TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

INSTITUTE FOR THEORETICAL PHYSICS

DIPLOMARBEIT

Sampling lattice field theories with score-based diffusion models

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Physik

eingereicht von

Thomas Ranner, BSc
Matrikelnummer 11771213

ausgeführt am Institut für Theoretische Physik
der Fakultät für Physik der Technischen Universität Wien

Betreuung
Betreuer: Privatdoz. Dipl.-Ing. Dr.techn. Andreas Ipp
Mitwirkung: Projektass.(FWF) Dipl.-Ing. Dr.techn. David Müller

Wien, 16.10.2024 _____ _____
(Unterschrift Verfasser) (Unterschrift Betreuer)

Zusammenfassung

Diffusionsmodelle sind die Basis moderner Bildgenerierungsmodelle. Wir verwenden sie zur Erzeugung von Feldkonfigurationen in Gitterfeldtheorien. Aufgrund der ähnlichen Datenstrukturen von digitalen Bildern und zweidimensionalen diskreten Feldern können generative Modelle, die ursprünglich für die Bildgenerierung entwickelt wurden, mit nur geringen Anpassungen für Gitterfeldtheorien verwendet werden. In dieser Arbeit fokussieren wir uns auf *score-based generative diffusion models (score models)*, eine spezielle Art von Diffusionsmodellen. Hierbei wird die Score-Funktion von einem neuronalen Netzwerk gelernt.

Nachdem wir uns kurz mit einem nulldimensionalen Beispielproblem beschäftigen, implementieren wir *score models* für die skalare ϕ^4 Theorie und die reine U(1)-Eichtheorie. Die Qualität unserer trainierten Modelle, gemessen an der effektiven Stichprobengröße, ist für das skalare Problem nahezu ideal. Für die Eichtheorie erreichen wir nur bei schwacher Kopplung gute Resultate, je stärker die Kopplung ist, desto schlechter wird die Qualität unserer Modelle. Ein großer Vorteil von Diffusionsmodellen gegenüber Modellen, die auf Markovketten basieren, ist, dass sie prinzipiell Konfigurationen mit verschwindender Autokorrelation zwischen einzelnen Proben erzeugen können. Es zeigt sich auch, dass Konfigurationen deutlich schneller erstellt werden können als mit unserer Hybrid-Monte-Carlo Implementierung.

Des Weiteren präsentieren wir eine Adaptierung der *score models*, das *action model*. Es basiert darauf, dass die Score-Funktion dem negativen Gradienten der Wirkung entspricht, womit wir dem neuronalen Netzwerk eine direkte physikalische Bedeutung geben können. Wir vergleichen diese beiden Modelle für das nulldimensionale Beispielproblem und für die skalare ϕ^4 Theorie. Abschließend demonstrieren wir eine weitere physikalische Interpretation von Diffusionsmodellen, indem wir den Fluss der Carosso-Renormierungsgruppe mithilfe eines *score models* invertieren. Damit zeigen wir eine direkte Verbindung zwischen generativen Diffusionsmodellen und Renormierungsgruppen.

Abstract

Diffusion models are state-of-the-art tools for machine-learning-based image generation. We apply them to lattice field theories to generate field configurations. Due to the similar data structures of digital images and two-dimensional discretized fields, generative models designed for image generation can be used for lattice field theories with only minor adaptions. In this work we focus on score-based generative diffusion models (score models), a particular type of diffusion models in which a neural network learns a quantity named 'score'.

After briefly reviewing a zero-dimensional toy model, we implement score models for scalar ϕ^4 theory and U(1) pure gauge theory. We find that the quality of our trained models, measured by the effective sampling size, is almost perfect in the scalar case. For U(1), the quality is still good for weak coupling, but decreases with increasing coupling strength. A notable advantage of diffusion models over Markov chain-based methods is that they can, in principle, generate configurations with vanishing autocorrelation between samples. We also find the generative speed to be significantly faster compared to our implementation of the Hybrid Monte Carlo (HMC) method.

Furthermore, we introduce the action model, an adaption of the score model. It uses the fact that the score corresponds to the negative gradient of the action, giving the trained neural network a direct physical meaning. Comparisons between both models are given for the zero-dimensional toy problem and scalar ϕ^4 theory. To take the physical interpretation of diffusion models even further, we use a score model to reverse the Carosso renormalization group flow, showing a direct connection between score-based generative diffusion models and renormalization group theory.

Contents

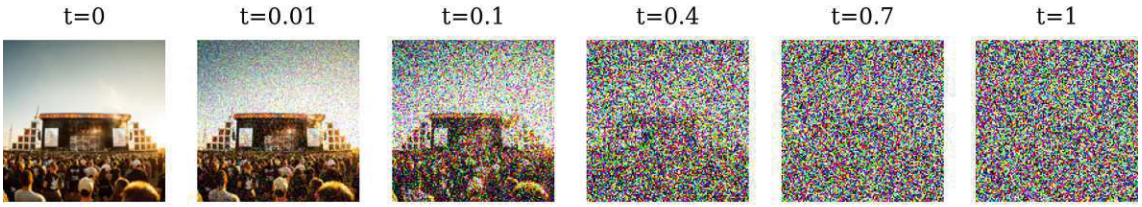
1 Introduction

Generative machine learning models have attracted significant scientific and societal attention in recent years. These generative models are trained with a dataset and can then generate new data which follows the probability distribution of the original dataset. Among them are diffusion models, whose most prominent application lies in text-to-image generation, being used e.g. in Stable Diffusion [1] or DALL-E [2]. While not initially developed for applications in physics, advances have been made to use generative models [3] and specifically diffusion models [4] in the context of lattice field theory. Due to the similar data structures of images and lattice field theory configurations (i.e. degrees of freedom sitting at intersections of a quadratic grid, with short and long scale correlations between these values), it seems promising to use diffusion models, being a prominent image generation method, for the generation of configurations in lattice field theories. Traditionally, Monte Carlo methods are being used to generate field configurations in lattice field theory. These methods face certain problems like critical slowing down and topological freezing [5]. Finding ways to generate configurations which circumvent these problems by having shorter autocorrelation times can help with more accurate calculations when going towards the continuum limit.

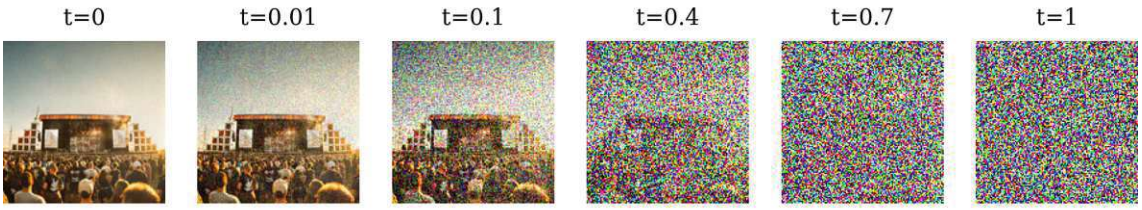
The exact type of diffusion model we will use was introduced in [6]. It is based on diffusion processes given by stochastic differential equations, which continuously change configurations and thus deform the probability density of configurations. The forward diffusion process turns data into noise and correspondingly a complex probability distribution (from which we want to draw samples) into a simple probability distribution that is independent of the initial data (where it is easy to draw samples from). By reversing this diffusion process we can generate data from noise and thus draw samples from the complex probability distribution. The reversal is possible with knowledge of a quantity called 'score', the gradient of the logarithm of the probability density, which will be learned by a neural network. An illustration of the diffusion processes in the context of image generation can be seen in fig. 1.1. The biggest drawback of this method is that it always needs a number of samples drawn from the target probability distribution through other means for training.

First, we illustrate the concept using a zero-dimensional toy problem. Then, we work on a two-dimensional scalar field theory, similar to what has been done in [4]. Finally, we apply the diffusion model to U(1) pure gauge theory. The last part is also intended to be a first step towards using diffusion models for SU(3) gauge theory in future works.

Since diffusion models have not been developed with physical applications in mind, they have initially not been related to any physical processes. To explore the physics



(a) The diffusion process, starting at $t = 0$.



(b) The generative process, starting at $t = 1$.

Figure 1.1: Comparison of the forward diffusion process going from left to right and the generative (reverse) diffusion process going from right to left. The generative process uses a score model similar as in chapter 4, trained only with a single image for illustrative purposes.

behind the diffusion processes in the generative model, we note that the score is equivalent to the gradient of the negative action of a system [4]. By adapting the diffusion model such that the network models the action instead of the score, we gain direct access to the action, along the whole diffusion process. In a different approach, the authors of [7] bring up a connection between diffusion models and a certain renormalization group scheme, namely the Carosso renormalization group (RG) flow [8], which we will study experimentally.

The code for this project using Python and PyTorch is available here: `https://github.com/Thomas-Ranner/LFT_DM`

2 Score model

2.1 Concept

Score-based generative models (e.g. [9, 10]) are a class of machine learning models with the aim of drawing samples from a configuration space with a (usually unknown) probability distribution. Although developed mainly for image generation (where generating an image of a cat corresponds to drawing a sample from the space of all images with a probability distribution that favours images of cats over all other possible images), they are restrained neither by the 'form' of the generated data nor by the complexity of the probability distribution. In this work we will implement score-based generative modelling through stochastic differential equations (SDEs), as introduced in [6].

The basic concept of score-based modelling with SDEs is shown in fig. 2.1. To draw a sample $\boldsymbol{\phi}_0$ from the target distribution $p_0(\boldsymbol{\phi}_0)$, we solve the so-called reverse-time SDE with initial condition $\boldsymbol{\phi}_T$. This $\boldsymbol{\phi}_T$ is drawn from another probability distribution $p_T(\boldsymbol{\phi}_T)$, which is typically very easy to sample from (e.g. a multivariate normal distribution). As the name suggests, this reverse-time SDE is the reverse of another SDE, which is called the forward SDE. Said forward SDE continuously adds noise to a sample drawn from $p_0(\boldsymbol{\phi}_0)$ such that after diffusion time T it follows the aforementioned simple distribution $p_T(\boldsymbol{\phi}_T)$. In short, the forward SDE turns data into noise and the reverse SDE turns noise into data.

The forward SDE, whose only aim is to destroy all information of the initial samples, is chosen in a simple analytical form. The presented generative approach

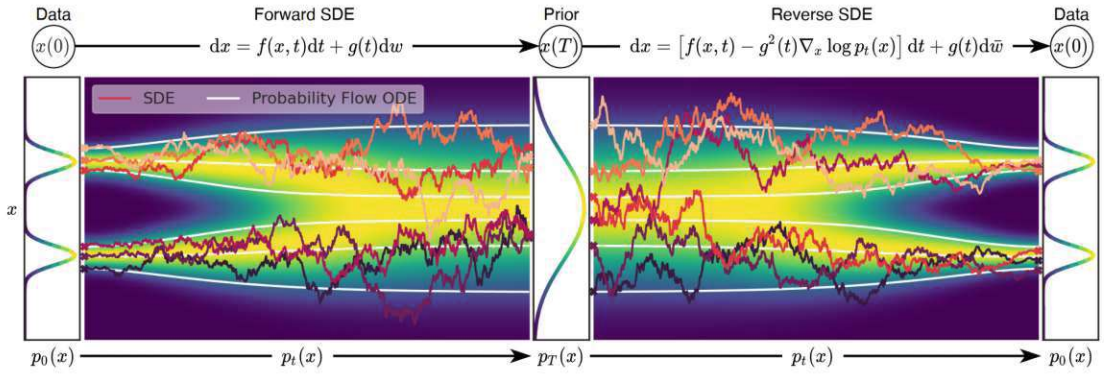


Figure 2.1: Illustration of score based generative modelling, taken from [6]. Note: what is labelled as $\mathbf{x}$ in the graphic is $\boldsymbol{\phi}$ in this text.

is based on the fact that with knowledge of the analytical form of the forward SDE and an additional quantity termed 'score' (which will be learned by a neural network), we are able to write down the SDE of the reverse process, and therefore solve it.

The general form of the mentioned stochastic differential equations is given as (see e.g. [11])

$$\mathrm{d}\boldsymbol{\phi} = \mathbf{f}(\boldsymbol{\phi}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w},\ \boldsymbol{\phi} \in \mathbb{R}^d, \tag{2.1}$$

where t is the diffusion time, which runs between 0 and T, $\mathbf{w}$ stands for the Wiener process, $\mathbf{f}(\boldsymbol{\phi}, t)$ is the vector-valued drift coefficient and $g(t)$ the scalar diffusion coefficient. The random Wiener process is a Gaussian process that can be defined by

$$\mathbb{E}[\eta_i(t)] = 0, \tag{2.2}$$
$$\mathbb{E}[\eta_i(t)\eta_j(s)] = \delta_{i,j}\delta(t-s), \tag{2.3}$$

where $\eta_i(t) = \mathrm{d}w_i/\mathrm{d}t$ and $\mathbb{E}$ denotes the expectation value.

While eq. (2.1) describes the time evolution of a single sample $\boldsymbol{\phi}$, the time evolution of the probability density function over $\boldsymbol{\phi}$, $p_t(\boldsymbol{\phi})$ is given by the corresponding Fokker-Planck equation,

$$\frac{\partial}{\partial t}p_t(\boldsymbol{\phi}) = -\sum_{i=1}^{d}\frac{\partial}{\partial \phi_i}\big(f_i(\boldsymbol{\phi}, t)p_t(\boldsymbol{\phi})\big) + \sum_{i=1}^{d}\frac{\partial^2}{\partial\phi_i\partial\phi_i}\left(\frac{g(t)^2}{2}p_t(\boldsymbol{\phi})\right). \tag{2.4}$$

2.2 Forward diffusion process

The purpose of the forward diffusion process is to transform the initial data such that it follows a simple probability distribution that is independent of the initial data. One possible and popular choice is to set $\mathbf{f}(\boldsymbol{\phi}, t) = 0$ and $g(t) = \sigma^t$ with $\sigma > 0$ in eq. (2.1), leading to

$$\mathrm{d}\boldsymbol{\phi} = \sigma^t\mathrm{d}\mathbf{w}. \tag{2.5}$$

This choice is termed as Variance Exploding SDE in [6], as the variance of p_t diverges for $t \to \infty$. An advantage of this particular choice is that the transition probability $p_{0t}(\boldsymbol{\phi}_t|\boldsymbol{\phi}_0)$ is given by a simple multivariate normal distribution,

$$p_{0t}(\boldsymbol{\phi}_t|\boldsymbol{\phi}_0) = \mathcal{N}\left(\boldsymbol{\phi}_0, \frac{1}{2\log(\sigma)}(\sigma^{2t}-1)\mathbf{I}\right), \tag{2.6}$$

with $\mathbf{I}$ being the identity matrix. Knowing the transition probability, the probability density at time t is given as

$$p_t(\boldsymbol{\phi}_t) = \int \mathrm{d}\boldsymbol{\phi}_0' p_0(\boldsymbol{\phi}_0')p_{0t}(\boldsymbol{\phi}_t|\boldsymbol{\phi}_0'). \tag{2.7}$$

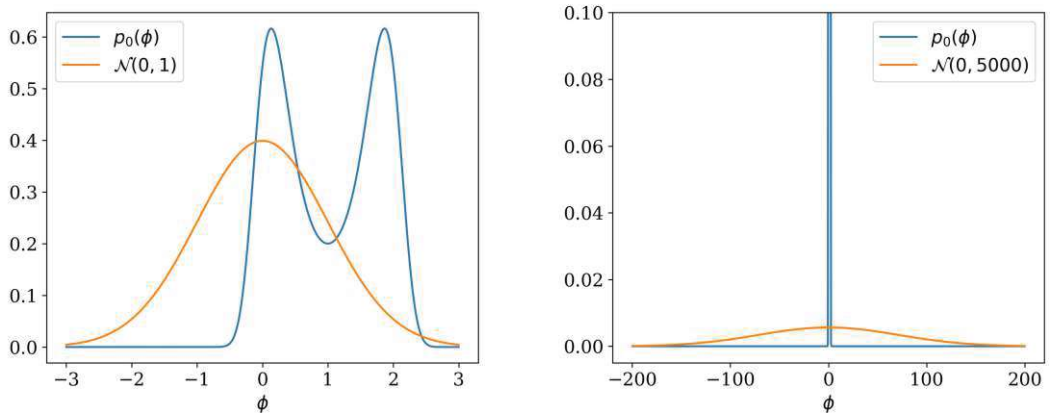


Figure 2.2: Comparison of a possible target distribution with two normal distributions of different variance. In the right case, the target distribution acts approximately as a delta function.

To verify eq. (2.6) in 1D, we first note that $p_{0t}(\boldsymbol{\phi}_t|\boldsymbol{\phi}_0)$ is equivalent to $p_t(\boldsymbol{\phi}_t)$ when assuming $\boldsymbol{\phi}_0$ as initial condition, i.e. inserting $p_0(\boldsymbol{\phi}_0') = \delta(\boldsymbol{\phi}_0' - \boldsymbol{\phi}_0)$ in eq. (2.7). Under this assumption we can write

$$p_t(\boldsymbol{\phi}_t) = \frac{1}{\sqrt{2\pi\Sigma_t^2}} e^{-\frac{1}{2}\frac{(\boldsymbol{\phi}_t - \boldsymbol{\phi}_0)^2}{\Sigma_t^2}} \tag{2.8}$$

with

$$\Sigma_t^2 = \frac{1}{2\log(\sigma)}(\sigma^{2t} - 1). \tag{2.9}$$

For eq. (2.6) to be the correct, $p_t(\boldsymbol{\phi}_t)$ has to be a solution of the Fokker-Planck equation corresponding to eq. (2.5),

$$\frac{\partial}{\partial t} p_t(\boldsymbol{\phi}_t) = \frac{\sigma^{2t}}{2} \frac{\partial^2}{\partial \boldsymbol{\phi}^2} p_t(\boldsymbol{\phi}_t). \tag{2.10}$$

Inserting eq. (2.8) above, and using $\partial_t \Sigma_t^2 = \sigma^{2t}$, we get

$$\frac{1}{\sqrt{2\pi\Sigma_t^2}} \frac{\sigma^{2t}}{2\Sigma_t^2} e^{-\frac{1}{2}\frac{(\boldsymbol{\phi}_t - \boldsymbol{\phi}_0)^2}{\Sigma_t^2}} \left(\frac{(\boldsymbol{\phi}_t - \boldsymbol{\phi}_0)^2}{2\Sigma_t^2} - 1 \right) = \frac{\sigma^{2t}}{2} \frac{1}{\sqrt{2\pi\Sigma_t^2}} \frac{1}{\Sigma_t^2} e^{-\frac{1}{2}\frac{(\boldsymbol{\phi}_t - \boldsymbol{\phi}_0)^2}{\Sigma_t^2}} \left(\frac{(\boldsymbol{\phi}_t - \boldsymbol{\phi}_0)^2}{2\Sigma_t^2} - 1 \right), \tag{2.11}$$

proving that eq. (2.8) is a solution of the Fokker-Planck equation.

For the generative process, it is necessary to know the probability density at time T, which we set to $T = 1$. If σ is sufficiently large, i.e. if the normal distribution p_{0t} is sufficiently broad, the initial probability distribution p_0 can be approximated as a delta function $p_0(\boldsymbol{\phi}_0) \approx \delta(\boldsymbol{\phi}_0)$ for $t \to 1$, see fig. 2.2. This leads to

$$p_T(\boldsymbol{\phi}_T) \approx \mathcal{N}\left(\mathbf{0}, \frac{1}{2\log(\sigma)}(\sigma^{2t} - 1)\mathbf{I}\right), \tag{2.12}$$

which is now independent of the initial data at $t = 0$ and thus straightforward to sample from.

2.3 Reverse diffusion process

For a diffusion process given by eq. (2.1), there exists a reverse diffusion process [12] which takes the form

$$\mathrm{d}\boldsymbol{\phi} = \left[\mathbf{f}(\boldsymbol{\phi}, t) - g(t)^2 \nabla_{\boldsymbol{\phi}} \log p_t(\boldsymbol{\phi})\right] \mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}, \tag{2.13}$$

with t flowing backwards from T to 0 and $\bar{\mathbf{w}}$ describing a time-reversed Wiener process. Starting point for this reverse-time SDE are samples drawn from $p_T(\phi_T)$. The only term in the reverse SDE not explicitly given by the form of the forward SDE is $\nabla_{\boldsymbol{\phi}} \log p_t(\boldsymbol{\phi})$, which is called 'score' and will subsequently be modelled by a neural network. For our choice of the forward SDE, the reverse SDE reads

$$\mathrm{d}\boldsymbol{\phi} = -\sigma^{2t} \nabla_{\boldsymbol{\phi}} \log p_t(\boldsymbol{\phi})\mathrm{d}t + \sigma^t \mathrm{d}\bar{\mathbf{w}}. \tag{2.14}$$

2.3.1 ODE formulation

In [6] it is shown that for a diffusion process given by eq. (2.1), there exists a deterministic process whose trajectories have the same marginal probability densities (i.e. the probability densities at each timepoint of the flow) as trajectories following the SDE. This deterministic process is characterized by an ordinary differential equation (ODE),

$$\mathrm{d}\boldsymbol{\phi} = \left(\mathbf{f}(\boldsymbol{\phi}, t)\mathrm{d}t - \frac{1}{2} g(t)^2 \nabla_{\boldsymbol{\phi}} \log p_t(\boldsymbol{\phi})\right) \mathrm{d}t, \tag{2.15}$$

which takes the form

$$\mathrm{d}\boldsymbol{\phi} = -\frac{1}{2} \sigma^{2t} \nabla_{\boldsymbol{\phi}} \log p_t(\boldsymbol{\phi})\mathrm{d}t \tag{2.16}$$

for our chosen SDE eq. (2.5). In this formulation, the forward and the reverse process are described by the same ODE.

By solving either eq. (2.13) or eq. (2.15) backwards in time from $t = T$ to $t = 0$, we can reverse the diffusion process given by eq. (2.1). Thus, to draw a sample from p_0, we can first draw a sample from p_T and then solve the reverse diffusion process.

2.4 Score matching

To solve the reverse diffusion process presented in the previous section, we need to calculate the score $\nabla_{\boldsymbol{\phi}} \log p_t(\boldsymbol{\phi})$ for arbitrary $\boldsymbol{\phi}$ and t. As we do not have an analytical expression for the score, we estimate it with a trained neural network. This neural network, $\mathbf{s}_\theta(\boldsymbol{\phi}, t)$, is termed score model.

Following eq. (2.16), the ODE we solve in the generative process is

$$\mathrm{d}\boldsymbol{\phi} = -\frac{1}{2}\sigma^{2t}\mathbf{s}_\theta(\boldsymbol{\phi}, t)\mathrm{d}t, \tag{2.17}$$

and the reverse SDE eq. (2.14) is changed analogously.

To train the network, we use denoising score matching [13], with the loss function given as in [6],

$$\mathbb{E}_{t\in\mathcal{U}(0,T)}\mathbb{E}_{p_0(\boldsymbol{\phi}_0)}\mathbb{E}_{p_{0t}(\boldsymbol{\phi}_t|\boldsymbol{\phi}_0)}\left[\lambda(t)\|\mathbf{s}_\theta(\boldsymbol{\phi}_t, t) - \nabla_{\boldsymbol{\phi}_t}\log p_{0t}(\boldsymbol{\phi}_t|\boldsymbol{\phi}_0)\|_2^2\right], \tag{2.18}$$

where $\mathcal{U}(0,T)$ denotes the uniform distribution over the interval $[0,T]$ and $\mathbb{E}_p$ the expectation value over the probability distribution p.

The weighting function $\lambda(t)$ is chosen as $\lambda(t) \propto 1/\mathbb{E}\|\nabla_{\boldsymbol{\phi}_t}\log p_{0t}(\boldsymbol{\phi}_t|\boldsymbol{\phi}_0)\|_2^2$ to ensure that the magnitude of the loss function remains constant across all t.

For our chosen SDE, p_{0t} is explicitly given by eq. (2.6), and we can write down the gradient of the logarithm of p_{0t}, using eq. (2.9):

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}_t}\log p_{0t}(\boldsymbol{\phi}_t|\boldsymbol{\phi}_0) &= \nabla_{\boldsymbol{\phi}_t}\log\left(\frac{1}{\Sigma_t\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{\boldsymbol{\phi}_t-\boldsymbol{\phi}_0}{\Sigma_t})^2}\right) \\
&= \nabla_{\boldsymbol{\phi}_t}\left(-\log\left(\Sigma_t\sqrt{2\pi}\right) - \frac{1}{2}\left(\frac{\boldsymbol{\phi}_t-\boldsymbol{\phi}_0}{\Sigma_t}\right)^2\right) \\
&= -\frac{\boldsymbol{\phi}_t-\boldsymbol{\phi}_0}{\Sigma_t^2}.
\end{aligned} \tag{2.19}$$

After setting $\lambda(t) = \Sigma_t^2$, we can write the loss function as

$$\mathbb{E}_{t\in\mathcal{U}(0,T)}\mathbb{E}_{p_0(\boldsymbol{\phi}_0)}\mathbb{E}_{p_{0t}(\boldsymbol{\phi}_t|\boldsymbol{\phi}_0)}\left[\|\Sigma_t\mathbf{s}_\theta(\boldsymbol{\phi}_t, t) + \frac{\boldsymbol{\phi}_t-\boldsymbol{\phi}_0}{\Sigma_t}\|_2^2\right]. \tag{2.20}$$

To account for $\mathbb{E}_{p_0(\boldsymbol{\phi}_0)}$, we need training samples drawn from p_0. The complete training algorithm with a given training data set $\{\boldsymbol{\phi}_0'\}$ is given in algorithm 1.

Algorithm 1 Training algorithm for the score model.

for n in number of training epochs **do**
  Shuffle training samples randomly
  for i in number of training samples / batch size **do**
    Use training samples from $i \cdot$ batch size to $(i+1) \cdot$ batch size
    Sample a batch of timesteps t from the uniform distribution over $[0,T]$
    Sample $\boldsymbol{\phi}_t'$ from p_{0t} eq. (2.6) with the respective $\boldsymbol{\phi}_0'$ and t
    Calculate the loss eq. (2.20)
    Update the parameters of the network $\mathbf{s}_\theta$ accordingly
  end for
end for

2.5 Model probability

For a given ODE $\mathrm{d}\boldsymbol{\phi} = f(\boldsymbol{\phi}, t)\mathrm{d}t$ with an initial condition for the probability density, it is possible to calculate the probability density after time evolution. With t going from T to 0, the formula reads [6]

$$\log p_0(\boldsymbol{\phi}_0) = \log p_T(\boldsymbol{\phi}_T) + \int_0^T \nabla_{\boldsymbol{\phi}_t} \cdot f(\boldsymbol{\phi}_t, t)\, \mathrm{d}t. \tag{2.21}$$

For our chosen ODE eq. (2.17) the log-likelihood is given as

$$\log p_0(\boldsymbol{\phi}_0) = \log p_T(\boldsymbol{\phi}_T) - \frac{1}{2}\int_0^T \sigma^t \nabla_{\boldsymbol{\phi}_t} \cdot \mathbf{s}_\theta(\boldsymbol{\phi}_t, t)\, \mathrm{d}t. \tag{2.22}$$

Since numerically calculating the divergence of a high-dimensional function can be very costly, we instead utilize the Skilling-Hutchinson trace estimator [14, 15]:

$$\nabla_{\boldsymbol{\phi}_t} \cdot \mathbf{s}_\theta(\boldsymbol{\phi}_t, t) = \mathbb{E}_{p(\epsilon)}\left[\epsilon^T \nabla_{\boldsymbol{\phi}_t} \mathbf{s}_\theta(\boldsymbol{\phi}_t, t)\epsilon\right] \quad \text{with} \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{2.23}$$

2.6 Action model

Score models were initially developed mainly for image generation and not for usage in physics applications. Even though the utilization of a diffusion process is inspired by physics, there is no direct connection between score models and the quantum field theories we aim to examine in this work.

To assign the neural network used for score based generative modelling a more direct physical interpretation, we propose a new model, which replaces the score model introduced in section 2.4. For this we first note that in a quantum field theory, the probability density of field configurations is determined by the action S of the theory,

$$p(\boldsymbol{\phi}) = \frac{1}{Z} e^{-S(\boldsymbol{\phi})}, \quad Z = \int \mathcal{D}[\boldsymbol{\phi}] e^{-S(\boldsymbol{\phi})}. \tag{2.24}$$

Looking at the 'score' of the theory, the gradient of the logarithm of the probability distribution gives

$$\nabla_{\boldsymbol{\phi}} \log p(\boldsymbol{\phi}) = -\nabla_{\boldsymbol{\phi}} S(\boldsymbol{\phi}). \tag{2.25}$$

This shows that the score model $\mathbf{s}_\theta(\boldsymbol{\phi}, t)$ approximates the negative gradient of the action of the field theory at each timestep of the diffusion process (note that effectively there is a different field theory at every timestep), as discussed in [4]. Inspired by this observation, we propose to have a neural network $S_\theta(\boldsymbol{\phi}, t)$ that approximates the action itself instead of its gradient. This results in modifications to the ODE eq. (2.17),

$$\mathrm{d}\boldsymbol{\phi} = -\frac{1}{2}\sigma^{2t} \nabla_{\boldsymbol{\phi}} S_\theta(\boldsymbol{\phi}, t)\mathrm{d}t, \tag{2.26}$$

analogously to the reverse SDE eq. (2.14) and to the loss function eq. (2.20),

$$\mathbb{E}_{t\in\mathcal{U}(0,T)}\mathbb{E}_{p_0(\boldsymbol{\phi}_0)}\mathbb{E}_{p_{0t}(\boldsymbol{\phi}_t|\boldsymbol{\phi}_0)}\left[\|\Sigma_t\nabla_{\boldsymbol{\phi}}S_\theta(\boldsymbol{\phi}_t,t)+\frac{\boldsymbol{\phi}_t-\boldsymbol{\phi}_0}{\Sigma_t}\|_2^2\right]. \tag{2.27}$$

Using this 'action model' instead of the score model gives us direct access to the action (apart from the normalization constant Z which is irrelevant for the behaviour of the system) along the whole diffusion process. In principle, one could also integrate over the score (which is done in [4] for a toy model), but that is computationally very costly for higher dimensional problems. Knowing the action along the diffusion process may help us connect generative diffusion models with renormalization group theory, which will also be discussed in chapter 6.

In contrast to image generation, when working on physical problems the action of the theory we want to sample from is known in analytical form, and one could incorporate this knowledge into the design of the neural network. Furthermore, symmetry conditions which the action of a theory must fulfil are valid along the whole diffusion process and can be built into the network architecture. Another advantage is that the output of the action model is a scalar, in contrast to it being a field for the score model, possibly reducing the complexity of the network architecture. A drawback of our method is that it is necessary to calculate the gradient of the network output during the generative process and second derivatives during training. On the other hand, the gradient of S_θ is definitely the gradient of a log probability, even before training, while the score model only ever approximates the gradient of a log probability. In the following sections we will compare the performance of both approaches.

3 Toy model

3.1 Theory

To illustrate the concepts introduced in the previous section, we first study them on a toy model, a zero-dimensional field theory with only a single degree of freedom, $\phi \in \mathbb{R}$. The action is given as

$$S(\phi) = a\phi^2 + b\phi^4, \quad a, b \in \mathbb{R}, \tag{3.1}$$

and thus the probability distribution of ϕ as

$$p(\phi) = \frac{1}{Z} e^{-(a\phi^2 + b\phi^4)} \tag{3.2}$$

with

$$Z = \int_{\infty}^{\infty} \mathrm{d}\phi \, e^{-(a\phi^2 + b\phi^4)}. \tag{3.3}$$

We require $b \geq 0$ as otherwise the integral in eq. (3.3) would diverge. Expectation values of observables are given by

$$\langle O(\phi) \rangle = \frac{1}{Z} \int_{\infty}^{\infty} \mathrm{d}\phi \, O(\phi) e^{-(a\phi^2 + b\phi^4)}. \tag{3.4}$$

Due to the simplicity of this theory, it is possible to calculate these integrals analytically, which we will use as reference values for our trained models. Specifically, we will look at the magnetization $\langle M \rangle = \langle \phi \rangle$, the susceptibility $\chi = \langle \phi^2 \rangle - \langle |\phi| \rangle^2$ and the Binder cummulant $U_L = 1 - \langle \phi^4 \rangle / (3 \langle \phi^2 \rangle^2)$.

3.2 Methods

To train our model, we first generate 100 000 samples drawn from eq. (3.2) using stochastic quantization [16], with parameters $a = -0.9$ and $b = 0.4$. We chose a negative a to have two separate maxima in the probability distribution eq. (3.2), as this is qualitatively more different from the normal distribution after the diffusion process eq. (2.12) compared to the probability distribution with positive a. We thus expect it to be harder to learn for the network. In fig. 3.1, we show the resulting distribution over ϕ.

We use a simple neural network architecture consisting of five linear layers. In the special case of a single degree of freedom, the architecture for the score model

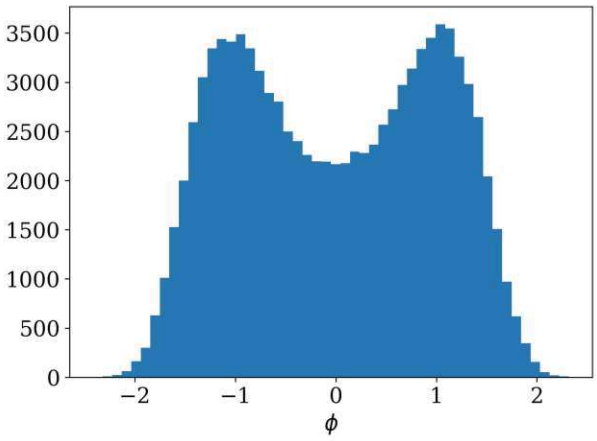


Figure 3.1: Histogram of the field values along ϕ of the training samples.

	Magnetization	Susceptibility	Binder cummulant
Analytical values	0	0.2203	0.4191
Training samples	0.001(4)	0.221(1)	0.418(2)
Score model	0.00(1)	0.220(4)	0.413(6)
Action model	0.00(1)	0.223(3)	0.422(5)

Table 3.1: Comparison of observables.

and the action model is identical. It takes the field value and the diffusion time as inputs and returns a single value, which models either the action or the derivative of the action.

For training, we use the Adam optimizer [17] with a learning rate of $5 \cdot 10^{-4}$ and choose a batch size of 512. After 1 000 training epochs, we are able to generate samples whose observables match the values directly calculated with eq. (3.4).

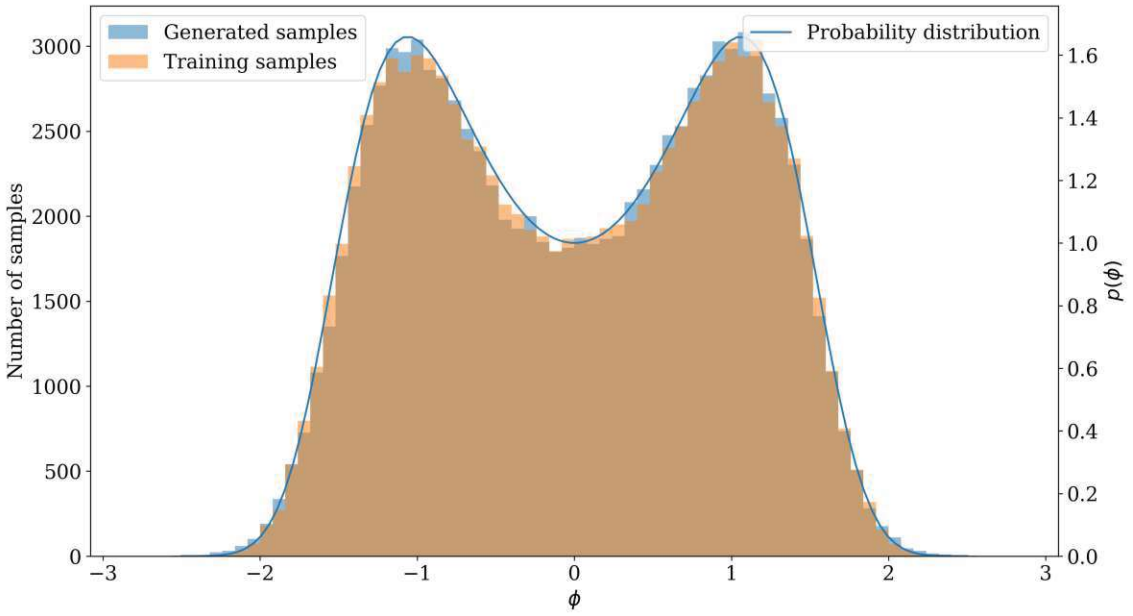
3.3 Results

We find that after training, both the score model and the action model are able to generate samples that follow the targeted probability distribution, as shown in fig. 3.2.

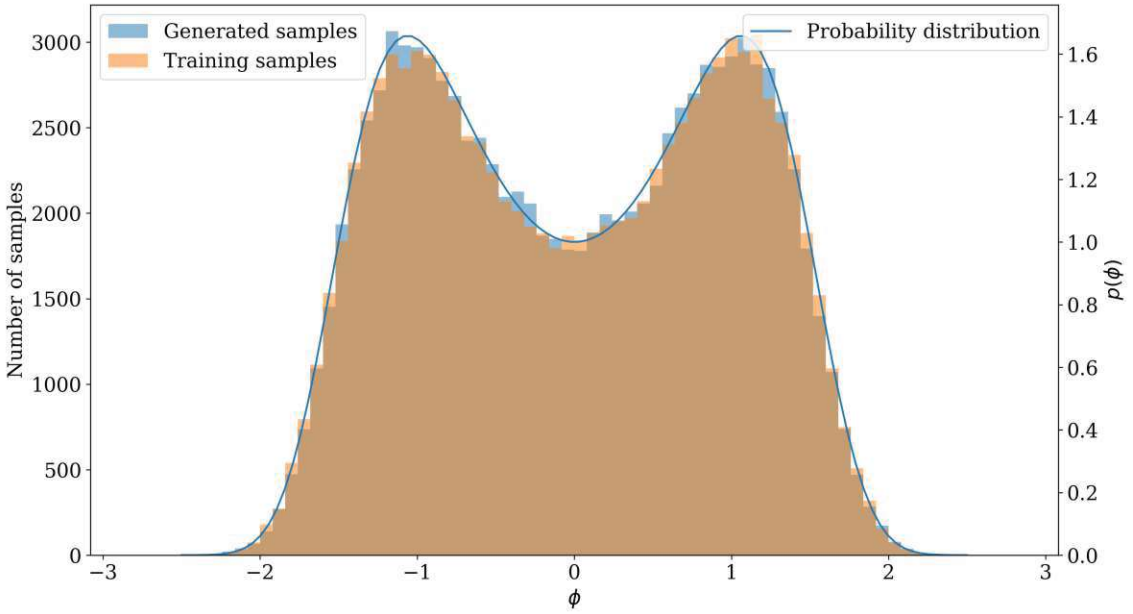
Furthermore, we calculate the expectation values of the previously introduced observables over 10 000 samples generated by the trained models and compare them to those of the training samples and the analytical values. The results are shown in table 3.1.

3.3.1 Learned action

With the trained action model, we can now observe the action along the whole diffusion process, simply by evaluating the neural network. The expected behaviour



(a) Score model



(b) Action model

Figure 3.2: Histogram of the field values of the samples generated with the trained models, compared with the training samples and the shape of the probability distribution given by the action eq. (3.2).

for the action is to continuously transform from the action of the target theory eq. (3.1) at $t = 0$ to the action corresponding to the normal distribution given by eq. (2.12) at $t = 1$. In fig. 3.3 we see that the network indeed exhibits this behaviour, in agreement with the expected actions at both ends of the diffusion process.

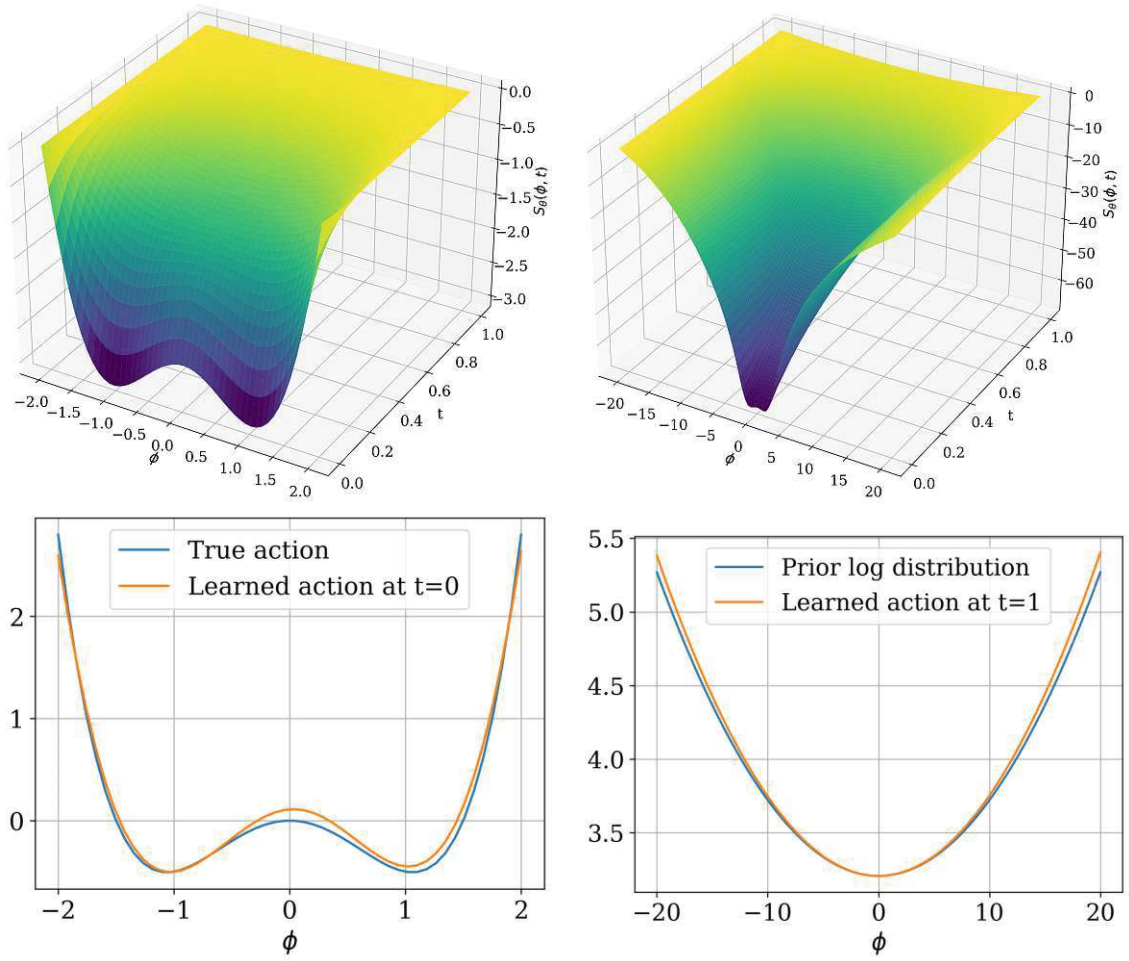


Figure 3.3: (Top) Network output depending on the diffusion time t and the field for two different value ranges of ϕ. For each time slice, the values are shifted by a constant such that the maximum value remains constant. This is only for better visualization, as mentioned a constant shift of the action has no effect on the theory it describes. (Bottom) The action learned by the network compared with the expected action at $t = 0$ and $t = 1$, shifted so that the minima of both curves take the same value.

4 Scalar field theory

4.1 Theory

We consider a scalar field theory in two dimensional Euclidean space with the action given as

$$S_E[\phi_0] = \int \mathrm{d}^2\mathbf{x}\left[\frac{1}{2}\sum_{\mu=1}^{2}(\partial_\mu\phi_0(\mathbf{x}))^2 + \frac{1}{2}m_0^2\phi_0(\mathbf{x})^2 + \frac{\lambda_0}{4!}\phi_0(\mathbf{x})^4\right]. \tag{4.1}$$

The field is discretized onto a two-dimensional lattice with N^2 grid-points denoted by position vectors $\mathbf{n} = (n_1, n_2)$, where $n_1, n_2 \in [1, N]$, and a lattice spacing of a, such that $\mathbf{x} = a\mathbf{n}$. We follow the convention presented in chapter 7 of [18], where the differentials are replaced by finite differences and the field is rescaled, such that the discretized action on the lattice can be written as

$$S_L(\phi) = \sum_{\mathbf{n}}\left[-2\kappa\sum_{\mu=1}^{2}\phi(\mathbf{n})\phi(\mathbf{n}+a\hat{\mu}) + (1-2\lambda)\phi(\mathbf{n})^2 + \lambda\phi(\mathbf{n})^4\right], \tag{4.2}$$

with $\hat{\mu}$ being the unit vector in direction μ. The rescaled field ϕ, the hopping parameter κ and the dimensionless coupling constant λ in eq. (4.2) are related to the parameters in eq. (4.1) as

$$\phi_0(\mathbf{x}) = \sqrt{2\kappa}\phi(\mathbf{x}), \tag{4.3}$$

$$\lambda_0 = \frac{6\lambda}{a^2\kappa^2}, \tag{4.4}$$

$$m_0^2 = \frac{1-2\lambda}{\kappa a^2} - \frac{4}{a^2}. \tag{4.5}$$

We assume periodic boundary conditions.

The probability density of the field is given by

$$p(\phi) = \frac{1}{Z}e^{-S_L(\phi)}, \tag{4.6}$$

with

$$Z = \prod_{\mathbf{n}}\int_{-\infty}^{\infty}\mathrm{d}\phi(\mathbf{n})\, e^{-S_L(\phi)} \tag{4.7}$$

and thus, observables by

$$\langle O \rangle = \frac{1}{Z} \prod_{\mathbf{n}} \int_{-\infty}^{\infty} \mathrm{d}\phi(\mathbf{n})\, O(\phi) e^{-S_L(\phi)}. \tag{4.8}$$

As in the previous chapter, we calculate some observables to assess the quality of our trained models. Namely the expectation value of the magnetization, the susceptibility and the Binder cumulant, which are defined as

$$M = \frac{1}{N^2} \sum_{\mathbf{n}} \phi(\mathbf{n}), \tag{4.9}$$

$$\langle M \rangle = \left\langle \frac{1}{N^2} \sum_{\mathbf{n}} \phi(\mathbf{n}) \right\rangle, \tag{4.10}$$

$$\chi_2 = N^2 \big(\langle M^2 \rangle - \langle |M| \rangle^2 \big), \tag{4.11}$$

$$U_L = 1 - \frac{1}{3} \frac{\langle M^4 \rangle}{\langle M^2 \rangle^2}. \tag{4.12}$$

We note that in the literature, the susceptibility is usually written down as $\chi_2 = N^2 \big(\langle M^2 \rangle - \langle M \rangle^2 \big)$, but the absolute value is essential for the correct behaviour as response function. Due to the structure of the action, $\langle M \rangle$ is always zero, hence the usual definition would effectively be $\chi_2 = N^2 \langle M^2 \rangle$, which does not exhibit a peak around the phase transition. Only when one is stuck within one of the two potential sinks, e.g. in a single Monte Carlo Markov chain, $\langle M \rangle^2 = \langle |M| \rangle^2$ and both definitions would be equivalent. Presumably, the widespread use of methods where configurations are generated through Markov chains led to the common usage of the alternative susceptibility definition. As samples generated by diffusion models are by construction independent of each other, we need to use eq. (4.10).

Scalar ϕ^4 theory in two dimensions exhibits two phases, with the phase transition in the classical limit at

$$\kappa_{c0} = \frac{1 - 2\lambda}{4}. \tag{4.13}$$

In the symmetric phase where $\kappa < \kappa_c$, the system is disordered, i.e. configurations are almost random noise with mean zero and thus $\langle |M| \rangle \approx 0$. In contrast to that, the field values of configurations of the broken phase ($\kappa > \kappa_c$) fluctuate around a non-zero value or minus that value. The sign is completely random (spontaneous symmetry breaking). Hence $\langle |M| \rangle$ is given by the absolute value of said number. Figure 4.1 shows the behaviour of both $\langle |M| \rangle$ and the susceptibility around the phase transition.

4.2 Methods

Training samples on a grid of 32×32 are generated using the Hybrid Monte Carlo (HMC) method [19]. We generate 10 000 training samples and another 1 000 test

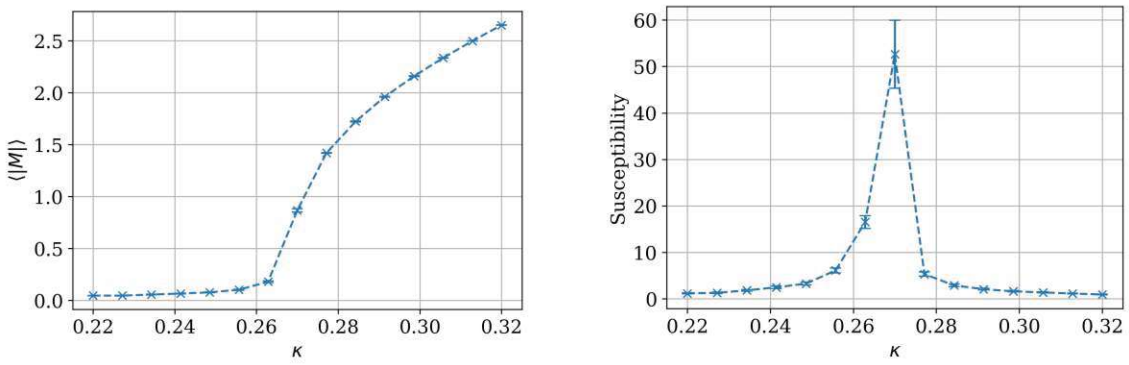


Figure 4.1: Illustration of the effect of the phase transition on the expectation value of the absolute value of the magnetization and the susceptibility for $\lambda = 0.02$ over varying κ. The phase transition lies roughly at $\kappa_c \approx 0.27$. These values were calculated from samples we generated using the Hybrid Monte Carlo (HMC) method we also use for generating our training samples. Note that the classical critical parameter given by eq. (4.13) is $\kappa_{c0} = 0.24$, which differs from the actual critical point.

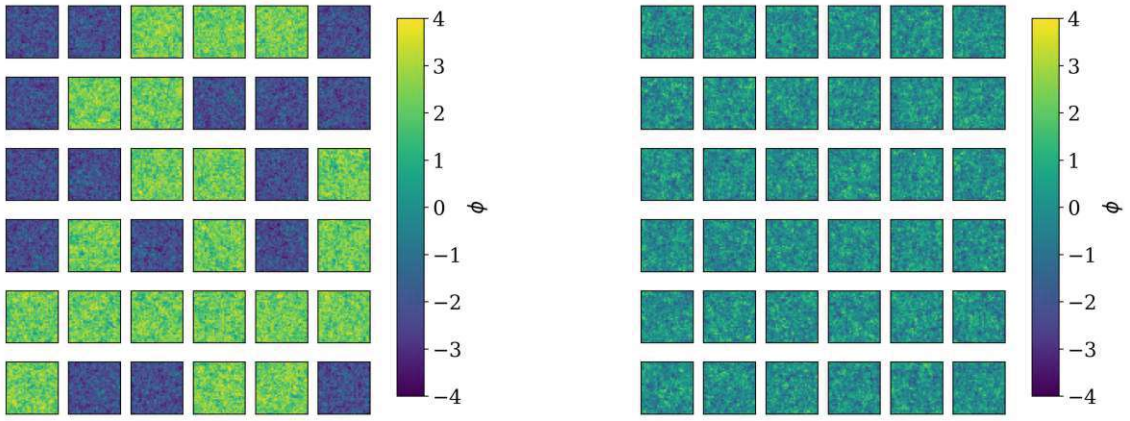
samples which are not used in the training process, but only for comparison with the trained models. This is done for both the broken phase and the symmetric phase, with parameters $\lambda = 0.02$ for both and $\kappa = 0.3$ ($\kappa = 0.22$) for the broken (symmetric) phase respectively. Example configurations are shown in fig. 4.2.

Digital images are described by three values at each pixel in the case of coloured images or one value by pixel for black and white images. The different degrees of freedom per pixel are commonly referred to as channels. The pixels are ordered in a rectangular two-dimensional grid. As it happens, configurations of a two-dimensional scalar field theory on the lattice have exactly the same shape as an image with one channel. Hence, we can directly employ the methods developed for image generation. (Note: pixel values for images are in the range of zero to one. To operate in a similar value range, we normalize the training samples of the scalar field theory by division with the largest field value in the whole set. This normalization is reversed after the generative process.)

4.2.1 Network architecture

For our score model, we can thus base the neural network architecture on the U-Net presented by an author of [6] for image generation[1], which we slightly adapt. The U-Net architecture was introduced in [20] and is a common choice in image generating models. The basic architecture can be seen in fig. 4.3. It consists of two subsequent parts: In the first part the number of grid points is decreased and the number of channels increased through a sequential application of convolutional layers

[1] in `https://colab.research.google.com/drive/120kYYBOVa1i0TD85RjlEkFjaWDxSFUx3?usp=sharing`, accessed 13.10.2024.



(a) Broken phase (b) Symmetric phase

Figure 4.2: Colourmap of the field values of our training samples.

and optionally also pooling layers. In the second part the initial shape is recovered through the application of convolutions and transpose convolutions. Additionally, in a U-Net the values at intermediate points within the first part are saved and later concatenated with the values of the same shape within the second part. Traditional U-Nets only depend on the two-dimensional grid of a configuration, but we require the dependence on an additional scalar parameter for the time information. Following the previously mentioned U-Net architecture, we employ sinusoidal position embedding [21] to incorporate the additional time parameter. Given a scalar input (the diffusion time t), this position embedding PE creates a vector using randomly initialized untrainable parameters:

$$PE_{2i} = \sin(2\pi t p_{2i}),\ p_{2i} \sim a\mathcal{N}(0,1), \tag{4.14}$$

$$PE_{2i+1} = \cos(2\pi t p_{2i+1}),\ p_{2i+1} \sim a\mathcal{N}(0,1). \tag{4.15}$$

We create a vector with 128 parameters and choose $a = 30$.

For the action model, we do not use a U-Net architecture, but rather a simple convolutional network, additional linear layers to achieve a scalar output and again sinusoidal position embedding for the time parameter. The detailed network architecture is shown in fig. 4.4. The convolutional layers in this build are chosen all with a kernel size of 3, a stride of 1 and circular padding of 1, such that they maintain translational invariance. Only in the last two linear layers the translation invariance is broken.

4.2.2 Training

We again use the Adam optimizer with a variable learning rate starting at 10^{-3}, which is being decreased by a factor of 0.995 every epoch, and use a batch size of 200. The training progress is visualized in fig. 4.5 by calculating observables during intermediate steps of the training process. We find that the action model converges

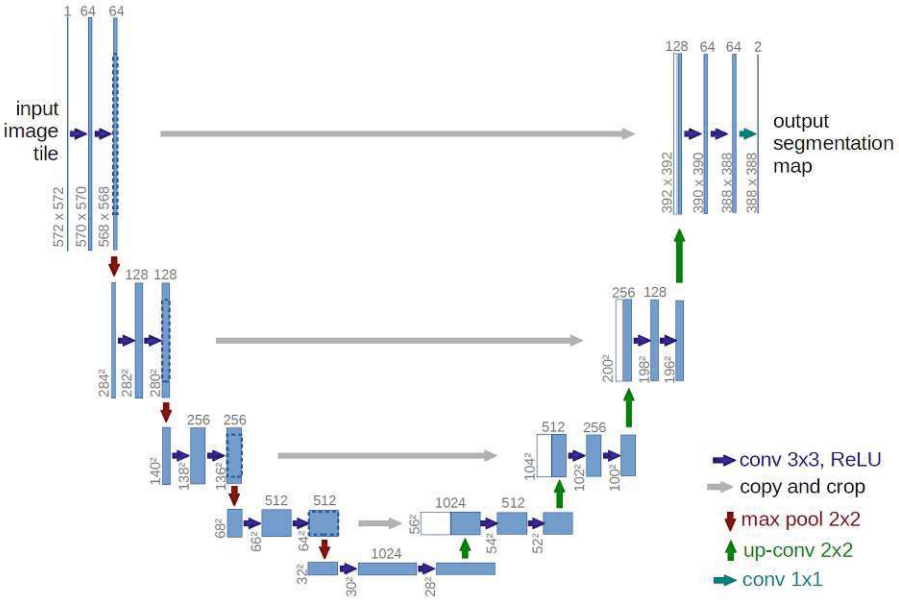


Figure 4.3: The original U-Net architecture presented in [20].

slightly faster and more continuously than the score model. This may be either due to it having less parameters to optimize or due to the fact that the score model also has to learn that it models the gradient of a scalar quantity. As mentioned previously, a drawback of the action model is that an additional derivative has to be taken compared to the score model, which also manifests itself in the training time. While a training epoch for the score model takes about 1.2 seconds, it takes 3.7 seconds for the action model, using a Nvidia GeForce RTX 3090.

4.2.3 SDE vs. ODE

Initially, the reverse ODE eq. (2.15) seems like a more attractive choice for sample generation, compared to the reverse SDE eq. (2.13), due to the broader availability of pre-built solvers for ordinary differential equations. As discussed in section 2.3.1, for a network that perfectly describes the score (or the action), both approaches are equivalent. But a neural network can only ever be trained to be almost perfect, and we found that using the SDE instead of the ODE results in measurable differences. To illustrate this, we train a network with training samples of the broken phase. In this case, the magnetization of the training samples is distributed either around a certain positive value or around minus that value. No configuration exhibits a magnetization around zero, as shown for the test samples in fig. 4.6. After having trained a network, we find that generating samples with the SDE yields the same distribution of the magnetization, but when using the ODE with the exact same network we also generate a few samples where the magnetization lies around zero, see fig. 4.6. The observable that is most sensitive to these outliers in the magnetization is the susceptibility. In fig. 4.7 we show the evolution of the susceptibility during the

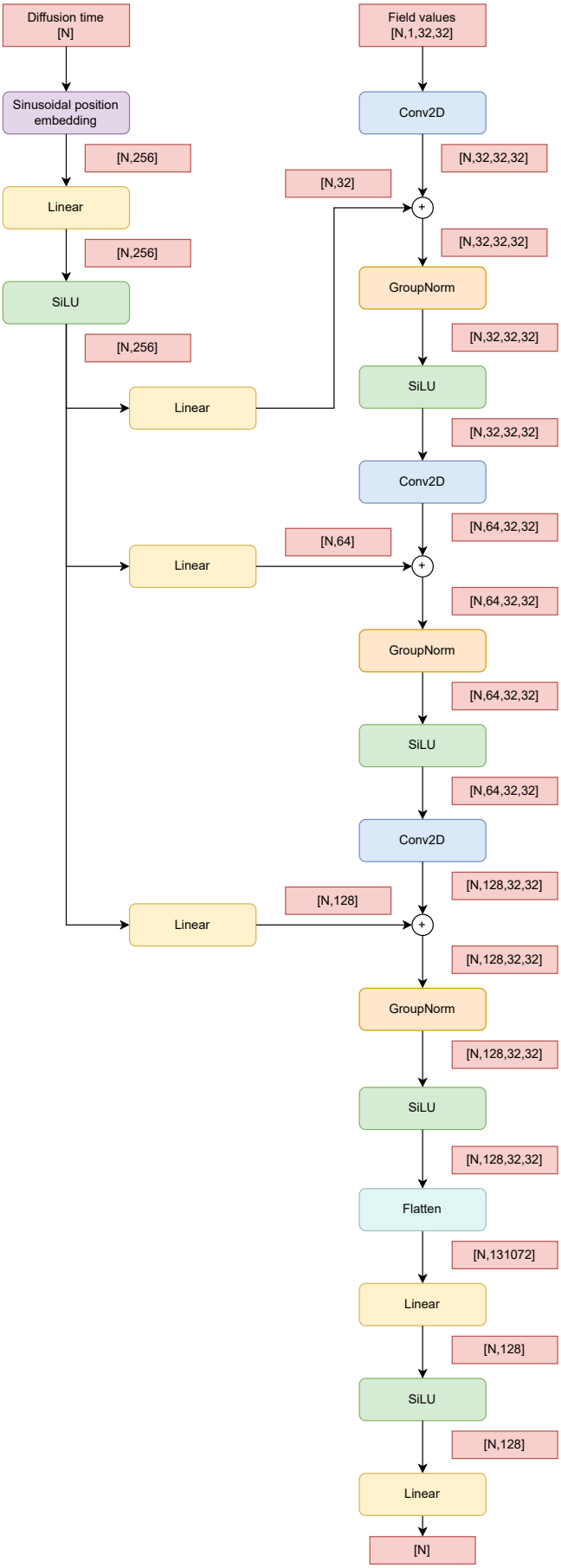


Figure 4.4: Network architecture of the action model. N is the batch size.

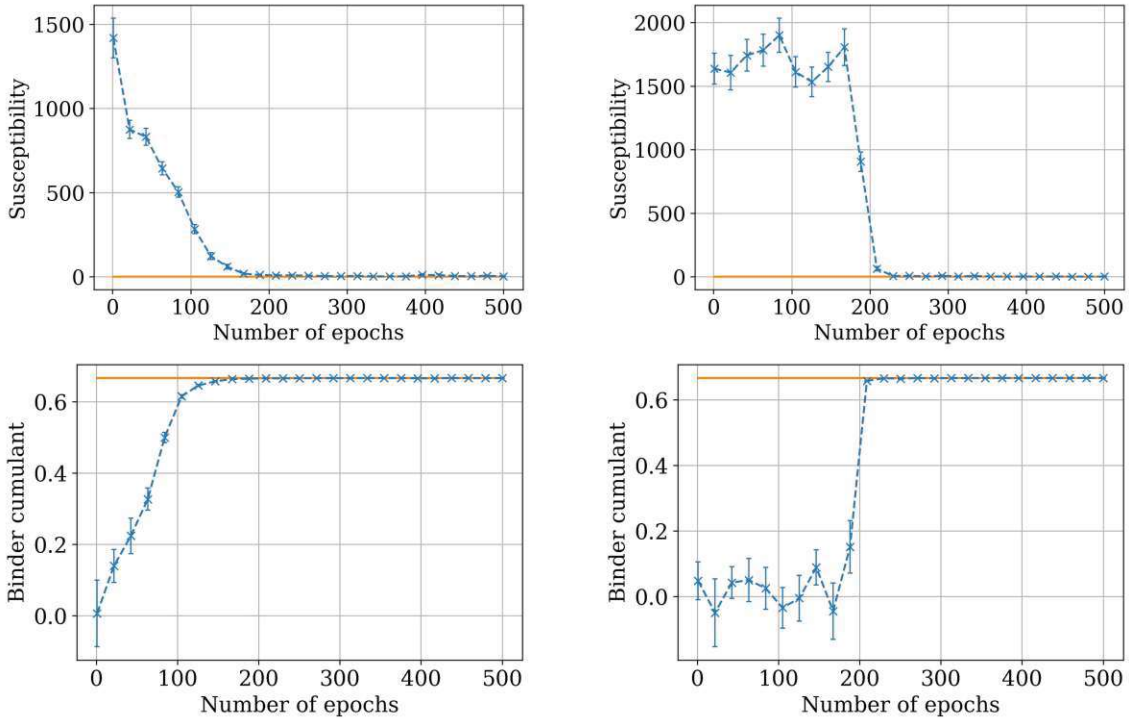
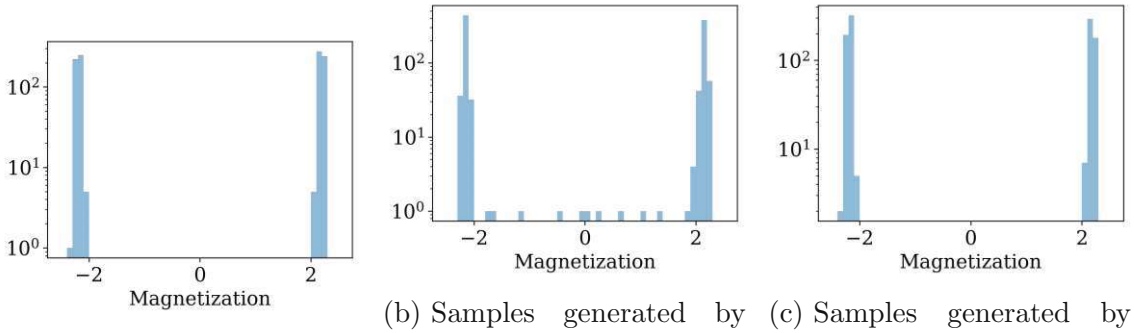


Figure 4.5: Comparison of the training progress in the broken phase for the action model (left) and the score model (right). Every twenty epochs, 1 000 samples are generated and used to calculate the susceptibility and the Binder cumulant. The orange line corresponds to the value of the training samples.



(a) Test samples

(b) Samples generated by solving the ODE

(c) Samples generated by solving the SDE

Figure 4.6: Histograms showing the distribution of the magnetization, for 1 000 samples each.

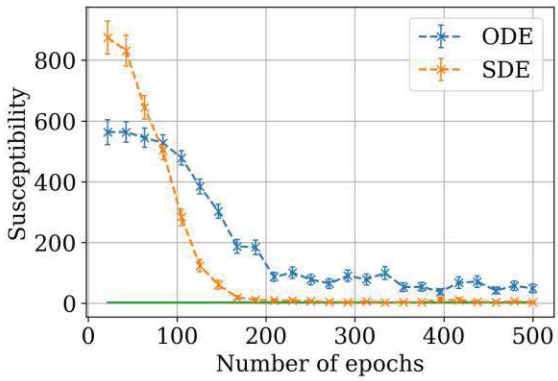


Figure 4.7: Comparison of the training progress for the SDE and the ODE. After every 20 training epochs, we generate 1 000 samples each with the SDE and the ODE approach, using the same network. The green line represents the susceptibility of the training samples.

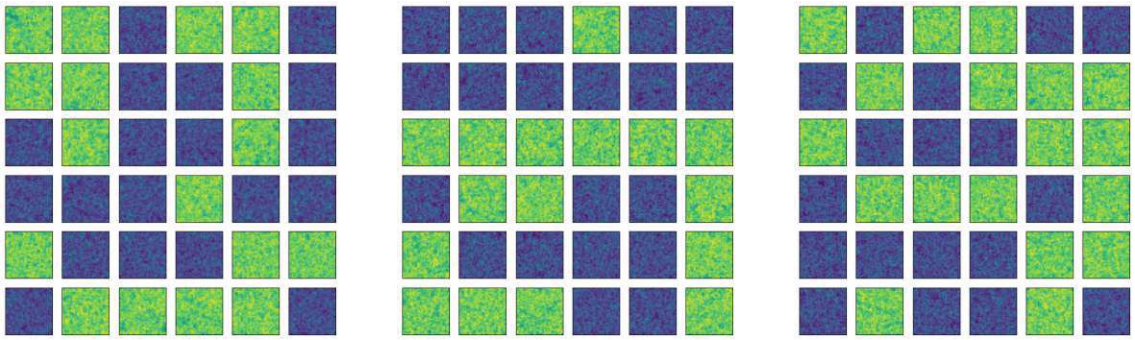
training process both for samples generated with the ODE and with the SDE. One can clearly see that the SDE approach generates samples which match the training samples more closely. Hence, we only use the SDE for sample generation going forward.

To solve the ODE, we use the fourth-order Runge-Kutta method RK4, and for the SDE the Euler-Maruyama method, both with a step size of $5 \cdot 10^{-3}$.

4.3 Results

We find that with both the action and the score model it is possible to generate samples which match the targeted distribution of field configurations. Examples for the broken phase are shown in fig. 4.8.

Comparing the distribution of the magnetization of generated samples and those of the test samples in fig. 4.9, we see very good alignment for both the symmetric



(a) Test samples (b) Action model (c) Score model

Figure 4.8: Comparison of randomly chosen test samples and samples generated with trained action and score models. The colours indicate the field values at each grid point.

	$\langle M \rangle$	χ_2	U_L
Training samples	0.0000(8)	1.17(3)	−0.01(2)
Test samples	0.002(3)	1.20(9)	0.05(7)
Score model	0.002(3)	1.3(2)	−0.5(7)
Action model	−0.001(3)	1.15(8)	0.00(6)

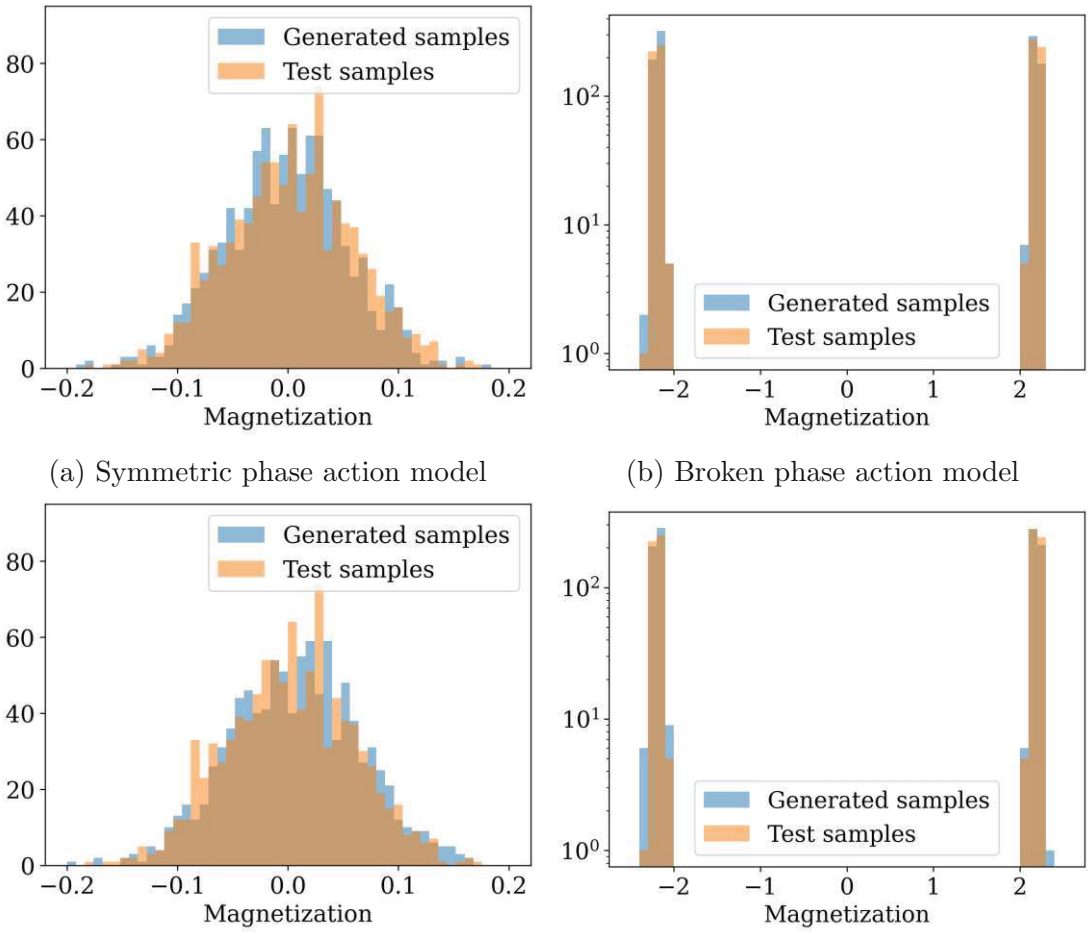
Table 4.1: Comparison of observables of the training samples, the test samples and the samples generated by our trained models, for the symmetric phase.

and the broken phase.

For a quantitative comparison, we list the calculated observables of the training samples, the test samples, and of 1 000 samples generated with the trained models in table 4.1 (symmetric phase) and table 4.2 (broken phase). The errors were calculated with the jackknife method. We find that observables of the generated samples agree with those of our test and training set. We can conclude that both models are able to generate configurations which have a probability distribution that is equivalent to the probability distribution of the configurations obtained with the HMC method.

	$\langle M \rangle$	χ_2	U_L
Training samples	0.02(3)	1.54(9)	0.66625(1)
Test samples	0.1(1)	1.6(1)	0.66624(3)
Score model	0.1(1)	1.8(1)	0.66618(3)
Action model	0.0(1)	1.8(1)	0.66618(3)

Table 4.2: Comparison of observables of the training samples, the test samples and the samples generated by our trained models, for the broken phase.



(a) Symmetric phase action model

(b) Broken phase action model

(c) Symmetric phase score model

(d) Broken phase score model

Figure 4.9: Comparison of the magnetization histograms for 1 000 test samples and 1 000 generated samples.

4.3.1 Sample likelihood and effective sampling size

As discussed in section 2.5, we are able to calculate the likelihood of a given sample to be generated by the trained model. Comparing the logarithm of this likelihood to the negative action tells us how well the model can draw samples according to the physical theory, $\log(p(\phi)) = -S(\phi) + \text{constant}$. In fig. 4.10 one can see that the probabilities of our models and the physical theory match very well.

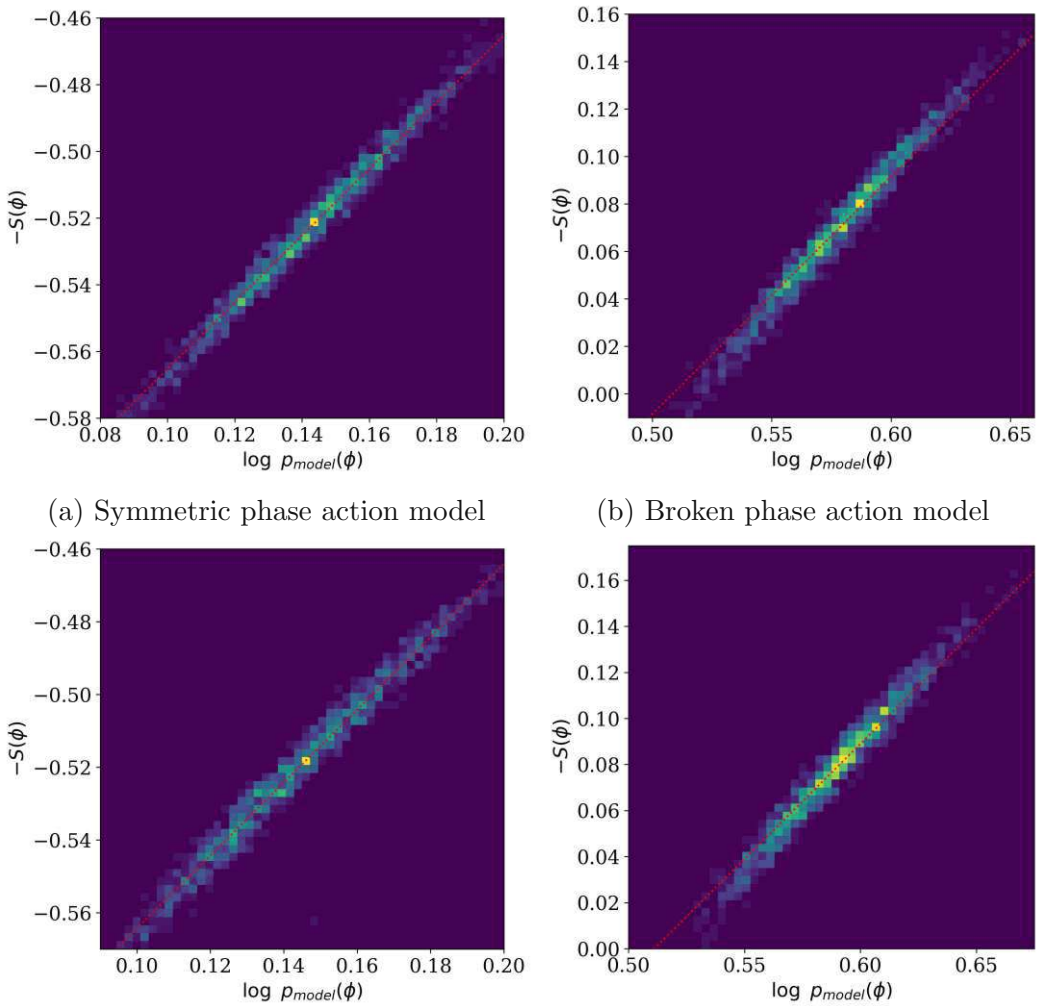
To put this observation on a numerical basis, we calculate the effective sampling size (ESS), which measures how well two probability distributions p and q match on a given set of samples. Another interpretation is that, given N samples drawn from q, the ESS gives the percentages of samples that can be used as an unbiased estimator for p. We use the definition given in [3],

$$\mathrm{ESS} = \frac{\left(\frac{1}{N}\sum_{i=1}^{N} p(\phi_i)/q(\phi_i)\right)^2}{\frac{1}{N}\sum_{i=1}^{N}\left(p(\phi_i)/q(\phi_i)\right)^2} \in [0,1]. \tag{4.16}$$

An ESS of 1 means that the probability of drawing ϕ_i from p and q is equal for all N samples. We use 1 000 generated samples for the calculation. In our case $p(\phi) = \exp(-S(\phi))$ (note that the ESS is independent of a possible multiplicative constant in p or q), and $q(\phi)$ is the estimated likelihood of our model to draw ϕ. For both the broken and the symmetric phase, we achieve an ESS of over 0.9999 with both the action and the score model.

4.3.2 Learned action

To visualize the action learned by the neural network in the action model case, we use field configurations where the field takes the same value ϕ at each lattice site. Results shown in fig. 4.11 demonstrate that the network approximates the target action eq. (4.2) for diffusion time $t \to 0$ and an action corresponding to a broad normal distribution for $t \to 1$. These results are qualitatively similar to what we achieved for the toy model in section 3.3.1, though the network output is now very coarse.



(a) Symmetric phase action model

(b) Broken phase action model

(c) Symmetric phase score model

(d) Broken phase score model

Figure 4.10: 2D histogram with the model log-likelihood on the abscissa and the negative action on the ordinate. If all values lie on a line with slope one (the red line in the plots) it means that the values match up to a constant. This type of visualization is taken from [3].

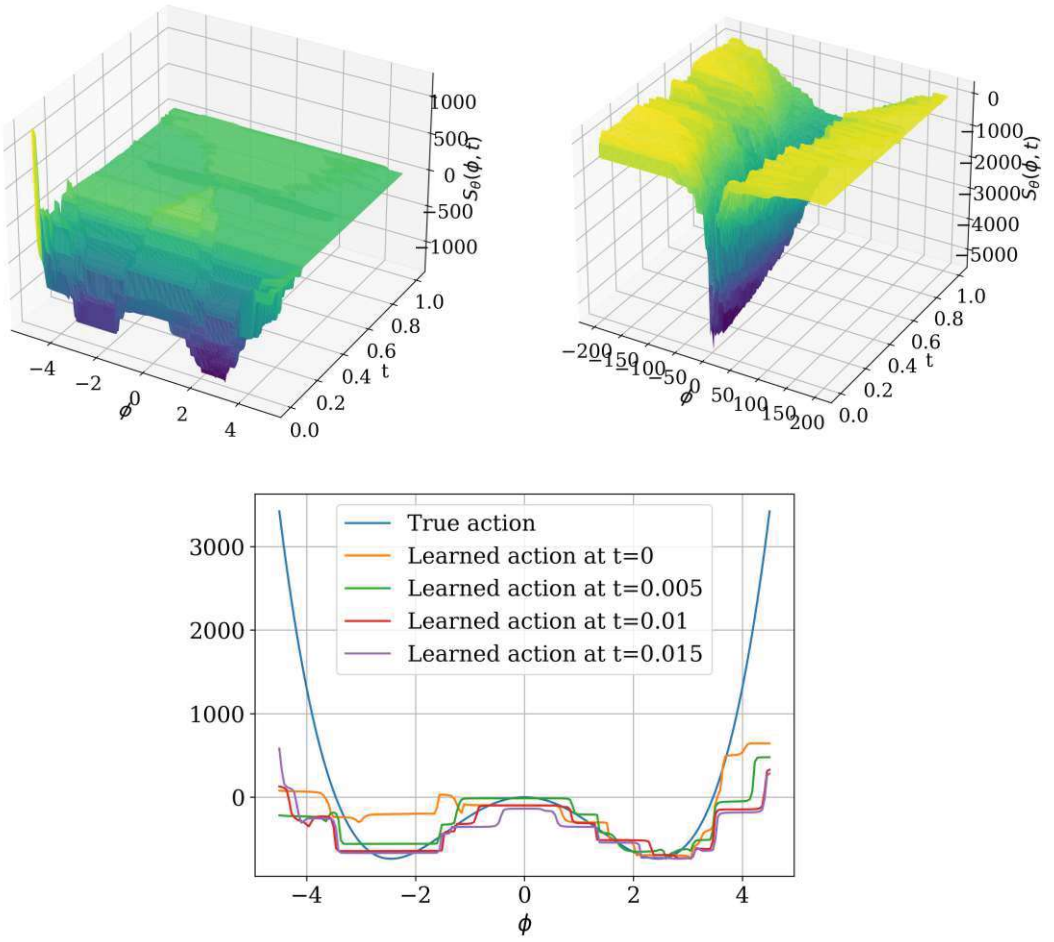


Figure 4.11: (Top) The output of the action model depending on ϕ and t for two different value-ranges of ϕ. For better visualization, at each time slice the values are shifted by a constant such that the action of ϕ_{max} remains constant. (Bottom) Comparison between the network output around diffusion time $t = 0$ and the target action given by eq. (4.2). The curves are shifted such that their minima all take the same value. These shifts are possible as a constant in the action has no physical relevance.

5 U(1) gauge field theory

5.1 Theory

We now turn our attention to lattice gauge theory, where we aim to apply the presented techniques on the simplest possible model, namely U(1) pure gauge theory. We again use a two-dimensional lattice with N^2 points, with position vectors $\mathbf{n} = (n_1, n_2)$, where $n_1, n_2 \in [1, N]$. At each lattice point, there are two so-called link variables $U_\mu(\mathbf{n}) \in \mathrm{U}(1)$, $\mu \in 1, 2$. We further define a quantity called plaquette, given by four link variables:

$$U_{\mu\nu}(\mathbf{n}) = U_\mu(\mathbf{n}) U_\nu(\mathbf{n} + \hat{\mu}) U_\mu(\mathbf{n} + \hat{\nu})^\dagger U_\nu(\mathbf{n})^\dagger \tag{5.1}$$

where $\hat{\mu}$ and $\hat{\nu}$ denote unit vectors in direction μ and ν. This plaquette is a gauge invariant object, which in the context of U(1) means that

$$U_{\mu\nu}(\mathbf{n}) = \Omega(\mathbf{n}) U_{\mu\nu}(\mathbf{n}) \Omega(\mathbf{n} + \hat{\mu})^\dagger, \tag{5.2}$$

with $\Omega(\mathbf{n})$ being an element of U(1) for each lattice site. We now use the plaquettes to write down the Wilson action [22] as

$$S_G(U) = -\beta \sum_{\mathbf{n}} \mathrm{Re}[U_{01}(\mathbf{n})]. \tag{5.3}$$

The link variables, as elements of U(1), can be written either in the complex representation, as

$$U_\mu(\mathbf{n}) \in \mathbb{C},\ |U_\mu(\mathbf{n})| = 1, \tag{5.4}$$

or in the angular representation, as

$$\theta_\mu(\mathbf{n}) \in [0, 2\pi), \text{ where } U_\mu(\mathbf{n}) = \exp(i\theta_\mu(\mathbf{n})). \tag{5.5}$$

The plaquettes in angular representation are given as

$$\theta_{\mu\nu}(\mathbf{n}) = \theta_\mu(\mathbf{n}) + \theta_\nu(\mathbf{n} + \hat{\mu}) - \theta_\mu(\mathbf{n} + \hat{\nu}) - \theta_\nu(\mathbf{n}), \tag{5.6}$$

and subsequently the action as

$$S_G(U) = -\beta \sum_{\mathbf{n}} \cos(\theta_{01}(\mathbf{n})). \tag{5.7}$$

Again, expectation values of observables are given by

$$\langle O \rangle = \frac{1}{Z} \int \mathcal{D}[U] O[U] e^{-S_G[U]}, \tag{5.8}$$

with

$$Z = \int \mathcal{D}[U] e^{-S_G[U]}. \tag{5.9}$$

Observables we will measure include the action, the topological charge Q and the topological susceptibility $\langle Q^2 \rangle$, where the topological charge is defined as

$$Q = \frac{1}{2\pi} \sum_{\mathbf{n}} \arg(U_{01}(\mathbf{n})). \tag{5.10}$$

5.2 Methods

As the variables on the lattice sites are now U(1) group elements and not just real scalars, the methodology of the score model has to be adapted accordingly. We developed methods for both the angular representation and the complex representation. While the angular representation closer resembles the case of real scalars, it is unique for U(1), hence the complex representation is a better basis to generalize these methods to SU(N) theories. The fact that there are now two variables at each grid point has no consequences for the diffusion processes.

5.2.1 Angular representation

In this case, the diffusion process itself remains unchanged, with the general SDE given as

$$\mathrm{d}\theta = \mathbf{f}(\theta, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}, \tag{5.11}$$

but one has to note that θ_μ is 2π periodic. This requires that the action, and therefore the neural network, is invariant under transformations of the form $\theta_\mu \to \theta_\mu + n \cdot 2\pi$, $n \in \mathbb{N}$. We ensure this by using the sine and cosine of the plaquettes as a input for the network. We use plaquettes calculated from the link variables as input for the network, as the action eq. (5.7) only depends on the plaquettes. For diffusion time $t \to 1$, in the scalar case the field value at each grid point would follow a broad normal distribution. Taking the 2π periodicity of θ_μ into account, this results in a constant distribution over $[0, 2\pi)$. Thus, at $t = 1$ we sample not from eq. (2.12) but from this constant distribution. No other modifications compared to the scalar field are necessary.

5.2.2 Complex representation

In this case, more modifications are needed, which we choose such that the diffusion processes are equivalent to those in the angular representation.

To obtain the diffusion equations, we use eq. (5.11) as a starting point and switch to the complex representation,

$$e^{i\mathrm{d}\theta} = e^{i\mathbf{f}(\theta,t)\mathrm{d}t} \cdot e^{ig(t)\mathrm{d}\mathbf{w}}, \tag{5.12}$$

leading to the SDE we use in the complex case:

$$\mathrm{d}U = e^{i\mathbf{f}(U,t)\mathrm{d}t} \cdot e^{ig(t)\mathrm{d}\mathbf{w}}. \tag{5.13}$$

One has to note that infinitesimal changes are induced as $U \cdot \mathrm{d}U$ instead of $U + \mathrm{d}U$, which we have to take into account when solving ODEs and SDEs.

The forward SDE can still be solved in one step. Knowing that in the angular representation

$$\theta_t = \theta_0 + \Sigma_t \eta \text{ with } \eta \sim \mathcal{N}(0,1) \tag{5.14}$$

and

$$\Sigma_t = \frac{1}{2\log(\sigma)}(\sigma^{2t} - 1), \tag{5.15}$$

we can solve the forward SDE in the complex representation as

$$U_t = U_0 e^{i\Sigma_t \eta} \text{ with } \eta \sim \mathcal{N}(0,1). \tag{5.16}$$

To solve the reverse ODE, we use a Runge-Kutta solver adapted for the complex representation, as presented in [23].

For the reverse SDE, we adapt the Euler-Maruyama method used previously. While for an SDE of the form eq. (5.11) the updating scheme is given as

$$\theta_{i+1} = \theta_n + \mathbf{f}(\theta_i, t_i)\Delta t + g(t_i)\sqrt{\Delta t}\eta \text{ with } \eta \sim \mathcal{N}(0,1), \tag{5.17}$$

for eq. (5.13) we use

$$U_{i+1} = U_i \cdot e^{i\Delta t \mathbf{f}(U_i,t_i)} e^{g(t_i)\sqrt{\Delta t}\eta} \text{ with } \eta \sim \mathcal{N}(0,1). \tag{5.18}$$

Just as in the scalar case (see section 4.2.3) we found that using the reverse SDE to generate configurations yields better results compared to the reverse ODE, hence we again use the reverse SDE for sample generation.

The prior distribution is given as a constant distribution along the complex unit circle, following what we mentioned for the angular representation.

For calculating the model probability, and in the action model case also for training and sample generation, we have to take the gradient of the neural network with respect to the complex link variables. For a given function $f : \mathbb{C} \to \mathbb{R}$ the autograd tool of PyTorch, which we use to obtain the derivatives, calculates the conjugate Wirtinger derivative,[1] given by

$$\bar{\partial} f = \frac{1}{2}\left(\frac{\partial}{\partial x} + i\frac{\partial}{\partial y}\right) f, \tag{5.19}$$

where $x = \mathrm{Re}(z)$, $y = \mathrm{Im}(z)$ and $z \in \mathbb{C}$. Since in our case z is a U(1) group element, we actually need the group derivative Df along the group manifold. It can be related to the conjugate Wirtinger derivative as

$$Df = -2\mathrm{Im}\left(z\overline{\bar{\partial} f}\right). \tag{5.20}$$

[1] see `https://pytorch.org/docs/stable/notes/autograd.html`, accessed 13.10.2024

To see this, consider the group derivative along U(1) which is defined as

$$Df(z) = \lim_{\epsilon \to 0} \frac{f(e^{i\epsilon} z) - f(z)}{\epsilon}. \tag{5.21}$$

The Taylor expansion of the first term for small ϵ is given by

$$\begin{aligned} f(e^{i\epsilon} z) &= f(z) + \partial f(z)(i\epsilon z) + \bar{\partial} f(z)(-i\epsilon \bar{z}) + O(\epsilon^2) \\ &= f(z) + \epsilon[-2\mathrm{Im}(z\partial f)] + O(\epsilon^2), \end{aligned} \tag{5.22}$$

which yields

$$Df(z) = -2\mathrm{Im}(z\partial f). \tag{5.23}$$

Since $f(z) \in \mathbb{R}$ we have $\overline{\bar{\partial} f} = \partial f$.

5.2.3 Network architecture

Some adaptions regarding the network architecture are made compared to what was presented for the scalar field in section 4.2.1. Given a configuration of link variables as input, the first step is to calculate the plaquettes, as the Wilson action depends only on the plaquettes and not the link variables directly. As mentioned before, in the angular representation we then take the sine and cosine, which we treat as two channels. Similarly, in the complex representation we use the real and the imaginary parts as channels. We continue with a convolutional network similar to the scalar case. While the time embedding is exactly as in the scalar case, we now restrict ourselves to convolutional layers with a kernel size of 3, a stride of 1 and circular padding of size 1, abandoning the U-Net approach. Convolutions of this special structure retain translational invariance leading to a translation-invariant network. Due to the use of plaquettes, it is also gauge invariant. Both are properties of the Wilson action which me manage to incorporate into the design of the network. The detailed structure of the score model is shown in fig. 5.1. The action model only differs in the last layer, where instead of a convolution we take the sum over all grid points and then reduce the number of channels with a linear layer.

Results presented in this chapter are achieved with the score model in complex representation, although we get similar results with the action model and in the angular representation.

5.2.4 Training

To train the network, we generate 10 000 samples each for $\beta = 1$, $\beta = 2$ and $\beta = 5$ using HMC. By comparison of the topological susceptibility of our training samples to the values presented in [24], we confirm that we implemented the HMC method correctly. We choose a 2D grid of 8^2 points. We optimize the network using the Adam optimizer starting at a learning rate of 10^{-3}, which decreases by a factor

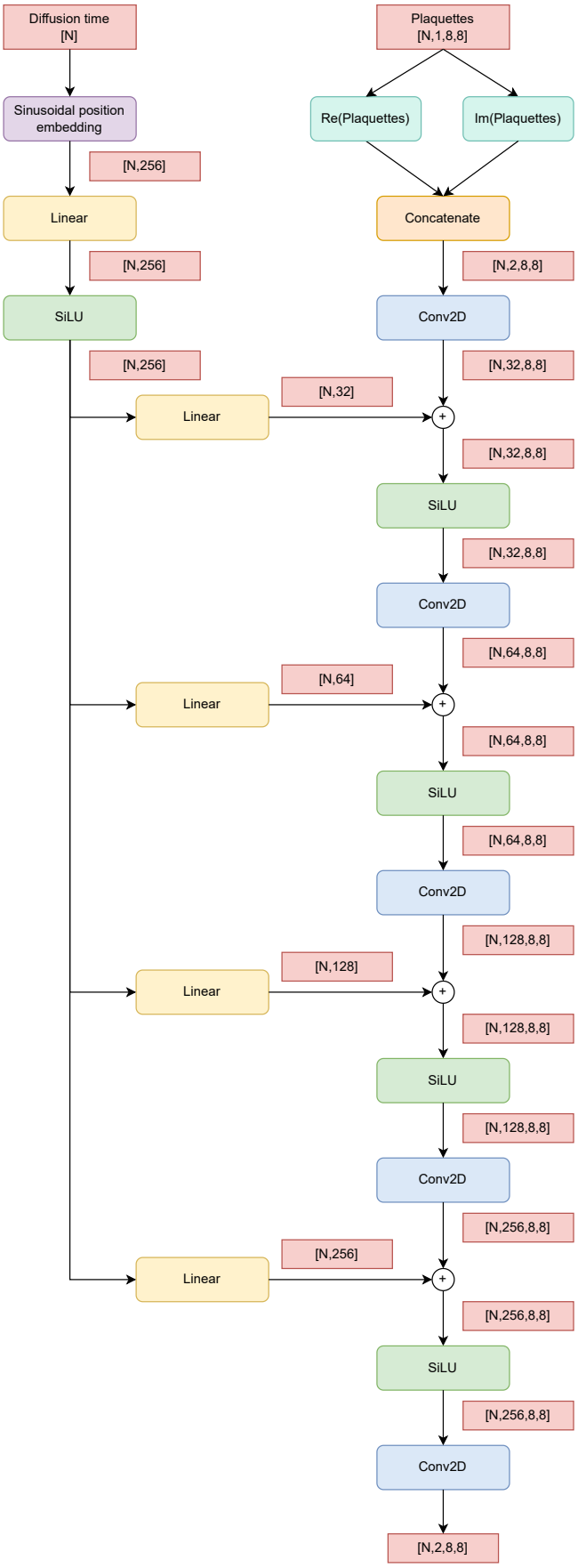


Figure 5.1: Network architecture of the score model. N is the batch size.

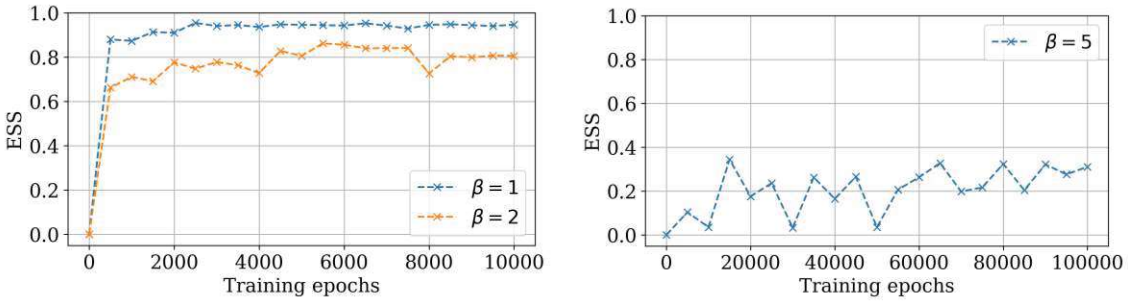


Figure 5.2: Evolution of the ESS during training. We always use 2 000 generated samples to calculate the ESS.

of 0.9997 every epoch. Using a batch size of 1 000, we do 10 000 training epochs for $\beta = 1$ and $\beta = 2$. At this point we find that the ESS eq. (4.16) does not significantly increase any further. This relates to a training time of roughly 45 minutes on a Nvidia GeForce RTX 3090. When training the model for $\beta = 5$ we train ten times as long and use a factor of 0.9999 to decrease the learning rate. Still, the ESS fluctuates significantly more than in the cases of lower β. These results are shown in fig. 5.2. We clearly see that training gets increasingly difficult for larger β. This is to be expected, as for $\beta \to 0$ link variables behave as independent random noise equivalent to the prior distribution. Hence the diffusion process is easier to learn for the network compared to configurations with high correlations between link variables.

5.3 Results

For smaller beta, we find that our trained models can generate field configurations such that expectation values of observables match with those of the training set. For larger beta, discrepancies remain after training, as is also reflected in the lower ESS shown in fig. 5.2. In table 5.1, we compare observables for $\beta = 2$ and in table 5.2 for $\beta = 5$. In fig. 5.3 we compare the action distribution and in fig. 5.4 the topological charge distribution of training samples and generated samples.

In the application for U(1) lattices, we start to see the performance advantage of using score models as sample generator. While our implementation of Hybrid Monte Carlo (using CUDA acceleration) needs about 17 minutes to generate 10 000 configurations, the trained score model only takes 45 seconds. Though one has to keep the initial training time of the score model in mind, which is in this case, as mentioned above, around 45 minutes.

5.3.1 Model probability

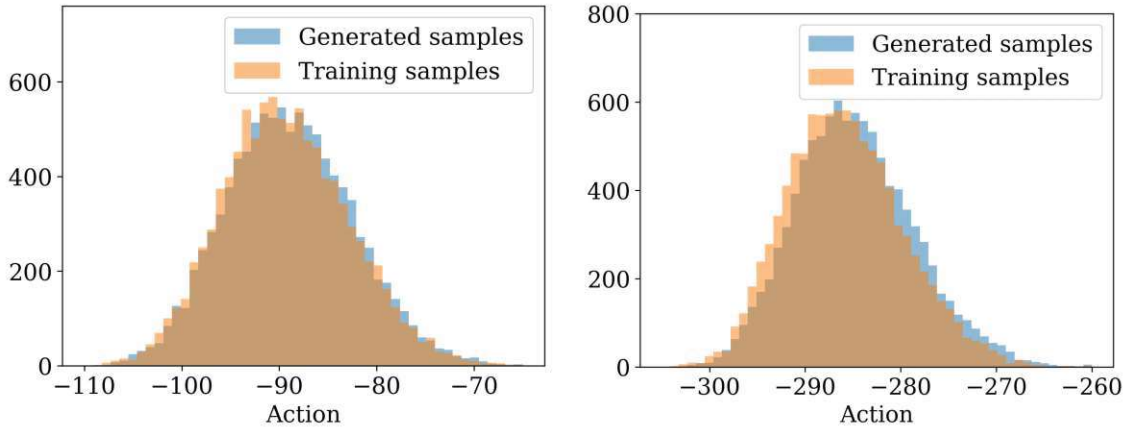
Again, we calculate the likelihood of our model to generate given samples and compare it with the action, see fig. 5.5. As already shown in fig. 5.2, we achieve an ESS

	$\langle S \rangle$	χ
Training samples	−89.32(9)	1.29(3)
Score model	−88.98(9)	1.30(3)

Table 5.1: Observables for $\beta = 2$, calculated over 10 000 configurations each.

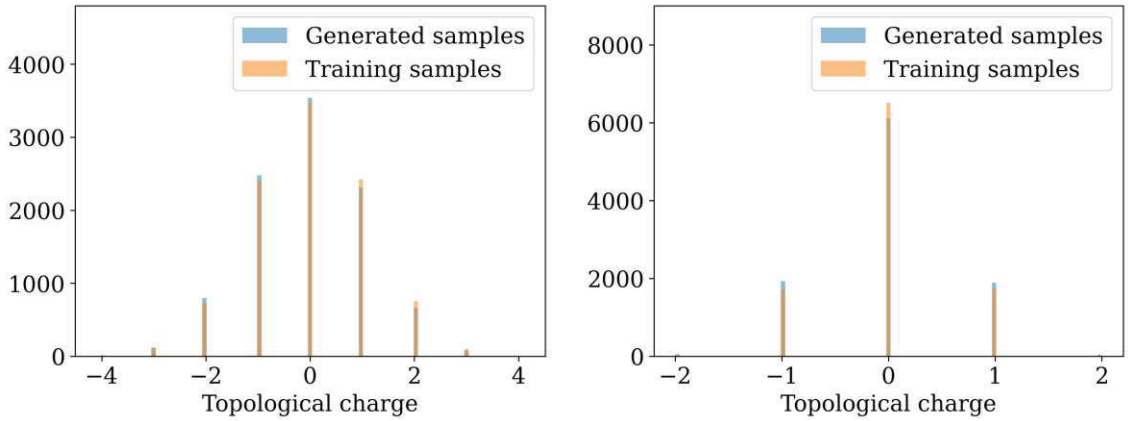
	$\langle S \rangle$	χ
Training samples	−285.81(9)	0.365(8)
Score model	−284.70(9)	0.419(8)

Table 5.2: Observables for $\beta = 5$, calculated over 10 000 configurations each.



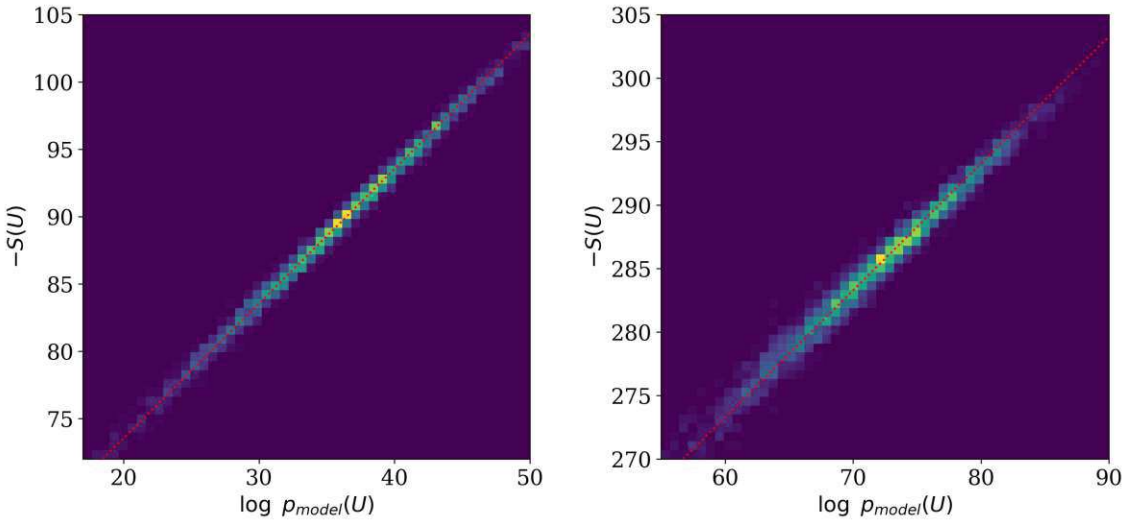
(a) $\beta = 2$ (b) $\beta = 5$

Figure 5.3: Comparison of the action distribution for 10 000 training samples and 10 000 samples generated with the trained score model.



(a) $\beta = 2$ (b) $\beta = 5$

Figure 5.4: Comparison of the topological charge distribution for 10 000 training samples and 10 000 samples generated with the trained score model.



(a) $\beta = 2$ (b) $\beta = 5$

Figure 5.5: 2D histogram with the model log-likelihood on the abscissa and the negative action on the ordinate, as in fig. 4.10.

of around 0.8 for $\beta = 2$ and 0.3 for $\beta = 5$.

To better compare the results to [3], we also calculate the rate of samples accepted by the Metropolis-Hastings algorithm. In this algorithm, after starting from one generated configuration U^0, we calculate the probability of accepting the following configuration U' by

$$p_{\text{accept}}(U'|U^{i-1}) = \min\left(1, \frac{q(U^{i-1})}{p(U^{i-1})}\frac{p(U')}{q(U')}\right), \tag{5.24}$$

where p is the target distribution and q the model distribution. We then draw a random number from a uniform distribution over $[0, 1)$. If the random number is smaller than p_{accept} we accept the new configuration and set $U^i = U'$, otherwise $U^i = U^{i-1}$. For $\beta = 2$ we reach an acceptance rate of around 75%, compared to 40% to 50% for the same problem in [3].

5.3.2 Increasing the grid size

A significant advantage of the convolutional network architecture we described in section 5.2.3 is that it is completely independent of the size of the grid it is applied to. Thus, we can use a network trained with training samples of size 8^2 and use it to generate samples on an arbitrarily sized lattice. We use the network trained with training samples of shape 8×8 for $\beta = 2$, as presented in the previous results, to generate samples with shape 16×16. Comparing these to samples of shape 16×16 generated using HMC, we find that the network is able to generate samples on larger lattices successfully, though there is a drop in quality. A comparison of the action

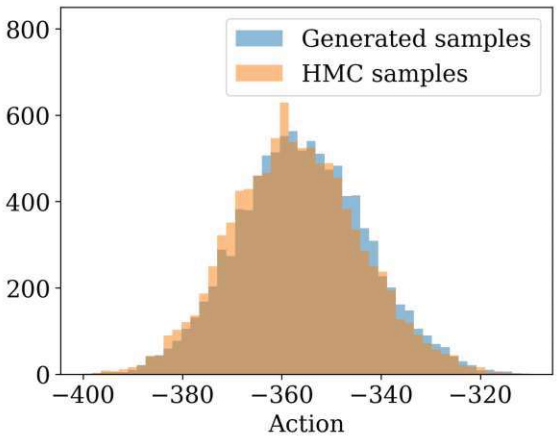


Figure 5.6: Comparison of the action of samples generated with HMC and with the trained score model on a 16×16 lattice.

	$\langle S \rangle$	χ
HMC samples	$-357.1(2)$	5.1(1)
Score model	$-355.6(2)$	5.2(1)

Table 5.3: Observables for $\beta = 2$ on a 16×16 lattice, calculated over 10 000 configurations each.

distribution can be seen in fig. 5.6 and of the observables in table 5.3. The ESS exhibits the most notable drop, going from 0.80 for the 8×8 grid to 0.46 for the 16×16 grid. We also repeated the procedure on a 25×25 lattice, here the ESS drops to 0.16.

6 Carosso flow

6.1 Introduction

The generative diffusion process corresponds to a continuous deformation of the action, which becomes apparent by looking at our trained action models (fig. 3.3 and fig. 4.11). At diffusion time $t = 0$ the action is given by the studied theory, and at $t = T$ the action corresponds to a normal distribution (i.e. an action with only a mass term). This procedure coincides with what is known as renormalization group (RG) flow in physics, see e.g. [25], where a time-dependent action is introduced, which continuously deforms a given action into an action that behaves like a mass term. In [7], the authors show a direct connection between generative models and a certain RG flow scheme, termed Carosso RG scheme [8], which is a renormalization group flow governed by a stochastic process.

6.2 Theory

Following the notation in [7], the Carosso RG flow is described by a stochastic differential equation, given as

$$\partial_t \phi_t(\mathbf{n}) = \left(\Delta - M^2\right) \phi_t(\mathbf{n}) + \eta_t(\mathbf{n}) \tag{6.1}$$

with initial condition $\phi_0(\mathbf{n})$ at $t = 0$. We operate on a lattice as described in chapter 4, but now in d dimensions, with N^d lattice points. The noise term η satisfies

$$\mathbb{E}[\eta_t(\mathbf{n})] = 0, \tag{6.2}$$

$$\mathbb{E}[\eta_t(\mathbf{n})\eta_s(\mathbf{m})] = \left(\frac{N}{L}\right)^d \delta(t-s)\delta_{\mathbf{n},\mathbf{m}}. \tag{6.3}$$

Here, L is the physical length of the system, such that L/N is the lattice spacing.

In momentum space, the drift term of the SDE takes a simpler form, namely without a derivative. Using the discrete Fourier transformation

$$\tilde{\phi}(\mathbf{p}) := \frac{1}{N^d} \sum_{\mathbf{n}\in\mathbb{Z}_N^d} e^{-i\frac{2\pi}{N}\mathbf{p}\cdot\mathbf{n}} \phi(\mathbf{n}) \tag{6.4}$$

we can write the SDE in momentum space as

$$\partial_t \tilde{\phi}_t(\mathbf{p}) = -\left(|\hat{\mathbf{p}}|^2 + M^2\right) \tilde{\phi}_t(\mathbf{p}) + \tilde{\eta}_t(\mathbf{p}) \tag{6.5}$$

with initial condition $\tilde{\phi}_0(p)$ at $t = 0$. The noise term $\tilde{\eta}$ satisfies

$$\mathbb{E}[\tilde{\eta}_t(\mathbf{p})] = 0, \tag{6.6}$$

$$\mathbb{E}[\tilde{\eta}_t(\mathbf{p})\tilde{\eta}_s(\mathbf{k})] = \left(\frac{1}{L}\right)^d \delta(t-s)\delta_{\mathbf{p},-\mathbf{k}} \tag{6.7}$$

and $\hat{\mathbf{p}}$ is given as

$$\hat{p}_i := \frac{2N}{L}\sin\left(\frac{\pi}{N}p_i\right), \quad i = 1, ..., d. \tag{6.8}$$

In momentum space, it is possible to write down the transition probability from $\tilde{\phi}_0$ to $\tilde{\phi}_t$,

$$p_{t,0}(\tilde{\phi}_t|\tilde{\phi}_0) \propto \prod_{\mathbf{p}\in\mathbb{Z}_{\mathbb{N}}} \exp\left(-\frac{L^d(|\hat{\mathbf{p}}|^2+M^2)}{1-e^{-2(|\hat{\mathbf{p}}|^2+M^2)t}}\left|\tilde{\phi}_t(\mathbf{p}) - \tilde{\phi}_0(\mathbf{p})e^{-(|\hat{\mathbf{p}}|^2+M^2)t}\right|^2\right). \tag{6.9}$$

Taking t to infinity, we get

$$p_{\infty,0}(\tilde{\phi}_\infty|\tilde{\phi}_0) \propto \prod_{\mathbf{p}\in\mathbb{Z}_{\mathbb{N}}} \exp\big(-L^d(|\hat{\mathbf{p}}|^2+M^2)|\tilde{\phi}_\infty(\mathbf{p})|^2\big), \tag{6.10}$$

where the transition probability is independent of the initial configuration. Thus the transition probability is also the probability distribution of $\tilde{\phi}_\infty$, from which we can now easily sample according to

$$\mathrm{Re}(\tilde{\phi}_\infty(\mathbf{p})), \mathrm{Im}(\tilde{\phi}_\infty(\mathbf{p})) \sim \mathcal{N}\left(0, \frac{1}{4L^d(|\hat{\mathbf{p}}|^2+M^2)}\right), \ \mathbf{p} \neq \mathbf{0} \text{ or } (N/2, ..., N/2), \tag{6.11}$$

$$\tilde{\phi}_\infty(\mathbf{0}), \tilde{\phi}_\infty((N/2, ..., N/2)) \sim \mathcal{N}\left(0, \frac{1}{2L^d(|\hat{\mathbf{p}}|^2+M^2)}\right). \tag{6.12}$$

Due to $\phi(\mathbf{n})$ being real-valued, it's Fourier transformation $\tilde{\phi}(\mathbf{p})$ has to fulfill $\tilde{\phi}(-\mathbf{p}) = \tilde{\phi}(\mathbf{p})^*$, which we also have to take into account when sampling.

6.2.1 Observables

Following [7], we will use the two-point correlator in momentum space,

$$\tilde{G}_2^{\text{Carosso}}(\mathbf{p}_1, \mathbf{p}_2) := \langle\tilde{\phi}(\mathbf{p}_1)\tilde{\phi}(\mathbf{p}_2)\rangle \tag{6.13}$$

to observe changes of the field configurations along the diffusion process. For small $\mathbf{p}$, the two-point correlator can be approximated as

$$\tilde{G}_2^{\text{Carosso}}(\mathbf{p}, -\mathbf{p}) \approx \frac{Z}{|\hat{\mathbf{p}}|^2 + r_t^2 m_R^2} \tag{6.14}$$

with the renormalized mass m_R and the wave function renormalization Z, which we will also measure. Following this approximation, one can also write down explicit formulas for Z and $r_t^2 m_R^2$ as

$$\frac{1}{r_t^2 m_R^2} \approx \frac{1}{4} \sum_{\mathbf{p}\in(1,0),(0,1),(-1,0),(0,-1)} \frac{1}{4\frac{N^2}{L^2}\sin^2(\pi/N)} \left(\frac{\tilde{G}_2^{\text{Carosso}}(\mathbf{0},\mathbf{0})}{\tilde{G}_2^{\text{Carosso}}(\mathbf{p},-\mathbf{p})} - 1 \right), \tag{6.15}$$

$$Z \approx r_t^2 m_R^2 \tilde{G}_2^{\text{Carosso}}(\mathbf{0},\mathbf{0}). \tag{6.16}$$

6.2.2 Analytical solution for the flow of the two-point correlator

Due to the simplicity of the SDE in momentum space, we are able to find an analytical solution for the two-point correlator along the Carosso flow. For this, we write down the solution of eq. (6.5) as if it was an ODE and the noise term an inhomogeneity:

$$\tilde{\phi}_t(\mathbf{p}) = e^{-(|\hat{\mathbf{p}}|^2+M^2)t}\tilde{\phi}_0(\mathbf{p}) + e^{-(|\hat{\mathbf{p}}|^2+M^2)t} \int_0^t e^{(|\hat{\mathbf{p}}|^2+M^2)t'} \tilde{\eta}_{t'}(\mathbf{p}) \mathrm{d}t'. \tag{6.17}$$

Given that, we also know the solution for the two-point correlator along the flow

$$\langle \tilde{\phi}_t(\mathbf{p})\tilde{\phi}_t(-\mathbf{p})\rangle = e^{-2(|\hat{\mathbf{p}}|^2+M^2)t}\tilde{\phi}_0(\mathbf{p})\tilde{\phi}_0(-\mathbf{p}) + \frac{1}{2L^d(|\hat{\mathbf{p}}|^2+M^2)}\left(1 - e^{-2(|\hat{\mathbf{p}}|^2+M^2)t}\right), \tag{6.18}$$

where we chose $\langle \tilde{\phi}_t(\mathbf{p})\tilde{\phi}_t(-\mathbf{p})\rangle$ due to its approximation eq. (6.14). We note, that here the expectation value only covers the noise term of the SDE $\tilde{\eta}$, though for the correlator it should also sum over all possible initial configurations $\tilde{\phi}_0$.

When looking at the solution for t going to infinity, we find

$$\langle \tilde{\phi}_\infty(\mathbf{p})\tilde{\phi}_\infty(-\mathbf{p})\rangle = \frac{1}{2L^d(|\hat{\mathbf{p}}|^2+M^2)}, \tag{6.19}$$

which is independent of the initial configuration (and thus also covers the previously mentioned expectation value over all initial configurations).

By comparison to eq. (6.14), we find the behaviour of Z and $r_t^2 m_R^2$ for $t \to \infty$ to be

$$Z = \frac{1}{2L^d}, \tag{6.20}$$

$$r_t^2 m_R^2 = M^2. \tag{6.21}$$

In our numerical simulations of the Carosso flow, shown in fig. 6.1, we can confirm that these values are approached for large flow times.

6.3 Adapting the score model

Even though the Carosso SDE eq. (6.1) is written in a different way, it is a realization of the general SDE used for score models, eq. (2.1), where $\eta_t \, \mathrm{d}t = g(t)\mathrm{d}w$ and thus $\mathbf{f}(\phi, t) = (\Delta - M^2)\phi_t(\mathbf{n})$ and $g(t) = (N/L)^d$. Thus, we can write down the SDE and the ODE of the diffusion process describing the reversed Carosso flow as

$$\mathrm{d}\phi = \left[(\Delta - M^2)\phi_t(\mathbf{n}) - \left(\frac{N}{L}\right)^{2d} \nabla_\phi \log p_t(\phi)\right] \mathrm{d}t + \left(\frac{N}{L}\right)^{d} \mathrm{d}\bar{\mathbf{w}}, \tag{6.22}$$

$$\mathrm{d}\phi = \left[(\Delta - M^2)\phi_t(\mathbf{n}) - \frac{1}{2}\left(\frac{N}{L}\right)^{2d} \nabla_\phi \log p_t(\phi)\right] \mathrm{d}t. \tag{6.23}$$

When using the exploding variance SDE, we have set the time of the diffusion process to $T = 1$, as that was sufficiently long for all information of the initial data to be destroyed. This is in general not the case for the Carosso SDE, in fact it depends on the choice of N/L and M. For our choice of parameters, $M = 1$ and $N/L = 1$, we found that $T = 4$ is a sufficiently long diffusion time.

We can use the same network architecture for the score model as in chapter 4. For training, recall the form of the loss function

$$\mathbb{E}_{t\in\mathcal{U}(0,T)}\mathbb{E}_{p_0(\phi_0)}\mathbb{E}_{p_{0t}(\phi_t|\phi_0)}\left[\lambda(t)\|\mathbf{s}_\theta(\phi_t, t) - \nabla_{\phi_t}\log p_{0t}(\phi_t|\phi_0)\|_2^2\right], \tag{6.24}$$

which includes $\nabla_{\phi_t}\log p_{0t}(\phi_t|\phi_0)$. From eq. (6.9), we can write

$$\begin{aligned}
&\frac{\partial}{\partial\phi_t(\mathbf{n}_i)}\log p_{0t}(\phi_t|\phi_0) \\
&= \frac{\partial}{\partial\phi_t(\mathbf{n}_i)}\sum_{\mathbf{p}\in\mathbb{Z}_\mathbb{N}} -\frac{L^d(|\hat{\mathbf{p}}|^2 + M^2)}{1 - e^{-2(|\hat{\mathbf{p}}|^2+M^2)t}}\left|\tilde{\phi}_t(\mathbf{p}) - \tilde{\phi}_0(\mathbf{p})e^{-(|\hat{\mathbf{p}}|^2+M^2)t}\right|^2 \\
&= \frac{\partial}{\partial\phi_t(\mathbf{n}_i)}\sum_{\mathbf{p}\in\mathbb{Z}_\mathbb{N}} -\frac{L^d(|\hat{\mathbf{p}}|^2 + M^2)}{1 - e^{-2(|\hat{\mathbf{p}}|^2+M^2)t}}\left(\tilde{\phi}_t(\mathbf{p}) - \tilde{\phi}_0(\mathbf{p})e^{-(|\hat{\mathbf{p}}|^2+M^2)t}\right) \\
&\cdot\left(\tilde{\phi}_t(\mathbf{p}) - \tilde{\phi}_0(\mathbf{p})e^{-(|\hat{\mathbf{p}}|^2+M^2)t}\right)^*.
\end{aligned} \tag{6.25}$$

Using

$$\frac{\partial}{\partial\phi(\mathbf{n}_i)}\tilde{\phi}(\mathbf{p}) = \frac{1}{N^d}e^{-i\frac{2\pi}{N}\mathbf{p}\cdot\mathbf{n}_i}, \tag{6.26}$$

we get

$$\begin{aligned}
&\frac{\partial}{\partial\phi_t(\mathbf{n}_i)}\log p_{0t}(\phi_t|\phi_0) \\
&= \sum_{\mathbf{p}\in\mathbb{Z}_\mathbb{N}} -\frac{L^d}{N^d}\frac{|\hat{\mathbf{p}}|^2 + M^2}{1 - e^{-2(|\hat{\mathbf{p}}|^2+M^2)t}}\left(e^{-i\frac{2\pi}{N}\mathbf{p}\cdot\mathbf{n}_i}\left(\tilde{\phi}_t^*(\mathbf{p}) - \tilde{\phi}_0^*(\mathbf{p})e^{-(|\hat{\mathbf{p}}|^2+M^2)t}\right)\right. \\
&\left. + e^{i\frac{2\pi}{N}\mathbf{p}\cdot\mathbf{n}_i}\left(\tilde{\phi}_t(\mathbf{p}) - \tilde{\phi}_0(\mathbf{p})e^{-(|\hat{\mathbf{p}}|^2+M^2)t}\right)\right),
\end{aligned} \tag{6.27}$$

and the gradient is given as

$$\nabla_{\phi_t} = \begin{pmatrix} \frac{\partial}{\partial \phi_t(\mathbf{n}_1)} \\ \dots \\ \frac{\partial}{\partial \phi_t(\mathbf{n}_{N^d})} \end{pmatrix}. \tag{6.28}$$

For the prefactor we choose

$$\lambda(t) = \frac{L^d}{N^d} \frac{|\hat{\mathbf{p}}|^2 + M^2}{1 - e^{-2(|\hat{\mathbf{p}}|^2 + M^2)t}} \tag{6.29}$$

By taking the Fourier transformations of ϕ_t and ϕ_0 and plugging them into eq. (6.27), we can compute the loss function and train the network just as we did in the previous chapters.

6.4 Results

We use the score model with Carosso SDE to generate configurations of a two-dimensional scalar field theory just as in chapter 4. We can employ the same network architecture and training samples generated analogously. The chosen parameters of the action are $\lambda = 0.02$ and $\kappa = 0.22$, and the size of the lattice is set to 20^2.

After training, we manage to generate configurations whose observables match with those of the training samples, see table 6.1 for details. While the results are similar to the exploding variance model, training takes much longer. A training epoch for the Carosso score model currently takes twenty times as long as one for the exploding variance model, but we have not yet fully optimized the code for the Carosso case. Currently, two discrete Fourier transformations are necessary for each training step, as we let the diffusion process run in position space and then need to transform to momentum space in the calculation of the loss function. In contrast to the exploding variance case, we also have not implemented direct sampling from p_{0t}, but solve the forward SDE for each training step. Changing these points should lead to a significant decrease of training time.

	$\langle M \rangle$	χ_2	U_L
Training samples	0.001(1)	3.24(6)	0.01(2)
Carosso score model	0.003(1)	3.19(7)	−0.04(3)

Table 6.1: Comparison of observables of the training samples and samples generated with the trained Carosso score model.

Besides generating correct configurations, we also want the flow of the reverse SDE given by the score model to be the exact reverse process of the Carosso flow. We verify this by calculating the two-point correlators eq. (6.13), Z and $r_t^2 m_R^2$ along both flows. That is, for the Carosso flow we take configurations generated by the

HMC and solve the Carosso SDE eq. (6.1) from diffusion time $t = 0$ to $t = 4$. For the reversed flow we sample configurations from eq. (6.11) and eq. (6.12) and then solve the reverse ODE eq. (6.23) from $t = 4$ to $t = 0$, using the trained score model. The comparison between both flows in fig. 6.1 shows very good correspondence between the two-point correlators. While the remaining small discrepancies lead to more significant differences for Z and $r_t^2 m_R^2$, we can still see that the generative process is a reversal of the Carosso flow.

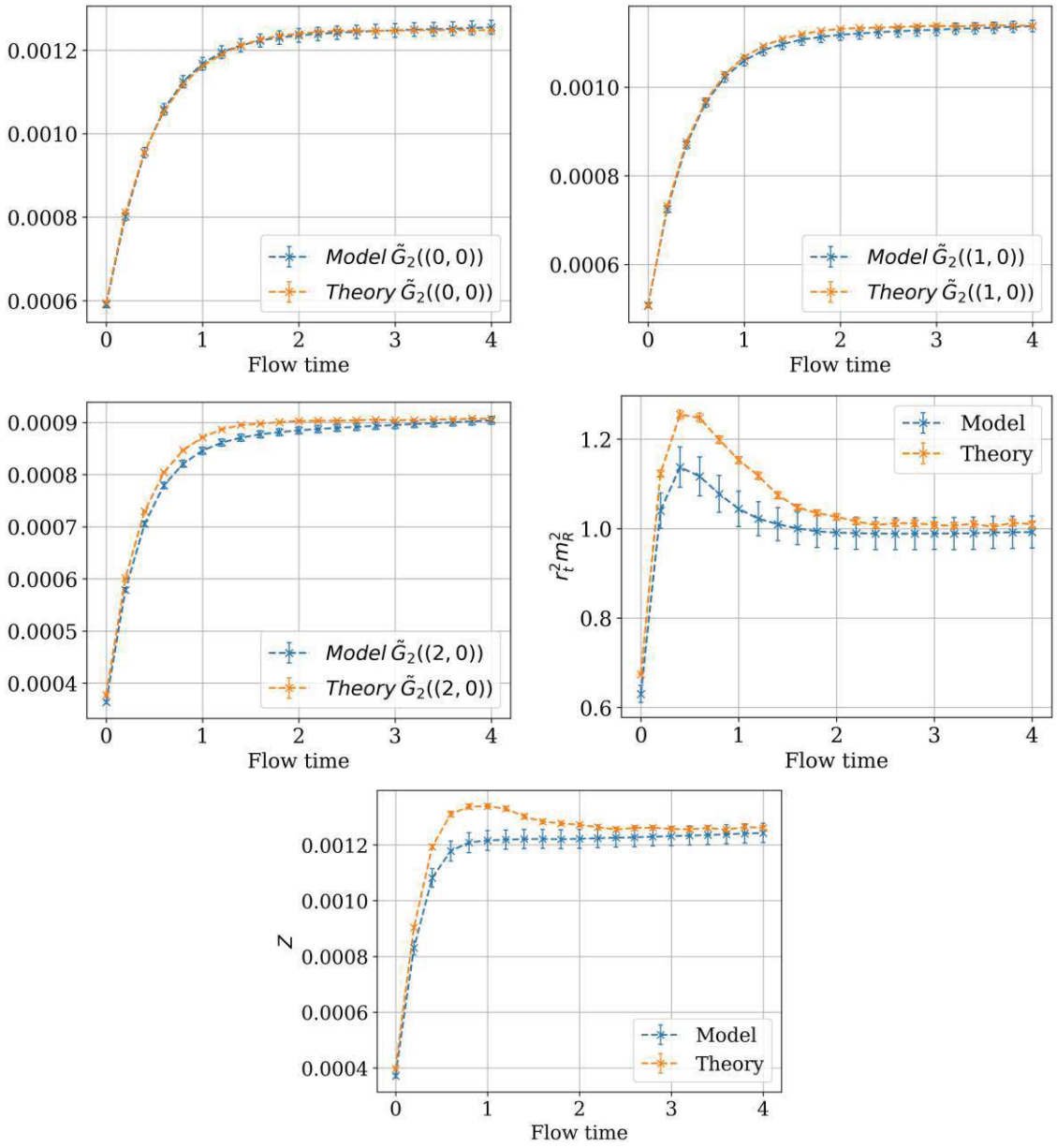


Figure 6.1: Comparison between the Carosso flow and the generative flow of the trained score model for $M = 1$ and $N = L = 20$.

7 Conclusion

In this work, we have shown that score-based generative models are capable of generating configurations of scalar ϕ^4 lattice field theory with a quality comparable to the hybrid Monte Carlo method, achieving an impressive effective sampling size of over 0.9999. We further demonstrated that it is possible to use score models to generate configurations of U(1) pure gauge theory. Although the results are only comparable to the HMC results for weak coupling, we manage to outperform the normalizing flow method as presented in [3] on the same problem, where we achieved a Metropolis acceptance rate of about 75% compared to a reported Metropolis acceptance rate of 40% to 50%. We suspect that results similar to the scalar case should be possible even in the strong coupling regime, by adapting the network architecture and increasing the training time. The biggest drawback of the presented method is the need for training samples generated through other methods, though by demonstrating the possibility to generate samples on bigger lattices compared to the training samples one may be able to explore lattice sizes which are too computationally expensive for other methods.

An obvious topic for future research is to apply score models to SU(2) and SU(3) gauge theories. Another possibility to explore is to use a parameter (e.g. κ in eq. (4.2) or β in eq. (5.3)) of the target action as an additional input parameter of the network. Seeing how the network then generalizes to parameter values not included in the training set, especially around a phase transition, might be interesting.

For now, calculating the model probability is computationally very costly. Finding improvements, possibly inspired by physical processes, would make using the Independence Metropolis algorithm on large sets of configurations generated by the diffusion models tractable.

Besides using the well-established score model, we also developed an adapted method, where the neural network learns the action instead of the score. We show that this action model performs just as good as the score model in the case of a scalar field. Additionally, with the trained action model we now have direct access to the action along the whole diffusion process. In future research this direct physical interpretability of the network may be used to improve network architectures, or the diffusion process itself. For example, one could base the network architecture on the general form of the action in the corresponding theory with trainable parameters as time-dependent prefactors.

Finally, we demonstrated the connection between the diffusion process in score-based generative models and renormalization group flows. We were able to design a generative diffusion process based on a known RG flow, namely the Carosso flow, and experimentally confirm that the flow learned by our network indeed matches

the Carosso flow. This direct connection may give rise to new diffusion model setups inspired by other RG flows and vice versa.

All in all, diffusion models seem to be a promising new method in lattice field theory and yield a number of interesting research topics in different directions.

Acknowledgements

The generative process of this thesis has been a challenging but also rewarding year-long journey, which would not have been possible without the continuous guidance of my supervisors. Therefore I would like to thank Andreas Ipp for letting me take on this project and providing so much valuable insight. The same appreciation goes to David Müller, whose ideas and knowledge are a main pillar of this thesis.

Furthermore I would like to thank Sabine Andergassen and Miriam Patricolo for providing the inspiration for and help with chapter 6 of this work.

Bibliography

[1] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2022. arXiv: 2112.10752 [cs.CV].

[2] Aditya Ramesh et al. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022. arXiv: 2204.06125 [cs.CV].

[3] Michael S. Albergo et al. *Introduction to Normalizing Flows for Lattice Field Theory*. 2021. arXiv: 2101.08176 [hep-lat].

[4] L. Wang, G. Aarts, and K. Zhou. "Diffusion models as stochastic quantization in lattice field theory". In: *Journal of High Energy Physics* 2024.5 (May 2024). ISSN: 1029-8479. DOI: 10.1007/jhep05(2024)060.

[5] Stefan Schaefer, Rainer Sommer, and Francesco Virotta. "Critical slowing down and error analysis in lattice QCD simulations". In: *Nuclear Physics B* 845.1 (Apr. 2011), pp. 93–119. ISSN: 0550-3213. DOI: 10.1016/j.nuclphysb.2010.11.020.

[6] Yang Song et al. *Score-Based Generative Modeling through Stochastic Differential Equations*. 2021. arXiv: 2011.13456 [cs.LG].

[7] Jordan Cotler and Semon Rezchikov. *Renormalizing Diffusion Models*. 2023. arXiv: 2308.12355 [hep-th].

[8] Andrea Carosso. "Stochastic renormalization group and gradient flow". In: *Journal of High Energy Physics* 2020.1 (Jan. 2020). ISSN: 1029-8479. DOI: 10.1007/jhep01(2020)172.

[9] Jascha Sohl-Dickstein et al. *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. 2015. arXiv: 1503.03585 [cs.LG].

[10] Yang Song and Stefano Ermon. *Generative Modeling by Estimating Gradients of the Data Distribution*. 2020. arXiv: 1907.05600 [cs.LG].

[11] C. W. Gardiner. *Handbook of stochastic methods for physics, chemistry and the natural sciences*. Third. Vol. 13. Springer Series in Synergetics. Springer-Verlag, 2004. ISBN: 3-540-20882-8.

[12] Brian D.O. Anderson. "Reverse-time diffusion equation models". In: *Stochastic Processes and their Applications* 12.3 (1982), pp. 313–326. ISSN: 0304-4149. DOI: https://doi.org/10.1016/0304-4149(82)90051-5.

[13] Pascal Vincent. "A Connection Between Score Matching and Denoising Autoencoders". In: *Neural Computation* 23.7 (July 2011), pp. 1661–1674. ISSN: 0899-7667. DOI: 10.1162/NECO_a_00142.

[14] John Skilling. "The Eigenvalues of Mega-dimensional Matrices". In: *Maximum Entropy and Bayesian Methods: Cambridge, England, 1988*. 1989, pp. 455–466. ISBN: 978-94-015-7860-8. DOI: 10.1007/978-94-015-7860-8_48.

[15] M.F. Hutchinson. "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines". In: *Communication in Statistics- Simulation and Computation* 18 (Jan. 1989), pp. 1059–1076. DOI: 10.1080/03610919008812866.

[16] G. Parisi and Yong-shi Wu. "Perturbation Theory Without Gauge Fixing". In: *Sci. China, A* 24.4 (1981), pp. 483–496. URL: http://cds.cern.ch/record/124815.

[17] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[18] Laurent Lellouch et al. *Modern Perspectives in Lattice QCD: Quantum Field Theory and High Performance Computing: Lecture Notes of the Les Houches Summer School: Volume 93, August 2009*. Oxford University Press, Aug. 2011. ISBN: 9780199691609. DOI: 10.1093/acprof:oso/9780199691609.001.0001.

[19] Simon Duane et al. "Hybrid Monte Carlo". In: *Physics Letters B* 195.2 (1987), pp. 216–222. ISSN: 0370-2693. DOI: https://doi.org/10.1016/0370-2693(87)91197-X.

[20] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[21] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[22] Kenneth G. Wilson. "Confinement of quarks". In: *Phys. Rev. D* 10 (8 Oct. 1974), pp. 2445–2459. DOI: 10.1103/PhysRevD.10.2445.

[23] Martin Lüscher. "Properties and uses of the Wilson flow in lattice QCD". In: *Journal of High Energy Physics* 2010.8 (Aug. 2010). ISSN: 1029-8479. DOI: 10.1007/jhep08(2010)071.

[24] Irais Bautista et al. "Measuring the topological susceptibility in a fixed sector". In: *Physical Review D* 92.11 (Dec. 2015). ISSN: 1550-2368. DOI: 10.1103/physrevd.92.114510.

[25] Adrian Koenigstein et al. "Numerical fluid dynamics for FRG flow equations: Zero-dimensional QFTs as numerical test cases. I. The O(N) model". In: *Physical Review D* 106.6 (Sept. 2022). ISSN: 2470-0029. DOI: 10.1103/physrevd.106.065012.