
Accelerated Simulation of the XY Model using Conditional Normalizing Flows

*A thesis submitted in fulfilment of the requirements
for the degree of Master of Technology*

by

Jaivardhan Kapoor

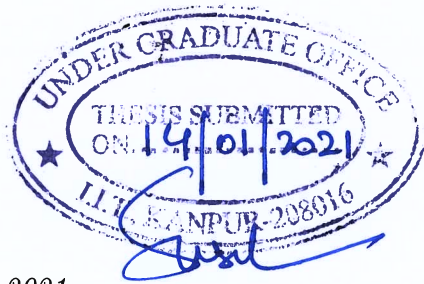


DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

January 2021

Certificate

It is certified that the work contained in this thesis entitled "Accelerated Simulation of the XY Model using Conditional Normalizing Flows" by "Jaivardhan Kapoor" has been carried out under my supervision and that it has not been submitted elsewhere for a degree.



January 2021

Vipul

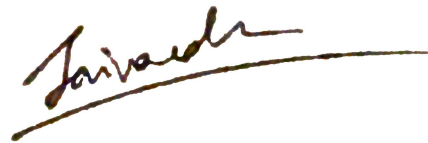
Vipul Arora

Professor

Department of Electrical Engineering
Indian Institute of Technology Kanpur

Declaration

This is to certify that the thesis titled "Accelerated Simulation of the XY Model using Conditional Normalizing Flows" has been authored by me. It presents the research conducted by me under the supervision of Professor Vipul Arora. To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for a degree. Further, due credit has been attributed to the relevant state-of-the-art and collaborations (if any) with appropriate citations and acknowledgements, in line with established norms and practices.

A handwritten signature in dark ink, appearing to read "Jaivardhan", is written over a horizontal line that extends across the width of the signature.

Signature

Name: Jaivardhan Kapoor

Programme: BT-MT

Department of Electrical Engineering

Indian Institute of Technology Kanpur

Kanpur – 208016

Abstract

Name of the student: **Jaivardhan Kapoor**

Roll No: **15807300**

Degree for which submitted: **M.Tech.**

Department: **Electrical Engineering**

Thesis title: **Accelerated Simulation of the XY Model using Conditional Normalizing Flows**

Thesis supervisor: **Vipul Arora**

Month and year of thesis submission: **January 2021**

Statistical Physics often requires computational simulations to study models. Simulating the XY model, a lattice model in Statistical Physics is performed by Markov Chain Monte Carlo. This method, although asymptotically exact, is resource and time-consuming. Recently, Normalizing flows, a class of probabilistic deep generative models in Machine Learning, have enjoyed success in simulating lattice models in both classical and quantum Physics.

In this thesis, we present a novel normalizing flow architecture that allows us to simulate lattices for the XY model. The proposed model can be conditioned on the temperature of the system, and hence can simulate lattices for a range of temperatures. The model is trained using Forward and Reverse KL objectives, therefore can be trained with or without pre-existing simulated data. Furthermore, the fully convolutional neural networks used in our model allow us to employ transfer learning to learn larger lattices by fine-tuning models for smaller lattices.

We quantitatively and qualitatively analyze the performance of the proposed model and compare it to recent works that also use deep generative models to simulate the XY model. We find that in computing physical observables, our model performs better than or on-par with recent works that also use deep generative models to simulate the XY model. We also

observe that due to the global continuous symmetry and topological phase transitions in the XY model, learning to simulate it is a hard problem. We show qualitatively that our model can transfer-learn larger lattice sizes and performs better than competing methods. Finally, we propose some possible future extensions to our method to better simulate such lattice models.

Acknowledgements

Throughout writing this thesis, I have received a great deal of help and support from my colleagues, friends, and family.

I would start by thanking my supervisor Dr. Vipul Arora for his constant support and immense patience. There have been times where I did not have confidence in myself, and Vipul Sir always led me to believe I could do it. I am grateful for the stimulating discussions with him, and his oft-accurate criticisms of my work.

I would also like to thank my fellow peers Vinay Kumar Verma, Japneet Singh, and Vikas Kanaujia for their help and collaboration in this project. Besides, I extend my thanks to Gurtej Kanwar, MIT for his helpful insights during our discussions.

My friends, who I consider my family in my stay at IIT Kanpur, have always been at my side, lifting my spirits and staying supportive in this endeavor. Thanks to Swapnil, Shivang, Krishan, Shivam, Dharendra, and Nishkarsh. Apologies to all those whose name I did not write but am equally grateful towards.



Finally, I am forever indebted to my family for supporting me always from behind the curtains. Thanks to my sister Ritika for providing encouraging words exactly when I needed them. My parents Anu and Rajesh Kapoor have always believed in me, and during the pandemic lockdown never let me fall into a slump. This is for you.

Contents

Acknowledgements	vi
List of Figures	ix
List of Tables	x
1 Introduction	1
2 Classical Spin Models: The XY Model	4
2.1 XY Lattice and its Hamiltonian	4
2.2 Markov Chain Monte Carlo - Simulating the XY model	5
2.3 Observables in the XY model	7
2.4 The Berezinskii–Kosterlitz–Thouless (BKT) Transition	8
2.5 Symmetries in the XY model	9
3 Normalizing Flows	11
3.1 Introduction	11
3.2 Transformation of Random Variables	11
3.2.1 Composing Multiple Transformations	12
3.3 Learning Normalizing Flows	14
3.3.1 Forward KL Divergence	14
3.3.2 Reverse KL Divergence	15
3.4 Normalizing Flows for Circular Variables	17
3.4.1 Circular Splines	18
3.5 Coupling-based Architecture for Multivariate Normalizing Flows	19
4 Related Work and Baselines	23
4.1 Using Tractable likelihood Models for Learning Physics	23
4.2 Machine Learning approaches applied to the XY model	24
4.3 Baselines	24
4.3.1 HG-VAE ([10])	24

4.3.2	ImplicitGAN ([47])	25
5	Proposed Model – Conditional Coupling-Circular Spline Flow	27
5.1	The Model	27
5.2	Training using Forward KL Divergence	29
5.2.1	Magnetization Normalization of the Training Data	30
5.3	Training using Reverse KL Divergence	31
6	Experiments	33
6.1	Metrics	33
6.1.1	Earth Mover Distance (EMD)	33
6.1.2	Percent Overlap ($\%OL$)	34
6.1.3	L_2 error	34
6.2	Experiment Setting	35
6.2.1	MCMC Simulation hyperparameters	35
6.2.2	Flow Settings	37
6.2.2.1	Training and Evaluation	37
6.3	Observations	37
6.3.1	Training the Flow with Forward KL	37
6.3.2	Training the Flow with Reverse KL	38
6.3.3	Comments on Performance	38
6.4	Transfer Learning for Larger Lattices – A Qualitative Analysis	41
7	Conclusion and Future Work	44
	Bibliography	46

List of Figures

2.1	Vortices and anti-vortices	8
2.2	Observables for $L = 8$ and $L = 16$. Shaded areas show ± 1 standard deviation from mean.	9
3.1	Testing CSF compositions on a 1D toy problem. Top left: the target density is a mixture of 4 von Mises distributions. Top right: bijections learned by the flow. Bottom left: Flow density optimized using Reverse KL, with an effective sample size (ESS) of 99.86%. Bottom right: Flow density optimized using Forward KL with an ESS of 99.61%. Flow is composed of 10 CSFs with $K = 5$ pieces each.	20
3.2	Jacobian structures for Autoregressive and Coupling Flow architectures. In both cases, simply multiplying the diagonal values of the Jacobian gives us the determinant in $\mathcal{O}(D)$ time.	21
3.3	Learning a toy problem $p^*(\mathbf{x}) \propto \exp\{4 \cos(x_1 - x_2)\}$. Left: real density. Right: flow learned density with an ESS of 98.4%. Flow consists of 8 transformations and uses a single hidden-layer Feedforward Neural Net with 16 hidden units. Training done using Reverse KL divergence.	22
5.1	Forward and inverse conditional flow compositions	28
5.2	Architecture of Conditional Coupling Circular Spline Flow with checkerboard mask  . A successive flow with the alternate checkerboard mask  is composed to form a single CC-CSF. For the inverse flow, simply use the inverse componentwise CSF f^{-1} instead of f	29
6.1	Observables for $L = 8$ and $L = 16$, computed using samples from a) MCMC, b) HG-VAE, c) ImplicitGAN, and d) CC-CSF trained using Forward KL. Susceptibility at low temperatures for HG-VAE was highly divergent, hence omitted.	39
6.2	Observables for $L = 8$ and $L = 16$, computed using samples from a) MCMC, b) HG-VAE, c) ImplicitGAN, and d) CC-CSF trained using Reverse KL. Susceptibility at low temperatures for HG-VAE was highly divergent, hence omitted.	40
6.3	Observables computed using samples from a) MCMC, b) CC-CSF transfer-learned from $L = 8$ flow, b) ImplicitGAN trained on 32×32 lattice data simulated using MH algorithm. Finetuning was done using Forward KL objective. For both models, dataset used contains 10k samples for each temperature.	43

List of Tables

6.1	Evaluated metrics for baselines and flow models. For reverse KL CC-CSF, $\alpha = 0.53$ for $L = 8$ and $\alpha = 0.55$ for $L = 16$. Mean and max values are computed over the range of temperature $T \in [0.025, 1.025]$, with 32 equally spaced intervals.	36
-----	--	----

Dedicated to my parents, Anu and Rajesh Kapoor

Chapter 1

Introduction

In recent times, Machine Learning (ML) has seen countless applications in various domains, owing to its ability to leverage pre-existing data and/or information about a problem to accelerate and improve predictions and inferences. The Sciences, in particular, have seen an explosion of applications of ML algorithms to augment and build upon traditional computational approaches. In this thesis, we will present one such application of contemporary ML techniques to accelerate a particular computational problem in Physics – simulating the XY model.

The rise in popularity of applied Machine Learning is in large part due to the advent of efficiently trainable Deep Neural Networks, which are central in the Deep Learning (DL) paradigm. In DL, we construct sequentially-applied nonlinear transforms that transform an input (images, waveforms, text, molecules, and much more) into the desired output, which may be simple (a binary output showing whether the image is of a particular class or not) or complex (a super-resolved version of the input image). Neural nets can be used as black box transforms in many other ML algorithms, increasing their expressivity and range of applications. This larger class of using Neural Nets with other ML algorithms falls broadly within the realm of differential programming, which utilized gradient descent-type algorithms to train these models.

ML algorithms, and their applications, can be broadly categorized into two different classes – supervised ML and unsupervised ML. The former requires labeled data, for example, the class of the image (whether it is of a cat, dog, car, and so on) with the image. The latter on the other hand only works on data points and does not require explicitly labeled data. Unsupervised ML is generally considered much more difficult, and a large portion of unsupervised ML algorithms utilize probabilistic constructions of a model. A

probabilistic style of model construction considers the input, output, as well as parameters and components of the model as random variables. Through probabilistic methods such as Bayesian inference, these random variables are then inferred and once the model is fully trained we can generate outputs from the model that is similar to the dataset we used. We can also use marginalization and various other properties of such probabilistic models to further analyze the problem.

A massive advantage of this probabilistic paradigm of unsupervised learning is that it blends well with applications to sciences. Analysis of scientific data requires uncertainty estimates along with point estimates to enable the practitioner to interpret the data. Since probabilistic approaches have an easier time providing uncertainty estimates of inferred properties and generated data, they are well-suited to problems in Physics, Chemistry, Biology, Geology, Medicine, and many more.

In Physics, many physical models require computational simulations to study their properties and investigate further. For example, in astrophysics, large scale simulations are run to map the evolution of galaxies and star systems. In the study of statistical physics and condensed matter Physics, microstates (individual molecules or units of matter) are simulated using various techniques to study the macroscopic properties of the medium. As an example, the Ising model [9] is a mathematical model containing up/down spins on a lattice (regularly spaced grid). It is used to study ferromagnetism, as the arrangement of the spins causes a net magnetization on different scales. The XY model [30] is a slightly more complex model with angular spins on a lattice. This model has certain additional properties and observables that are of interest to physicists. These lattice models are also found in Quantum Mechanics, where they are used to study Lattice Field Theory (LFT). In our work, we will solely focus on the XY model, which is a classical model of Statistical Physics.

Lattice models in more than a single dimension are usually intractable and cannot be studied analytically, therefore we must simulate them. Usually, these models have a Hamiltonian (or action in the case of LFT) that describes statistically the different possible microstates. Thus they have a tractable probability density associated with each configuration of spins in the lattice. We use statistical simulation methods such as Markov Chain Monte Carlo (MCMC) [2] to generate these configurations and then compute the observables, such as heat capacity, energy, magnetization, and so on.

MCMC is asymptotically correct but tends to be slow when the temperature of the system is low, or the lattice is large, or the model is complex. On the other hand, machine learning approaches allow us to use parallel processing hardware in the form of GPUs to quickly

generate samples. The challenge then lies in learning the physical model itself so that the generative ML model can reproduce the macroscopic observables. Several works have been published applying this concept to various lattice models [10, 32, 22, 16, 1, 24, 5, 21, 47].

In this work, we will exclusively look at accelerated simulation of the XY model. We will be employing Normalizing Flows [43, 40, 44] to learn and generate the XY model, and the model will be able to be conditioned on the temperature of the system. This model allows us to directly model the circular spins on the lattice through a sequence of transformations of random variables, which we can then use to generate new lattice configurations. Our contributions are as follows:

1. we introduce a novel Temperature-conditioned Coupling-based Circular Spline Flow model to learn and generate lattices for the XY model,
2. we introduce implicit and explicit regularization techniques to train the flow model,
3. we compare against state-of-the-art baselines and obtain competitive metrics, often beating state-of-the-art, and finally
4. we test the transfer learning capabilities of our model to learn and generate larger lattice sizes.

In Chapter 2 we will review the XY model, its properties, and how to simulate it. Chapter 3 provides an expository introduction to the theory of normalizing flows, their extension for modeling circular data, and multivariate architectures for efficient implementation. We then review some related work and baselines to compare our work against in Chapter 4. In Chapter 5 we construct our flow model and introduce techniques to train it efficiently. Finally, we evaluate the model's performance and features and compare with baselines in Chapter 6 and state our conclusions in Chapter 7.

Chapter 2

Classical Spin Models: The XY Model

2.1 XY Lattice and its Hamiltonian

In a 2D XY model [30], we have a two-dimensional lattice of $L \times L$ particles, with positions denoted by the tuple i, j . Each position in the lattice interacts with its nearest neighbors through the spin values. The spin is itself characterized by an angle, $x \in (-\pi, \pi]$. The Hamiltonian (or the configuration energy) of the lattice in terms of the spins $\mathbf{x} = \{x_{i,j}\}$ is then characterized as

$$H(\mathbf{x}) = -J \sum_{i,j} \sum_{\text{NN}_+(i,j)} \cos(\mathbf{x}_{i,j} - \mathbf{x}_{i',j'}), \quad (2.1)$$

where the outer sum is over all lattice positions and the inner sum over the nearest neighbors of each position. NN_+ contains the nearest neighbours of a lattice position in the positive direction, so that each interaction is only counted once.

Since the lattice is finite dimensional, periodic boundary conditions are established in the nearest neighbour interactions as is performed in mathematical simulations of physical models. The Hamiltonian may be written as follows:

$$H(\mathbf{x}) = -J \sum_{\substack{i \in \{1 \dots L\} \\ j \in \{1 \dots L\}}} \left(\cos(\mathbf{x}_{i,j} - \mathbf{x}_{i,j \oplus 1}) + \cos(\mathbf{x}_{i,j} - \mathbf{x}_{i \oplus 1,j}) \right), \quad \text{where} \quad (2.2)$$

$$a \oplus b = (a + b) \bmod L \quad (2.3)$$

The modulus operators include the periodic boundary conditions.

The lattice \mathbf{x} can take several configurations with the space of possible configurations in $[-\pi, \pi]^{L,L}$. Using a statistical mechanical perspective, the probability density of the lattice existing in a given state can be written as a Boltzmann distribution:

$$p(\mathbf{x}|T) = \frac{1}{Z(T)} \exp\left(-\frac{H(\mathbf{x})}{kT}\right) \quad (2.4)$$

where T is the temperature of the system, k is the Boltzmann constant, and $Z(T)$ the partition function given by normalizing the exponential term across all spin configurations. The coupling constant J in the Hamiltonian in our case is taken to be $0.5k$. Changing J is equivalent to scaling the temperature T correspondingly. It is this probabilistic formulation of the XY model that will allow us to simulate samples from it, and consequently, apply Machine Learning techniques to simulate it.

2.2 Markov Chain Monte Carlo - Simulating the XY model

We simulate the XY model by drawing samples from its Boltzmann distribution. Once we have samples, that correspond to high-probability configurations, we can then compute any observed quantity of the XY model by Monte Carlo (MC) averaging.

Suppose we want to accurately compute an observable O at temperature T . the observable will be different for different configurations, so usually we require an average over all configurations. Its expectation is computed as $\langle O \rangle = \int_{\mathbf{x}} O(\mathbf{x}) \frac{1}{Z(T)} \exp\left(-\frac{H(\mathbf{x})}{kT}\right) d\mathbf{x}$. Since this integral is not tractable, we resort to MC averaging. Assuming we have exact samples from the Boltzmann distribution, the MC estimate is computed as:

$$\langle O \rangle_{\text{MC}} = \frac{1}{N} \sum_{\substack{\mathbf{x}_1 \dots \mathbf{x}_N \\ \mathbf{x}_i \sim p(\mathbf{x})}} O(\mathbf{x}_i) \quad (2.5)$$

Due to samples being from the true distribution, the MC estimate is an unbiased estimate of the true value. In the case of the XY model, however, even sampling from the Boltzmann distribution is not possible, due to the unknown partition function $Z(T)$. We can only evaluate the *unnormalized* density given a configuration.

Markov Chain Monte Carlo (MCMC) allows us to precisely overcome this problem. MCMC [2] is a general class of algorithms that allow us to asymptotically sample from an intractable

distribution. In simple terms, MCMC works by creating a Markov chain of samples, so that the samples mimic the target probability distribution.

The *Metropolis Hastings* (MH) algorithm [8] is the most popular MCMC algorithm, and indeed several popular MCMC methods can be reinterpreted as special cases of the MH algorithm. We will be using the MH algorithm to generate samples of the XY model.

For an MCMC algorithm we require a proposal distribution (or more formally, the Markov Transition Kernel). The proposal should be easy to sample from and evaluate, and of a form such that we can condition the distribution given a value x' . Let's call the proposal $\tilde{p}(x|x')$ and the target distribution $p(x)$. The general recipe of the MH algorithm is:

1. Initialize the chain with the first sample $x^{(1)}$.
2. For $i = 2 \dots N$, generate the sample $x^{(i)}$ using the following steps.
 - (a) Given previous sample $x^{(i-1)}$, generate $\tilde{x}^{(i)} \sim \tilde{p}(x|x^{(i-1)})$.
 - (b) Compute acceptance probability

$$a = \min \left\{ 1, \frac{p(\tilde{x}^{(i)})\tilde{p}(x^{(i-1)}|\tilde{x}^{(i)})}{p(x^{(i-1)})\tilde{p}(\tilde{x}^{(i)}|x^{(i-1)})} \right\}$$

Note that the unnormalized target density suffices to compute this.

- (c) Set

$$x^{(i)} = \begin{cases} \tilde{x}^{(i)} & \text{w.p. } a \\ x^{(i-1)} & \text{w.p. } 1 - a \end{cases}$$

3. Return the chain $\{x^{(i)} \dots x^{(N)}\}$.

The XY model is simulated for a given temperature T by setting the target as the (unnormalized) Boltzmann distribution. We sample from the proposal by 1) sampling a lattice position randomly, that is, $u \sim \text{Uniform}(1 \dots L)$, $v \sim \text{Uniform}(1 \dots L)$, and then 2) perturbing the spin at position (u, v) by sampling from a normal distribution $\mathcal{N}(\mathbf{x}_{u,v}^{(i-1)}, \sigma^2)$ (out of boundary values are taken modulo 2π .) The unnormalized proposal density for acceptance probability computation is simply the Normal density. Because the proposal distribution is symmetric, the proposal terms in the numerator and denominator cancel out in the acceptance probability computation, leaving the target ratio.

Usually, we run the chain for a number of steps before collecting the samples. This is done to allow the chain to settle in a high-probability region, thereby getting rid of transient effects. This phase is called the warmup or burn-, also referred to as thermalization in physics literature. Finally, for the XY model there exist sampling algorithms such as the Wolff cluster algorithm [50], which perturb clusters of spins at once. Using our method, we can in principle model and learn any similar lattice model with angular spins, therefore our method is more universal in sampling lattices for custom densities.

2.3 Observables in the XY model

In this section, we list some of the observables of the XY model. These observables may be considered as measurable macroscopic properties of the model, where the microscopic quantities represent the individual spins in the lattice. We will usually compute the variance and mean of these observables to compare in our experimental section. Each observable is computed with the (unbiased) Monte Carlo estimate from the MCMC samples $\{\mathbf{x}^{(i)}\}_{i=1\dots N}$ of the XY model. The observables are computed given a specific temperature of the system.

Mean Energy (E) - The mean energy of the XY model at temperature T is simply the hamiltonian divided by the number of lattice points,

$$\langle E|T \rangle = \frac{1}{N} \sum_{i=1\dots N} \frac{1}{L^2} H(\mathbf{x}^{(i)}) \quad (2.6)$$

As the temperature increases, the energy of the system also increases, as the spins start to inhabit values farther away from neighboring sites, and drive the energy upward.

Mean Magnetization (M) - Magnetization is defined as

$$M(\mathbf{x}) = \frac{\sqrt{\sum_{u,v} \cos^2 \mathbf{x}_{u,v} + \sum_{u,v} \sin^2 \mathbf{x}_{u,v}}}{L} \quad (2.7)$$

The mean magnetization is the average of magnetization over all samples.

$$\langle M|T \rangle = \frac{1}{N} \sum_{i=1\dots N} M(\mathbf{x}^{(i)}) \quad (2.8)$$

Mean magnetization decreases with increase in temperature, as spins become uncorelated with each other. For larger lattices, the drop in magnetization becomes steeper near the



FIGURE 2.1: Vortices and anti-vortices

critical temperature. It is essentially the magnitude of the average direction of magnetization of spins of the lattice.

Mean Vorticity (V) - If we traverse a loop or clique of neighboring spins in clockwise direction, we will encounter vortices or antivortices. Visually show in Figure 2.1, if we go along the loop adding the changes in spin values modulo 2π , a total of less than -2π results in antivortices, while more than 2π is a vortex. If the sum remains within these values, there is no (anti)vortex present. Vortex-antivortex pairs may be encountered in neighboring loops, and we shall later see that unbinding of these pairs occurs at a critical temperature, leading to the Kosterlitz-Thouless transition.

Mean vorticity is the number of vortices in a lattice divided by the number of lattice sites. The MC estimate mean and variance are computed in simulations.

Magnetic Susceptibility (χ) - It is computed as the variance of the magnetization across lattice samples.

$$\chi = \text{Var}(M) \tag{2.9}$$

Magnetic susceptibility peaks at a temperature close to the transition temperature. For progressively large lattice which approximate the continuum XY model, the peak gets sharper.

All observables computed with the MH algorithm are shown in Figure 2.2.

2.4 The Berezinskii–Kosterlitz–Thouless (BKT) Transition

This section briefly describes a phenomenon special to the two-dimensional lattice of the XY model. The BKT transition [30, 42] is a phase transition as a result of unbinding of vortex-antivortex pairs at a critical temperature T_c . For coupling constant $J = 0.5k$,

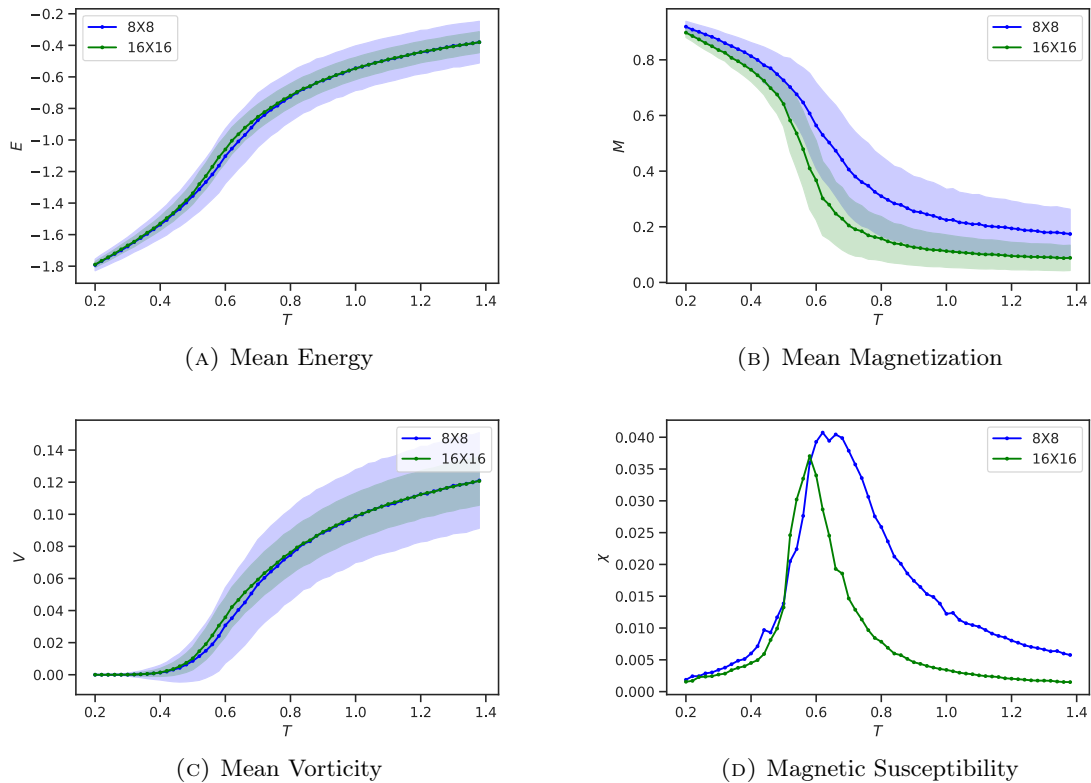


FIGURE 2.2: Observables for $L = 8$ and $L = 16$. Shaded areas show ± 1 standard deviation from mean.

$T_c \approx 0.45$. Near T_c , called the critical region, simulation of the XY model is harder than it is at lower or higher temperatures. This transition is first-order, resulting in divergence of magnetic susceptibility and other similar order parameters at the transition. The BKT transition is specific to the XY model, although some other statistical physics models may exhibit similar phase transitions. This transition makes it significantly harder to learn the XY model, compared to models that have a smoothly changing higher-order parameter with temperature. Later in the experimental section of this work, we will look at extrapolation analyses to see if our model correctly simulates the model in the critical region.

2.5 Symmetries in the XY model

The XY model has $U(1)$ symmetry, that is, if we rotate each spin in the lattice by a constant angle, the resulting configuration will be equivalent to the original one. To see this, recall that the XY model is characterized by its Hamiltonian. To a configuration $\mathbf{x} = \{\mathbf{x}_{i,j}\}$, we add a constant angle $x_0 \in [-\pi, \pi]$, such that the new configuration $\mathbf{x}' = \{\mathbf{x}_{i,j} \oplus x_0\}$, where

$a \oplus b = (a + b) \bmod 2\pi$. The Hamiltonian of the new configuration is

$$\begin{aligned}
 H(\mathbf{x}') &= -J \sum_{i,j} \sum_{\text{NN}_+(i,j)} \cos(\mathbf{x}'_{i,j} - \mathbf{x}'_{i',j'}) \\
 &= -J \sum_{i,j} \sum_{\text{NN}_+(i,j)} \cos(\mathbf{x}_{i,j} + \mathcal{P}\sigma - \mathbf{x}_{i',j'} - \mathcal{P}\sigma) \\
 &= H(\mathbf{x})
 \end{aligned} \tag{2.10}$$

Thus the Boltzmann distribution remains the same, as do the observables. Note that this is a global symmetry, as opposed to the more "local" gauge symmetries.

$U(1)$ symmetry of the XY model poses challenges in learning the model through ML techniques, as there is a manifold of equivalent optima (referred to in the physics literature as Goldstone modes). Later we shall see how to overcome this issue.

Another symmetry the XY model exhibits is discrete translational symmetry. Due to periodic boundary conditions, any translation of lattice positions in the x-dimension or y-dimension results in the same lattice. We take advantage of this symmetry through convolutional neural networks which are translation-invariant.

Chapter 3

Normalizing Flows

3.1 Introduction

In this chapter, we look at Normalizing Flows (NFs). NFs [40, 43] are a composition of differentiable invertible functions that map a random variable to another random variable. They are a very powerful tool for generative machine learning. Using the formula for the transformation of probability densities, we can compute the exact likelihood (probability density) of the original r.v. as well as the transformed r.v. In the next sections, we look at the building blocks of normalizing flows and also the transforming functions that allow us to efficiently learn and generate samples from the desired distribution.

3.2 Transformation of Random Variables

Consider a continuous random variable z . Let there be a bijective function f that maps z to another random variable x , $x = f(z)$. the the probability density for x is given by

$$p_x(x) = p_z(z) \cdot \left| \frac{1}{\frac{\partial f(z)}{\partial z}} \right|$$

Since $z = f^{-1}(x)$, this can be rewritten as

$$p_x(x) = p_z(f^{-1}(x)) \cdot \left| \frac{\partial f^{-1}(x)}{\partial x} \right|$$

For a multivariate random variable $\mathbf{z} \in \mathbb{R}^D$, with a bijective mapping $f : \mathbb{R}^D \rightarrow \mathbb{R}^D$ to $\mathbf{x} \in \mathbb{R}^D$, the multivariate version of this formula uses the absolute value of the determinant of the jacobian \mathbf{J}_f instead. The Jacobian of a vector-valued function of a multivariate input is the matrix of its partial derivatives.

$$\mathbf{J}_f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_n(\mathbf{x})}{\partial x_n} \end{bmatrix} \quad (3.1)$$

Using this Jacobian, we can write the probability transformation formula for a multivariate bijection as

$$p_{\mathbf{x}}(\mathbf{x}) = p_{\mathbf{z}}(\mathbf{z}) \cdot \left| \mathbf{J}_f^{-1}(\mathbf{z}) \right| = p_{\mathbf{z}}(\mathbf{z}) \cdot |\mathbf{J}_f(\mathbf{z})|^{-1} \quad (3.2)$$

where $|\cdot|$ denotes the absolute value of the determinant function. Similarly, using the identity $\mathbf{z} = f^{-1}(\mathbf{x})$ instead, we get

$$p_{\mathbf{x}}(\mathbf{x}) = p_{\mathbf{z}}(f^{-1}(\mathbf{x})) \cdot |\mathbf{J}_{f^{-1}}(\mathbf{x})| \quad (3.3)$$

The probability $p_{\mathbf{z}}(\cdot)$ is a simple parametric one that is easy to evaluate and sample from. Thus we start from a simple density and move toward a more complex and expressive one through transformations. By taking different forms of the invertible function f , we can apply normalizing flows to several unsupervised learning tasks. In the next section, we look at how to learn a generative model from data using normalizing flows.

3.2.1 Composing Multiple Transformations

In this section we look at compositions of transformations. Without loss of generality, we look at 2 transformations f_1, f_2 . Their composition is given by $f(\mathbf{x}) = f_2(f_1(\mathbf{x})) = f_2 \circ f_1(\mathbf{x})$. The Jacobian of the composition is given by $\mathbf{J}_f = \mathbf{J}_{f_2} \cdot \mathbf{J}_{f_1}$.

If the transformations are bijective, the inverse transformation can be written as $f^{-1}(\mathbf{x}) = f_1^{-1}(f_2^{-1}(\mathbf{x})) = f_1^{-1} \circ f_2^{-1}(\mathbf{x})$. Note that the order of composition has been reversed. The Jacobian of the composed inverse transformation is written in the same way as above.

We know that deep neural networks are powerful precisely because several transformations are composed on top of each other, resulting in powerful representations as data is passed through more and more layers or transformations. The same principle applies to normalizing

flows. By composing together multiple (relatively simple) transformations, we can create highly expressive transformations for random variables, resulting in a large learning capacity for such flows.

Starting with a random variable \mathbf{z} , consider a composition of flow transformations $f = f_n \circ f_{n-1} \circ \dots \circ f_2 \circ f_1$. The composition is computed sequentially starting with f_1 and ending with f_n . Let the intermediate outputs be denoted as

$$\begin{aligned}\mathbf{x}^{(0)} &= \mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z}) \\ \mathbf{x}^{(i)} &= f_i(\mathbf{x}^{(i-1)}) \quad \forall i \in \{1 \dots n\} \\ \mathbf{x} &= \mathbf{x}^{(n)}.\end{aligned}$$

The probability transformation formula is then written as

$$\begin{aligned}p_{\mathbf{x}}(\mathbf{x}^{(n)}) &= p_{\mathbf{z}}(\mathbf{z}) \cdot \prod_{i=1}^n \left| \mathbf{J}_{f_i}^{-1}(\mathbf{x}^{(i-1)}) \right| \\ &= p_{\mathbf{z}}(\mathbf{z}) \cdot \prod_{i=1}^n \left| \mathbf{J}_{f_i}(\mathbf{x}^{(i-1)}) \right|^{-1}\end{aligned}\tag{3.4}$$

Similarly, for an inverse transformation with the equivalent composition $f^{-1} = f_1^{-1} \circ f_2^{-1} \circ \dots \circ f_{n-1}^{-1} \circ f_n^{-1}$, the intermediate outputs are computed like so:

$$\begin{aligned}\mathbf{z}^{(0)} &= \mathbf{x} \sim p_{\mathbf{x}}(\mathbf{x}) \\ \mathbf{z}^{(i)} &= f_{(n+1-i)}^{-1}(\mathbf{z}^{(i-1)}) \quad \forall i \in \{1 \dots n\} \\ \mathbf{z} &= \mathbf{z}^{(n)},\end{aligned}$$

and the probability transformation formula is written as

$$p_{\mathbf{x}}(\mathbf{x}) = p_{\mathbf{z}}(\mathbf{z}^{(n)}) \cdot \prod_{i=1}^n \left| \mathbf{J}_{f_{n+1-i}^{-1}}(\mathbf{z}^{(i-1)}) \right|\tag{3.5}$$

The parameters of the flow composition are the set of parameters of the individual transformations, $\phi = \{\phi_i\}_{i=1}^n$.

3.3 Learning Normalizing Flows

We will look at 2 methods of learning normalizing flows. Both methods utilize the Kullback-Leibler (KL) divergence between two distributions.

3.3.1 Forward KL Divergence

The forward KL divergence is defined as

$$\mathbb{KL}[p(\mathbf{x})||q(\mathbf{x})] = \int_{\mathbf{x}} \log \frac{p(\mathbf{x})}{q(\mathbf{x})} p(\mathbf{x}) d\mathbf{x} = \mathbb{E}_{p(\mathbf{x})} \left[\log \frac{p(\mathbf{x})}{q(\mathbf{x})} \right] \geq 0 \quad (3.6)$$

If we are given some data and want to learn a flow that generates samples similar to the data, we can use the forward KL divergence to optimize the flow to do so. Assume the data follows a distribution $p^*(\mathbf{x})$, such that

$$\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^N \stackrel{i.i.d.}{\sim} p^*(\mathbf{x}), \quad (3.7)$$

where \mathcal{D} is the dataset.

For a composition of flow transformations f^{-1} , used as $\mathbf{z} = f^{-1}(\mathbf{x}; \phi)$, we also define the distribution $p_{f^{-1}}(\mathbf{x}) \stackrel{\text{def}}{=} p_{\mathbf{x}}(\mathbf{x})$ as the distribution of the flow outputs, which are related to the simple starting distribution $p_{f^{-1}}(\mathbf{z}) \stackrel{\text{def}}{=} p_{\mathbf{z}}(\mathbf{z})$ as in Equation 3.5. The flow distribution $p_{f^{-1}}(\mathbf{x})$ contains the variational parameters ϕ , which are the parameters of the flow bijection f^{-1} .

We try to optimize ϕ such that the forward KL divergence between $p^*(\mathbf{x})$ and $p_{\mathbf{x}}(\mathbf{x})$ is minimized. More formally, our objective is:

$$\begin{aligned} & \min_{\phi} \mathbb{KL}[p^*(\mathbf{x})||p_{f^{-1}}(\mathbf{x})] \\ & \iff \min_{\phi} \mathbb{E}_{p^*(\mathbf{x})} [-\log p_{f^{-1}}(\mathbf{x})] \end{aligned} \quad (3.8)$$

The second objective here follows from the fact that $p^*(\mathbf{x})$ is independent of ϕ .

The central issue in this minimization problem is that the data density is usually unknown. Even if we have an explicit unnormalized data density $\tilde{p}^*(\mathbf{x})$ such that $p^*(\mathbf{x}) = \frac{\tilde{p}^*(\mathbf{x})}{Z^*}$, only in trivial cases do we have an analytical form of the integral. Therefore, to alleviate this

issue, we use the Monte Carlo (MC) estimate of the forward KL [35]. This is written as

$$\mathbb{KL} [p^*(\mathbf{x})||p_{f^{-1}}(\mathbf{x})] \approx \frac{1}{N} \sum_{\mathbf{x}_i \in \mathcal{D}} -\log p_{f^{-1}}(\mathbf{x}) \quad (3.9)$$

This is equivalent to negative log-likelihood minimization of the flow. The MC estimator is asymptotically unbiased, as are its gradients. Furthermore, it is well suited for stochastic optimization settings, as a randomly sampled minibatch of the dataset also results in an unbiased estimator.

Because we have access to the observations \mathbf{x} and not the latents \mathbf{z} , we used the transformation formula in Equation 3.5 and not Equation 3.4. The algorithm then amounts to:

1. Choose a random minibatch of data $\mathcal{D}_i = \mathbf{x}_1 \dots \mathbf{x}_k$, and transform the data through the inverse flow transformation f^{-1} to get $\mathbf{z}_1 \dots \mathbf{z}_k$ and corresponding intermediate transformations.
2. Compute the MC estimator in Equation 3.9.
3. Backpropagate through the free variables ϕ and repeat for several iterations.

At the end of the training procedure, assuming the objective is sufficiently optimized, we can generate new data \mathbf{x} that match the target data distribution through the forward flow transformation $f = f_n \circ f_{n-1} \circ \dots \circ f_2 \circ f_1$.

3.3.2 Reverse KL Divergence

KL divergence is a non-symmetric divergence, that is, $\mathbb{KL} [p||q] \neq \mathbb{KL} [q||p]$. This means that changing the order of the densities results in different objectives that provide different results. Reverse KL divergence is simply reversing the density orders in the forward KL divergence. It is defined as

$$\mathbb{KL} [q(\mathbf{x})||p(\mathbf{x})] = \int_{\mathbf{x}} \log \frac{q(\mathbf{x})}{p(\mathbf{x})} q(\mathbf{x}) d\mathbf{x} = \mathbb{E}_{q(\mathbf{x})} \left[\log \frac{q(\mathbf{x})}{p(\mathbf{x})} \right] \geq 0 \quad (3.10)$$

Note that compared to Equation 3.6, the expectation is over q instead of p . In learning settings, this may result in drastically different results. Forward KL results in mode collapse, whereas Reverse KL results in mode covering behaviour.

Perhaps the most important requirement for using Reverse KL is that we must be able to compute $p^*(\mathbf{x})$, or rather the unnormalized density $\tilde{p}^*(\mathbf{x})$. This is because one of the terms inside the expectation is $\log p^*(\mathbf{x})$. An exact value of this is required for an unbiased MC estimator of the Reverse KL.

On the flip side, a massive advantage of this is that we do not need data to train a flow, the log density of the target suffices. Therefore this is ideal for settings when procuring data for the learning task is expensive but computing the target density is cheap.

Reverse KL divergence is used for learning normalizing flows that produce samples that follow a prescribed density. It is very well suited to applications in the sciences, where we want samples but only have the (unnormalized) log probability. For example, if we have the Hamiltonian of a system, we may be able to train flows that generate samples from the Boltzmann distribution of the system. This training method is also known in the ML literature as **probability density distillation** [38, 37].

The optimization objective when Reverse KL is used is

$$\begin{aligned} \min_{\phi} \mathbb{KL} [p_f(\mathbf{x}) || p^*(\mathbf{x})] & \quad (3.11) \\ \iff \min_{\phi} \mathbb{E}_{p_f(\mathbf{x})} [\log p_f(\mathbf{x}) - \log p^*(\mathbf{x})]. \end{aligned}$$

Note that for $\mathbf{x} = f(\mathbf{z})$, $\mathbb{E}_{p_{\mathbf{x}}} [g(\mathbf{x})] = \mathbb{E}_{p_{\mathbf{z}}} [g(f(\mathbf{z}))]$. Using this identity, we can rewrite the objective as

$$\min_{\phi} \mathbb{E}_{p_f(\mathbf{z})} [\log p_f(f(\mathbf{z})) - \log p^*(f(\mathbf{z}))]. \quad (3.12)$$

where the expectation is over the simpler distribution $p_{\mathbf{z}}(\mathbf{z})$ and the first term in the expectation is computed as in Equation 3.4. Here, both terms are a function of the free variables ϕ and the second term provides the information about the target density.

The MC estimator of the reverse KL is computed by sampling from the base distribution $p_{\mathbf{z}}(\mathbf{z})$, and is of the form

$$\mathbb{KL} [p_f(\mathbf{x}) || p^*(\mathbf{x})] \approx \frac{1}{N} \sum_{\substack{\mathbf{z}_i \sim p_{\mathbf{z}}(\mathbf{z}) \\ i=1 \dots N}} [\log p_f(f(\mathbf{z}_i)) - \log p^*(f(\mathbf{z}_i))] \quad (3.13)$$

In case we can only compute the unnormalized target density $\tilde{p}^*(\mathbf{x})$, the log normalizing constant $\log Z^*$ separates out of the estimator as a constant, making no difference to the

gradient of the MC estimator. The algorithm for training with the reverse KL divergence is as follows:

1. Sample $\mathbf{z}_i \sim p_{\mathbf{z}}(\mathbf{z})$ for $i = 1 \dots k$. Here $p_{\mathbf{z}}(\mathbf{z})$ is simple, for example a standard Normal or Uniform distribution.
2. Transform \mathbf{z} through the flow transformation f to get $\mathbf{x}_1 \dots \mathbf{x}_k$ and corresponding intermediate transformations.
3. Compute the log target density for $\mathbf{x}_1 \dots \mathbf{x}_k$.
4. Compute the MC estimator in Equation 3.13.
5. Backpropagate through the free variables ϕ and repeat for several iterations.

Once training is completed and the objective is converged, new samples are generated by following steps 1 and 2.

3.4 Normalizing Flows for Circular Variables

Rezende and Mohamed in [43] prescribe some standard bijections for real valued variables. A linear flow bijection is of the form $f(\mathbf{x}) = \mathbf{a} + h(\mathbf{B}\mathbf{x} + \mathbf{c})$, where $h(\cdot)$ is a component-wise application of a bijection such as a logistic function or tanh. We have similar forms for radial flows.

Normalizing flows in non-Euclidean domains [17, 48, 4, 44] are not as easy to construct. For compact spaces such as angular data, bijections need to be constrained to a compact space. Additionally, these bijections need to be composable, and flows should increase in their expressivity with the number of compositions they are made up of. In our case, lattice sites are angle-valued, meaning $\mathbf{x} \in \mathbb{T}^D$, where \mathbb{T}^D is a product space that factorizes as $\prod^D \mathbb{S}^1$ with \mathbb{S}^1 as a circle.

In this section we will review the method introduced in [44] that constructs flows on circular domains. The work shows that a bijection f acting on a circular variable $x \in \mathbb{S}^1$ must have

the following properties:

$$f(-\pi) = -\pi \quad (3.14)$$

$$f(\pi) = \pi \quad (3.15)$$

$$\nabla_x f(x) > 0 \quad (3.16)$$

$$\nabla_x f(x)|_{x=-\pi} = \nabla_x f(x)|_{x=\pi} \quad (3.17)$$

The fixed points $-\pi, \pi$ can be shifted by a translation operator (which is a bijection with identity Jacobian). To create such a bijection f satisfying the above 4 properties, the authors adopted the use of monotonic rational quadratic splines [20] as in Neural Spline Flows [15].

3.4.1 Circular Splines

Splines are functions that are defined piecewise, with a continuity constraint where the functions meet. Formally, a spline $s : [a, b] \rightarrow \mathbb{R}$ can be written as:

$$s(x) = \begin{cases} s_0(x) & x \in [a, w_0] \\ s_i(x) & x \in [w_{i-1}, w_i] \quad \forall i \in \{1 \dots k-1\} \\ s_k(x) & x \in [w_{k-1}, b], \end{cases} \quad (3.18)$$

with the additional constraint that $s_i(w_i) = s_{i+1}(w_i)$, and in our case, $\nabla_x s_i(x)|_{w_i} = \nabla_x s_{i+1}(x)|_{w_i}$.

Rational quadratic splines [20] use the rational quadratic form for each piecewise function,

$$s_i(x) = \frac{a_i^{(1)}x^2 + b_i^{(1)}x + c_i^{(1)}}{a_i^{(2)}x^2 + b_i^{(2)}x + c_i^{(2)}}$$

In our case, we have an additional constraint on the spline to be monotonically increasing. For a given interval in the spline domain $[w_{i-1}, w_i]$, output interval $[h_{i-1}, h_i]$, derivative values at the end $[d_{i-1}, d_i]$, and positive derivative everywhere, we can represent the parameters $\{a^{(1)}, b^{(1)}, c^{(1)}, a^{(2)}, b^{(2)}, c^{(2)}\}$ in terms of $\{w_{i-1}, w_i, h_{i-1}, h_i, d_{i-1}, d_i\}$. (For derivation refer [15], Section 3.1.)

To employ these splines as bijections for circular variables, we need to enforce the bijection conditions for circular variables. The first to third conditions can be easily enforced as

above. The fourth condition of equal gradient at $-\pi, \pi$ can be satisfied by simply enforcing the derivative of s_0 at $-\pi$ to be the same as that of s_k at π .

Thus a circular spline $s : [-\pi, \pi] \rightarrow [-\pi, \pi]$ is composed of K piecewise rational quadratic splines. The spline can be parameterized by three vectors: 1) W containing the widths of each of the pieces with the vector's sum equal to 2π , 2) H containing the corresponding heights with its sum also equal to 2π , and 3) D containing the (positive) derivatives at each knot (or intersection) between the pieces. Note that the number of knots, including the ends, is $K + 1$. However, the fourth bijection condition requires the derivatives the first and last knots to be the same. Increasing the number of pieces K results in a more expressive bijection. We shall henceforth refer to these bijections as **Circular Spline Flows (CSF)**.

For an input z , computing the output x of a CSF f is straightforward. First the piece index k is computed such that z lies in the k -th piece of the CSF. This can be done in $\log k$ time using binary search. Then the corresponding piecewise function is applied to give the output, along with the derivative at that point.

We can compose these splines together as in Section 3.2.1 to construct extremely expressive flows. Figure 3.1 shows the target density and model density for a learned composition of CSFs. The flow manages to learn perfectly the multimodal density.

3.5 Coupling-based Architecture for Multivariate Normalizing Flows

While constructing practical bijections in multivariate settings, we run into several problems. Most of these are concerning the computation of the determinant of the Jacobian as a part of the model density. Firstly, the Jacobian should be invertible such that the determinant is non-zero (hence the bijection). Secondly, for a dense Jacobian matrix of a function in D dimensions, the determinant computation scales as $\mathcal{O}(D^3)$. This is prohibitively expensive for high-dimensional data such as images, and in our cases, reasonably sized lattices.

To alleviate this, researchers have come up with clever tricks to construct flows so that they are simultaneously highly expressive while allowing for efficient density evaluations. Two major methods ubiquitous in literature are **autoregressive flows** [39, 28, 45] and **coupling flows** [12, 13, 14, 15]. Both these methods leverage designs that constrain the Jacobian to be *lower-triangular* and hence computable in $\mathcal{O}(D)$ time. (See Figure 3.2 for how Jacobians are structured in the two approaches.) In this work, we will look at coupling

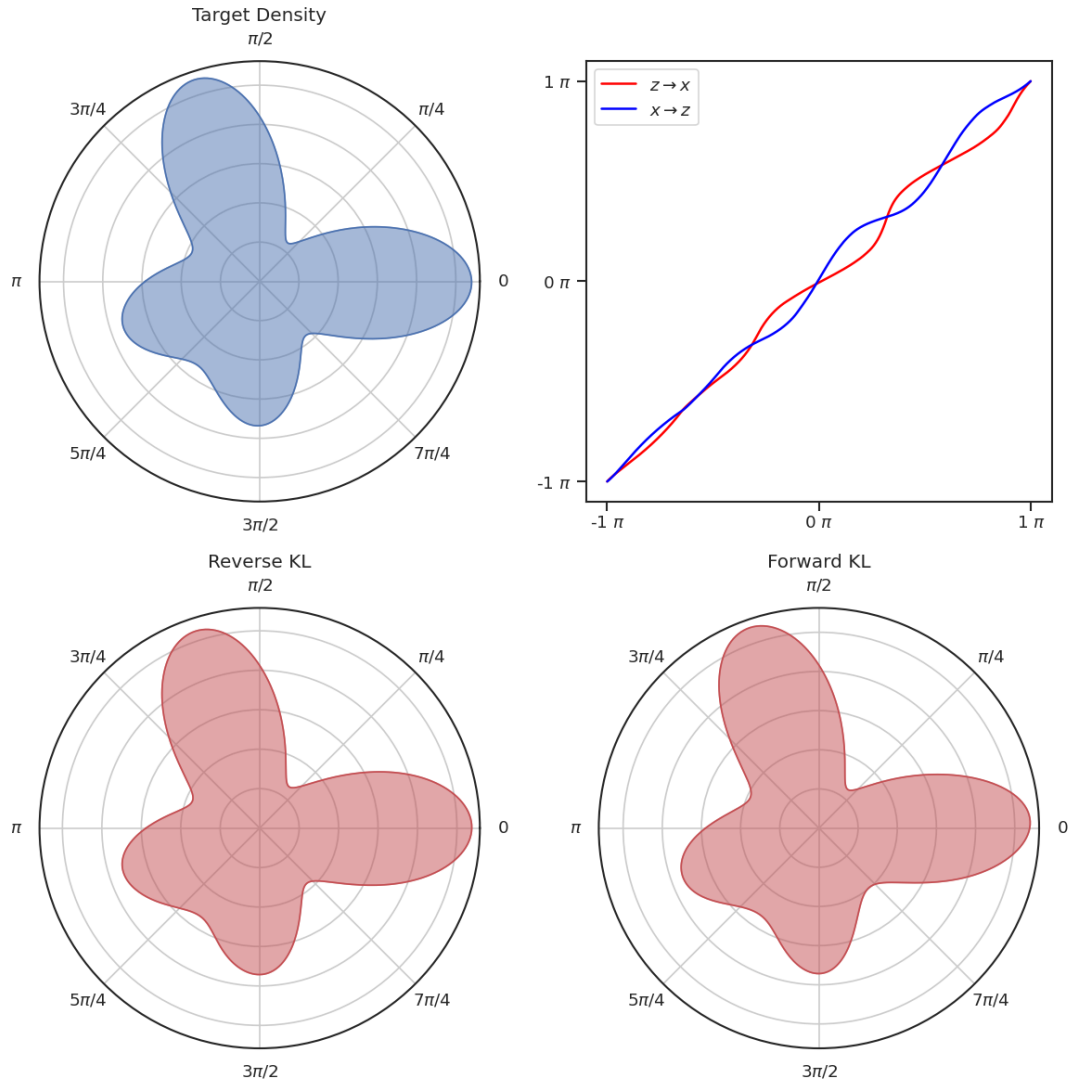


FIGURE 3.1: Testing CSF compositions on a 1D toy problem. Top left: the target density is a mixture of 4 von Mises distributions. Top right: bijections learned by the flow. Bottom left: Flow density optimized using Reverse KL, with an effective sample size (ESS) of 99.86%. Bottom right: Flow density optimized using Forward KL with an ESS of 99.61%. Flow is composed of 10 CSFs with $K = 5$ pieces each.

flows and employ them for learning the XY model. In the following we will look at coupling layers and how in [44] coupling based CSFs are created for multivariate angular data with the domain as \mathbb{T}^D .

Coupling flows split the random variable into two components $\mathbf{z} = [\mathbf{z}_{(1)}, \mathbf{z}_{(2)}]$. Given a 1-dimensional bijection $f(z; \theta)$ where θ are the parameters for the bijection, the flow outputs

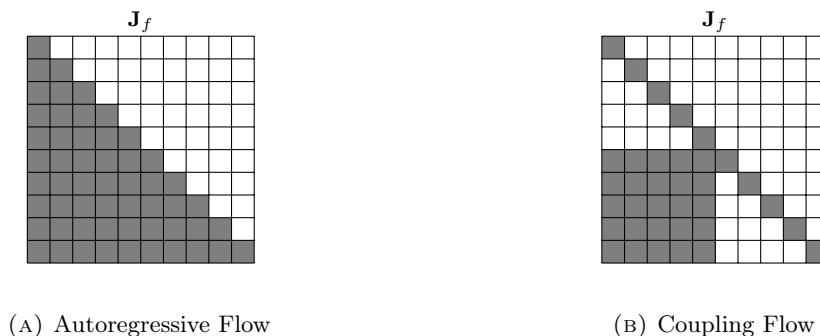


FIGURE 3.2: Jacobian structures for Autoregressive and Coupling Flow architectures. In both cases, simply multiplying the diagonal values of the Jacobian gives us the determinant in $\mathcal{O}(D)$ time.

two equal sized components as $\mathbf{x} = [\mathbf{x}_{(1)}, \mathbf{x}_{(2)}]$. The computation happens like so:

$$\mathbf{x}_{(1)} = \mathbf{z}_{(1)} \quad (3.19)$$

$$\theta = \text{NN}_\phi(\mathbf{z}_{(1)}) \quad (3.20)$$

$$\mathbf{x}_{(2)} = f(\mathbf{z}_{(2)}; \theta) \quad (3.21)$$

In our case, f is a CSF with $\theta = \{W, H, D\}$ as CSF parameters. NN_ϕ is a Neural Net with parameters ϕ that takes as input the first component of the input and outputs the parameters of the CSF. Then the outputted parameters are used to transform the second component (the transform is applied component-wise as it is a scalar bijection). Also, because we did not transform the first component, the inverse is very easy to compute.

$$\mathbf{z}_{(1)} = \mathbf{x}_{(1)} \quad (3.22)$$

$$\theta = \text{NN}_\phi(\mathbf{x}_{(1)}) \quad (3.23)$$

$$\mathbf{z}_{(2)} = f^{-1}(\mathbf{x}_{(2)}; \theta) \quad (3.24)$$

The inputs and outputs of the neural net remain the same in both cases, and all that changes is the usage of f^{-1} instead of f .

The power of coupling flows arises from the fact that arbitrarily powerful neural nets can be used for modeling dependency between the two components. Although the first component remains unchanged, we can compose additional coupling flows in which the first component is the one that is transformed now. Stacking these flows allows us to learn highly complex multidimensional distributions.

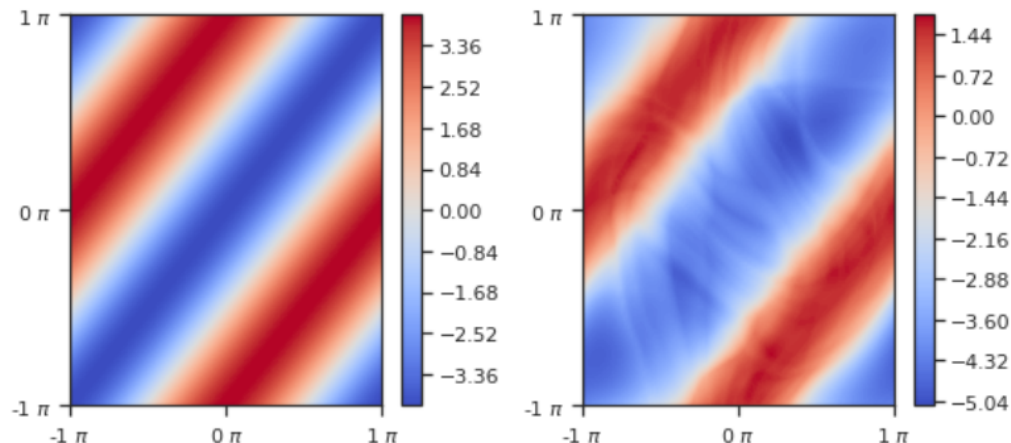


FIGURE 3.3: Learning a toy problem $p^*(\mathbf{x}) \propto \exp\{4 \cos(x_1 - x_2)\}$. Left: real density. Right: flow learned density with an ESS of 98.4%. Flow consists of 8 transformations and uses a single hidden-layer Feedforward Neural Net with 16 hidden units. Training done using Reverse KL divergence.

Here the components we create by splitting the vector x in half. In practice, the splitting can be arbitrary (but fixed for a specific flow transformation). We split the variable using a mask $m \in \{0, 1\}^D$. Multiplying with the mask turns the masked values to be zero which are then passed to the neural net. This gives us

$$\mathbf{x} = (1 - m) \cdot \mathbf{z} + m \cdot f(\mathbf{z}; NN_\phi((1 - m) \cdot \mathbf{z})) \quad (3.25)$$

The mask is then flipped for alternate coupling flow transformations which are composed. When the component-wise transformation f is a CSF, we call this flow transformation as **Coupling-Circular Spline Flow (C-CSF)**. A C-CSF-learned density for a 2D toy problem is shown in Figure 3.3.

The Jacobian of (flattened outputs of) C-CSF has the block structure as

$$\mathbf{J}_{\text{C-CSF}} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{A} & \mathbf{B} \end{bmatrix} \quad (3.26)$$

Visually the Jacobian has a structure shown in Figure 3.2b. \mathbf{B} is a diagonal matrix with diagonal elements as the derivatives of the unidimensional CSF for the d -th dimension of $\mathbf{z}_{(2)}$. As the Jacobian is lower-triangular, its absolute value of determinant is $|\mathbf{J}_{\text{C-CSF}}| = |\prod_i \mathbf{B}_{ii}|$.

In the next chapter, we create a *temperature-conditioned* C-CSF for learning to simulate the XY model given a temperature.

Chapter 4

Related Work and Baselines

Our work employs deep generative models for physics. This topic has many recent works that our work has built upon. We will look at some work that either uses machine learning on the XY model or learns adjacent physics models through various techniques that we have used. Most of this work uses either normalizing flows, variational autoencoders, or generative adversarial networks as architectures.

4.1 Using Tractable likelihood Models for Learning Physics

Several models with the lattice formulation, both classical and quantum, have had significant work on trying to simulate them using ML techniques. Normalizing flows for provably and asymptotically unbiased simulation of physical models were introduced in [37]. This work used normalizing flows with affine coupling layers to learn and generate states for n-body problems and molecular simulation. Our work also uses the same framework, with the addition of splines for learning angular spins and temperature-conditioning. [29] also introduce flows that respect translational and rotational symmetry in Euclidean space for n-body systems.

In particular, sampling lattices has benefitted greatly from using normalizing flows as the proposal distribution. Lattice Field Theory models, such as the scalar ϕ^4 lattice model [23], the complex-valued Schwinger model [49], and many other non-Abelian Lattice Gauge Theory models (with $SU(N)$ gauge symmetry) have been simulated by normalizing flow-based deep generative models.

[1] use RealNVP flows [13] to generate scalar ϕ^4 lattices. Their model, though, works on lattices with real-valued fields, and they test for physical parameters that do not exhibit spontaneous symmetry breaking. In [24], the authors employ Circular Splines with a tile-based masking pattern to create gauge-equivariant flows that respect the $U(1)$ gauge symmetry present in the Schinger model. This is perhaps the closest model to ours, with the exception that the XY model exhibits a global $U(1)$ symmetry and our flow can be conditioned on the temperature. Once again, the Schwinger model does not exhibit spontaneous symmetry breaking for the choice of parameters it was tested on in this work. A follow-up work [5] was published recently, creating equivariant flows for larger gauge symmetry groups such as $SU(N)$. This class of flow architectures was used for learning Gauge Lattice Theory models. On the other hand, [21] used affine flows [13] to generate the Ising model through the dual formulation of its Hamiltonian.

In addition to these tractable likelihood models, several works [10, 47, 41] have used implicit likelihood generative models such as Variational Autoencoders (VAEs) [26, 27] and Generative Adversarial Networks (GANs) [34].

4.2 Machine Learning approaches applied to the XY model

There also exists previous work on using supervised and unsupervised learning on the XY model. One of the earliest applications of deep learning to the XY model was published in [10], where the authors used Variational Autoencoders to generate microstates of the XY model, conditioned on the temperature. At the critical temperature, learning to identify vortex-antivortex pairs was also performed using supervised machine learning approaches in [3]. Finally, recent work in [47] employs ImplicitGANs [11] to generate XY model microstates. In the next section, we will briefly review the baselines [47, 10] we have chosen to compare with our model.

4.3 Baselines

4.3.1 HG-VAE ([10])

The HG-VAE, as named in the paper, is a Conditional Variational Autoencoder (C-VAE) that generates XY model lattices. The model consists of an encoder-decoder architecture, where the encoder neural net f takes as input MCMC configurations of the XY model and

outputs the parameters $f(x)$ of a latent variable (in this case, the Normal distribution.) Latent variables $z \sim \mathcal{N}(z|f(x))$ are then sampled from in the latent space and passed through a decoder neural net g to generate new lattice microstates \hat{x} .

The loss function is essentially the sum of the reconstruction error (between the input and output lattices), the KL divergence between the outputted parameters from the encoder and a standard Normal, and an additional loss term concerned with the computed energy of the lattice.

$$\begin{aligned} Loss_{\text{HG-VAE}} = & \|x - \hat{x}\|^2 + KL\left[\mathcal{N}(z|f(x))\|\mathcal{N}(0,1)\right] \\ & + \|E(x) - E(\hat{x}' \sim N(\hat{x}, \sigma^2))\|^2 \end{aligned} \quad (4.1)$$

The last 2 terms act as regularizers to cover the space of possible lattice configurations. The entire architecture is conditioned on a scalar temperature T to make the model temperature dependent.

4.3.2 ImplicitGAN ([47])

This approach uses a Generative Adversarial Net, with modifications to the loss functions, to generate XY model microstates x conditioned on a temperature T . It consists of two networks, a generator G and a discriminator D . Training is performed by generating a viable lattice sample through G by inputting noise z and conditioning on the temperature, $x = G(z; T)$. This generated lattice sample is then passed through the discriminator D conditioned on the same temperature T , which outputs the probability of the sample being real or fake, that is, if it is from the data distribution or not. This creates a minimax game between G and D , where G tries to generate realistic samples and D tries to distinguish them from actual training data. When properly trained, the generator produces samples of sufficient quality such that a properly trained discriminator may not be able to distinguish them from the training data. New samples are then generated from G whenever required.

GANs by themselves often suffer from mode collapse – they are not able to cover the entire high-mass region of the distribution they are attempting to learn. To counter this, [11] introduced a change in the loss function that approximately maximizes the entropy of the distribution of samples from $G(z; T)$. This means that the output distribution tends toward more diffuse states and hence mode-covering behavior increases. [47] add this trick into their architecture to increase the accuracy of the ImplicitGAN model. This requires a third auxiliary network A to match T and $\hat{T} = A(x)$, where x may either be real or from G .

Furthermore, training is regularized by minimizing the y component of mean magnetization of the output lattices, thus getting rid of the global rotational symmetry of lattice spins.

Additional details and derivatives for the training objective and hyperparameters may be found in the respective papers. All networks in these models use convolutional layers and fully-connected layers.

Chapter 5

Proposed Model – Conditional Coupling-Circular Spline Flow

Chapters 2 and 3 were expositions on the prerequisites required for our proposed methods. In this chapter, we propose a conditional flow architecture for learning the XY model. We also look at data preprocessing and training procedures for training with Forward KL as well as Reverse KL. We will justify the architectural and optimization choices that we make as inductive choices according to the XY model.

In the next section, we propose our model. Then we look at the data and training algorithm used for Forward KL optimization, and after that for Reverse KL.

5.1 The Model

In Sections 3.4 and 3.5 we explained how to create flow compositions for learning multivariate circular variables, resulting in the Coupling Circular Spline Flow (C-CSF) near the chapter's end. In this section, we will augment the C-CSF by conditioning the flows on the temperature of the system. We call the augmented model as **Conditional Coupling Circular Spline Flow (CC-CSF)**.

In the flow composition $F = F_n \circ \dots \circ F_1$, each C-CSF F_i is conditioned on the temperature. Thus we have $F(\mathbf{z}|T) = F_n(F_{n-1}(\dots(F_1(\mathbf{z}|T)\dots|T)|T))$. The inverse of this is simply the temperature conditioned inverse composition. The composition of CC-CSF is shown in Figure 5.1. Figure 5.1a shows the composition for forward generation conditioned on temperature T , while Figure 5.1b shows the inverse generation for the same.

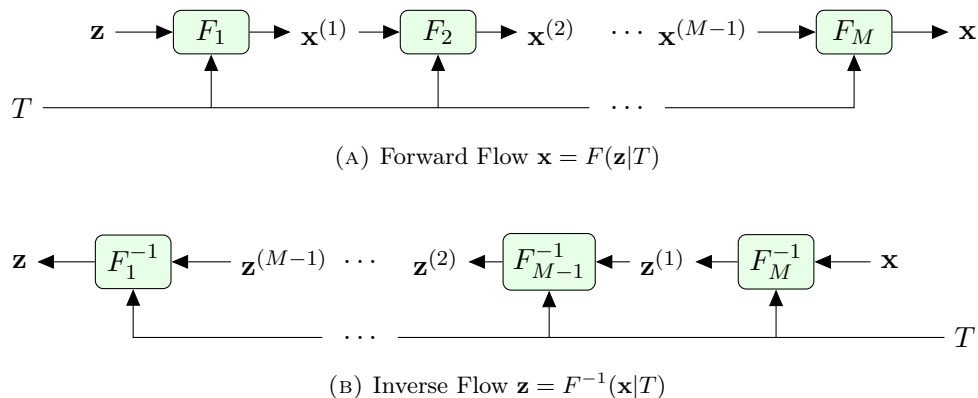


FIGURE 5.1: Forward and inverse conditional flow compositions

We use coupling layers with the same form as Equation 3.25. For the mask m we use a checkerboard pattern $\begin{bmatrix} \blacksquare & \square \\ \square & \blacksquare \end{bmatrix}$ that masks alternating locations in the 2D lattice. This pattern is used as lattice spins are likely to be correlated with their nearest neighbors and then the higher-order neighbors. Checkerboard masking enables the flows to learn the dependency between these components. Furthermore, we compose two flows with alternate checkerboard masking ($\begin{bmatrix} \blacksquare & \square \\ \square & \blacksquare \end{bmatrix}, \begin{bmatrix} \square & \blacksquare \\ \blacksquare & \square \end{bmatrix}$) to create a single CC-CSF flow layer that we see in Figure 5.1.

The neural network used inside the flow is a 2-layer fully-Convolutional Neural Net(CNN), with periodic padding before each convolutional operator to enforce periodic boundary conditions of the XY lattice. This fully convolutional architecture fits very well with the checkerboard masking pattern. Additionally, along with periodic padding before each convolution operator this architecture respects the discrete translational symmetry of the XY model as mentioned in Section 2.5. As an input to the net, we concatenate the cosine and sine components of the lattice (while setting the masked components of the lattice to 0) with a 2D lattice filled up with the scalar temperature T , making a 3-channel input. The output consists of $3K$ channels for each lattice site, resulting in a tensor of size $(3K, L, L)$, where L is the lattice size. Therefore for each lattice size, we obtain a $3K$ length vector containing the CSF parameters. The CSF parameters are applied to each lattice point, but only the unmasked positions are set to their transformed value. The masked positions are kept the same as the input. A diagram of the CC-CSF layer is shown in Figure 5.2.

Finally, the starting distribution $p_z(\mathbf{z})$ of the complete flow composition is set to be a Uniform distribution with support as $[-\pi, \pi]^{L \times L}$. Therefore the unnormalized probability density of the model is simply the sum of the log of absolute-determinants of Jacobians of all flow components.

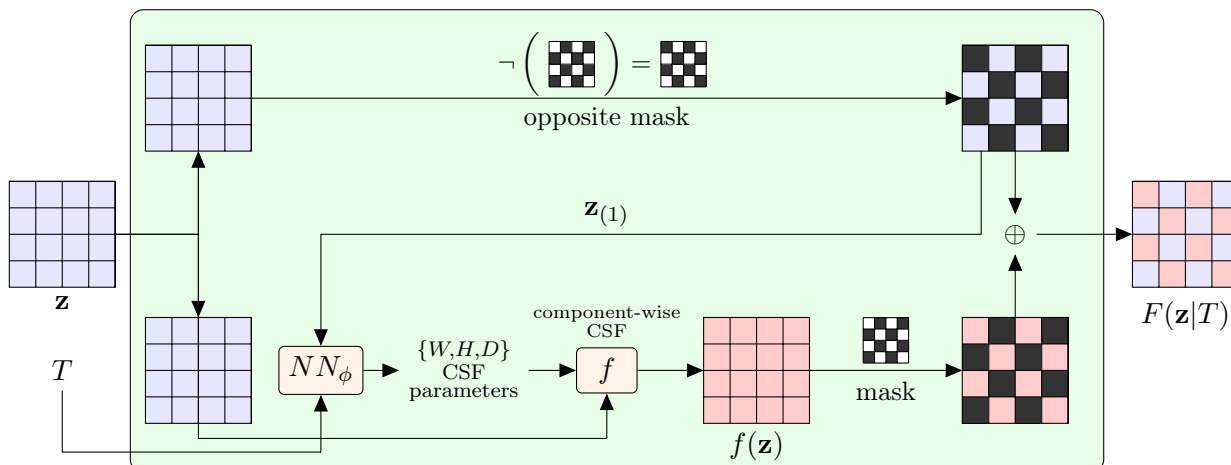


FIGURE 5.2: Architecture of Conditional Coupling Circular Spline Flow with checkerboard mask $\begin{bmatrix} \blacksquare & \square \\ \square & \blacksquare \end{bmatrix}$. A successive flow with the alternate checkerboard mask $\begin{bmatrix} \square & \blacksquare \\ \blacksquare & \square \end{bmatrix}$ is composed to form a single CC-CSF. For the inverse flow, simply use the inverse componentwise CSF f^{-1} instead of f .

Because we have access to the unnormalized true density of the XY model, we can train this flow using 2 methods: Forward KL and Reverse KL. The discrete translational symmetry of the XY model is incorporated into the flow using convolutional layers and periodic padding. The other symmetry, $U(1)$ global rotational symmetry (Section 2.5), is a much more difficult one to handle. As we shall see in the experimental section too, this symmetry results in difficulties training the models using both Forward and Reverse KL. There has been recent work on constructing normalizing flows that are equivariant to certain symmetries. However, to the best of our knowledge, there hasn't been any work on equivariant flows for $U(1)$ global rotational symmetry. Therefore, we attempt to employ augmentations in data and training objectives to go around this problem. In the next section, we will see the two different training methods for CC-CSF.

5.2 Training using Forward KL Divergence

The Forward KL divergence objective is the same as in Equation 3.8, except an outer expectation over T is applied to the temperature-conditioned Forward KL. Assume that the temperature is samples using probability $p_T(T)$. The Forward KL objective in this case is

$$\min_{\phi} \mathbb{E}_{p_T} [\mathbb{KL} [p^*(\mathbf{x}|T) || p_{F^{-1}}(\mathbf{x}|T)]] \quad (5.1)$$

where ϕ are the free variables which now consist of the neural network parameters for the constituent flows.

Using the MC approximation of the KL gives us

$$\min_{\phi} \frac{1}{N} \sum_{\substack{\{\mathbf{x}_i, T_i\} \\ \mathbf{x}_i \sim p^*(\mathbf{x}|T_i)}} -\log p_{F^{-1}}(\mathbf{x}_i|T_i) \quad (5.2)$$

The gradient of this objective is independent of the terms that are constant with respect to ϕ , therefore we can simplify the objective by only writing the sum of log-absolute determinants of the Jacobians of each flow $F_1^{-1}(\cdot|T) \dots F_M^{-1}(\cdot|T)$. (The starting distribution $p_{\mathbf{z}}(\mathbf{z})$ is a Uniform, therefore the gradient of the logarithm is zero.)

The data $\{\mathbf{x}_i, T_i\}$ that is used to optimize the above objective is generated through the MH algorithm described in Section 2.2.

5.2.1 Magnetization Normalization of the Training Data

Recall that according to Equation 2.10, the Hamiltonian of the XY model is invariant to addition of a constant angle to every spin in the lattice. Thus the Boltzmann distribution is constant along a manifold in the $(L \times L)$ dimensional space. In other words, the Boltzmann distribution of the XY model can be factorized into two components:

$$p^*(\mathbf{x}) = p_{\parallel}^*(\mathbf{x})p_{\perp}^*(\mathbf{x}), \quad (5.3)$$

such that the magnetization of the entire lattice $((\sum_{u,v} \cos \mathbf{x}_{u,v}, \sum_{u,v} \sin \mathbf{x}_{u,v}))$ is in a fixed direction for samples from p_{\parallel}^* , and the second component p_{\perp}^* adds the randomness back to the samples' directions of magnetization through a uniformly-sampled angle.

For a lattice \mathbf{x} , we can subtract a x_0 from every spin in \mathbf{x} such that the resulting lattice $\mathbf{x} - x_0$ has a horizontally-aligned magnetization, i.e.,

$$\sum_{u,v} \sin(\mathbf{x}_{u,v} - x_0) = 0. \quad (5.4)$$

We can apply this to every data point in our dataset for all temperatures, and thus get rid of an unwanted degree of freedom. The resultant dataset is still distributed according to the Boltzmann distribution, but lies on an $(L^2 - 1)$ -dimensional manifold over the entire

domain $[-\pi, \pi]^{L \times L}$. As we shall later see, this results in a substantially smaller search space of possible parameters for the flow and results in better generative models.

To implement this, we find the angle x_0 according to Equation 5.4 and subtract it from the lattice spins for every lattice in the dataset. We call this preprocessing step **Magnetization Normalization (magnorm)**.

5.3 Training using Reverse KL Divergence

Similar to the previous section, we change the Reverse KL objective in Equation 3.12 by adding an outer expectation over temperatures, and changing the densities inside the reverse KL by conditioning them on the temperature.

$$\min_{\phi} \mathbb{E}_{p_T} [\mathbb{E}_{p_{\mathbf{z}}(\mathbf{z})} [\log p_F(F(\mathbf{z}|T)) - \log p^*(F(\mathbf{z}|T))]] \quad (5.5)$$

Since, $\log p^*(\mathbf{x}) = -H(\mathbf{x}|T) - \log Z_T$, we can write the MC estimator of the shifted Reverse KL in similar vein to Equation 3.13, as:

$$\min_{\phi} \frac{1}{N} \sum_{\substack{T_i \sim p_T(T) \\ \mathbf{z}_i \sim p_{\mathbf{z}}(\mathbf{z})}} [\log p_F(F(\mathbf{z}_i|T_i)) + H(F(\mathbf{z}_i|T_i)|T_i)] \quad (5.6)$$

where ϕ are the parameters of the neural nets in the flow composition F .

To reduce the parameter space of the flow in this case, we augment the Reverse KL loss using a *regularization term* $\mathcal{R}(\mathbf{x}) = \lambda \|\mathbf{x}\|^2$. Furthermore, we noticed that tempering the target distribution $-H(F(\mathbf{z}_i|T_i)|T_i)$ by multiplying it by a factor $(1 - \alpha)$ allowed us to learn the flow and observables much better than the original Reverse KL objective. Combining these two regularizations together, we denote the *augmented* Reverse KL objective :

$$\min_{\phi} \frac{1}{N} \sum_{\substack{T_i \sim p_T(T) \\ \mathbf{z}_i \sim p_{\mathbf{z}}(\mathbf{z})}} [\log p_F(F(\mathbf{z}_i|T_i)) + (1 - \alpha) \cdot H(F(\mathbf{z}_i|T_i)|T_i) + \mathcal{R}(F(\mathbf{z}_i|T_i))] \quad (5.7)$$

with $\alpha \in [0, 1]$. The regularizer \mathcal{R} forces the spins to be aligned toward 0, and hence breaks the symmetry in the objective due to $U(1)$ global rotational symmetry. The training process works by sampling temperature $T \sim p_T(T)$ and starting variable $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$, passing through the forward flow, computing the Hamiltonian of the resulting lattice, and backpropagating through the above augmented loss.

In the next chapter, we will evaluate the proposed model and training objectives qualitatively and quantitatively, and compare them to relevant baselines.

Chapter 6

Experiments

In this chapter, we will evaluate our models against specified baselines in Chapter 4. We will also compute metrics to compare the accuracy of observables with respect to the "ground truth", which is the observables from the MCMC simulation in Chapter 2. We will also exploit the fully convolutional structure of our flow to perform transfer learning on larger lattice sizes, and compare the transfer-learned model with baselines.

6.1 Metrics

We shall evaluate our models' performance on the observables mentioned in 2.3. Since we can generate individual lattice configurations from the flow, we can compute these observables from them, and compute various metrics to determine if the observables match the ground truth MCMC observables.

Each of the metrics needs to account for the mean as well as the variance of the computed observables for each temperature. Since we evaluate over a range of different temperatures, we will use an averaged value and maximum value for the entire range of temperatures. Below we define the metrics used.

6.1.1 Earth Mover Distance (EMD)

EMD, also known as the Wasserstein Metric, computed the distance between two probability distributions. In simple terms, EMD computed the minimum amount of mass needed to be displaced to convert a probability distribution into another. The smaller the EMD is,

the better the probability distributions match. In our case, for a specific temperature, we compute an observable for all lattices generated from the flow and compute the EMD against the histogram for the observable computed from MCMC samples. For scalar observable $\mathcal{O} = y$ at temperature T , the (normalized) histogram of the flow is denoted as $\mathcal{H}_F(\mathcal{O} = y|T)$, and that of the MCMC samples as $\mathcal{H}_M(\mathcal{O} = y|T)$.

$$\text{EMD}(\mathcal{H}_F(\mathcal{O}|T), \mathcal{H}_M(\mathcal{O}|T)) = \sum_{x=-\infty}^{+\infty} \left| \sum_{y=-\infty}^x (\mathcal{H}_F(\mathcal{O} = x|T) - \mathcal{H}_M(\mathcal{O} = y|T)) \right| \quad (6.1)$$

For a list of temperatures $\{T_1 \dots T_R\}$ we compute the *mean EMD* as

$$\frac{1}{R} \sum_{i=1}^R \left(\text{EMD}(\mathcal{H}_F(\mathcal{O}|T_i), \mathcal{H}_M(\mathcal{O}|T_i)) \right), \text{ and the } \textit{max EMD} \text{ as } \max \left(\text{EMD}(\mathcal{H}_F(\mathcal{O}|T_i), \mathcal{H}_M(\mathcal{O}|T_i)) \right).$$

6.1.2 Percent Overlap (%OL)

%OL is a simpler metric to compute the similarity between two histograms. For each index in the histogram, we compute the minimum value of the 2 histogram values, and sum over all indices.

$$\%OL(\mathcal{H}_F(\mathcal{O}|T), \mathcal{H}_M(\mathcal{O}|T)) = \sum_{x \in \text{bins}} \min(\mathcal{H}_F(\mathcal{O} = x|T), \mathcal{H}_M(\mathcal{O} = y|T)) \quad (6.2)$$

%OL is not as accurate as EMD, as relatively accurate observables may still get a small percent overlap, but it is useful in cases where the spread of the observable is large for a certain temperature. For a range of temperatures, we use the *mean* and *min* versions of this metric in a similar way as in EMD. In our experiments, we use 40 bins for each 1.0 increment range for our histograms to compute %OL.

6.1.3 L_2 error

This metric is used for observables obtained as a scalar instead of a histogram for a specific temperature, for example, Magnetic Susceptibility χ . It is simply denoted as $\|x - y\|_2$. For a range of temperatures, we take the average norm.

6.2 Experiment Setting

Now we will state the hyperparameters for each architecture and training of the flow composition used, along with the temperature ranges for the flow. We will also state the MCMC algorithm's hyperparameters.

6.2.1 MCMC Simulation hyperparameters

We generate the 'ground truth' for lattices using MCMC as in Section 2.2. We consider the range of temperatures in $[0.025, 1.025]$ with equal increments for a total of 32 temperatures. This consists of low temperatures, the critical region as well a slice of high-temperature regions, and is sufficiently representative of the model for training purposes. We simulate for lattice lengths of 8 and 16, although we observed a slight bias in magnetic susceptibility at low temperatures for larger lattices. This is expected as

For simulating the lattice states, we use MH sampling. We thermalize the chain with a burn-in of 200k samples. Afterward, we save samples every 400 steps to reduce autocorrelation between samples, gathering a total of 10k samples for each temperature.

We also optionally perform magnetization normalization (Section 5.2.1) in cases where it may be required for training.

Model	L	EMD Energy		EMD Mag.		EMD Vort.		%OL Energy		%OL Mag.		%OL Vort.		L_2 χ
		Mean	Max	Mean	Max	Mean	Max	Mean	Min	Mean	Min	Mean	Min	
HG-VAE[10]	8	0.63	0.79	0.15	0.22	0.054	0.079	11.22	4.01	58.73	27.60	49.17	22.65	3.8146
	16	0.593	0.751	0.124	0.202	0.046	0.064	0.255	0.0	46.53	2.27	39.19	5.13	2.9339
Implicit GAN[47]	8	0.035	0.087	0.059	0.157	0.003	0.007	76.94	7.60	63.98	33.23	93.88	86.39	0.0999
	16	0.037	0.074	0.091	0.274	0.0028	0.0064	63.38	12.55	43.40	12.88	94.17	83.18	0.2147
CC-CSF Forw. KL	8	0.039	0.087	0.028	0.093	0.004	0.010	84.76	64.93	86.40	71.46	93.42	83.85	0.0547
	16	0.054	0.113	0.048	0.172	0.007	0.016	67.19	40.61	70.39	34.98	81.99	59.53	0.2138
CC-CSF Rev. KL	8	0.025	0.055	0.039	0.125	0.003	0.011	80.61	50.93	80.98	62.56	95.90	84.66	0.0630
	16	0.023	0.075	0.106	0.302	0.0036	0.0119	85.32	62.31	50.46	7.08	90.30	64.29	0.2167

TABLE 6.1: Evaluated metrics for baselines and flow models. For reverse KL CC-CSF, $\alpha = 0.53$ for $L = 8$ and $\alpha = 0.55$ for $L = 16$. Mean and max values are computed over the range of temperature $T \in [0.025, 1.025]$, with 32 equally spaced intervals.

6.2.2 Flow Settings

The flow used contains a total of 15 compositions, each of these further containing 2 flow compositions each with alternating checkerboard masking. Thus in total, the flow consists of 30 transformations. Each transformation uses the Temperature-conditioned Coupling CSF introduced in Section 5.1. The Circular Splines for each transformation contain a total of $K = 5$ piecewise functions. The neural network for outputting parameters within each transformation consists of a single hidden layer fully-convolutional neural net with 2D filters of size 3. We use periodic padding for these convolution layers to preserve the periodic boundary conditions of the lattice. As we shall later see, this fully convolutional neural net allows us to train flows for larger lattice sizes through transfer learning.

6.2.2.1 Training and Evaluation

We train the flows broadly with 2 methods and their regularization techniques, which are outlined in Sections 5.2, 5.3. For Forward KL training, we train the flow with the above specifications for a total of $\sim 6k$ iterations. During training through reverse KL, we train the flow for longer, $\sim 15k$ iterations. We use the ADAM [25] optimizer, with a learning rate of 2×10^{-4} and a cosine learning rate scheduler [19]. We monitor the observable metrics and loss while training for tuning relevant training hyperparameters.

While evaluating the flows, we generate 1k samples for each temperature in the temperature range described previously. We then compute the various metrics introduced in Section 6.1, and have tabulated these results compared to the baselines considered in Section 4.3 in Table 6.1.

6.3 Observations

6.3.1 Training the Flow with Forward KL

Training using Forward KL, a) without magnorm, and b) with magnorm produces vastly different results. We noticed that without magnorm, the model produces lattice observables that are far from the MCMC samples. We conjecture that this is because the model is not powerful enough to model a continuum of identical modes in the probability space. magnorm essentially collapses these modes into one single mode by eliminating all of the other ones entirely from the dataset.

With magnorm, the flow can learn sufficiently close observables to the MCMC samples.

All observables computed with the MH algorithm are shown in Figure 2.2.

6.3.2 Training the Flow with Reverse KL

We observe that with the Reverse KL objective, the flow is not able to learn the distribution and produces poor observables and metrics if we do not employ explicit regularization as introduced in Section 5.3. With regularization enabled, the flow performs on par with Forward KL-trained flow, and is competitive with and often beats ImplicitGAN. A constant value of $\lambda = \frac{1}{2\pi^2}$ was used in Equation 5.7, while a simple grid search for the hyperparameter α was employed while training the flow. It seems like the tempering of the target density (the XY model's Boltzmann distribution) was crucial in stabilizing the training and increasing the accuracy of the Reverse KL-trained flow. The additional term of minimizing the sine component of the spins additionally stabilized training, albeit a large weight to this term hindered training.

6.3.3 Comments on Performance

We list here some observations by analyzing Table 6.1 and Figures 6.1,6.2. In general, the flow performs on par or better than ImplicitGAN in all metrics, while being significantly better than HG-VAE. Also note that usually the flow trained using Reverse KL is better at matching vorticity than the one trained using Forward KL, while the latter is better at the other observables.

The mean and max EMD for vorticity is better in the case of ImplicitGAN, however, the flows perform equally well and the values themselves are so small that the discrepancy may be attributed to randomness between the runs. Also, it is clear that magnetization is better learned by the Forward KL flow, and energy by the Reverse KL trained one.

Magnetic susceptibility, however, is harder to learn as it peaks more near the critical temperature as the lattice size increases. In Figures 6.1 and 6.2, it can be seen that in the case of flows, the peak for model-produced lattices occurs at a higher temperature than it should. In contrast, ImplicitGAN has those peaks at roughly the same temperatures as in MCMC. Nevertheless, according to L_2 metric, both the flow and ImplicitGAN are close in reproducing the susceptibility observable. There needs to be more analysis performed

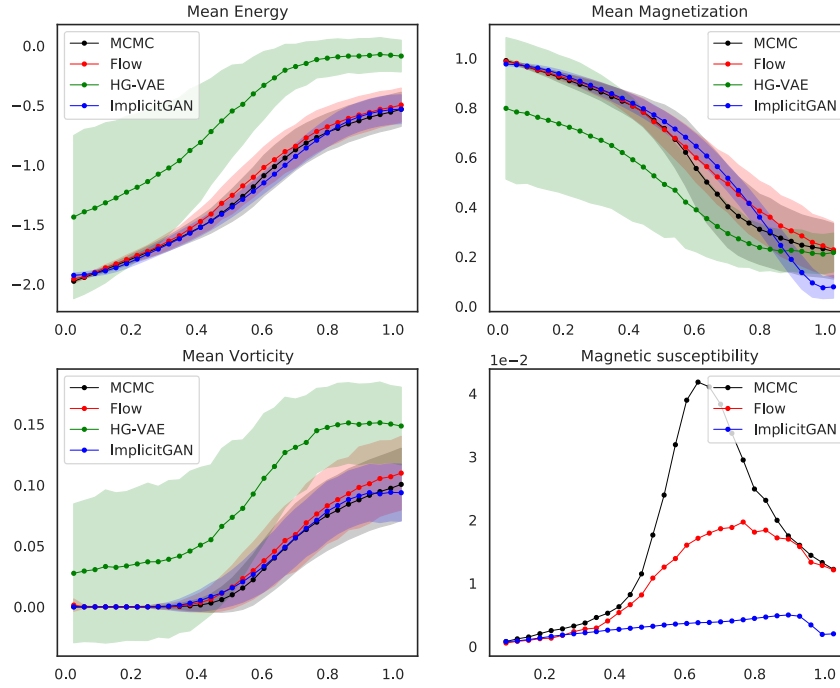
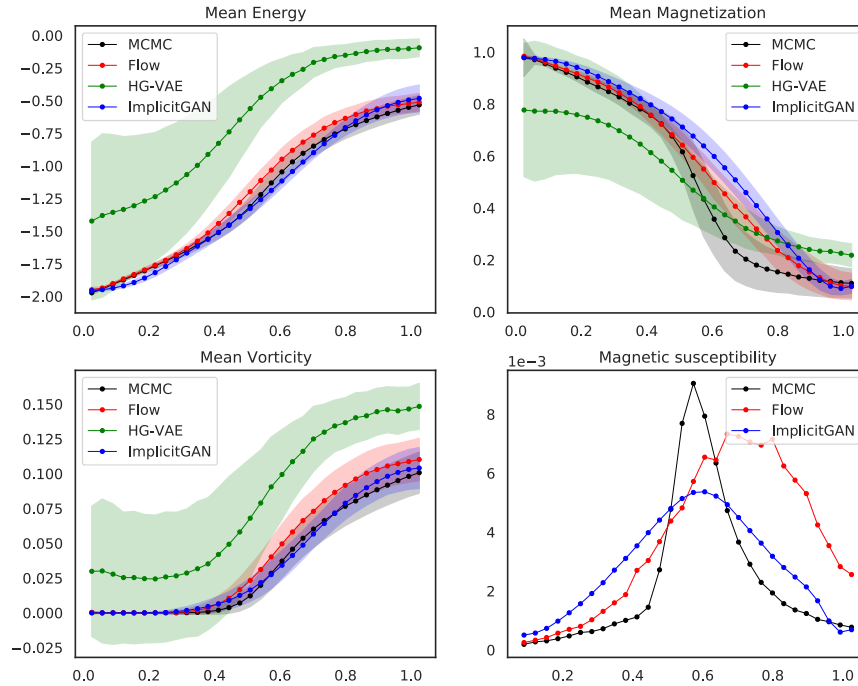
(A) Observables for $L = 8$ (B) Observables for $L = 16$

FIGURE 6.1: Observables for $L = 8$ and $L = 16$, computed using samples from a) MCMC, b) HG-VAE, c) ImplicitGAN, and d) CC-CSF trained using Forward KL. Susceptibility at low temperatures for HG-VAE was highly divergent, hence omitted.

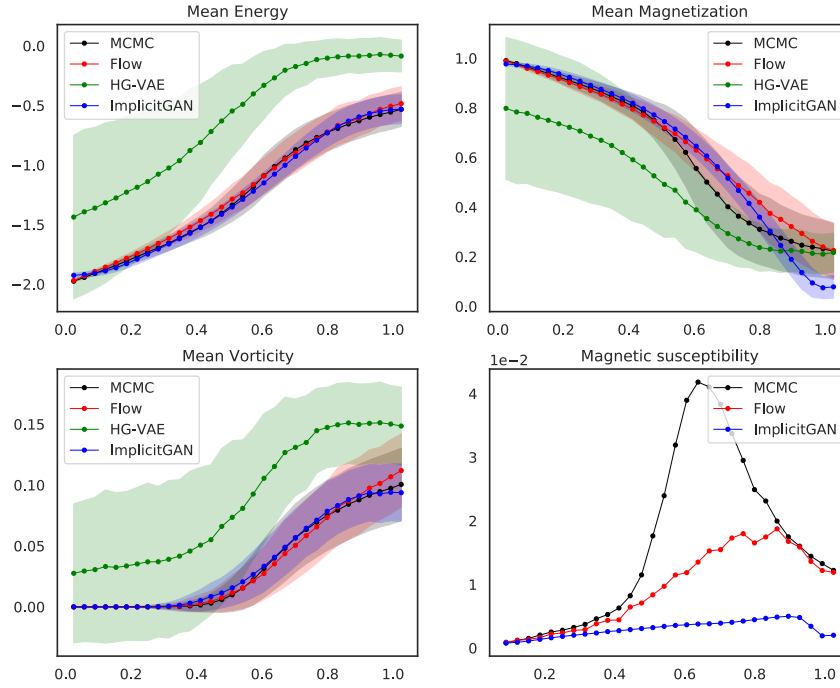
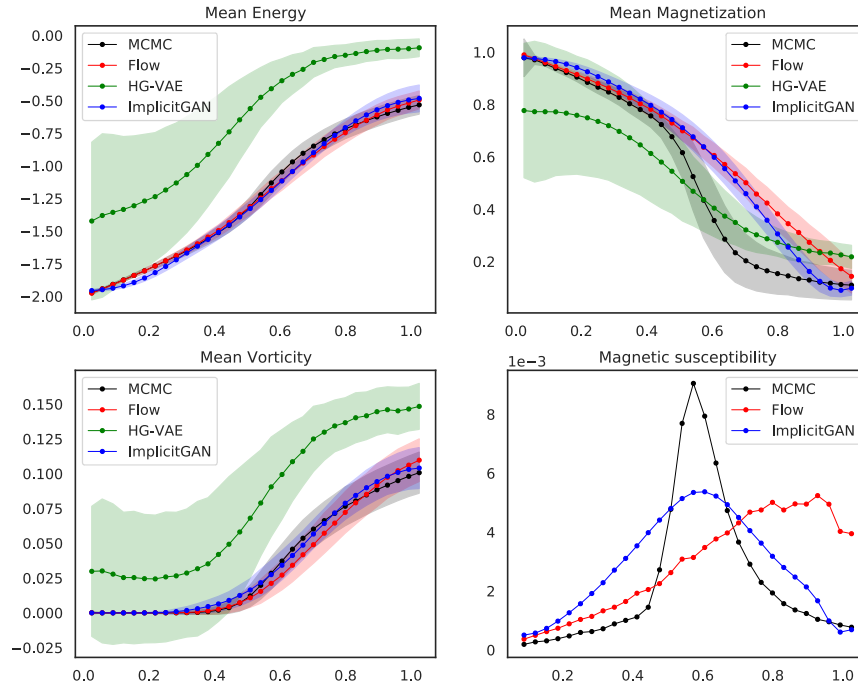
(A) Observables for $L = 8$ (B) Observables for $L = 16$

FIGURE 6.2: Observables for $L = 8$ and $L = 16$, computed using samples from a) MCMC, b) HG-VAE, c) ImplicitGAN, and d) CC-CSF trained using Reverse KL. Susceptibility at low temperatures for HG-VAE was highly divergent, hence omitted.

in our case for these higher-order observables, especially since they tend to diverge in the critical region and are thus harder to learn.

A question then arises: when should we use the Forward KL vs Reverse KL for training the flow? It seems that if our goal is to approximate magnetization, we should use the Forward KL as the training method. In any case, if we do not have access to MCMC samples beforehand, we will have to use the Reverse KL method, as it only requires the Hamiltonian formula of the XY model. If we do have the samples, we can simply resort to using Forward KL, as the training is slightly faster. In the sample generation phase during testing, the flows were slightly slower than the ImplicitGAN and HG-VAE (owing to multiple sequential transformations), but both were orders of magnitude faster than MCMC. Training the ImplicitGAN and HG-VAE was slightly faster than our flow models. However, it is also worth pointing out that training of flow models is relatively simple, as normalizing flows are somewhat more robust to training initializations and hyperparameters settings than the competing methods. We found that ImplicitGAN was significantly harder to train, requiring multiple training runs due to problems with mode collapse.

6.4 Transfer Learning for Larger Lattices – A Qualitative Analysis

Since our flow architecture is fully convolutional, it does not depend on the size of the lattice we want to generate. Therefore, simply changing the size of the checkerboard masking pattern to be bigger for a bigger lattice size means that we can apply the same filters we learned at lower lattice sizes and use them to further learn larger lattice models. A recent work [5] performs this transfer learning on Lattice Field Theory models. In this section, we shall see a brief demonstration of whether transfer learning is possible in the XY model.

We want to learn a flow to generate lattices with size 32×32 . Since even MCMC struggles to generate such large lattices fast enough, we shall use the flow that we trained on smaller 8×8 sized lattices. In this experimental setting, we used a larger temperature range of $T \in [0.36, 1.40]$, as generating $L = 32$ sized lattices in the low-temperature regime turns out to be slow for MCMC. Furthermore, the MCMC samples at such low temperatures are not sufficiently thermalized, and therefore impact the training of the flow. This is an area we will have to investigate further.

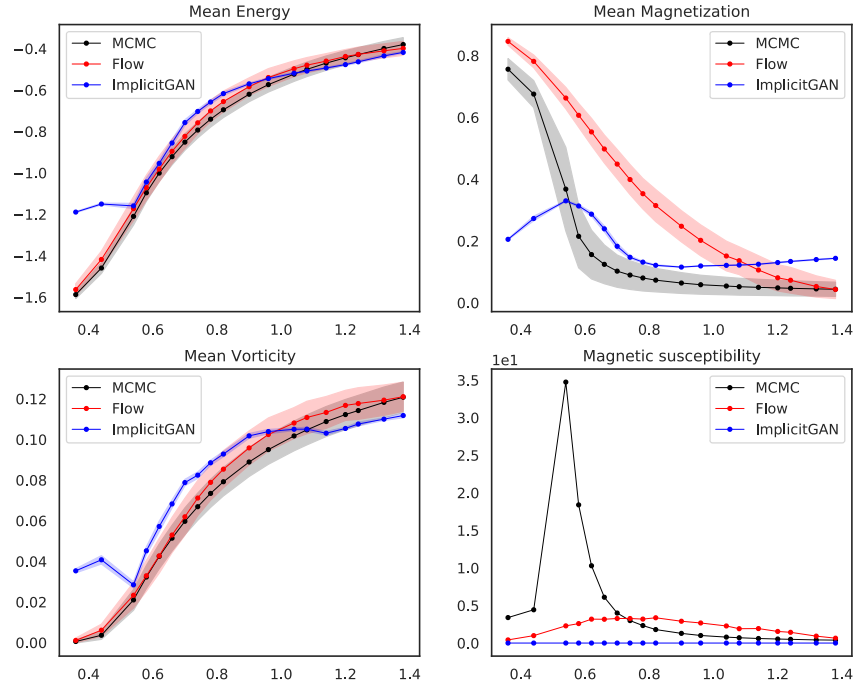
We take the layers of the Flow $F_{8 \times 8}$ and create a new flow composition $F_{32 \times 32}$ with these same layers. The Forward KL objective is used for finetuning of the model on MCMC

samples of 32×32 size. The only thing that we need to change is the checkerboard mask m , which is now 32×32 instead of 8×8 . Figure 6.3a shows observables from samples generated from $F_{32 \times 32}$, without any finetuning with new data. Figure 6.3b shows the observables after finetuning with MCMC data for a few epochs.

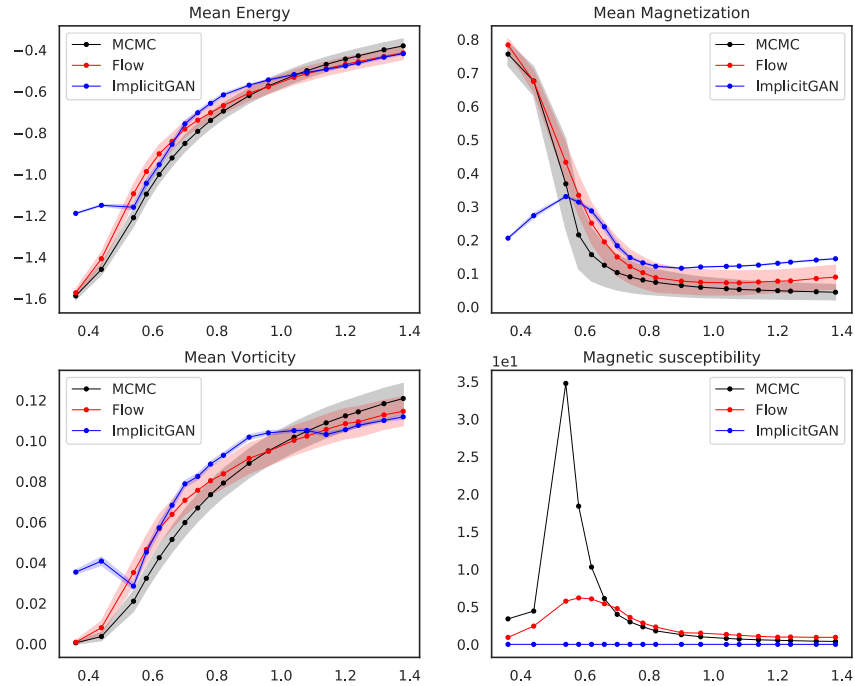
Qualitatively, we observe that the flow already matches the energy quite well even without finetuning. Magnetization is observed to be inaccurate for the new flow, as it drops even more rapidly with temperature for higher lattice sizes. However, after finetuning, we see that the magnetization has an even better fit to the MCMC data than before. Due to the larger lattice size, magnetic susceptibility peaks very sharply at the critical temperature, and the flow is inadequate for modeling the peak.

Interestingly, ImplicitGAN fails to learn the 32×32 sized lattice, giving a significantly worse fit to the MCMC data. ImplicitGAN also gives extremely sharp predictions at each temperature, where the variance of the observables is very small, suggesting mode collapse behaviour at larger lattice sizes. At lower temperatures, the observables are very inaccurate, which might be due to less thermalization of the lattice samples at low temperatures. It seems that ImplicitGAN thus requires very accurate MCMC samples to produce the same degree of accuracy as our model.

One can similarly perform this transfer learning with Reverse KL as the training objective. More work on this front needs to be done before this method of generating larger lattices is scalable with enough accuracy.



(A) Flow model before finetuning.



(B) Flow model after finetuning.

FIGURE 6.3: Observables computed using samples from a) MCMC, b) CC-CSF transfer-learned from $L = 8$ flow, b) ImplicitGAN trained on 32×32 lattice data simulated using MH algorithm. Finetuning was done using Forward KL objective. For both models, dataset used contains 10k samples for each temperature.

Chapter 7

Conclusion and Future Work

This work demonstrated the application of normalizing flows for accelerating simulation of the XY model. Experimental analyses were presented for lattice sizes of 8×8 and 16×16 . We observed performance beating or competitive with the state-of-the-art accelerated simulation approaches. We also demonstrated the capability of transfer learning to large lattice sizes.

It is worth noting that all the baselines, and our proposed model, only approximate the XY model, and are not asymptotically unbiased. Our model, being a tractable likelihood model, has the capacity for unbiased generation of samples through post-hoc correction through MCMC or Importance Sampling [1, 24, 5, 37]. However, the implicit and explicit regularizations we employ hinder the flow’s capability of density matching, sacrificing it for sample fidelity. Training with no regularization does not match either density or samples from the XY model. As a result, applying post-hoc correction techniques to match the XY model’s Boltzmann distribution in an exact manner does not work – the density outputted by the flow and the Boltzmann distribution suffer from a mismatch. As the physical models that are learned in the abovementioned related work do not contain continuous symmetry and topological phase transitions together, this leads us to hypothesize that lattice models with topological phase transitions and global continuous symmetry constraints are significantly harder to learn than those with no phase transitions or discrete symmetry. Thus, we may require more specialized architectures to effectively model its symmetries.

As future work, we are in the process of baking in the $U(1)$ global rotational symmetry constraint into the XY model. In preliminary results, we find that an equivariant flow model constructed in a manner similar as in [24] enabled us to learn perfectly a 1-D XY model spin chain, with relatively simple network architecture and vanilla Reverse KL objective.

Our current model construction was unable to do so without regularization. However, the 2-D XY model was unable to be learned efficiently at low temperatures, while it was easy to learn at high temperatures – this is without any regularization applied. More analysis needs to be done for learning such models at lower temperatures and enable post-hoc correction mechanisms.

Bibliography

- [1] MS Albergo, G Kanwar, and PE Shanahan. Flow-based generative models for markov chain monte carlo in lattice field theory. *Physical Review D*, 100(3):034515, 2019.
- [2] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003.
- [3] Matthew JS Beach, Anna Golubeva, and Roger G Melko. Machine learning vortices at the kosterlitz-thouless transition. *Physical Review B*, 97(4):045207, 2018.
- [4] Avishek Joey Bose, Ariella Smofsky, Renjie Liao, Prakash Panangaden, and William L Hamilton. Latent variable modelling with hyperbolic normalizing flows. *arXiv preprint arXiv:2002.06336*, 2020.
- [5] Denis Boyda, Gurtej Kanwar, Sébastien Racanière, Danilo Jimenez Rezende, Michael S Albergo, Kyle Cranmer, Daniel C Hackett, and Phiala E Shanahan. Sampling using $su(n)$ gauge equivariant flows. *arXiv preprint arXiv:2008.05456*, 2020.
- [6] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4):045002, 2019.
- [7] Juan Carrasquilla and Roger G Melko. Machine learning phases of matter. *Nature Physics*, 13(5):431–434, 2017.
- [8] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The American Statistician*, 49(4):327–335, 1995.
- [9] Barry A Cipra. An introduction to the ising model. *The American Mathematical Monthly*, 94(10):937–959, 1987.
- [10] Marco Cristoforetti, Giuseppe Jurman, Andrea I Nardelli, and Cesare Furlanello. Towards meaningful physics from generative models. *arXiv preprint arXiv:1705.09524*, 2017.

-
- [11] Adji B Dieng, Francisco JR Ruiz, David M Blei, and Michalis K Titsias. Prescribed generative adversarial networks. *arXiv preprint arXiv:1910.04302*, 2019.
- [12] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.
- [13] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.
- [14] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Cubic-spline flows. *arXiv preprint arXiv:1906.02145*, 2019.
- [15] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems*, pages 7511–7522, 2019.
- [16] Stavros Efthymiou, Matthew JS Beach, and Roger G Melko. Super-resolving the ising model with convolutional neural networks. *Physical Review B*, 99(7):075113, 2019.
- [17] Mevlana C Gemici, Danilo Rezende, and Shakir Mohamed. Normalizing flows on riemannian manifolds. *arXiv preprint arXiv:1611.02304*, 2016.
- [18] Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.
- [19] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. *arXiv preprint arXiv:1810.13243*, 2018.
- [20] JA Gregory and R Delbourgo. Piecewise rational quadratic interpolation to monotonic data. *IMA Journal of Numerical Analysis*, 2(2):123–130, 1982.
- [21] Gavin S Hartnett and Masoud Mohseni. Self-supervised learning of generative spin-glasses with normalizing flows. *arXiv preprint arXiv:2001.00585*, 2020.
- [22] Hong-Ye Hu, Shuo-Hui Li, Lei Wang, and Yi-Zhuang You. Machine learning holographic mapping by neural network renormalization group. *Physical Review Research*, 2(2):023369, 2020.
- [23] Max Jensen and Kilian Nickel. ϕ^4 theory on the lattice. 2011. URL <https://www.hiskp.uni-bonn.de/uploads/media/phi4.pdf>.
- [24] Gurtej Kanwar, Michael S Albergo, Denis Boyda, Kyle Cranmer, Daniel C Hackett, Sébastien Racanière, Danilo Jimenez Rezende, and Phiala E Shanahan. Equivariant flow-based sampling for lattice gauge theory. *arXiv preprint arXiv:2003.06413*, 2020.

-
- [25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [26] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [27] Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.
- [28] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in Neural Information Processing Systems*, 29:4743–4751, 2016.
- [29] Jonas Köhler, Leon Klein, and Frank Noé. Equivariant flows: sampling configurations for multi-body systems with symmetric energies. *arXiv preprint arXiv:1910.00753*, 2019.
- [30] J M Kosterlitz. The critical properties of the two-dimensional xy model. *Journal of Physics C: Solid State Physics*, 7(6):1046–1060, mar 1974. doi: 10.1088/0022-3719/7/6/005. URL <https://doi.org/10.1088/0022-3719/7/6/005>.
- [31] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [32] Shuo-Hui Li and Lei Wang. Neural network renormalization group. *Physical Review Letters*, 121(26):260601, 2018.
- [33] Jaechang Lim, Seongok Ryu, Jin Woo Kim, and Woo Youn Kim. Molecular generative model based on conditional variational autoencoder for de novo molecular design. *Journal of Cheminformatics*, 10(1):1–9, 2018.
- [34] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.
- [35] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *arXiv preprint arXiv:1906.10652*, 2019.
- [36] Kim A. Nicoli, Shinichi Nakajima, Nils Strodthoff, Wojciech Samek, Klaus-Robert Müller, and Pan Kessel. Asymptotically unbiased estimation of physical observables with neural samplers. *Phys. Rev. E*, 101:023304, Feb 2020. doi: 10.1103/PhysRevE.101.023304. URL <https://link.aps.org/doi/10.1103/PhysRevE.101.023304>.

-
- [37] Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457): eaaw1147, 2019.
- [38] Aaron Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, et al. Parallel wavenet: Fast high-fidelity speech synthesis. In *International Conference on Machine Learning*, pages 3918–3926. PMLR, 2018.
- [39] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.
- [40] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.
- [41] Jan M Pawłowski and Julian M Urban. Reducing autocorrelation times in lattice simulations with generative adversarial networks. *Machine Learning: Science and Technology*, 1(4):045011, 2020.
- [42] AST Pires, LS Lima, and ME Gouvea. The phase diagram and critical properties of the two-dimensional anisotropic xy model. *Journal of Physics: Condensed Matter*, 20(1):015208, 2007.
- [43] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.
- [44] Danilo Jimenez Rezende, George Papamakarios, Sébastien Racanière, Michael S Albergo, Gurtej Kanwar, Phiala E Shanahan, and Kyle Cranmer. Normalizing flows on tori and spheres. *arXiv preprint arXiv:2002.02428*, 2020.
- [45] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.
- [46] Samuel S Schoenholz. Combining machine learning and physics to understand glassy systems. In *Journal of Physics: Conference Series*, volume 1036, page 012021, 2018.
- [47] Japneet Singh, Vipul Arora, Vinay Gupta, and Mathias S Scheurer. Generative models for sampling and phase transition indication in spin systems. *arXiv preprint arXiv:2006.11868*, 2020.

-
- [48] Prince Zizhuang Wang and William Yang Wang. Riemannian normalizing flow on variational wasserstein autoencoder for text modeling. *arXiv preprint arXiv:1904.02399*, 2019.
- [49] Wikipedia contributors. Schwinger model — Wikipedia, the free encyclopedia, 2020. URL https://en.wikipedia.org/w/index.php?title=Schwinger_model&oldid=993425224.
- [50] Ulli Wolff. Collective monte carlo updating for spin systems. *Physical Review Letters*, 62(4):361, 1989.