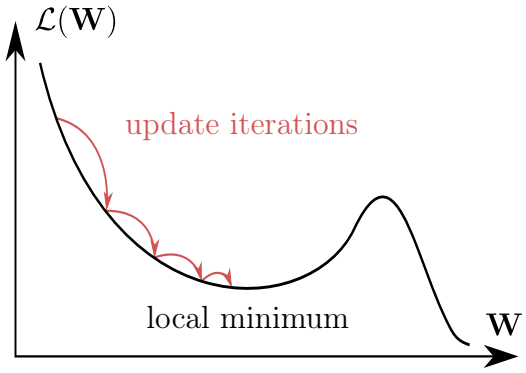


Figure 2.4.: Illustration of the SGD optimizer with iterative updates.

2.4. Generative adversarial networks

The concept of a generative adversarial network (GAN) [4] is special type of neuronal network architecture. GANs are generative unsupervised learning approaches where we do not supply a training set of input-output tuples $(\mathbf{x}, \mathbf{y})$. Instead, we sample data from an unknown distribution (e.g. taking landscape photos) and note it as real data. Our GAN then learns to generate samples from the hidden distribution. Hence, the GAN learns the distribution implicitly. The method is capable to generate unseen photo realistic images of provided real data [8]. This inspires us to apply GAN techniques to physical systems.

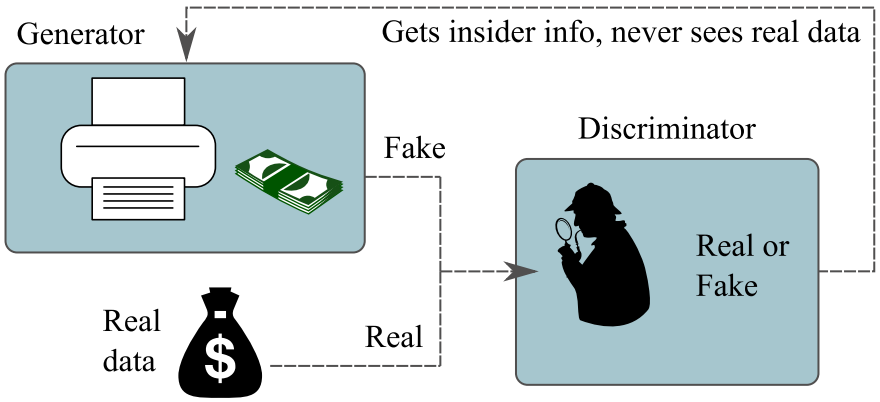


Figure 2.5.: Conceptual illustration of a generative adversarial network (GAN), which goal is to produce fake data that is indistinguishable from the training set. The generator learns to produce these fake samples and gets feedback from the discriminator. The training data is only provided to the discriminator, that has to classify samples as real or fake. These players must stay in equilibrium for convergence.

The GAN architecture specifies two connected networks called generator and discriminator. The generator is the part that generates fake images and the discriminator's goal is to judge if a provided sample is real or fake. In Figure 2.5 a conceptual illustration of the GAN working principle is shown.

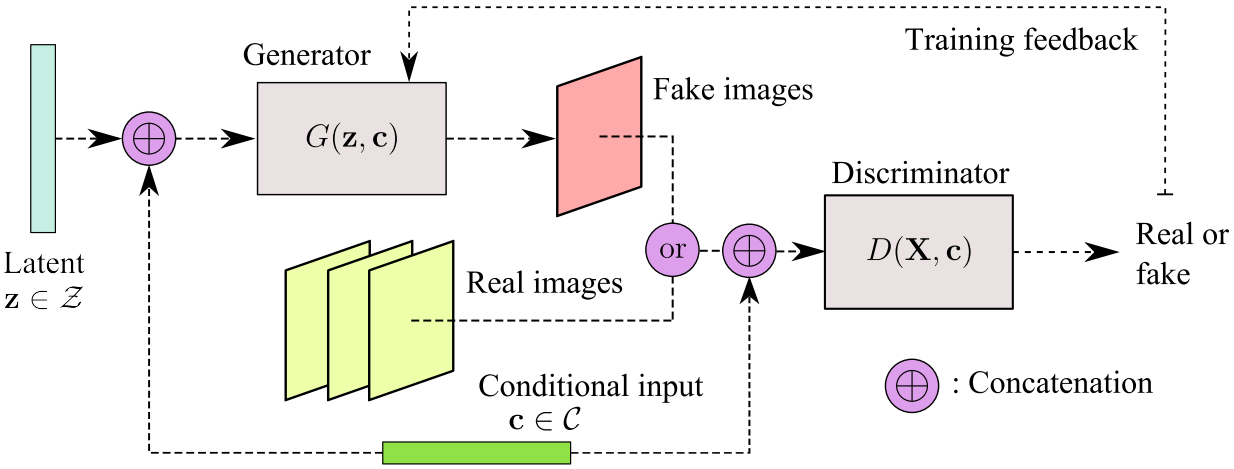


Figure 2.6.: Conditional GAN. The generator gets the concatenation of a latent vector $\mathbf{z}$ and a conditional vector $\mathbf{c}$ as input. The condition $\mathbf{c}$ with a data sample $\mathbf{X}$ is presented to the discriminator. This tuple is either generated or from the training set.

2.4.1. Mathematical description

We are interested in the conditional generative adversarial network, a variant of GAN. Here, we extend the GAN with an additional input which allows us to specify what data we want to generate. The conditional input $\mathbf{c}$ can consist of any known property of the real data. In our case the condition will be the temperature of the physical system. A block diagram of the conditional GAN is presented in Figure 2.6. The generator $G : (\mathbf{z}, \mathbf{c}) \mapsto \mathbf{X}$ takes as input a latent vector $\mathbf{z}$ sampled from the latent space $\mathcal{Z}$. Additionally, in the case of a conditional GAN also a condition $\mathbf{c}$ from condition space $\mathcal{C}$. The discriminator $D : (\mathbf{X}, \mathbf{c}) \mapsto (0, 1)$ receives a real or fake sample $\mathbf{X}$ and optionally the corresponding specifier $\mathbf{c}$. Furthermore, we extend the architecture with an auxiliary network $A : \mathbf{X} \mapsto \mathbf{c}$ that trains on real data to classifies the condition $\mathbf{c}$. The pre-trained auxiliary network then classifies fake samples and provides more information to the generator. It was shown, this is helping to stabilize the GAN network training [16, 20]. Mathematical we write the specific GAN optimization goal of this thesis as:

Definition 2.4.1 (Conditional GAN, player goals)**.**
Generator:

$$\min_G \Big(\quad \mathbb{E}_{\mathbf{z}\sim p_z,\, \mathbf{c}\sim p_c} \left[\mathcal{L}_G(D(G(\mathbf{z}, \mathbf{c}), \mathbf{c}), \text{Real})\right] \tag{2.11}$$

$$+ \mu \quad \mathbb{E}_{\mathbf{z}\sim p_z,\, \mathbf{c}\sim p_c} \left[\mathcal{L}_A(A(G(\mathbf{z}, \mathbf{c})), \mathbf{c})\right] \Big) \tag{2.12}$$

Discriminator:

$$\min_D \Big(\quad \mathbb{E}_{\mathbf{z}\sim p_z,\, \mathbf{c}\sim p_c} \left[\mathcal{L}_D(D(G(\mathbf{z}, \mathbf{c}), \mathbf{c}), \text{Fake})\right] \tag{2.13}$$

$$+ \quad \mathbb{E}_{(\mathbf{X},\mathbf{c})\sim p_{\text{data}}} \left[\mathcal{L}_D(D(\mathbf{X}, \mathbf{c}), \text{Real})\right] \tag{2.14}$$

$$+\frac{\gamma}{2} \quad \mathbb{E}_{(\mathbf{X},\mathbf{c})\sim p_{\text{data}}} \left[\left\|\boldsymbol{\nabla}_{(\mathbf{X},\mathbf{c})} D(\mathbf{X}, \mathbf{c})\right\|^2\right] \Big) \tag{2.15}$$

Auxiliary network:

$$\min_A \Big(\; \mathbb{E}_{(\mathbf{X},\mathbf{c})\sim p_{\text{data}}} \left[\mathcal{L}_A(A(\mathbf{X}), \mathbf{c})\right] \Big) \tag{2.16}$$

Here μ,$\gamma \in \mathbb{R}^+$ are hyperparameters. The expressions should be minimized with varying the model parameters of G, D and A. Each player has a corresponding loss function $\mathcal{L}$. Note that, the generator and discriminator have contradictory goals and need to be in equilibrium. The Equation 2.15 is a regularizer term. It penalizes large discriminator gradients evaluated for real data. With this GAN convergence was shown [14], when model parameters are initialized sufficiently close to the equilibrium point.

2.4.2. Convergence

Generative adversarial networks are still objects of ongoing research. Hence, there exist a few not completely solved common problems.

In flat areas of the loss function landscape small gradients occur. Small derivatives get multiplied together because of the chain rule and the final result gets tiny. Weight optimizer use the gradient to update the model parameters. Is the gradient value vanishing then we get stuck because the weights will not change noticeable. We call this the vanishing gradient problem. Some authors suggest this happens that if the discriminator is too good [1].

GAN equilibrium states are fleeting. Is the generator good enough to generate perfect fake images the discriminator can only guess if a presented state is real or fake. This random feedback will destroy the equilibrium states of the generator and the model performance collapses.

mode collapse
The generator maps the latent space only to a subset of the desired image space.

3. Ising model

Advancements in *quantum theory*, especially the inference of the spin concept, made a basic understanding of para- and ferromagnetism in solid matter possible. In 1924 a statistical physics model of ferromagnetism was proposed by Ernst Ising [7], namely the broadly known Ising model. The mathematical model describes spins at discrete lattice sides that can be considered either classically or quantum theoretically. We restrict ourselves here to the classical Ising model where the spin at lattice point i is discrete either up or down $\sigma_i \in \{\pm 1\}$. The general Hamiltonian is given with:

$$H = -\sum_{\langle i,j \rangle} J_{i,j}\ \sigma_i \sigma_j - \sum_i h_i \sigma_i. \tag{3.1}$$

Here $\langle i,j \rangle$ means the sum over all adjacent lattice sides and $J_{i,j}$ is the exchange interaction strength between spin i and j. The influence of an external field is denoted as h_i. This Hamiltonian can be studied for an any dimensional lattice where the number of nearest neighbors z is the only thing that changes, e.g. for two dimensions a spin has four neighboring spins $z = 4$ and in three dimensions $z = 6$.

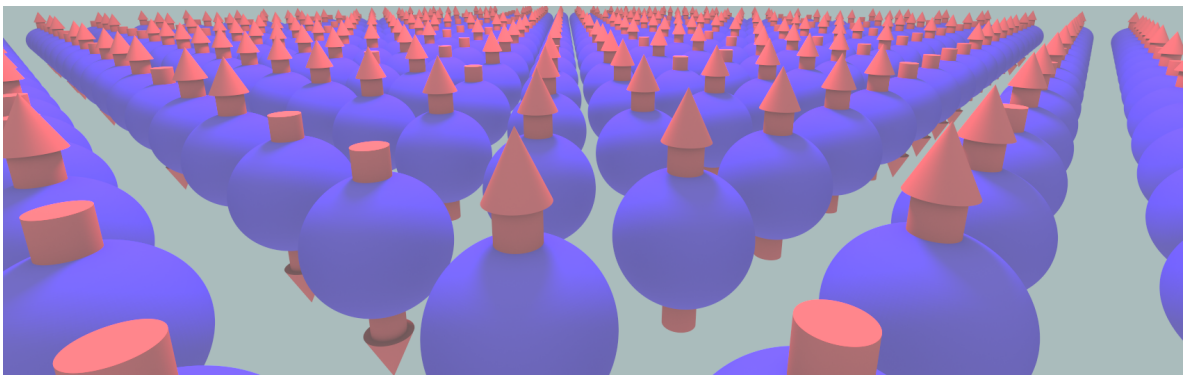


Figure 3.1.: Illustration of a two-dimensional lattice with spins either pointing up or down.

In this thesis we focus on the two-dimensional case with a global interaction strength $J = J_{i,j} > 0$ and no external field $h_i = 0$. Therefore, Equation 3.1 reduces to

$$H = -J \sum_{\langle i,j \rangle} \sigma_i \sigma_j. \tag{3.2}$$

This was the first model to exhibit a continuous phase transition for positive temperatures $T > 0$. Statistical physics yields with the canonical ensemble a critical temperature T_c, which divides the temperature range in two phases. For $T < T_c$ the spins will be

mostly aligned and the system ferromagnetic. On the other hand, for $T_c < T$ the thermodynamic fluctuations randomly flip the spins and the system behaves paramagnetic with no mean magnetization. In 1944 an analytic solution of the critical temperature was derived by Lars Onsager [17]

$$T_c = J\frac{2}{k_\mathrm{B}\ln\left(1+\sqrt{2}\right)} \approx 2.269\frac{J}{k_\mathrm{B}}, \tag{3.3}$$

where k_B is the Boltzmann constant. For further consideration the lattice is squared and finite with L spins per side and $N = L^2$ total spins. We note a configuration or microstate of the system with $\mathbf{\Sigma}$. For example for $L = 4$ and $N = 16$ we use two notations

$$\mathbf{\Sigma} := \begin{pmatrix} \sigma_{(0,0)} & \sigma_{(0,1)} & \sigma_{(0,2)} & \sigma_{(0,3)} \\ \sigma_{(1,0)} & \sigma_{(1,1)} & \sigma_{(1,2)} & \sigma_{(1,3)} \\ \sigma_{(2,0)} & \sigma_{(2,1)} & \sigma_{(2,2)} & \sigma_{(2,3)} \\ \sigma_{(3,0)} & \sigma_{(3,1)} & \sigma_{(3,2)} & \sigma_{(3,3)} \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} \sigma_0 & \sigma_1 & \sigma_2 & \sigma_3 \\ \sigma_4 & \sigma_5 & \sigma_6 & \sigma_7 \\ \sigma_8 & \sigma_9 & \sigma_{10} & \sigma_{11} \\ \sigma_{12} & \sigma_{13} & \sigma_{14} & \sigma_{15} \end{pmatrix}. \tag{3.4}$$

We consider periodic boundary conditions for our finite system sizes.

3.1. Important observables

Statistical physics and thermodynamics specifies many observables. Here, the most important one is the magnetization m that is the order-parameter for this system. The energies E are the eigenvalues of the Hamiltonian and specify the macrostates. Further, the magnetic susceptibility χ and the correlation length ξ characterize the critical temperate T_c at which their values diverge in the thermodynamic limit.

3.1.1. Magnetization

The magnetization m is the order-parameter of the Ising model and given with the normalized sum over the spins

$$m := \frac{1}{N}\sum_{i=0}^{N-1}\sigma_i. \tag{3.5}$$

Naively looking at the $\mathbb{Z}_2$ symmetry of the Hamiltonian may temp one to think m is always vanishing on average. The argument is that every microstate has a mirrored state with the same energy and therefore the same probability. This is only the case for temperatures above the critical temperature. Below T_c spontaneous symmetry breaking occurs and $m \neq 0$. A observable with this behaviour is called order-parameter.

Corollary 3.1.1. *The mean of one spin is the mean of the total magnetization*

$$\langle\sigma_i\rangle = \frac{1}{N}\sum_{i=0}^{N-1}\langle\sigma_i\rangle = \left\langle\frac{1}{N}\sum_{i=0}^{N-1}\sigma_i\right\rangle = \langle m\rangle\,. \tag{3.6}$$

Proof. One uses the translation invariance of the Ising model to apply the spatial average and then the linearity of the mean. □

3.1.2. Energy

The energy E classifies a macrostate. It includes all microstates $\mathbf{\Sigma}$ with the same energy. We get the energy by simply inserting the state into the Hamiltonian

$$E := \frac{1}{N} H(\mathbf{\Sigma}) = -\frac{J}{N} \sum_{\langle i,j \rangle} \sigma_i \sigma_j. \tag{3.7}$$

To compare the energy for different system sizes we normalize the energy per spin.

3.1.3. Magnetic susceptibility

The magnetic susceptibility χ is the linear response coefficient of the system to an applied magnetic field

$$\chi := \left(\frac{\partial m}{\partial h} \right)_{T,\, h=0}. \tag{3.8}$$

For the direct usage this definition we have to consider the general Hamiltonian from Equation 3.1 and simulate the system for different external field strengths. Statistical physics allows us to rewrite χ such that the simpler Equation 3.2 is sufficient.

Theorem 3.1.1. *The magnetic susceptibility can be evaluated with fluctuations of the magnetization in thermodynamic equilibrium*

$$\chi = \left(\frac{\partial m}{\partial h} \right)_{T,\, h=0} = \frac{N}{k_B T} \operatorname{Var}\left[|m| \right] = \frac{N}{k_B T} \left(\left\langle m^2 \right\rangle - \left\langle |m| \right\rangle^2 \right). \tag{3.9}$$

Proof. Either use the fluctuation-response theorem for $M = Nm$ or evaluate the derivative directly. Notating $Z = Z(T, h)$ the partition function and $\beta = 1/k_{\mathrm{B}} T$:

$$\begin{aligned}
\left(\frac{\partial m}{\partial h} \right)_{T,\, h=0} &= \left(\frac{\partial}{\partial h} \frac{1}{NZ} \operatorname{Tr}\left[M \exp(-\beta H) \right] \right)_{T,\, h=0} \\
&= \left(-\frac{1}{NZ^2} \operatorname{Tr}\left[M \exp(-\beta H) \right] \frac{\partial Z}{\partial h} + \frac{1}{NZ} \operatorname{Tr}\left[M \frac{\partial}{\partial h} \exp(-\beta H) \right] \right)_{T,\, h=0} \\
&= \left(-\frac{\beta}{NZ^2} \operatorname{Tr}\left[M \exp(-\beta H) \right]^2 + \frac{\beta}{NZ} \operatorname{Tr}\left[M^2 \exp(-\beta H) \right] \right)_{T,\, h=0} \\
&= -N\beta \left\langle m \right\rangle^2 + N\beta \left\langle m^2 \right\rangle \\
&= N\beta \left(\left\langle m^2 \right\rangle - \left\langle |m| \right\rangle^2 \right).
\end{aligned} \tag{3.10}$$

In the last step we used the substitution $m \mapsto |m|$. We are allowed to do so because nothing changes. In the paramagnetic phase where $m = 0$ it is clear and in the ferromagnetic regime the sign of $m \neq 0$ is not relevant for the physical behaviour, the up down orientation is arbitrary. The substitution reason is the Monte-Carlo-simulation, where states with the same energy have the same probability and therefore $\langle m \rangle$ would be always zero $\forall T$. In that sense the spontaneous symmetry breaking is missed out and we account for that with the absolute magnetization. □

3.1.4. Spin-spin correlation

The spin-spin correlation is defined as the covariance between two spin variables

$$G_c(i,j) := \mathrm{Cov}\left[\sigma_i, \sigma_j\right] = \langle \sigma_i \sigma_j \rangle - \langle \sigma_i \rangle \langle \sigma_j \rangle . \tag{3.11}$$

Recall the spin σ_i is located at the lattice point $\mathbf{r}_i$. Our Ising Hamiltonian Equation 3.2 is clearly translation invariant. Hence, the correlation function only depends on the relative vector $\mathbf{r} = \mathbf{r}_j - \mathbf{r}_i$ between the spin positions

$$G_c(\mathbf{r}) = G_c(\mathbf{r}_i, \mathbf{r}_i + \mathbf{r}). \tag{3.12}$$

Because G_c is independent of $\mathbf{r}_i$ we average over all combinations where $\mathbf{r}$ occurs and therefore improve our estimation. With Corollary 3.1.1 we get

$$G_c(\mathbf{r}) = \frac{1}{N} \sum_{i,j=0}^{N-1} \delta_{(\mathbf{r}_i - \mathbf{r}_j),\mathbf{r}} \ \left[\langle \sigma_i \sigma_j \rangle - \langle m \rangle^2 \right]. \tag{3.13}$$

Theorem 3.1.2. *The spin-spin correlation function can be calculated efficiently with a fast Fourier transform:*

$$G_c(\mathbf{r}) = (\mathcal{F}^{-1} g_c)(\mathbf{r}) \quad \text{with} \quad g_c(\mathbf{k}) = \frac{1}{N} \begin{cases} \left\langle \left| (\mathcal{F}\sigma_i)(\mathbf{k}) \right|^2 \right\rangle : & \mathbf{k} \neq 0 \\ 0 : & \mathbf{k} = 0 \end{cases}. \tag{3.14}$$

Proof. With Equation 3.13 we could calculate G_c directly for all $\mathbf{r}$ with a time effort of $\mathcal{O}(N^2)$. We are able to improve this, with applying a discrete Fourier transform

$$\begin{aligned}
g_c(\mathbf{k}) = (\mathcal{F} G_c)(\mathbf{k}) &= \sum_{\mathbf{r}} \exp\left(-\mathrm{i} \frac{2\pi}{L} \mathbf{r} \cdot \mathbf{k} \right) G_c(\mathbf{r}) \\
&= \sum_{\mathbf{r}} \exp\left(-\mathrm{i} \frac{2\pi}{L} \mathbf{r} \cdot \mathbf{k} \right) \frac{1}{N} \sum_{i,j=0}^{N-1} \delta_{(\mathbf{r}_i - \mathbf{r}_j),\mathbf{r}} \ \left[\langle \sigma_i \sigma_j \rangle - \langle m \rangle^2 \right] \\
&= \frac{1}{N} \sum_{i,j=0}^{N-1} \exp\left(-\mathrm{i} \frac{2\pi}{L} \left[\mathbf{r}_i - \mathbf{r}_j \right] \cdot \mathbf{k} \right) \left[\langle \sigma_i \sigma_j \rangle - \langle m \rangle^2 \right] \\
&= \frac{1}{N} \left\langle \sum_{i=0}^{N-1} \exp\left(-\mathrm{i} \frac{2\pi}{L} \mathbf{r}_i \cdot \mathbf{k} \right) \left[\sigma_i - \langle m \rangle \right] \cdot \sum_{j=0}^{N-1} \exp\left(\mathrm{i} \frac{2\pi}{L} \mathbf{r}_j \cdot \mathbf{k} \right) \left[\sigma_j - \langle m \rangle \right] \right\rangle \\
&= \frac{1}{N} \left\langle \left| \sum_{i=0}^{N-1} \exp\left(-\mathrm{i} \frac{2\pi}{L} \mathbf{r}_i \cdot \mathbf{k} \right) \left[\sigma_i - \langle m \rangle \right] \right|^2 \right\rangle \\
&= \frac{1}{N} \left\langle \left| \left(\mathcal{F} \left[\sigma_i - \langle m \rangle \right] \right)(\mathbf{k}) \right|^2 \right\rangle .
\end{aligned} \tag{3.15}$$

Note, in the last line the Fourier transform is d-dimensional because the σ_i are arranged in a lattice with positions $\mathbf{r}_i$. In our case it will be a 2D transform over a microstate $\mathbf{\Sigma}$. A further simplification can be made with substituting $\sigma_i - \langle m \rangle \mapsto \sigma_i$. We get

$$g_c'(\mathbf{k}) = \frac{1}{N} \left\langle \left| (\mathcal{F}\sigma_i)(\mathbf{k}) \right|^2 \right\rangle . \tag{3.16}$$

If we fix $g_c'(0) = 0$ then $g_c' = g_c$. This results from the definition of the Fourier transform where $\mathbf{k} = 0$ is just the average. In other words we remove the offset $\langle m \rangle$ afterwards.

In the implementation we calculate the fast Fourier transform (FFT) of σ_i for samples $\mathbf{\Sigma}$ and take the elements-wise absolute square. Then we evaluate Equation 3.16 and set $g_c'(0) = 0$. The final result can then be inverse fast Fourier transformed. This yields directly $G_c(\mathbf{r})$ with a time effort of $\mathcal{O}(N \log N)$. □

The Hamiltonian is also rotation invariant. Hence, the correlation function only depends on the absolute value of the distance vector $\mathbf{r}$

$$G_c(r) = G_c(\mathbf{r}) \quad \text{with} \quad r = |\mathbf{r}|. \tag{3.17}$$

We improve the estimator with averaging over all angular directions in distance of r. Note that $r \leq N/2$. This is because we utilize periodic boundary conditions and the correlation function is also therefore also periodic in r. Analytic results from *mean field theory* state for $T \neq T_c$

$$G_c(r) \propto \exp\left(-\frac{r}{\xi}\right), \tag{3.18}$$

where ξ is defined as the correlation length. We estimate ξ with fitting Equation 3.18 to the angular averaged result of Equation 3.17.

3.2. Phase transition

The Ising model without external field undergoes a phase transition at T_c for a dimension $d \geq 2$. The transition is classified by the continuous order-parameter m.

At high temperatures $T_c < T$ the mean magnetization vanishes to thermodynamic fluctuations $\langle m \rangle = 0$ and the external field response is paramagnetic. Low temperatures allow spontaneous symmetry breaking to occur and align most of the spins. A mean magnetization $\langle m \rangle \neq 0$ lets the system behave ferromagnetic.

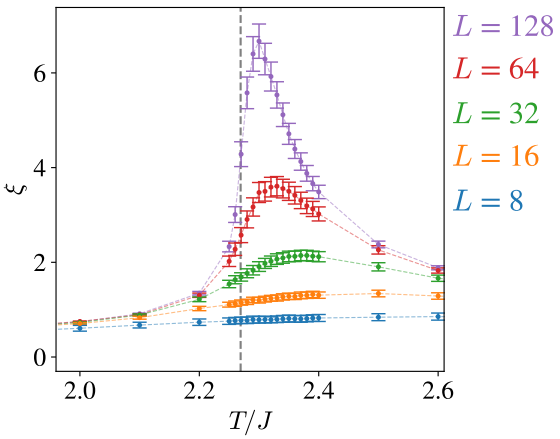


Figure 3.2.: Correlation length ξ for different system sizes L. The critical temperature T_c is marked with a vertical line. Samples generated with Wolff algorithm.

In Figure 3.2 we see peaks in the correlation length approaching T_c from the right for increasingly large finite systems. At the critical temperate the susceptibilities and the correlation length diverge in the thermodynamic limit of an infinite system. In fact, the follow power laws at T_c. This can be shown in *mean field theory* and is further analyzed in the concept of *universality classes*. For our case this is not relevant. We utilize the 2D Ising model as a benchmark system with analytically known critical temperate given in Equation 3.3.

4. Monte Carlo simulation

Many broadly used computational-physics simulations are based on Monte Carlo (MC) concepts [9, 18, 21]. In numerical Monte Carlo algorithms the sampling is done randomly. The probabilistic approach allows to solve a large set of problems, from optimization to generating samples from any probability distribution.

Machine learning approaches in general and specially generative adversarial networks are techniques that rely heavily on training data. We have to present data to the GAN from which it learns the underlying distribution. In this thesis, we use Monte Carlo techniques to generate states of the Ising model following the probability distribution from the canonical ensemble. These Monte Carlo states are used as the mentioned training set.

The direct sampling of the state distribution is not possible. Hence, we construct a Markov-chain with all the possible states of our physical model. The chain is a sequence where we create a configuration and move to the next one until we reach the desired equilibrium. For stable convergence the condition of detailed balance [21] must be fulfilled for all state pairs A and B:

$$p_A \cdot w(A \to B) = p_B \cdot w(B \to A). \tag{4.1}$$

Here p_A is the probability to find the system in state A and with $w(A \to B)$ we mean the transition rate from state A to B. Two solutions to Equation 4.1 are given with the Metropolis–Hastings and Wolff algorithms. The generation of the state-sample-set is done, regardless of the update algorithm, with the following steps:

1. Initialize a random/arbitrary state $\mathbf{\Sigma}$.
2. Do Update$(\mathbf{\Sigma}, T, J)$ until the thermodynamic equilibrium is reached.
3. In order to reduce autocorrelation, do n-times Update$(\mathbf{\Sigma}, T, J)$.
4. Store $\mathbf{\Sigma}$ and/or calculate observables of the current state.
5. Repeat from step 3 until the number of desired sampled states is reached.

Here the Update$(\mathbf{\Sigma}, T, J)$ function moves along the Markov-chain of the Ising model and creates new states $\mathbf{\Sigma} \to \mathbf{\Sigma}'$ from existing ones. The control parameter T and model constant J are additional inputs. The definition of Update$(\mathbf{\Sigma}, T, J)$ is varying by the used algorithm and declared in the next sections.

4.1. Metropolis–Hastings algorithm

The Metropolis–Hastings algorithm [5] is based on local MC updates. With local we mean that updates only take into account the state of nearest neighboring spins. Considering the system in the canonical ensemble yields for each state $\mathbf{\Sigma}$ the probability

$$p(\mathbf{\Sigma}) = \frac{1}{Z} \exp(-\beta H(\mathbf{\Sigma})), \tag{4.2}$$

where Z is the partition function, H the Hamiltonian and $\beta = 1/k_\mathrm{B}T$. The key idea of Metropolis–Hastings is to consider a random (uniformly sampled) single spin flip $\sigma_i \to (-\sigma_i) \quad (\mathbf{\Sigma} \to \mathbf{\Sigma}')$. Using Equation 4.2 we get the proposal flip probability of the spin at lattice point i with nearest neighbors $\mathrm{NN}(i)$

$$p^{(i)}_\mathrm{proposal} = \frac{p(\mathbf{\Sigma}'(i))}{p(\mathbf{\Sigma})} = \exp(-\beta \Delta E(i)) = \exp\left(-2\beta J \sigma_i \sum_{j \in \mathrm{NN}(i)} \sigma_j\right). \tag{4.3}$$

Hence, the flip probability only depends on the energy difference $\Delta E(i)$ between the configurations before and after the spin i flipped. Is $\Delta E(i)$ negative we always flip, else we sample according to $p^{(i)}_\mathrm{proposal}$. Mathematically the probability is written as $p^{(i)}_\mathrm{flip}$

$$p^{(i)}_\mathrm{flip} = \min\left[1,\ p^{(i)}_\mathrm{proposal}\right]. \tag{4.4}$$

Consider N single flip attempts, following p_flip, a Metropolis–Hastings update step. The UpdateMetropolis$(\mathbf{\Sigma}, T, J)$ function is defined in Algorithm B.1.

4.2. Wolff algorithm

The Wolff algorithm [23] is in the set of cluster based update methods. Local MC algorithm suffer from critical slow down when sampling near to the critical temperature T_c [9]. This results from the connection between the autocorrelation time $\tau_\mathrm{auto} \sim \xi^z$ $(z > 1)$ and the diverging correlation length ξ (see subsection 3.1.4). The effect can be reduced with cluster updates that change larger portions of the configuration with a single update. With this the phase space is sampled more even. Thus, less updates between measuring configurations are needed. The algorithm efficiency improves a lot near T_c compared to local MC updates.

The key idea of Wolff is to consider two isolated parallel spins next to one another. Flipping both changes nothing. If only one spin flips the energy cost yields a flip probability of $p = \exp(-2\beta|J|)$. We flip entire clusters, with spins of the same orientation, for only the energy cost of the unaligned cluster edges. Hence, we enlarge a cluster to neighboring parallel spins with the inverse probability

$$p_\mathrm{append} = 1 - \exp(-2\beta|J|). \tag{4.5}$$

The UpdateWolff$(\mathbf{\Sigma}, T, J)$ function, Algorithm B.2, begins with a random initial spin, extends the cluster according to Equation 4.5 and flips the whole cluster.

4.3. Training set

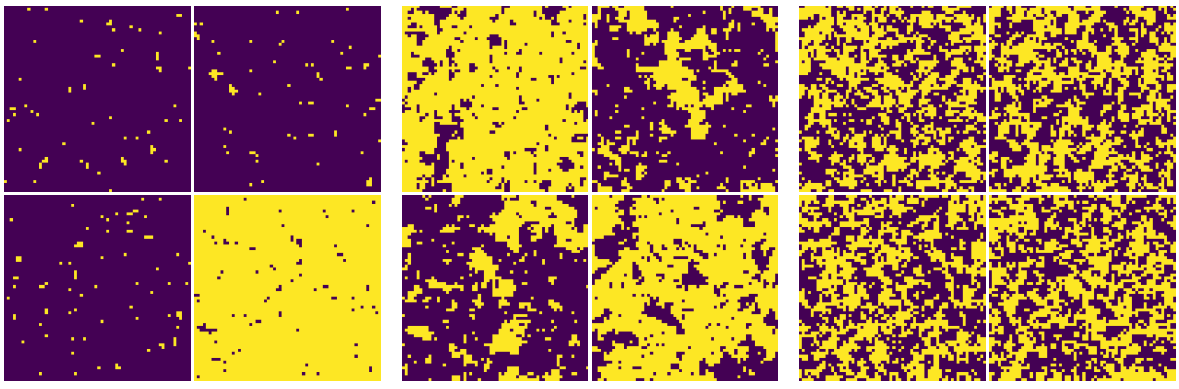
The training set is the distribution sample the generative adversarial network learns to reproduce. We generate configurations of the Ising model from chapter 3 using the Wolff algorithm. For all simulations we choose the exchange interaction constant $J = 1$. The system size L can be considered arbitrary. For the thermodynamic control parameter T we sample at 15 distinct temperatures

$$T \in \{1.0, 1.5, 1.8, 2.0, 2.1, 2.2, 2.25, 2.3, 2.35, 2.4, 2.5, 2.6, 2.8, 3.0, 3.4\} . \tag{4.6}$$

The range includes temperatures at all phases of the Ising model. Around the critical temperature $T_c \approx 2.269$ (see Equation 3.3) the density is chosen higher. The simulation steps are listed in chapter 4 and the Wolff updates are explained in section 4.2. We have three simulation parameters

$$n_{\text{thermal}}(T) = 10^3 \cdot T \qquad \ldots \text{list step 2} , \tag{4.7}$$
$$n_{\text{sweeps}}(T) = 10^2 \cdot T^2 \qquad \ldots \text{list step 3} , \tag{4.8}$$
$$n_{\text{samples}} = 10^4 \qquad \ldots \text{list step 5} . \tag{4.9}$$

Here n_{thermal} are the updates until we consider the system in equilibrium. An estimation of this number is possible, one plots the energy in dependence of the updates and looks for convergence. n_{sweeps} is the number of updates between measurements of the current state. The latter can be estimated with a binning analysis. The dependence of n_{thermal} and n_{sweeps} on T is because of the Wolff algorithm. The clusters get smaller with larger temperature, clearly visible in Equation 4.5. To account for that we scale the two simulation parameters with T. We simulate n_{samples} per temperature. In Figure 4.1 a random sample of training states is presented. One can get a feeling how the Ising model configurations look at different temperatures.



(a) $T = 1.8$. **(b)** $T = 2.3$. **(c)** $T = 3.4$.

Figure 4.1.: Presented are randomly sampled states from the training set. The states are generated using Wolff algorithm. Starting from the left, one sees the ferromagnetic, transition and paramagnetic phase. Utilized is a system size $L = 64$ and an interaction constant $J = 1$.

5. GAN architectures

1. define the problem. what is the goal here? 2. define in out variables 3. what next??

5.1. DC-GAN

[19]

Dis in Figure 5.1 and without injection layer in Figure 5.2.

5.1.1. Improvements

periodic boundary, amplitude 0.7 of states -> tanh in 0 and 1 numerical more stable/accurate, wasserstein at generator, (lower dims than celeb data), dis gradient penalty, aux network

5.2. StyleGAN2

[8]

5.3. SpinGAN

injection layer

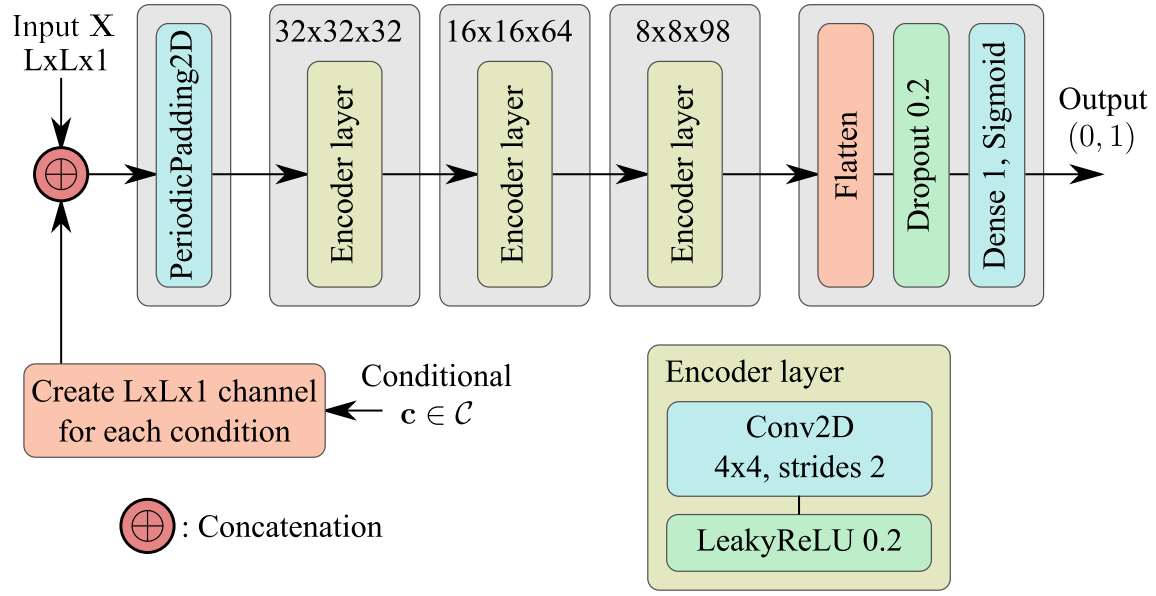


Figure 5.1.: SpinGAN's discriminator. Numbers in the top of the gray boxes correspond to the resolution and filter/channel size of the layer. Inputs are the images **X** and condition **c**.

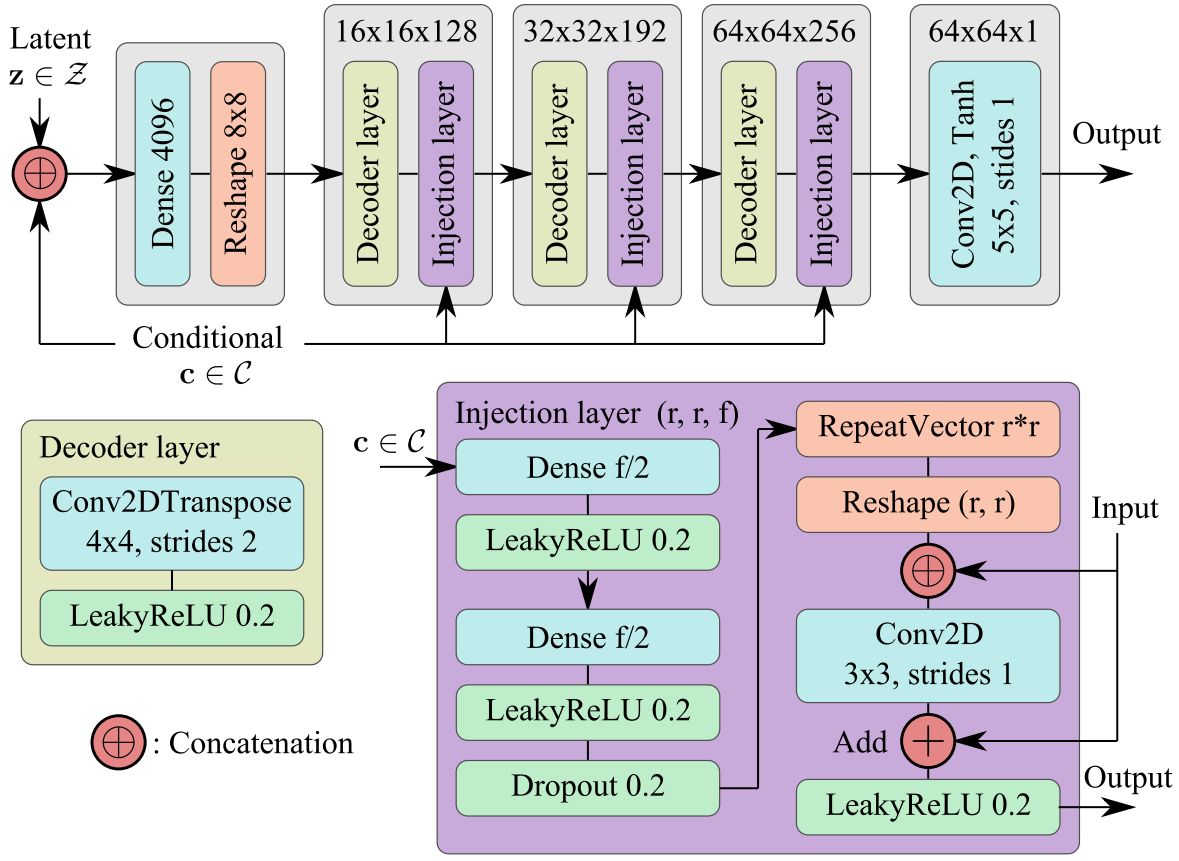


Figure 5.2.: SpinGAN's generator. Numbers in the top of the gray boxes correspond to the resolution and filter/channel size of the layer. Inputs are the latent vectors **z** and condition **c**.

6. Numerical experiments

6.1. Metrics

Generative networks behave differently during training. For normal classification networks the performance generally increases and converges with longer training time, not considering overfitting. GANs on the other hand are more complex in their performance evolution over the training epochs.

Percentage overlap: The percentage overlap POL of two normalized binned distributions p_1 and p_2 is defined as

$$\mathrm{POL}(p_1, p_2) := \sum_{i}^{\mathrm{bins}} \min\left[p_1^{(i)}, p_2^{(i)}\right]. \tag{6.1}$$

Here i is the bin index and $p_{1,2}^{(i)}$ the value of the i-th bin of the corresponding distribution.

Obs distance: For physical application the generator should generate data that includes the correct observable averages.

$$\text{Obs distance} := \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_{\mathrm{GAN},i} - x_{\mathrm{MC},i}}{\sigma_{\mathrm{MC},i}}\right)^2}, \tag{6.2}$$

where x_i are means of the considered observables and σ_i the corresponding standard deviations. Either from the GAN or the reference Monte Carlo (MC). In our case we utilize the observables $|m|, E, \chi, \xi$ from section 3.1 and additional variables U_2, κ_3 to account for higher moments. The latter are the Binder cumulant U_2 [2] and third cumulant κ_3 of the magnetization, defined as:

$$U_2 := \frac{3}{2}\left(1 - \frac{\langle m^4\rangle}{3\langle m^2\rangle^2}\right) \tag{6.3}$$

$$\kappa_3 := \frac{N}{T}\Big(\langle |m|^3\rangle - 3\langle m^2\rangle\langle |m|\rangle + 2\langle |m|\rangle^3\Big) \tag{6.4}$$

6.2. Gan fidelity

A conditional GAN is able to generate states at any temperature very quickly and efficiently.

$$\mathcal{F}_{\mathrm{GAN}}(T) := \mathbb{E}_{\mathbf{z}\sim p}\left[\left|\frac{\partial G(\mathbf{z}, T)}{\partial T}\right|\right] \tag{6.5}$$

Here the exact derivation is performed and does not need to be approximated. The reason is that neural-network-frameworks are extremely optimized in calculating exact gradients (autograd). This allows us to differentiate actual states depending on thermodynamic control parameters.

6.3. Implementation details

python, c++

tensorflow 2, rtx3090, amd 5900X.

time for train, memory , cluster, ...

7. Model performance

add images, to comparison of three architectures

Table 7.1.: OOP.

	OOP
DCGAN	95.4 ± 55.8
StyleGAN2	846.2 ± 1761.3
SpinGAN	32.6 ± 18.3

7.1. Architecture comparison

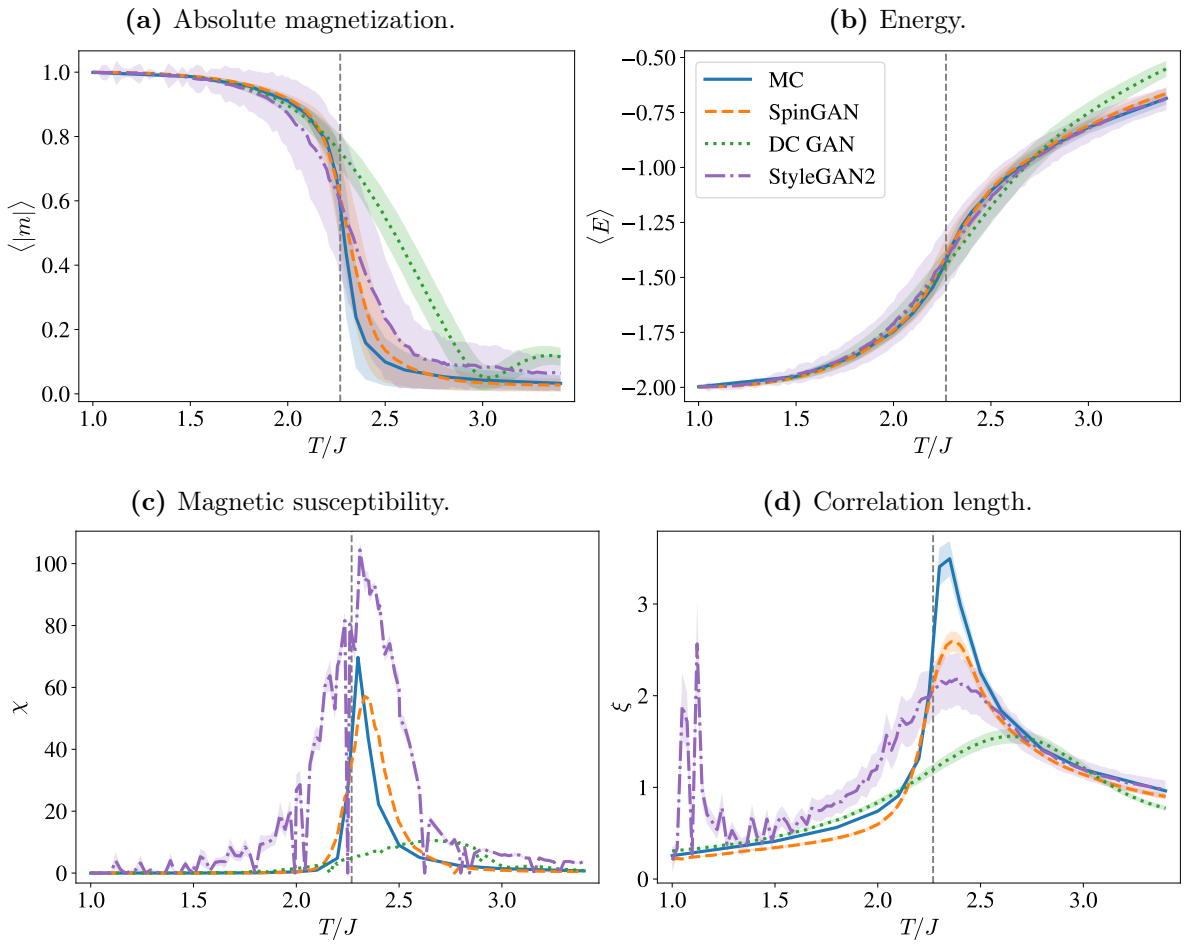


Figure 7.1.: Presented are the mean values of the observables for different GAN architectures. The shaded areas are the respective standard deviation. Monte Carlo (MC) data is used as the ground truth and a larger overlap corresponds to a better method. The critical temperature T_c is marked as a vertical line. The considered system size is $L = 64$.

7.2. Mode collapse and overshoot

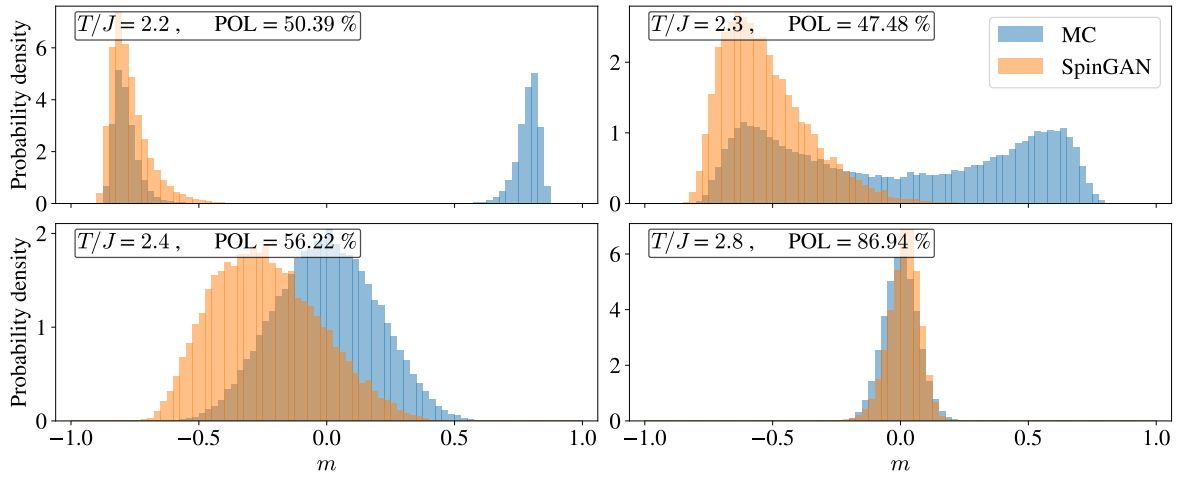


Figure 7.2.: Histograms of the magnetization for the system size $L = 64$ and different temperatures. Monte Carlo (MC) data is used as the ground truth and a larger overlap is better.

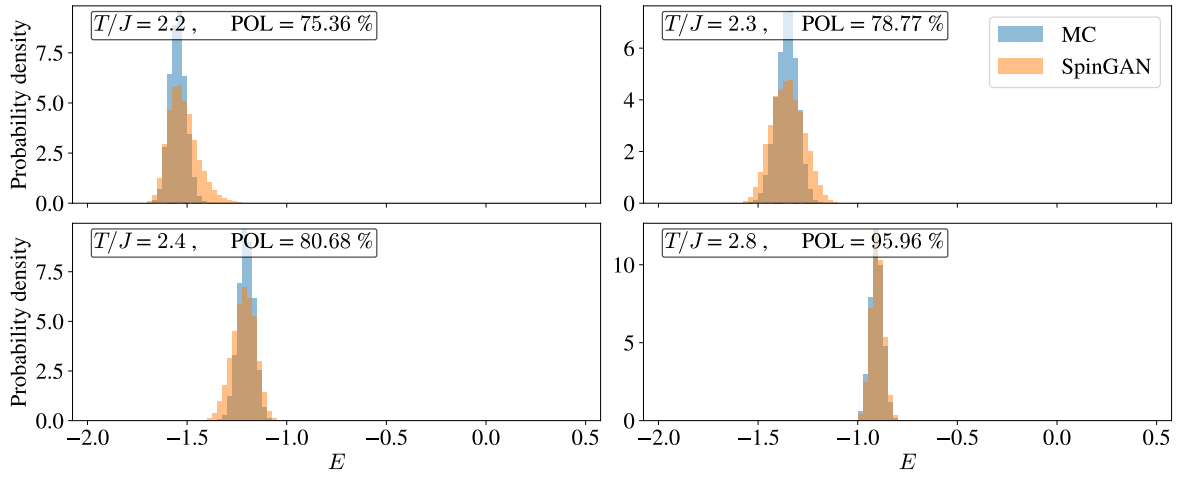


Figure 7.3.: Histograms of the energy for the system size $L = 64$ and different temperatures. Monte Carlo (MC) data is used as the ground truth and a larger overlap is better.

7.3. State evolution

The generator can be evaluated very efficiently and allows us to sample many states at an arbitrary temperature for cheap. A remarkable property of our conditional generative model is that we are able to evolve single states completely deterministic through the temperature range. All we have to do is to fix a latent vector z and change the conditional input T. This is shown in Figure 7.4 and Figure A.1. We can follow a state and see how clusters emerge and the Ising model changes its phase. With classic Monte Carlo methods an evolution of this kind is not possible. Randomness is introduced because of the stochastic property and MC simulations have to thermalize at every temperature step before measuring.

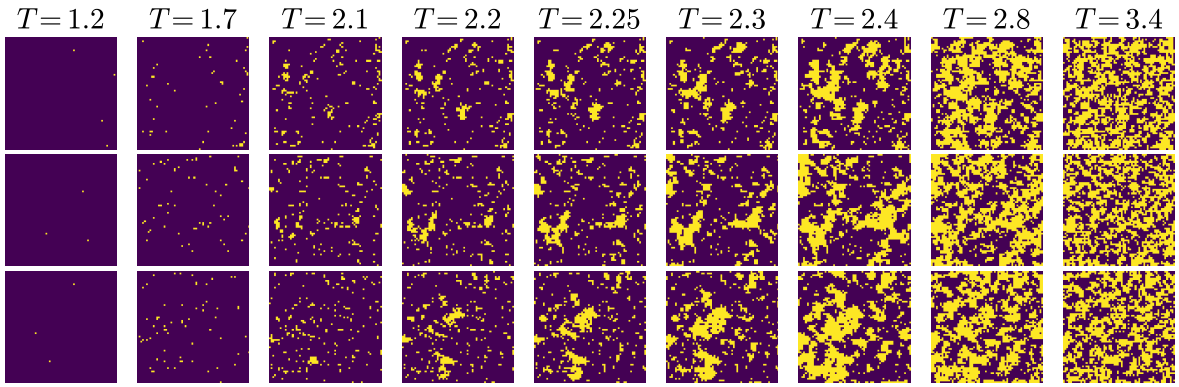


Figure 7.4.: States are generated with SpinGAN, utilizing a system size of $L = 64$. The cols represent the same temperature T. For the rows three fixed latent vectors z are used. With the fixation in z and varying in T a single state is evolved through the temperature range. On the left we see the ferromagnetic case with most spins aligned. Following a row to the right makes the emerging of clusters visible, until thermodynamic fluctuations dominate and the paramagnetic phase takes over.

7.4. GAN fidelity

We evaluate Equation 6.5 for SpinGAN at different lattice sizes L. This is presented in Figure 7.5.

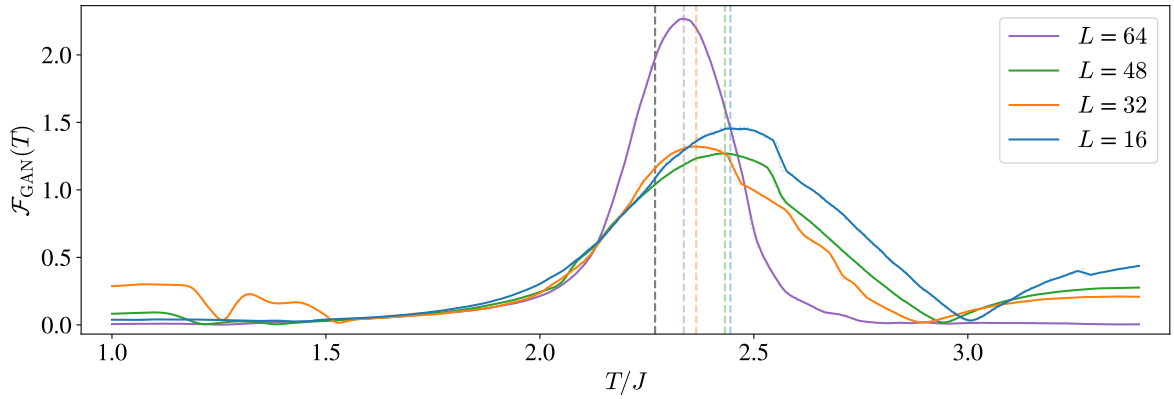


Figure 7.5.: The GAN fidelity $\mathcal{F}_{\mathrm{GAN}}$ of the SpinGAN architecture evaluated for different system sizes L. The critical temperature T_c is marked as a gray vertical line. Colored vertical lines represent the maximum of their corresponding curve. The focus lies on the purple line with $L = 64$, where the most time for optimizing hyperparameters was spent.

8. Summary, Conclusion and Outlook

summary

what can be done further(cluster not home pc), 3d ising other spin models

problems

References

[1] M. Arjovsky and L. Bottou. "Towards Principled Methods for Training Generative Adversarial Networks". In: (Jan. 17, 2017). arXiv: 1701.04862 [stat.ML].

[2] K. Binder. "Critical Properties from Monte Carlo Coarse Graining and Renormalization". In: *Phys. Rev. Lett.* 47 (1981), pp. 693–696. DOI: 10.1103/PhysRevLett.47.693.

[3] K. Fukushima. "Neocognition: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". English. In: *Biological Cybernetics* 36 (1980), pp. 193–202. ISSN: 0340-1200. DOI: 10.1007/BF00344251.

[4] I. J. Goodfellow et al. "Generative Adversarial Networks". In: (June 10, 2014). arXiv: 1406.2661 [stat.ML].

[5] W. K. Hastings. "Monte Carlo Sampling Methods Using Markov Chains and Their Applications". In: *Biometrika* 57 (1970), pp. 97–109. DOI: 10.1093/biomet/57.1.97.

[6] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators". In: 2 (1989), pp. 359–366. ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8.

[7] E. Ising. "A contribution to the theory of ferromagnetism". German. In: *Zeitschrift für Physik* 31 (1925), pp. 253–257. ISSN: 0939-7922. DOI: 10.1007/BF02980577.

[8] T. Karras et al. "Analyzing and Improving the Image Quality of StyleGAN". In: *CoRR* abs/1912.04958 (2019). arXiv: 1912.04958. URL: http://arxiv.org/abs/1912.04958.

[9] H. G. Katzgraber. "Introduction to Monte Carlo Methods". In: (May 11, 2009). arXiv: 0905.1629 [cond-mat.stat-mech].

[10] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv e-prints*, arXiv:1412.6980 (Dec. 2014). arXiv: 1412.6980 [cs.LG]. URL: https://ui.adsabs.harvard.edu/abs/2014arXiv1412.6980K.

[11] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86 (11 1998), pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791.

[12] Z. Liu, S. P. Rodrigues, and W. Cai. "Simulating the Ising Model with a Deep Convolutional Generative Adversarial Network". In: (Oct. 13, 2017). arXiv: 1710.04987 [cond-mat.dis-nn].

[13] W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity". In: 5 (1943), pp. 115–133. ISSN: 0007-4985. DOI: 10.1007/bf02478259.

[14] L. Mescheder, A. Geiger, and S. Nowozin. "Which Training Methods for GANs do actually Converge?" In: *International Conference on Machine Learning 2018* (Jan. 13, 2018). arXiv: 1801.04406 [cs.LG].

[15] K. Mills and I. Tamblyn. "Phase space sampling and operator confidence with generative adversarial networks". In: *arXiv e-prints*, arXiv:1710.08053 (Oct. 2017). arXiv: 1710.08053 [cond-mat.stat-mech]. URL: https://ui.adsabs.harvard.edu/abs/2017arXiv171008053M.

[16] A. Odena, C. Olah, and J. Shlens. "Conditional Image Synthesis With Auxiliary Classifier GANs". In: (Oct. 30, 2016). arXiv: 1610.09585 [stat.ML].

[17] L. Onsager. "Crystal statistics. 1. A Two-dimensional model with an order disorder transition". In: *Phys. Rev.* 65 (1944), pp. 117–149. DOI: 10.1103/PhysRev.65.117.

[18] J. Qiang. "Monte Carlo Simulation Techniques". In: (June 18, 2020). arXiv: 2006.10506 [physics.comp-ph].

[19] A. Radford, L. Metz, and S. Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: (Nov. 19, 2015). arXiv: 1511.06434 [cs.LG].

[20] J. Singh et al. "Generative models for sampling and phase transition indication in spin systems". In: *SciPost Phys. 11, 043 (2021)* (June 21, 2020). DOI: 10.21468/SciPostPhys.11.2.043. arXiv: 2006.11868 [cond-mat.stat-mech].

[21] J.-C. Walter and G. Barkema. "An introduction to Monte Carlo methods". In: (Apr. 1, 2014). DOI: 10.1016/j.physa.2014.06.014. arXiv: 1404.0209 [cond-mat.stat-mech].

[22] P. Werbos. "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science. Thesis (Ph. D.)." PhD thesis. Appl. Math. Harvard University, Jan. 1974.

[23] U. Wolff. "Collective Monte Carlo Updating for Spin Systems". In: *Phys. Rev. Lett.* 62 (4 Jan. 1989), pp. 361–364. DOI: 10.1103/PhysRevLett.62.361. URL: https://link.aps.org/doi/10.1103/PhysRevLett.62.361.

[24] A. Yu et al. "Arbitrary-Depth Universal Approximation Theorems for Operator Neural Networks". In: (Sept. 23, 2021). arXiv: 2109.11354 [cs.LG].

A. Additional performance data

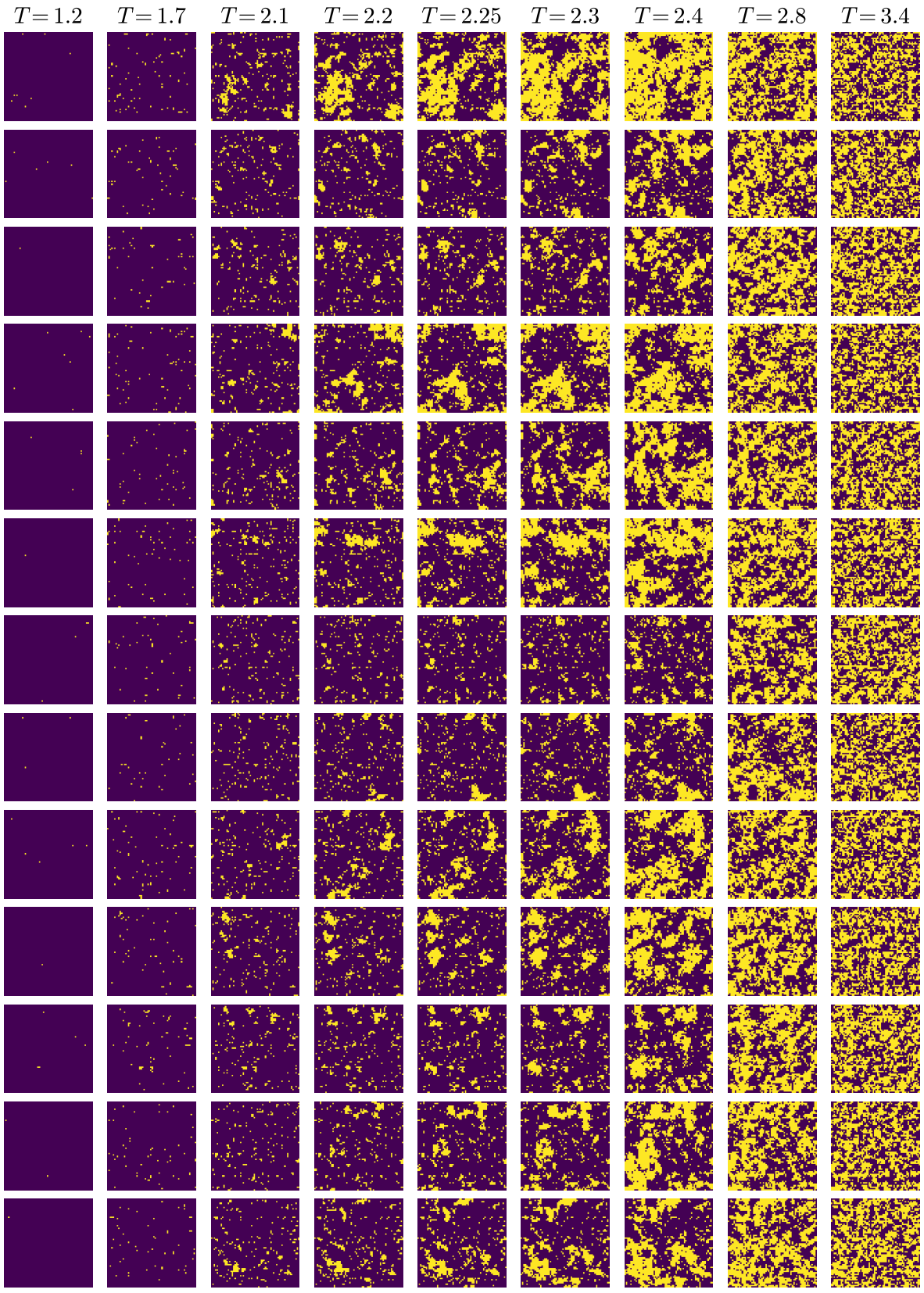


Figure A.1.: States are generated with SpinGAN, utilizing a system size of $L = 64$. The cols represent the same temperature T. For the rows three fixed latent vectors z are used. With the fixation in z and varying in T a single state is evolved through the temperature range.

A.1. SpinGAN

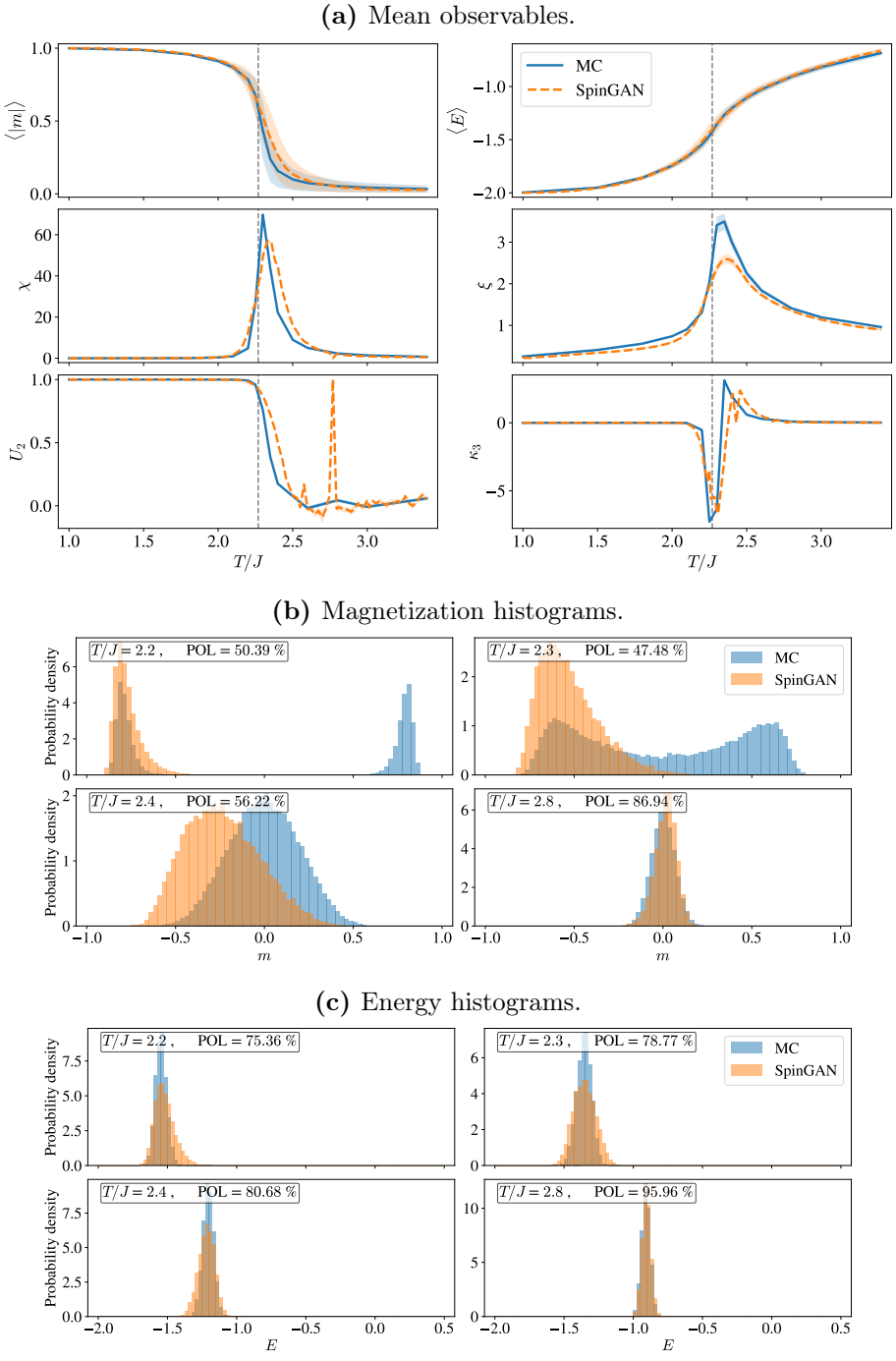


Figure A.2.: SpinGAN; (a) mean values of the observables, where shaded areas are the respective standard deviation. Histograms of the magnetization (b) and energy (c). In (a) the critical temperature T_c is marked as a vertical line. Monte Carlo (MC) data is used as the ground truth, a larger overlap is better. The used system size is $L = 64$.

A.2. StyleGAN2

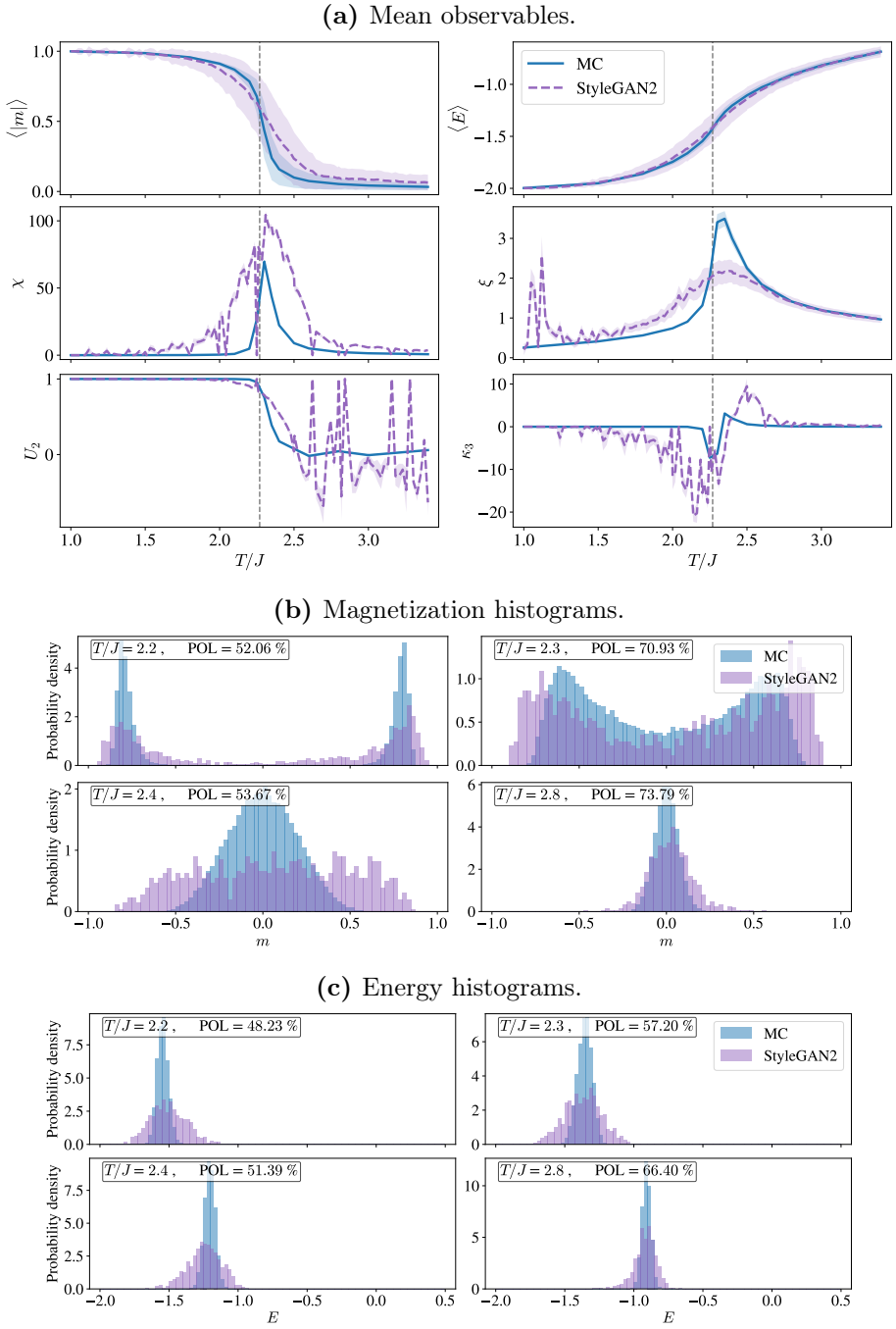


Figure A.3.: StyleGAN2; (a) mean values of the observables, where shaded areas are the respective standard deviation. Histograms of the magnetization (b) and energy (c). In (a) the critical temperature T_c is marked as a vertical line. Monte Carlo (MC) data is used as the ground truth, a larger overlap is better. The used system size is $L = 64$.

A.3. DCGAN

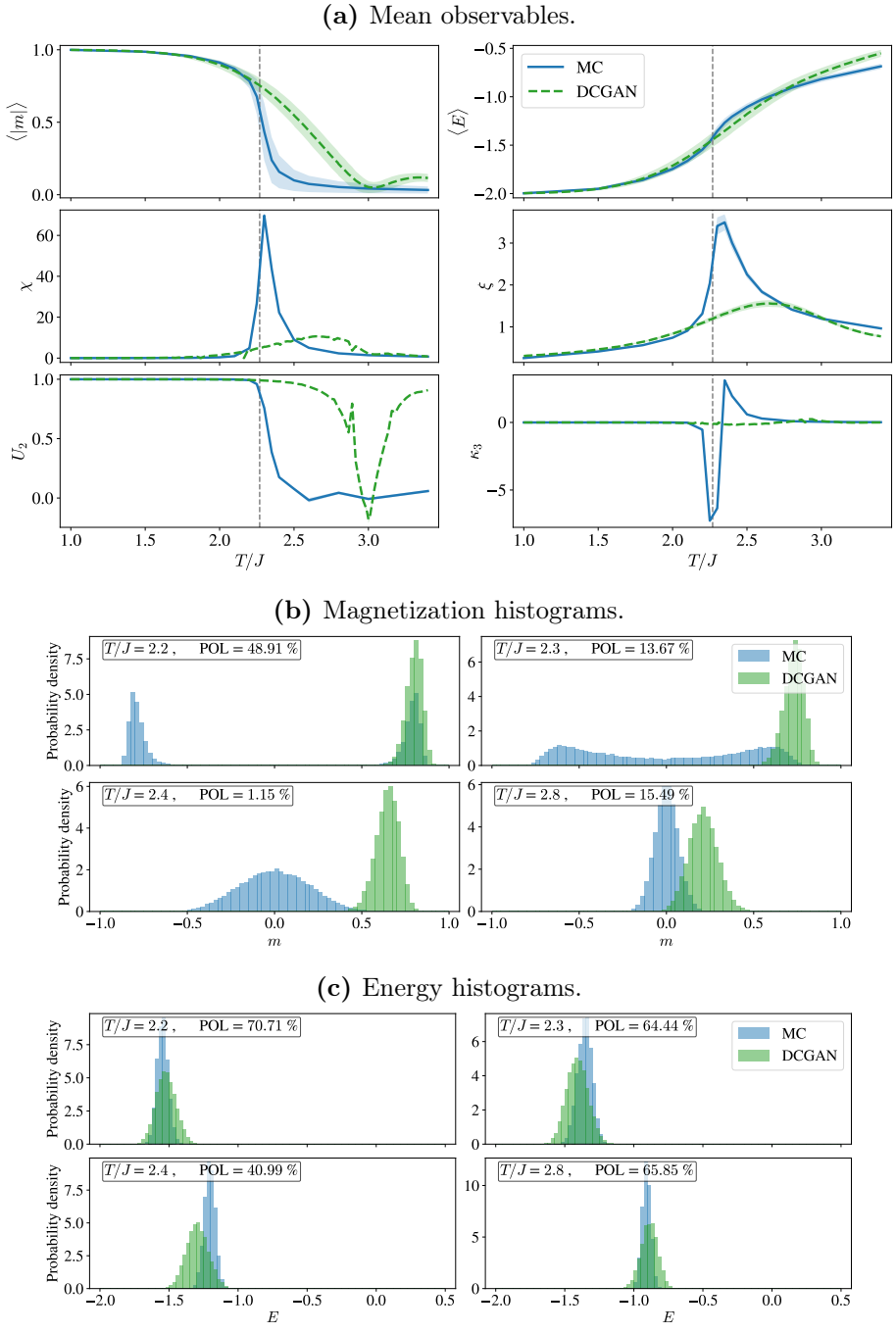


Figure A.4.: DCGAN; (a) mean values of the observables, where shaded areas are the respective standard deviation. Histograms of the magnetization (b) and energy (c). In (a) the critical temperature T_c is marked as a vertical line. Monte Carlo (MC) data is used as the ground truth, a larger overlap is better. The used system size is $L = 64$.

B. Algorithms

Listing of the algorithms explained in chapter 4.

Algorithm B.1 Metropolis–Hastings algorithm, update

```
1:  function UpdateMetropolis(Σ, T, J)
2:      for x=0, N-1 do                                   ▷ N flip attempts
3:          i ← rndInteger[0, N − 1]                      ▷ Choose random spin
4:
5:          p^(i)_proposal ← ...                          ▷ Probability of flipping spin i
6:
7:          if (rndFloat[0, 1] ≤ p^(i)_proposal) then     ▷ Sample p^(i)_proposal
8:              σ_i ← (−σ_i)                              ▷ Flip spin i, update Σ
9:          end if
10:     end for
11: end function
```

Algorithm B.2 Wolff algorithm, update

```
1:  function UpdateWolff(Σ, T, J)
2:      p_append ← [1 − exp(−2β|J|)]                      ▷ Probability of extending the cluster
3:
4:      i ← rndInteger[0, N − 1]                          ▷ Choose random cluster starting spin
5:      σ_init ← σ_i                                      ▷ Store initial spin for alignment checks
6:      σ_i ← (−σ_i)                                      ▷ Flip starting spin
7:      cluster ← add spin i                              ▷ Add first spin to the cluster list
8:
9:      while cluster.size > 0 do
10:         k ← cluster.popBack                           ▷ Pop last entry in cluster list
11:
12:         for all neighbors of side k do
13:             if (σ_init == σ_neighbor) and (rndFloat[0, 1] ≤ p_append) then
14:                 cluster ← add neighbor spin           ▷ Extend the cluster
15:
16:                 σ_neighbor ← (−σ_neighbor)            ▷ Flipping spins on the fly
17:             end if
18:         end for
19:     end while
20: end function
```