universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Enhancing Boltzmann-Generators with Coarse-Grained Latent Distributions“

verfasst von / submitted by

Jakob Schindelwig

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 876
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Physik
Betreut von / Supervisor:	Univ. Prof. Christoph Dellago

Acknowledgments

My biggest thanks go to Sebastian Falkner, for all the time he spent explaining and debating and helping whenever I needed him. One could not wish for better support. I also thank Christoph Dellago for being a great supervisor and for welcoming me into his splendid research group, I really felt at home in the office. Thanks to all the group members and especially my fellow students, who wrote their Master-Thesis parallel to me, for their excellent company! Without all of you, this thesis would not have come to be.

Abstract

Predicting observables of equilibrium distributions of atomic and molecular systems is, among many others, a problem, for which samples representing the equilibrium distribution of said systems are needed. In recent years Boltzmann-Generators were introduced as a promising new tool to obtain samples from the equilibrium distribution. These are generative, deep neural networks used to generate uncorrelated configurations of physical systems according to Boltzmann statistics. A learned mapping transforms samples from either Gaussian or uniform probability distribution to the configuration space while allowing for reweighting of the generated distribution to the reference distribution. We enhance Boltzmann-Generators by instead transforming a mix of coarse-grained configurations generated by traditional sampling methods together with Gaussian or uniform distributions and study the effects on the training process and the overall performance. Testing this approach on Alanine-Dipeptide and a ond-dimensional XY-Model, we conclude that under the right circumstances this approach accelerates the training and increases the performance.

Kurzfassung

Die Vorhersage von Observablen atomarer und molekularer Systeme ist unter anderem ein Problem, für das die Gleichgewichtsverteilung repräsentierende Konfigurationen benötigt werden. Vor wenigen Jahren wurden Boltzmann-Generatoren als ein vielversprechendes neues Werkzeug zur Generierung solcher Konfigurationen eingeführt. Dabei handelt es sich um generative, tiefe neuronale Netze, die zur Erzeugung unkorrelierter Konfigurationen physikalischer Systeme gemäß der Boltzmann-Verteilung verwendet werden. Ein erlerntes Mapping transformiert Stichproben aus einer Gaußschen oder uniformen, latenten Wahrscheinlichkeitsverteilung in den Konfigurationsraum und ermöglicht dabei eine Neugewichtung der generierten Verteilung gegenüber der Referenzverteilung. Wir verbessern Boltzmann-Generatoren, indem wir statt der herkömmlichen latenten Verteilungen eine Mischung aus durch traditionelle Sampling-Methoden erzeugte coarse-grained Konfigurationen zusammen mit Gaußschen- oder uniformen Verteilungen transformieren. Anschließend untersuchen wir die Auswirkungen auf den Trainingsprozess und die Leistung des Netzwerkes. Wir testen diesen Ansatz an Alanin-Dipeptiden und einem eindimensionalen XY-Modell und kommen zu dem Schluss, dass dieser Ansatz unter den richtigen Umständen das Training beschleunigt und die Leistung erhöht.

Contents

Acknowledgments i

Abstract iii

Kurzfassung v

1. Introduction 1

2. Theory 5
2.1. Maxwell-Boltzmann Statistics . . . 5
2.2. The Sampling Problem . . . 5
2.2.1. Monte Carlo Algorithms . . . 6
2.2.2. Molecular Dynamics Simulations . . . 7
2.2.3. Coarse-Graining of Molecules . . . 8
2.3. Deep Neural Networks . . . 9
2.3.1. Feedforward Neural Networks (FNN) . . . 10
2.4. Generative Models . . . 11
2.5. Normalizing Flows . . . 12
2.5.1. Normalizing Flow architectures . . . 14
2.5.2. Real-NVP Transformation Networks . . . 16
2.5.3. Neural Spline Flows . . . 17
2.5.4. Boltzmann generators . . . 17

3. Results 23
3.1. XY-Spinchain . . . 23
3.1.1. MC-Simulation for the Generation of Training Data . . . 24
3.1.2. Gaussian Mixture Model . . . 25
3.1.3. Network Architecture . . . 26
3.1.4. Performance Metrics of the Gaussian Mixture Prior . . . 28
3.2. The Alanine Dipeptide Model . . . 37
3.2.1. MD-Simulation for the Generation of Reference Data . . . 37
3.2.2. Latent Vector - Coarse Graining . . . 38
3.2.3. Transformation Layers . . . 40
3.2.4. Training the Boltzmann generator on the Alanine Dipeptide . . . 41
3.3. Performance with Coarse Graining in Latent Distribution . . . 43
3.3.1. Proof Of Concept . . . 43
3.3.2. Performance of Boltzmann Generators with Coarse Graining . . . 43

3.3.3. Reweighting And Resampling the Alanine Dipeptide Configurations 46

3.3.4. Temperature Exchange 47

4. Discussion **51**

Bibliography **53**

List of Figures **63**

A. Appendix **67**

A.1. Algorithms 67

A.1.1. MC-Simulation of the XY-Model 67

A.1.2. Gaussian Mixture Model 67

A.1.3. XY-Model Network 67

A.1.4. Alanine Dipeptide MD-Simulation 68

A.1.5. Coarse Grained Simulation 68

A.1.6. Alanine Dipeptide Network 68

1. Introduction

One core challenge in many scientific fields consists of sampling states reflecting the equilibrium distributions of atomic and molecular systems. One needs these samples of the equilibrium distribution to understand all kinds of molecular processes such as explaining phase transitions in condensed matter physics, predicting a drug's behavior in the human body or understanding a protein's behavior under various circumstances. Only for very simple systems the equilibrium state distribution can be sampled based on analytical solutions, for more complex systems, numerical methods are required. These numerical methods for generating samples reflecting the equilibrium distribution of a system are usually started using an initial configuration, which is then propagated using various algorithms. Hence, samples are drawn not independently but rather sequentially, which leads to correlations between them. The two most used methods are Monte Carlo (MC) simulations and Molecular dynamics (MD) simulations [1]. In both cases, the degree of perturbation applied to a previously visited configuration is limited, by acceptance in the case of MC and by the fastest motions in the case of MD. Therefore, both of those methods have similar limitations, both do not scale well with the degrees of freedom (DOFs) of the system under study, and both perform poorly simulating systems with high energetic or entropic barriers between high probability regions. In complex systems long correlation times can be obvserved for both, meaning that many steps are necessary until a statistically independent sample is generated. These limitations can partially be overcome with different methods for enhancing sampling [2, 3, 4].
Coarse graining (CG) [5] is a widely used approach tackling the aforementioned limitations. By replacing groups of atoms with interaction centers and introducing force fields mimicking the actual forces, the DOFs are reduced and perturbation steps can be chosen larger due to a regularized potential energy surface. Therefore, the correlation time also decreases. A prominent example of this approach is the Martini model, a CG model, which was used to simulate an entire cell [6].
To overcome the limitations set by energy barriers much larger than $k_{\mathrm{B}}T$ that lead to rare transitions and, hence, long correlation times, advanced sampling methods have been developed. Three examples are the replica exchange method [4], umbrella sampling [2, 7], and metadynamics [3, 8]. The replica exchange method starts simulations at different temperatures, and, if certain criteria are fulfilled, exchanges the current configurations between two simulations. This allows low-temperature simulations to explore the whole phase space without waiting for rare transitions by undergoing transitions at high temperatures before switching to a lower temperature again. For umbrella sampling methods, first, a collective variable capturing the process of interest has to be defined. A bias potential (mostly harmonic potentials) as a function of this collective variable is then added and simulations are run for different bias positions. This way, different simula-

tions are set to explore different regions of phase space, which are in the end combined by reweighting the samples from each window. Similar to umbrella sampling, also in metadynamics simulations a bias potential is introduced along one or more before defined collective variables. Here Gaussian bias kernels are added successively after a fixed number of iterations at the current collective variable position pushing the walker out of local minima over time. This allows the exploration of higher energy regions along the collective variable after some time.

Recently, machine learning became increasingly popular in various physical fields such as studying out-of-equilibrium systems [9], finding better force fields for MD simulations [10], in high energy physics [11] and protein structure prediction [12]. This rise in popularity is mostly due to the increasing power of graphics cards in recent years, which in turn led to steep scientific progress in the field of deep learning. Among other advances, new stochastic gradient descent algorithms have been developed [13] in turn improving neural networks performances. These improved training methods allowed for larger and more complex networks and larger models, shifting the interest toward deep learning.

Deep learning has been successfully employed for image recognition [14], speech recognition [15], natural language understanding [16] and many other tasks. Since the first introduction of a deep generative model, the Deep Belief network [17], many more network architectures were introduced [18] and they were improved to the point of being now used by many people in everyday life to generate pictures [19] and text [20].

For a long time the absence of a probabilistic architecture with sufficient generative performance while allowing for estimation of densities made tackling the sampling problem with generative deep learning impossible, but the recent invention of Normalizing-Flow networks [21] has opened new chapters in statistical physics. In 2019, Noè and coworkers [22] proposed utilizing the Normalizing-Flow model to generate uncorrelated equilibrium samples of complex molecules quickly and referred to their invention as Boltzmann generators (BGs).

BGs learn an invertible transformation of an easy-to-sample-from probability distribution to the equilibrium distribution of the system under study. The training is conducted in two directions, meaning the network learns to generate configurations mimicking train data, and it alters the transformation so that the free energy of the distribution of the transformed samples is minimized. BGs have since also been used to model atomic solids [23], to generate configurations at different temperatures [24], leveraging internal coordinates [25], to generate transition paths [26], to model liquids [27] and to generate samples from an isobaric, isothermal ensemble [28]. However, previous works [22, 23] worked on small scale systems due to the suboptimal scaling of the BGs performance with the number of degrees of freedom.

In this thesis, we aim to support BGs with coarse-graining to overcome the said limitation. We propose exchanging the easy-to-sample-from probability distribution, with other latent distributions closer to the Boltzmann distribution. For one approach tested on the alanine dipeptide we use a combination of a coarse-grained simulation- and Gaussian distributions instead of only Gaussian distributions as latent distribution. The other approach we tested utilizes Gaussian-Mixtures-Models (GMMs). We compare the performance of

a Boltzmann generator transforming a purely uniform latent distribution with a BG transforming a combination of a GMM distribution and uniform distributions.

2. Theory

2.1. Maxwell-Boltzmann Statistics

In statistical physics, there are various approaches to describe systems depending on their connection to an ideal surrounding. An isolated system, where the energy is constant, is characterized by the microcanonical ensemble. A system, where heat exchange is allowed, which leads to a constant temperature, is well described with the canonical ensemble. Open systems can be described by the grand canonical ensemble, for which temperature and pressure are constant.
In this thesis we only look at closed systems (no variation in particle number) in thermal equilibrium, so we use the canonical ensemble. The idea here is the following: A system has a fixed number of constituents j with k degrees of freedom (DOF) described by a vector $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_j)$ based on which the Hamiltonian $\mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_j)$ of the system can be analytically calculated. One important quantity is the canonical partition function Z (eq. 2.1).

$$Z = \int \frac{1}{\xi h^{3j}} d^k\mathbf{x}_1 d^k\mathbf{x}_2 ... \mathbf{x}_j^k e^{-\beta\mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)} = e^{-\beta F} \tag{2.1}$$

Here, h is the Planck constant, which renders the partition function dimensionless and ξ is an over counting factor becoming meaningful when different points in the $j \cdot k$ dimensional space of DOF describe the same state due to identical particles. F is the equilibrium free energy of the system and $\beta = \frac{1}{k_\mathrm{B} T}$ with k_B being the Boltzmann constant and T being the temperature. With different operations basically all relevant quantities describing a system, e.g. entropy and heat capacity, can be derived from Z. The probability to find the system in a certain state $\mathbf{x}$ is given by equation 2.2:

$$\rho(\mathbf{x}) = \frac{1}{Z} e^{-\beta\mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_j)} \tag{2.2}$$

The equilibrium distribution of a system is the one that minimizes the free energy F. Since it is impossible to derive the canonical partition function analytically for most systems due to the complexity of the integral in equation 2.1, the PDF is also not analytically tractable. However, numerous approaches have been developed to enable sampling from the Boltzmann distribution while circumventing the calculation of the partition function [1].

2.2. The Sampling Problem

Sampling states of a system according to the Boltzmann distribution is a difficult task for complex systems since no expression of the PDF is analytically tractable. To circumvent

this central problem, the most common approach is employing algorithms that generate new samples step by step based on previous samples which, in the limit of infinite steps, visit each state according to their statistical weight.

2.2.1. Monte Carlo Algorithms

Monte Carlo methods are a common tool in statistical physics for exploring the phase space of systems at certain temperatures while the formulation of the algorithm guarantees that generated samples will follow the desired distribution. One famous and here used algorithm is the Metropolis algorithm [29]. The algorithm is started at a random point $\mathbf{x}_0$ in phase space with the energy $E(\mathbf{x}_0) = E_0$. Then a random perturbation is applied to $\mathbf{x}_0$ leading to the phase space point $\mathbf{x}_1$ and the new energy $E(\mathbf{x}_1) = E_1$ is calculated. If $E_1 \leq E_0$ the step is accepted and the new point is used as the initial point in the next iteration. If $E_1 > E_0$ a random number $0 \leq p < 1$ is sampled from a uniform distribution. If $e^{-(\beta E_1 - \beta E_0)} < p$ the step is accepted. Otherwise, the step is rejected and the old point in phase space is retained. This procedure is repeated N times and therefore N phase space points are saved. The acceptance criterion is derived from the detailed balance condition equation 2.3, which states that the probability density of one state $\mathcal{N}(A)$ times the transition matrix element $\pi(AB)$, the transition probability to get to from state A to state B equals the probability density of that region times the transition probability $\pi(BA)$:

$$\mathcal{N}(A)\pi(AB) = \mathcal{N}(B)\pi(BA) \tag{2.3}$$

The transition matrix π contains all transition probabilities between states, which for all states C and D is defined as $\pi(CD) = P(X_t = D | X_{t-1} = C)$. To derive an acceptance probability for the step $\mathrm{acc}(AB)$ we define $\pi(AB) = \alpha(AB) * \mathrm{acc}(AB)$. If the generation probability α is symmetric, the relation 2.4 holds.

$$\mathcal{N}(A) \cdot \mathrm{acc}(AB) = \mathcal{N}(B) \cdot \mathrm{acc}(BA) \tag{2.4}$$

Put together this results in

$$\frac{\mathrm{acc}(AB)}{\mathrm{acc}(BA)} = \frac{\mathcal{N}(B)}{\mathcal{N}(A)} = \exp(-\beta(E_B - E_A)) \tag{2.5}$$

To arrive at the method mentioned above $\mathrm{acc}(AB)$ and $\mathrm{acc}(BA)$ are chosen such that the larger one is equal to one. The step acceptance probability p_{acc} can be rewritten as

$$p_{\mathrm{acc}} = \min(1, \exp(-\beta(E_B - E_A)) \tag{2.6}$$

In the limit $n \to \infty$ every point in the phase space is sampled according to its probability by fulfilling the detailed balance condition. In addition, the condition leads to ergodic sampling of phase space as long as the perturbation method is sufficiently expressive. This means, that in the limit of infinite sampling time every point in phase space is reached and, therefore, the ensemble and time average are equivalent.

An observable A under the equilibrium probability distribution $p(\mathbf{x})$ can be calculated exactly from infinite samples and approximated from a set of samples $\{\mathbf{x}_i\}_{i\in[1,\dots N]} \sim p(\mathbf{x})$:

$$\langle A(\mathbf{x})\rangle_{\mathbf{x}\sim p(\mathbf{x})} = \int \mathrm{d}\mathbf{x}\, p(\mathbf{x})A(\mathbf{x}) = \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} A(\mathbf{x}_i) \approx \frac{1}{N}\sum_{i=1}^{N} A(\mathbf{x}_i) \tag{2.7}$$

This approximation's accuracy depends on how well the set of samples represents the underlying distribution. The representative power of an MC-simulation generated sample-set suffers in the presence of high energy barriers considerably larger than $k_\mathrm{B}T$, since then transitions between phase space regions become very unlikely. The curse of dimensionality affects also this method, since as the phase space volume scales with the number of DOF, so does the number of steps n needed for equally representative sample set [1].

2.2.2. Molecular Dynamics Simulations

In the realm of atoms, quantum mechanics reign. But since analytical computations of systems other than H and H_2 are impossible, one needs numerical algorithms to obtain information on complex atomic systems. With density functional theory, one can very accurately, albeit very slowly, simulate molecular systems with chemical accuracy. A widespread alternative approach is to model atomic interactions with classical effective force fields in a way that mimics their actual behavior neglecting any quantum mechanical effects. Those effects are not negligible when simulating light particles, chemical reactions or low temperature systems (uncertainty principle and tunnel effect).
Since for more many-body systems no analytical solution for the classical equations of motion is tractable, other procedures must be introduced. The most intuitive and simple one is to go from continuous time to discrete time. This means, that at every time step, all the forces in a system are calculated. Employing an integrator (e.g. the Verlet-integrator [30]) the equations of motion are solved numerically and particle positions and velocities are advanced accordingly. To simulate the canonical ensemble, one introduces a thermostat for coupling the system to a heat bath. Methods simulating this coupled heat bath differ in accuracy and computational cost. Langevin dynamics for example introduce random collisions following some distribution mimicking the effects of solvent particles at a set temperature [1].
The time step Δt needs to be small enough to keep up with the fastest movements of particles, usually $\Delta t = 1 - 2\mathrm{fs}$. For example, when a particle is placed in a harmonic potential $U(x) = \frac{1}{2}kx^2$, its energy is given by $E(x,v) = \frac{1}{2}kx^2 + \frac{1}{2}mv^2$. Here k is a constant, m is the mass, x is the position, and v is the velocity. The maximal velocity therefore is $\sqrt{\frac{2E}{m}}$. Now the maximal distance Δx is $\sqrt{\frac{2E}{k}}$, meaning that Δt must be smaller than $\sqrt{\frac{m}{k}}$. Hence for lighter particles and steeper potentials, the time steps must be selected smaller.
In the limit of infinite time steps, the number of times a certain state is visited divided by the total step number should reflect the probability of a state from which the correct PDF can be calculated (eq. 2.7). Equivalent to the Monte Carlo algorithm, also the Molecular

Dynamics (MD) simulation algorithm needs to be ergodic to give the correct results. MD simulations also share the problem of correct sampling in the presence of potentials large compared to $k_{\mathrm{B}}T$ and the scaling with DOF with MC simulations [1].

2.2.3. Coarse-Graining of Molecules

To increase the possible integration timestep and to reduce the number of particles and thereby allow for a more efficient simulation, a popular method used to speed up the above-mentioned algorithms is coarse-graining the system [5]. One form of coarse-graining was already introduced in the previous paragraph, which is, modeling atom interactions using classical mechanics with effective interactions instead of a potential energy surface derived from quantum mechanics. Going one step further in the coarse-graining ladder, another widely used example is grouping multiple atoms of a molecular residue in a single particle parameterized using effective forces.
When simulating molecules, one usually replaces groups of atoms with interacting mass points [5]. If thereby the number of interacting mass points is reduced by a factor of N, the computation exhibits a computational speedup of the order of N^2. Also, since the fastest oscillating DOF are effectively integrated over, the simulation time steps can be enlarged to usually $\Delta t = 5 - 20$fs. The coarse-graining procedure consists of two steps. First, one maps atoms in a certain way to your new mass points, whereby the way depends on the physical properties under study. Then you introduce force fields coupling the mass points together.
The force fields can be derived either via a bottom-up or a top-down approach. For this thesis it suffices to look at the bottom-up approach: The parameters of the force fields are tuned, so the coarse-grained (CG) simulation reproduces the microscopic, statistical, and thermodynamic properties of an all-atom simulation or experimental data.

Coarse-grained potentials of mean force

To reproduce molecular behavior in CG approaches, force fields are assigned to beads representing groups of atoms. These new force fields lead to the beads experiencing the same forces, the group of atoms collectively would be exposed to on average. The interaction potentials, that together with bead parameters define the force fields generally consist of two terms, the bonded and non-bonded interaction potentials:

$$U_{total}^{CG}(\mathbf{x}) = \sum U_{bond}^{CG}(\mathbf{x}) + \sum U_{nb}^{CG}(\mathbf{x}) \tag{2.8}$$

For our purposes, it suffices to look into the bonded interactions, since in our test-system we simulate a single molecule and incorporate non-bonded interactions into the coarse-grained potential definition. Those are either represented by commonly used potentials like harmonic potentials with adjustable parameters or with potentials of mean force explained in the following. When looking at the conformational DOF distribution $P(r, \theta, \phi, T)$ derived from an all-atom simulation, where r are the distances, θ the angles, ϕ the dihedral angles and T the temperature, one can make the following independence

assumption, which depends heavily on mapping scheme and cannot be generalized:

$$P(r,\theta,\phi,T) = P(r,T)P(\theta,T)P(\phi,T) \tag{2.9}$$

To obtain effective potentials to parameterize the interaction between coarse-grained beads, these probability distributions can be Boltzmann-inverted and depict temperature-dependent bonded interaction potentials as potentials of mean force:

$$U_{bond}(r,T) = -k_{\mathrm{B}}T\log\left(\frac{P(r,T)}{r^2}\right) + c_r \tag{2.10}$$

$$U_{angle}(\theta,T) = -k_{\mathrm{B}}T\log\left(\frac{P(\theta,T)}{\sin(\theta)}\right) + c_\theta \tag{2.11}$$

$$U_{dihedral}(\phi,T) = -k_{\mathrm{B}}T\log\left(P(\phi,T)\right) + c_\phi \tag{2.12}$$

Here the c's are freely chosen constants unimportant for the physical behavior. The validity of the assumption (eq. 2.9) depends on the mapping scheme. The potentials are also only valid for one temperature. The name potential of mean force stems from the fact, that this approach is equivalent to when for example looking at the dihedral angle potential (eq. 2.12), setting the negative gradient of the potential at a certain value of ϕ_0 to the expectation value of the force at $\phi = \phi_0$ (eq. 2.13).

$$\nabla U_{dihedral}(\phi_0) = -\langle F(r,\theta,\phi)\rangle_{\phi=\phi_0,T} \tag{2.13}$$

2.3. Deep Neural Networks

Artificial neural networks consist of a set of nodes $N = \{u_1, u_2, ...\}$ connected via a set of directed connections or edges $H \subseteq N \times N$. Each node can experience an activation x_i either by external input (input layer) or by other nodes (hidden and output layers). In the second case, the activation is $x_i = f_{act}(y_i)$ with $y_i = \sum_{j\in N_i} x_j w_{j,i}$ (additive case). f_{act} is an activation function and $\{w_{j,i}\}_{j\times i\in H}$ are parameters or weights attributed to every edge and $N_i \subseteq N$ are all nodes having an edge pointing at node i and $\{x_j\}_{j\in N_i}$ are the activations of so connected nodes. The activation function is an easy to evaluate non-linear function that ensures that the overall transformation applied by the network is not linear. The sum can be replaced by a product (multiplicative case). One distinguishes between recurrent (RNN) [31] and feedforward neural networks (FNN)[32, 33]. FNNs are acylic, while RNNs are cyclic. This means, that for one input, the activation of some nodes in an RNN can change, and hence the output can change. FNNs have a time-independent output. There exist many more kinds of neural networks such as the convolutional neural network [34] better equipped for tasks like image recognition. More complex tasks call for more complex network architectures, often consisting of a combination of neural networks. There are different kinds of activation functions, the one used throughout this thesis and a generally very popular one due to its simplicity is the ReLU-function (short for the rectified linear unit), which is defined as $f(x) = x$ for $x > 0$ and $f(x) = 0$ otherwise. [35]

2.3.1. Feedforward Neural Networks (FNN)

FNNs can be applied for a variety of problems such as:
"*pattern recognition, clustering, and classification, function approximation, control, bioinformatics, signal processing, speech processing, etc.* "[33]
A basic scheme of an FNN and how outputs are generated can be seen in figure 2.1. An

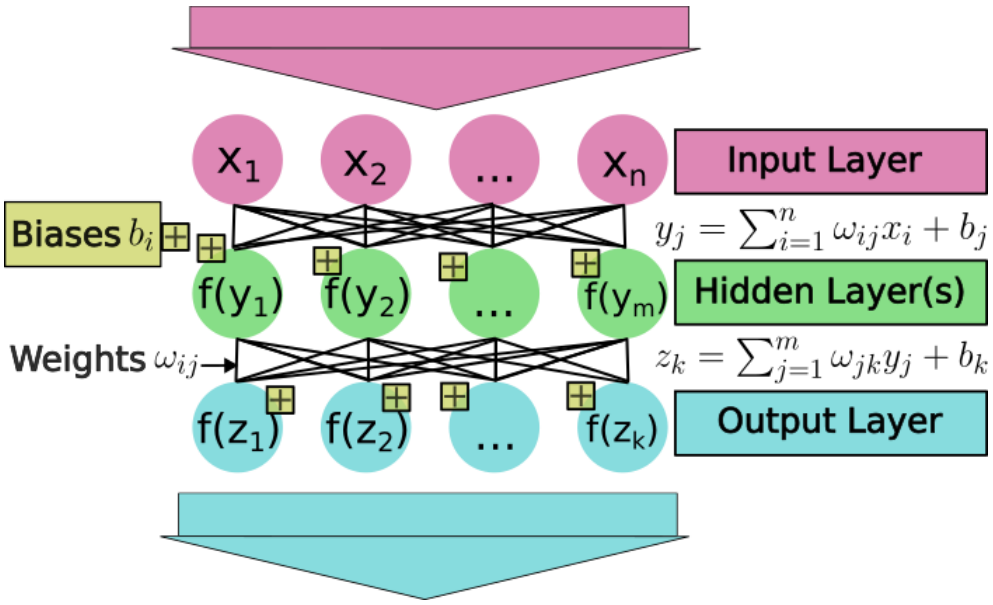

Figure 2.1.: Scheme of a feedforward neural network (FNN). An n component vector is used as input $\{x_i\}_{i\in[0,1,2,...,n]}$. With m biases, $n \cdot m$ weights and these inputs $\{y_j\}_{j\in[1,2,...,m]}$ are calculated. The activation function f is applied on those giving the values of the nodes in the first hidden layer. This procedure is repeated for all hidden layers and finally once more resulting in the output $\{z_i\}_{i\in[1,2,...,k]}$.

FNN consists of an input-, an output- and arbitrarily many hidden layers consisting of multiple nodes. In the fully connected case used in this work, each node from one layer is connected to each node of the next layer. Activations are passed forward as aforementioned, with the extension, that each node gets an additional bias parameter $b_j \in \mathbb{R}, j \in N_i$ leading to $y_k = b_j + \sum_{j\in N_i} x_j w_{j,k}$. Here N_i is the number of nodes in layer i. In this way, the input is mapped to a specific output depending on the weight and bias parameters, which are adjusted during the training process to obtain the desired output. To train a neural network, one first needs to define an error- or loss function, a function measuring how well the network is performing at its task. The worse the performance, the larger the loss function. During training, it is derived with respect to all network parameters in a process called backpropagation. A stochastic gradient descent method is then used to alter network parameters until a convergence of the loss function is evident. Among others, [13] one widely used algorithm is the ADAM algorithm [36]. As a first

and second-order momentum-based optimizer with a stochastic component, the algorithm avoids getting stuck in suboptimal local minima. It should be mentioned, that there are alternatives to backpropagation-based optimizations like evolutionary algorithms [37] and Bayesian optimization [38].
One distinguishes between supervised and unsupervised training. Unsupervised networks [39] learn on unlabelled training data and are used for e.g. data clustering, associations between data characteristics, dimensionality reduction, and anomaly detection. To conduct supervised training, labeled train data is necessary. The loss function here measures the difference between a label assigned to train data and the label given to the train data by the network. An example of a loss function in the supervised case is the Euclidean distance between labels and network-given values of output nodes.

2.4. Generative Models

In the book [18] David Foster defines generative models as:
"Generative modeling is a branch of machine learning that involves training a model to produce new data that is similar to a given dataset."
They are probabilistic models, that generate data either autoregressively or by transformation of a latent distribution such as Gaussian noise. When compared to discriminative models, deep learning models trained to distinguish data features, the data generating task is more complex. For a discriminative task, models learn $p(\mathbf{y}|\mathbf{x})$, whereby $\mathbf{y}$ is a low-dimensional set of features and $\mathbf{x}$ is a vector in a higher dimensional data space. The data space might for example consist of 64×64 pixels, and, therefore, have 4096 dimensions, while the feature space in a simple example might have just one dimension discrete dimension $\mathbb{I} = [0, 1, ..., 9]$, if the task is to recognize digits. Compared to that, generative models have to learn $p(\mathbf{x})$ or $p(\mathbf{x}|\mathbf{y})$. So while discriminative models learn a low-dimensional probability distribution depending on many variables, generative models represent a high-dimensional probability distribution, that might depend on fewer variables. This is in general a much more complex task, David Foster [18] explains this by claiming that discriminative models only have to learn some correlations within the train data, while generative models need to learn all. Nonetheless, generative models have been among others successfully used for image generation [40, 41], text generation [42, 20], music generation [43], voice generation [44] and exploring macromolecular structures [45].
According to [18] there are at the moment six basic kinds of generative models.

- **Variational Autoencoders:** They consist of an encoder and a decoder network. The encoder maps data to a latent space representation, that has fewer dimensions and is in many cases Gauss distributed. It should capture data-defining features. The decoder Network generates data from latent space vectors. The network is trained in two ways. It maximizes the similarity between generated data and data mapped from/to the same latent space vector and it increases the encoded train data distribution similarity to a Gaussian distribution. [46, 47, 48]

- **Generative Adversarial Networks:** They consist of two networks: a generator and a discriminator. While the discriminator learns to separate train data from fake data generated by the generator, the generator learns to trick the discriminator. The generator makes data from noise. [49, 50, 51, 52]

- **Autoregressive Models:**

  They sequentially produce output one step at a time using conditional probabilities, whereby for train data, the log-likelihood gets maximized: $-\log p(\mathbf{x}) = -\sum_i^n \log p(x_i|x_1, ..., x_{i-1})$ [53, 54, 20]

- **Energy-Based Models:** Energy-Based models, e.g. Boltzmann machines employ energy depending on network state and parameters. This energy first is made small for train data by adjusting parameters. Then, when generating data, the parameters stay fixed and new network states are explored using a Monte Carlo algorithm. [55, 56]

- **Diffusion Models:**

  The same network repeatedly adds noise to data until the transformed data is Gaussian distributed. The inverse process then generates data by denoising samples drawn from this normal distribution. The denoising process is trained by increasing the similarity of generated data to train data. [57, 58, 40]

- **Normalizing Flows:**

  Only using invertible transformations, normalizing flow models transform samples drawn from some probability distribution to data that looks similar to train data. The learning process maximizes the log-likelihood, to which an evidence lower bound consisting of the Kullback-Leibler (KL) divergence and the log-determinant of the Jacobian of the transformation. The invertibility and exact probability density are two advantages of this model making it especially useful for physical problems [59, 60, 22].

2.5. Normalizing Flows

The idea of normalizing flows as first introduced in [21] is the following. To get a good density estimation, build a map that transforms normal (prior) distributions to a more complex distribution. Adjust the parameter-dependent map, so the observed distributions and the posterior (transformed prior) distribution's similarity converges. To overcome the limitations of simple maps, put N parametric, invertible, and differentiable maps in series, the transformation can then become arbitrarily complex (eq. 2.14):

$$\mathbf{z}_N = f_{\theta_N} \circ f_{\theta_{N-1}} \circ ... \circ f_{\theta_1}(\mathbf{z}_0) \tag{2.14}$$

Here $\theta = (\theta_1, \theta_2, ...\theta_N)$ are parameters of the maps f. The probability density of a sample can then be evaluated by transforming it back to the prior normal distribution and

computing the product of the back-transformed sample under the normal distribution and the transformation-associated change of volume. While in [59] further ground work such as introducing some transformations has been done, the next big step in the development of normalizing flows was combining them with variational inference [60], using those networks to model probability distributions from samples. The advantage of using normalizing flows is that they are capable of not only producing samples but also computing their probability density.
Consider $p_\theta(\mathbf{x})$ the marginal likelihood of the observations $\mathbf{x}$ using a probabilistic model depending on the parameters θ and latent variables $\mathbf{z}$ over which we need to integrate:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} \tag{2.15}$$

In order to use variational inference, an approximate posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$, where ϕ are the parameters of this variational distribution, is introduced. The goal is to minimize the KL divergence $\mathbb{D}_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$ between the approximate and the true posterior distribution $p(\mathbf{z}|\mathbf{x})$ which is equivalent to maximizing the following approximation:

$$\log p_\theta(\mathbf{x}) = \log \int \frac{q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} \leq -\mathbb{D}_{KL}[q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})] + \mathbb{E}_q[\log p_\theta(\mathbf{x}|\mathbf{z})] = -\mathcal{F} \tag{2.16}$$

$\mathcal{F}$ henceforth serves as evidence of lower bound (ELBO) to the negative logarithm of the marginal likelihood. This ELBO is used to optimize a normalizing flow consisting of sequentially connected smooth mappings. When looking at an invertible smooth mappings $f : \mathbb{R}^d \to \mathbb{R}^d$ with $\mathbf{z'} = f(\mathbf{z})$ transforming the distribution $q(\mathbf{z})$ one gets:

$$q_{\mathbf{z}'}(\mathbf{z}') = q_{\mathbf{z}}(\mathbf{z}) \left| \det \frac{\partial f^{-1}}{\partial \mathbf{z}} \right| = q_{\mathbf{z}}(\mathbf{z}) \left| \det \frac{\partial f}{\partial \mathbf{z}} \right|^{-1} \tag{2.17}$$

When combining a series of maps as in equation 2.14 one arrives at equation 2.18:

$$\log q_N(\mathbf{z}_N) = \log q_0(\mathbf{z}_0) - \sum_{n=1}^{N} \log \left| \det \frac{\partial f_n}{\partial \mathbf{z}_{n-1}} \right| \tag{2.18}$$

The determinants of the Jacobians account for the aforementioned volume changes. The approximate posterior distribution is parameterized as a flow of length N $q_\phi(\mathbf{z}|\mathbf{x}) = q_N(\mathbf{z}_N)$, the law of the unconscious statistician (LOTUS) then can be stated as:

$$\mathbb{E}_{q_N}[h(\mathbf{z})] = \mathbb{E}_{q_0}[h(f_N \circ f_{N-1} \circ ... \circ f_1(\mathbf{z}_0))] \tag{2.19}$$

The free energy from equation 2.16 can be rewritten using equation 2.18 and equation 2.19 :

$$\begin{aligned} \mathcal{F} &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{x}, \mathbf{z})] = \mathbb{E}_{q_0(\mathbf{z}_0)}[\log q_N(\mathbf{z}_N) - \log p(\mathbf{x}, \mathbf{z}_N)] \\ &= \mathbb{E}_{q_0(\mathbf{z}_0)}[\log q_0(\mathbf{z}_0)] - \mathbb{E}_{q_0(\mathbf{z}_0)}[\log p(\mathbf{x}, \mathbf{z}_N)] - \sum_{n=1}^{N} \log \left| \det \frac{\partial f_n}{\partial \mathbf{z}_{n-1}} \right| \end{aligned} \tag{2.20}$$

In figure 2.2 a normalizing flow model learning in two directions, where $q_0(\mathbf{z}_0)$ is not parameterized, is depicted.

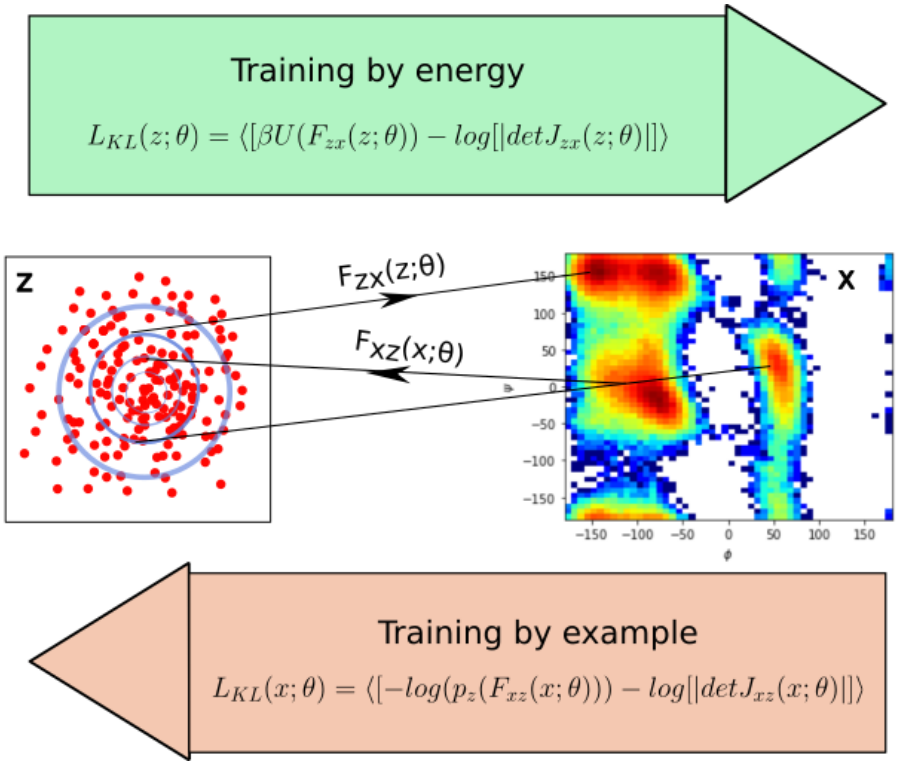

Figure 2.2.: Scheme of a normalizing flow. A bijective function F_{zx} maps every point z of an easy-to-sample from probability distribution p_z to a point x in some physical configuration space. The function can be altered by changing the components of the parameter vector θ. θ is optimised by minimising the Kullback-Leibler losses L_{KL}. In the formulas $\beta = \frac{1}{k_B T}$, U is the potential energy function of the physical system and $J_{zx} = J_{xz}^{-1}$ are the Jacobian matrices of transformations from $\{z\}$ to the configuration space.

Since their introduction different kinds of normalizing flows were introduced, an overview over which can be found in [61].

2.5.1. Normalizing Flow architectures

According to Ivan Kobyzev et al. [61], there are seven fundamentally different normalizing flow architectures:

- Elementwise bijections: Each element in a vector is transformed by an invertible scalar function $h : \mathbb{R} \to \mathbb{R}$. There is no mixing of variables [62, 63].
- Linear flows: Here $\mathbf{x}$ is transformed by the linear map $g(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$. The problem

with these transformations is their lacking expressiveness, e.g. a Gaussian under these transformations still is a Gaussian [21, 60, 64].

- Planar and radial flows: They are not widely used, since their inverse is difficult to compute. Planar flows use the transformation $g(\mathbf{x}) = \mathbf{x} + \mathbf{u}h(\mathbf{w}^T\mathbf{x} + b)$, where $\mathbf{u}, \mathbf{w} \in \mathbb{R}^D$, $b \in \mathbb{R}$ and $h : \mathbb{R} \to \mathbb{R}$ a smooth non-linearity. Radial flows use $g(\mathbf{x}) = \mathbf{x} + \frac{\beta}{\alpha+||\mathbf{x}-\mathbf{x}_0||}(\mathbf{x} - \mathbf{x}_0)$ with $\alpha, \beta \in \mathbb{R}$, $\alpha > 0$ [60].

- Coupling flows: An input vector $\mathbf{x}$ $in\mathbb{R}^D$ is partitioned into two vectors $\mathbf{x}^A \in \mathbb{R}^{D-d}$ and $\mathbf{x}^B \in \mathbb{R}^d$ transformed in the following way:

$$\begin{aligned} \mathbf{y}^A &= h(\mathbf{x}^A; \Theta(\mathbf{x}^B)) \\ \mathbf{y}^B &= \mathbf{x}^B \end{aligned} \tag{2.21}$$

  $h(\cdot; \theta) : \mathbb{R}^{D-d} \to \mathbb{R}^{D-d}$ is a bijective function parameterized by θ and Θ any function solely depending on $\mathbf{x}^B$. $\mathbf{y}^B$ is then transformed by the next map in the same fashion [65].

  This thesis deals with networks with two different coupling functions, which are discussed in more detail in sections 2.5.2 and 2.5.3.

- Autoregressive flows: They use scalar bijective transformations $h(\cdot; \theta) : \mathbb{R} \to \mathbb{R}$ parameterized by θ to transform each component with parameters depending on previous input components:

$$y_t = h(x_t; \Theta_t(x_{1:t-1})) \tag{2.22}$$

  The inverse calculation of these transformations is inherently sequential. Therefore, contrary to coupling flows, the universality property has been proven [66, 67].

- Residual flows: Residual Networks use maps of the form:

$$g(\mathbf{x}) = \mathbf{x} + F(\mathbf{x}) \tag{2.23}$$

  where $F(\cdot)$ is an FNN of any kind. Computing the inverse and the Jacobian can pose problems and be computationally expensive respectively for such networks [68, 69, 70].

- Infinitesimal flows: They either use ordinary differential equations (ODE) or stochastic differential equations (SDE). Other normalizing flows can be seen as a discretization of a first-order ODE:

$$\frac{d}{dt}\mathbf{x}(t) = F(\mathbf{x}(t), \theta(t)) \tag{2.24}$$

  $F : \mathbb{R}^D \times \Theta \to \mathbb{R}^D$ is the dynamics defining evolution function and $\theta : \mathbb{R} \to \Theta$ a parameterization. SDE basically adds Brownian motion to an ordinary ODE. Using the Kolmogorov and Kolmogorov backward equations, one can propagate a probability density in two directions[71, 72, 60].

2.5.2. Real-NVP Transformation Networks

In [65] Dinh and coworkers introduced coupling flows with real-valued non-volume preserving (real NVP) transformations tackling the problem of high-dimensional and highly structured probabilistic data generation claiming their method is exact and efficient due to a straightforward evaluation of the determinant of the Jacobian. First, an input vector $\mathbf{x}$ is partitioned into two vectors $\mathbf{x}^A$ and $\mathbf{x}^B$ of dimensions a and b with $a + b = D$. Now an affine coupling layer is used as a simple and tractable bijection $g^A : \mathbb{R}^D \to \mathbb{R}^D$:

$$\begin{aligned} \mathbf{y}^A &= \mathbf{x}^A \\ \mathbf{y}^B &= \mathbf{x}^B \odot \exp\big(s(\mathbf{x}^A)\big) + t(\mathbf{x}^A) \end{aligned} \tag{2.25}$$

$\odot$ denotes an element wise product, $s, t : \mathbb{R}^a \to \mathbb{R}^b$ functions. If as an intermediate step we define the vector $\mathbf{x}_{\text{arr}} = (\mathbf{x}_1^A, \mathbf{x}_2^A, ..., \mathbf{x}_a^A, \mathbf{x}_1^B, ...\mathbf{x}_b^B)$ and $\mathbf{y}_{\text{arr}}$ equivalently, the Jacobian of this transformation $\frac{d\mathbf{y}_{\text{arr}}}{d\mathbf{x}_{\text{arr}}^T}$ is triangular:

$$\frac{d\mathbf{y}_{\text{arr}}}{d\mathbf{x}_{\text{arr}}^T} = \begin{bmatrix} \mathbb{I}_a & 0 \\ \frac{\partial \mathbf{y}_{\text{arr},a+1:D}}{\partial \mathbf{x}_{\text{arr},1:a}^T} & \text{diag}\left(\exp[s(\mathbf{x}_{\text{arr},1:a})]\right) \end{bmatrix} \tag{2.26}$$

The determinant of this transformation can therefore be calculated efficiently as $\sum_{j=1}^{a} \exp\big[s(\mathbf{x}^A)_j\big]$. Since the calculation of a Jacobian of either s or t is not needed, those functions can be arbitrarily complex. Dinh et al. [65] used deep convolutional networks, however, the functions can likewise be FNNs without any need to change the calculation of the determinant. The transformation is then repeated switching the arguments, meaning $g^B : \mathbb{R}^D \to \mathbb{R}^D$ is defined equivalent to g^A but with A and B swapped, to transform all vector components. The determinant of $g(\mathbf{x}) = g^B \circ g^A(\mathbf{x})$ is $\det\big[g^B \circ g^A(\mathbf{x})\big] = \det\big[g^B(\mathbf{x}_b = g^A(\mathbf{x}))\big] \cdot \det\big[g^A(\mathbf{x})\big]$
The inverse of g is $g^{-1} = g^{A,-1} \circ g^{B,-1}$ and $g^{A,-1} : \mathbb{R}^D \to \mathbb{R}^D$

$$\begin{aligned} \mathbf{x}^A &= \mathbf{y}^A \\ \mathbf{x}^B &= (\mathbf{y}^B - t(\mathbf{y}^A)) \odot \exp\big(-s(\mathbf{y}^A)\big) \end{aligned} \tag{2.27}$$

The partitioning of the vectors is done either by splitting the input vector or by using a mask. Relevant for us and used in [65] is the binary splitting $\mathbf{b}$, which has values 0, if the (sum of the) coordinate(s) is even and 1 if it is odd and is of the same dimensionality as $\mathbf{x}$. In the case of splitting, g^A can be rewritten as:

$$\mathbf{y} = \mathbf{b} \odot \mathbf{x} + (1 - \mathbf{b}) \odot (\exp(s(\mathbf{b} \odot \mathbf{x})) + t(\mathbf{b} \odot \mathbf{x})) \tag{2.28}$$

The authors of [65] used a multiscale architecture inspired by [73] to save computational and memory costs and trainable parameters for their image generation network. That means, that first a small picture is produced from some Gaussians via some coupling layers. Then those small pictures and some more Gaussians are transformed into a larger picture with more pixels, again with some coupling layers. This is then repeated once again. They also used different masks at different scales in order to not get redundant partitioning.

2.5.3. Neural Spline Flows

A more flexible flow defined on bounded intervals instead of $\mathbb{R}$ like affine transformations is the so-called spline flow. Coupling functions need to fulfill two criteria: They need to be invertible and both, the function and its inverse need to be differentiable. Durkan et al. [74] have introduced monotonic rational-quadratic transforms, functions fulfilling the criteria mentioned above. They are piecewise defined and therefore depend on more parameters than affine coupling functions. They transform values from some bound interval $[-B, B]$ to values within the same interval.
One defines this function the following way: Introduce $K > 2$ knots with coordinates $\{(x^{(k)}, y^{(k)})\}_{k=0}^{K}$. They must increase strictly monotonically between $(x^{(0)}, y^{(0)}) = (-B, -B)$ and $(x^{(K)}, y^{(K)}) = (B, B)$. At each knot, the spline has an arbitrary positive derivative with values $\{\delta^{(k)}\}_{k=1}^{K-1}$, just at the borders $\delta^{(0)} = \delta^{(K)} = 1$ in order to match linear tails.
The transformation introduced first in reference [75] is defined as follows in the k^{th} bin:

$$\frac{\alpha^{(k)}(\xi)}{\beta^{(k)}(\xi)} = y^{(k)} + \frac{(y^{(k+1)} - y^{(k)})[s^{(k)}\xi^2 + \delta^{(k)}\xi(1-\xi)]}{s^{(k)} + [\delta^{(k+1)} + \delta^{(k)} - 2s^{(k)}]\xi(1-\xi)} \tag{2.29}$$

Here, $s^{(k)} = (y^{k+1} - y^k)/(x^{k+1} - x^k)$ and $\xi^{(k)} = (x - x^k)/(x^{k+1} - x^k)$. The expressiveness of the spline transformation comes at the cost of a more complicated expression for the inverse and derivative. Therefore, we refer the reader to [74] for those.
Since there are $K + 1$ knots, there are K bins with a height and a width and $K - 1$ knots with adjustable derivatives, amounting to a total of $3K - 1$ adjustable parameters. The width and height parameters are first passed through a soft-max function respectively to ensure a normalized width and height strictly greater than 0 and then multiplied by $2B$, thereby assuring the sum of widths and heights to be $2B$. The parameters defining the derivatives are just passed through a soft plus function assuring positivity.
These transformations are more flexible than affine transformations [61], which leads to a smaller number of coupling layers necessary for representing distributions with the same level of complexity.

2.5.4. Boltzmann generators

Boltzmann generators (BGs), as first described by Noe and coworkers [22], use so-called mixed transformations and affine coupling normalizing flows to generate configurations of molecular systems according to their statistical weight in the equilibrium distribution. Using the benefits of normalizing flows, BGs can be trained in both directions, meaning they not only learn to generate data similar to train data, but they also learn to minimize the relative entropy between the generated distribution and the reference Boltzmann distribution. The basic structure of a BG can be seen in figure 2.3.
Following the derivations by [22], we introduce a physical system with D degrees of freedom. Each configuration is described by a D-dimensional vector $\mathbf{x}$. The function $U : \mathbb{R}^D \to \mathbb{R}$ maps every configuration to its corresponding energy. As mentioned in section 2.1, the

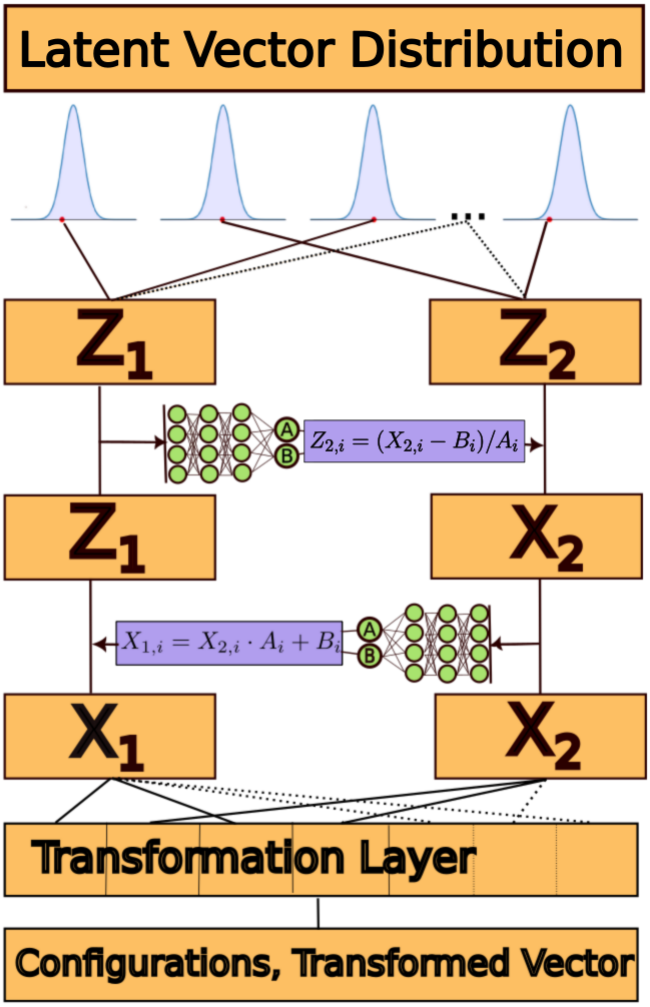

Figure 2.3.: Scheme of a Boltzmann generator. From a simple latent vector distribution, uncorrelated samples are drawn. The block also provides probabilities for each vector. The uneven components become part of the vector Z_1, the even ones part of Z_2. Z_1 is used as input values of a FNN giving the vectors **A** and **B**. Those then transform Z_2 to X_2. The same procedure gives values transforming Z_1 to X_1. This block is repeated n times. The final X_1 and X_2 are recombined to a vector fed to a mixed transformation layer, an invertible function transforming configurations into more well-behaved distributions and vice versa. The bottom box provides configurations from some other simulation and maps the energy to every vector.

probability density of each configuration is given by $p_X(\mathbf{x}) = \exp(-\beta U(\mathbf{x})/Z)$ with $\beta = 1/k_\mathrm{B}T$ and the partition function Z.
Let $\{p_Z(\mathbf{z})\}_{\mathbf{z}\in\mathbb{R}^D}$ be an easy-to-sample-from latent probability distribution, e.g. a multivariate Gaussian distribution and $F_{zx} : \mathbb{R}^D \to \mathbb{R}^D, \mathbf{z} \mapsto \mathbf{x}$ a bijective map. Then we define the distributions q_X and q_Z as a transformation of the Boltzmann and the latent distribution respectively:

$$\begin{aligned} p_X(\mathbf{x}) &\xrightarrow{F_{xz}} q_Z(\mathbf{z}) \\ p_Z(\mathbf{z}) &\xrightarrow{F_{zx}} q_X(\mathbf{x}) \end{aligned} \tag{2.30}$$

Here $F_{xz} = F_{zx}^{-1}$. Also let $u(\mathbf{x}) = \frac{U(\mathbf{x})}{k_\mathrm{B}T}$ be the reduced energy and $R_{zx}(\mathbf{x}) = |\det \mathbf{J}_{zx}(\mathbf{x})|$ be the determinant of the Jacobian of the transformation F_{zx}. R_{xz} is defined equivalently. The training goal is now to increase the overlap between $q_X(\mathbf{x})$ and $p_X(\mathbf{x})$ as much as possible. This is achieved by minimizing the Kullback-Leibler (KL) divergence, the information gained when sampling from one distribution when expecting another. It is 0 when the distributions are identical and becomes increasingly larger with increasing difference between the distributions reaching a maximum when there is no overlap. It is defined as:

$$\mathrm{KL}(q||p) = \int q(\mathbf{x})[\log q(\mathbf{x}) - \log p(\mathbf{x})]\mathrm{d}\mathbf{x} = -H_q - \int q(\mathbf{x}) \log p(\mathbf{x})\mathrm{d}\mathbf{x} \tag{2.31}$$

where H_q is the entropy of the distribution q. We can define a KL divergence with respect to the latent space and the configuration space since the KL-divergence is not symmetric. To derive an expression for the training loss, in latent space, one can perform the following operations when using equation 2.17 and then inserting for p_X:

$$\begin{aligned} \mathrm{KL}[p_Z||q_Z] =& - H_{\mathbf{z}\sim p_Z(\mathbf{z})} - \int p_Z(\mathbf{z}) \log q_Z(\mathbf{z})\mathrm{d}\mathbf{z} \\ =& - H_{\mathbf{z}\sim p_Z(\mathbf{z})} - \int p_Z(\mathbf{z})[\log p_X(F_{zx}(\mathbf{z})) + \log R_{zx}(\mathbf{z})]\mathrm{d}\mathbf{z} \\ =& - H_{\mathbf{z}\sim p_Z(\mathbf{z})} + \log\big(\mathrm{Z}_{\mathbf{x}\sim p_x(\mathbf{x})}\big) + \mathbb{E}_{\mathbf{z}\sim p_Z(\mathbf{z})}[u(F_{zx}(\mathbf{z})) - \log R_{zx}(\mathbf{z})] \end{aligned} \tag{2.32}$$

Only the mapping F_{zx} and thereby also R_{zx} depend on the network parameters, meaning that $H_{\mathbf{z}\sim p_Z(\mathbf{z})}$ and the partition function $\mathrm{Z}_{\mathbf{x}\sim p_x(\mathbf{x})}$ (eq. 2.1) are constants. Therefore, the resulting loss function is:

$$J_\mathrm{KL} = \mathbb{E}_{\mathbf{z}\sim p_Z(\mathbf{z})}[u(F_{zx}(\mathbf{z})) - \log R_{zx}(\mathbf{z})] = U - H_{\mathbf{x}\sim q(\mathbf{x})} + H_{\mathbf{z}\sim p(\mathbf{z})} \tag{2.33}$$

whereby U stands for the potential energy and H for the Entropy. Here the last equivalence holds because $H_{\mathbf{x}\sim q(\mathbf{x})} - H_{\mathbf{z}\sim z(\mathbf{z})} = \mathbb{E}_{\mathbf{z}\sim p_Z(\mathbf{z})}[\log R_{zx}(\mathbf{z})]$ shown in the supplementary material of [22]. The last form of equation 2.33 is equal to the free energy $F = U - H_{\mathbf{x}\sim q(\mathbf{x})}$ of the transformed latent vector in configuration space up to the constant $H_{\mathbf{z}\sim z(\mathbf{z})}$
Similarly for the KL divergence in configuration space one gets:

$$\mathrm{KL}[p_X||q_X] = H_{\mathbf{x}\sim p(\mathbf{x})} + \log\big(\mathrm{Z}_{\mathbf{z}\sim p_z(\mathbf{z})}\big) + \mathbb{E}_{\mathbf{x}\sim p_X(\mathbf{x})}[-\log p_Z(F_{xz}(\mathbf{x})) - \log R_{xz}(\mathbf{x})] \tag{2.34}$$

Again only the last term can be minimized and rewritten as the loss:

$$\begin{aligned} J_{\mathrm{ML}} &= \mathbb{E}_{\mathbf{x}\sim p_X(\mathbf{x})}[-p_Z(F_{xz}(\mathbf{z})) - \log R_{xz}(\mathbf{z})] \\ &= -\mathbb{E}_{\mathbf{x}\sim p_X(\mathbf{x})}[p_Z(F_{xz}(\mathbf{z}))] - H_{\mathbf{z}\sim q(\mathbf{z})} + H_{\mathbf{x}\sim p(\mathbf{x})} \end{aligned} \tag{2.35}$$

The Boltzmann generator is first trained on configurations from another sampling method using J_{ML} and the gradient descent method to adjust the parameters in the FNNs defining the parameters of the coupling transformations. After some training, data in latent space are generated and the network begins to also learn on J_{KL}:

$$J_{\mathrm{total}} = w_{\mathrm{ML}} J_{\mathrm{ML}} + w_{\mathrm{KL}} J_{\mathrm{KL}} \tag{2.36}$$

Here w_{ML} and w_{KL} are weights, determining the influence of example-, and energy based loss on the training. Usually, w_{KL} is increased during training.

Reweighting and Resampling

Given a convergence of the loss function indicating succesfull training, the network can generate uncorrelated configurations by sampling points in latent space and then transforming them to configuration space. This transformation is not perfect and the generated samples do usually not exactly follow Boltzmann statistics. However, since we know the exact probability that a configuration is generated $q(\mathbf{x}) = p_Z(\mathbf{z})R_{zx}(\mathbf{z})$ and since we know that $p_X(\mathbf{x}) \propto \exp(-u(\mathbf{x}))$, we can map a weight ω up to a factor to every produced configuration:

$$\omega_X(\mathbf{x}) = \frac{p_X(\mathbf{x})}{q_X(\mathbf{x})} = \frac{q_Z(\mathbf{z})}{p_Z(\mathbf{z})} \propto \exp\left[-u_X(F_{zx}(\mathbf{z})) - \log p_Z(\mathbf{z}) + \log R_{zx}(\mathbf{z})\right] \tag{2.37}$$

In figure 2.4 one can see examples of energies and weights corresponding to configurations produced by a BG.
The weights indicate the degree of over- or underrepresentation of configuration in the generated distribution compared to the reference distribution. With the help of those weights, equilibrium expectation values can be computed as:

$$\mathbb{E}[O] \approx \frac{\sum_{i=1}^{N} \omega_X(\mathbf{x}) O(\mathbf{x})}{\sum_{i=1}^{N} \omega_X(\mathbf{x})} \tag{2.38}$$

The mixed transformation layer used to transform coordinates into a network-friendly representation introduced in [22] is also used in this thesis and is therefore discussed in section 3.2.3. The research into BGs for sampling from the equilibrium distribution of physical systems gained traction fast after the fundamental work by Noé and coworkers [22]. Examples include works by Wirnsberger et al [23] which used RQ splines to model atomic solids, and Dibak et al. [24] which introduced a BG producing configurations at different temperatures. Additionally, liquids were modeled by [27] and transition paths were sampled by [26] using BGs.

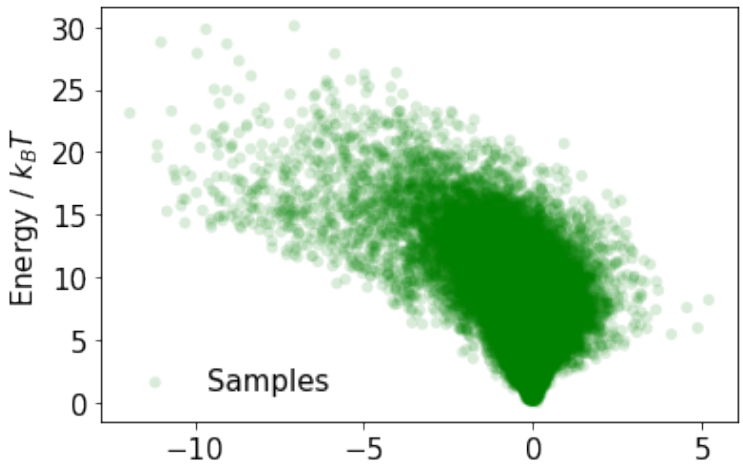

Figure 2.4.: The weights (in log scale) and energies of configurations produced by a BG.

3. Results

3.1. XY-Spinchain

Boltzmann generators are a promising new development for the sampling of many-body systems in the canonical ensemble. However, previous works [22] worked on small scale systems due to the suboptimal scaling of the BGs performance with the number of degrees of freedom. Therefore, in this thesis, we propose supporting the BG by adjusting the Prior distribution to better fit the configuration distribution and by that making the transformation less complicated. We first study this approach using an XY-model [76] and supporting the BG with a Gaussian mixture model used for learning in ref. [77].
We use a one-dimensional, only nearest neighbor interaction, XY-spin chain with N spins, where each spin is represented by one rotational degree of freedom (fig. 3.1).

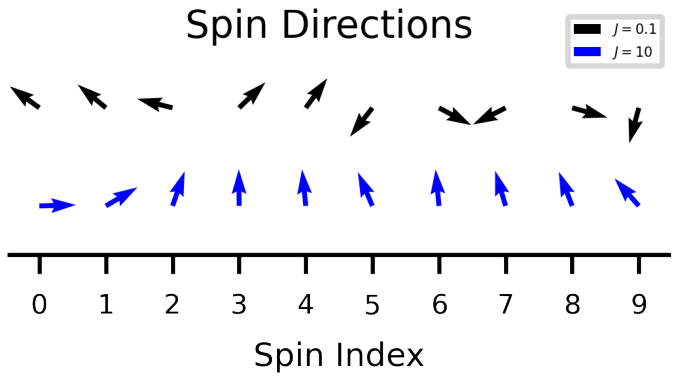

Figure 3.1.: Depiction of two configurations of the XY-model with different interaction strengths.

Since we do not include any external magnetic field we here define the Hamiltonian as:

$$H = \sum_{\langle i,j \rangle} -J\mathbf{s}_i\mathbf{s}_j = \sum_{k=1}^{N-1} -J\cos(\theta_k) \tag{3.1}$$

where $\langle i, j \rangle$ denotes all pairs of neighboring spins, J is the coupling strength, $\{\mathbf{s}_i\}_{1\in\{1,2,\dots N\}}$ are the spins with $|\mathbf{s}_i| = 1$ and $\{\theta_k\}_{k\in\{1,2,\dots N-1\}}$ are the angles between those spins. The boundaries are open, which means that the spins at the edge have only one neighbor.

Since this system is analytically tractable the partition function Z is:

$$Z = (2\pi)^N (I_0(\beta J))^{N-1} \tag{3.2}$$

where I_0 is the modified special Bessel function of the first kind and $\beta = \frac{1}{k_B T}$. To compare the BG-generated and reference distributions, we compare ensemble averages of observables. One is the correlation function between two spins i and j, which is given by:

$$\langle \mathbf{s}_i \mathbf{s}_j \rangle = \mu(|i-j|, \beta J) = \left(\frac{I_1(\beta J)}{I_0(\beta J)} \right)^{|i-j|} \tag{3.3}$$

Additionally, we include the heat capacity in the analysis, which is, in the thermodynamic limit of $N \lim_{N\to\infty}$ the heat capacity c per spin, defined as:

$$\frac{c}{k_B} = (\beta J)^2 \left(1 - \frac{\mu(1, \beta J)}{\beta J} - \mu(1, \beta J)^2 \right) \tag{3.4}$$

The above-defined quantities are illustrated in figure 3.2.

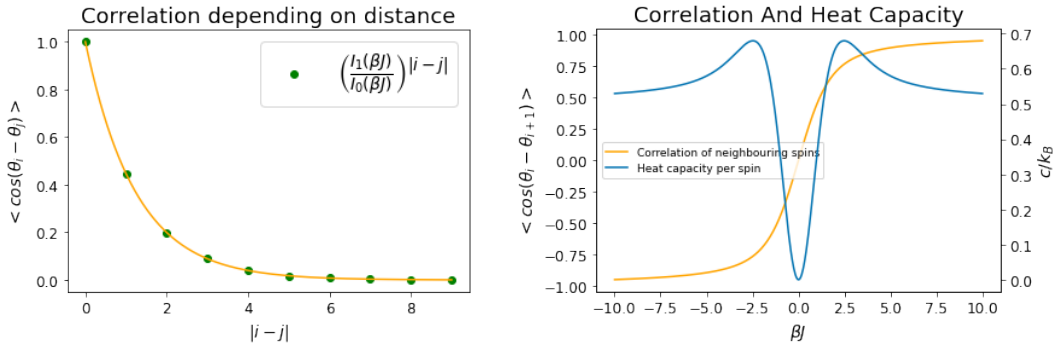

Figure 3.2.: Left: Exponentially decreasing correlation of direction of arrows with $\beta J = 1$, formula inset. Right: Correlation of neighboring arrows and heat capacity per site in the infinite arrows limit both plotted against the interaction strength divided by $k_B T = \frac{1}{\beta}$.

3.1.1. MC-Simulation for the Generation of Training Data

For training the BG by example, configurations of the system following the Boltzmann distribution are required. These are used in the initial stages of the training to shape the latent space distribution by example. To generate these configurations, we used a Markov chain Monte Carlo simulation (details in Appendix A.1.1). The resulting training data can be seen in figure 3.3.
As expected, the distributions of the spins without the removal of one rotational symmetry look close to uniform. It is important to note that the round ends of the violin plot

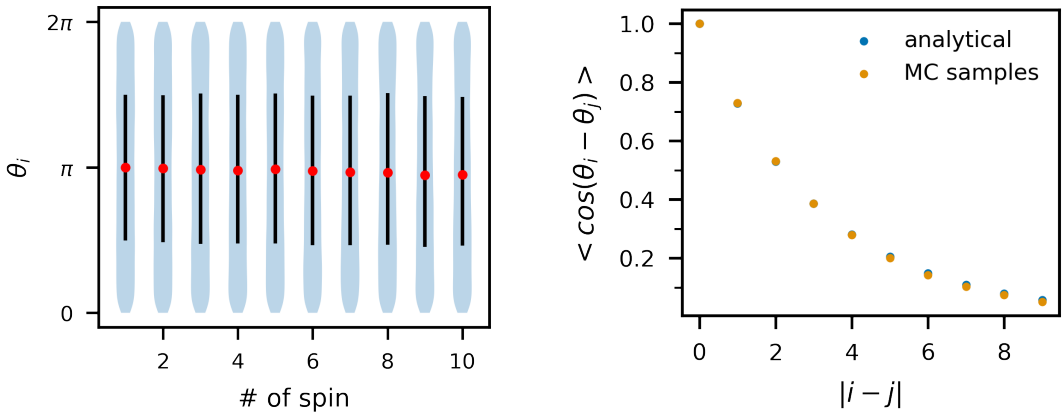

Figure 3.3.: Left: Violin plot of distributions of spin angles θ. Right: Correlation between spin pairs at distance $|i-j|$, analytically calculated and estimated from samples. Samples for plots were generated with an MC-simulation.

distributions are due to the use of Gaussian kernels instead of uniform bins for density estimation. The analytical and the sample-derived correlations can hardly be distinguished, indicating that the sampling time for the training data is sufficient. Also, the analytical average energy and heat capacity lie within the error of the average energies and heat capacities estimated based on the generated configurations. Hence it is safe to claim that the MC-simulation generates configurations representing the Boltzmann distribution.

3.1.2. Gaussian Mixture Model

Boltzmann generators transform a latent space sample of dimensionality $\mathbb{R}^d$ to a configuration of the same dimensionality, therefore, the scaling to a high number of DOFs is problematic since, in comparison to VAEs, no reduction in degrees of freedom is possible. Previous works on Boltzmann generators usually aimed to transform a prior uniform or Gaussian distribution to the Boltzmann distribution [24, 22]. To increase the performance of a BG, we attempt to employ prior distributions that are already closer to the desired Boltzmann distribution. In the case of the XY-model we used Gaussian mixture models (GMM).
A GMM consists of n multivariate Gaussian distributions (eq. 3.5) fitted to best represent a complex distribution.

$$f_X(x_1, x_2, ...x_k) = \sum_{i=1}^{n} w_i (2\pi)^{-k/2} \det |\mathbf{\Sigma}_i|^{(-1/2)} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^{\mathrm{T}} \mathbf{\Sigma}_i^{-1} (\mathbf{x}-\boldsymbol{\mu}_i)\right) \quad (3.5)$$

The covariance matrices $\{\mathbf{\Sigma}_i\}_{i\in\{1,...,n\}}$, means $\{\boldsymbol{\mu}_i\}_{i\in\{1,...,n\}}$ and weights $\{w_i\}_{i\in\{1,...,n\}}$ consist of d^2, d and 1 parameters respectively. Multivariate Gaussian distributions can also be called joint Gaussian distributions and one example can be seen in figure 3.4. To

fit the Gaussians to the configurations, we used the scikit python package [78], details can be found in the appendix A.1.2.

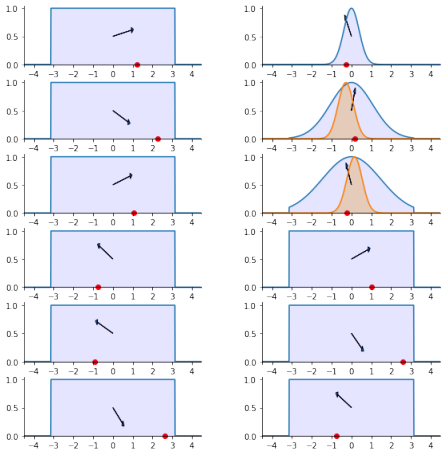

Figure 3.4.: Left: Uniform latent vector distribution. Right: Joint Gaussian distributions mixed with uniform distributions as latent vector distribution. If in the top most distribution the red dot is sampled, the second distribution becomes the orange one. Right and left the dots stand for possible samples with corresponding arrows.

We use a GMM as a prior distribution for different DOFs, for example for the first three spins as in figure 3.4. Since Gaussian distributions are defined on $\mathbb{R}$, but the angles $\theta \in]0; 2\pi]$, there is an inherent problem to this method. We tackle this by rejecting any sample that is out of bounds, leading to sharp edges in the resulting distribution, as can be seen in the third graph from the top on the right side in figure 3.4. A remaining problem is that the resulting distribution is not perfectly periodic like the circular symmetry that the model imposes. When sampling, first a Gaussian is drawn according to the weights w and then therefrom $\mathbf{x}$ is drawn. The probability density of sample $\mathbf{x}$ is proportional to $f_X(\mathbf{x})$ (eq. 3.5), which is enough information for our purposes.

3.1.3. Network Architecture

A schematic of the Boltzmann generator used for this model is depicted in figure 3.5. It uses either a prior distribution consisting only of uniform distributions or a mixture of uniform and Gaussian mixture model distributions as seen in figure 3.4.
Similar to [23], we employ splines in the split-coupling transformations. In the so called transformation layer before the configuration enters the neural network, we transform the configurations from the MCMC simulation to a zero-centered, rotational symmetry-depleted distribution. The symmetry of the system is removed by rotating the samples so the first spin has the value π and can therefore be removed when passing the configuration to the network. The subtraction of π leading to a zero-centered distribution is necessary

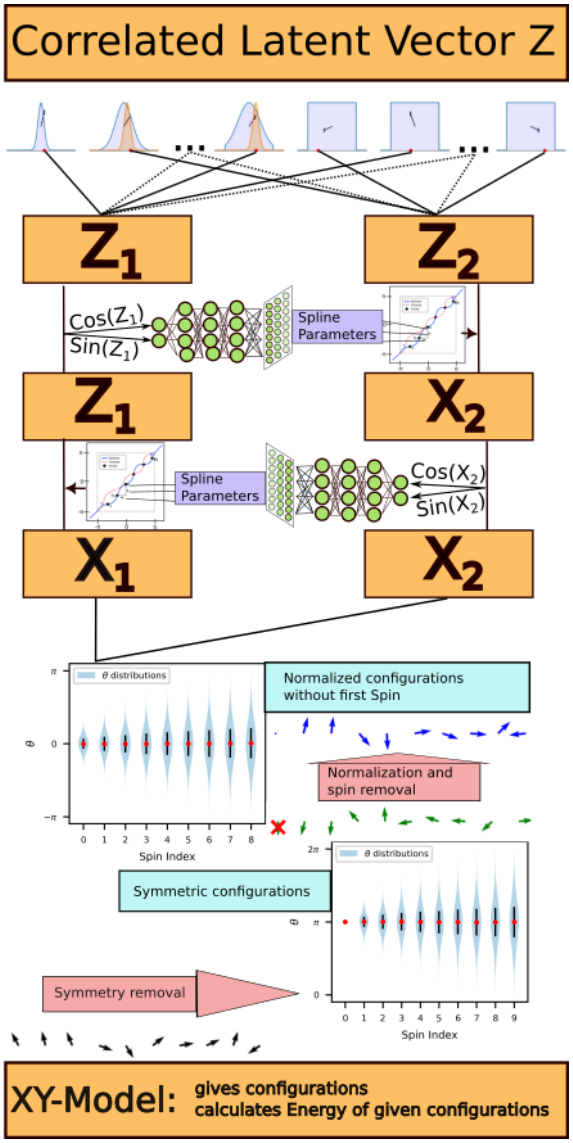

Figure 3.5.: The architecture of the BG used for the XY-model. A latent vector drawn from the latent distribution (fig. 3.4) is evenly split into two vectors. The sine and cosine of one vector components are used as inputs of an FNN, which outputs the parameters of a bijective spline function transforming the other vector. This is repeated in the other direction and this whole block can be repeated arbitrarily often. The configurations of an MC-simulation are rotated, so the first spin points in direction π. Then π is subtracted from every value and the first arrow is removed. The topmost block both provides samples and, if receiving a sample, provides its probability of occurring. Similarly, the XY-model provides samples from an MC-simulation and gives the energy of any configuration it receives.

for the spline flows to be used in the aforementioned way. For the network to understand the periodicity of the model we follow an approach similar to [79], which is that the FNNs train only on the sines cosines of the coupling vector components. Furthermore, the usage of circular spline flows [80] assures periodic transformations.
The energies necessary to evaluate the sample probabilities the network is trained on are on the one side provided by the Hamiltonian of the model and on the other side the probability is either constant in the uniform case or a product of a constant and the respective $f_X(\mathbf{x})$ in the uniform GMM mixed case. For further implementation details see Appendix A.1.3.

3.1.4. Performance Metrics of the Gaussian Mixture Prior

In this result section we first show the results and performance of a Network with a certain set of parameters to establish a baseline. We then compare this performance to networks with different parameters and finally to networks with uniform GMM mixed priors.

Uniformly Distributed Latent Vector Results

In the following figure 3.6, one can see the statistics of configurations produced by a Boltzmann generator consisting of three coupling blocks, connected by FNNs with three hidden layers consisting of eight nodes each defining parameters for a four-bin spline. The model consists of ten spins and the interaction strength per temperature $\beta J = 3$.
The unresampled configurations result in a regular, smooth density estimate for each spin in the violin plot. When comparing the analytic correlations to the ones from the generated configurations, one notices slight deviations. Those deviations almost vanish when looking at the resampled configurations, but in this case, the corresponding violin plot is less smooth. This issue can be solved by looking at the statistics of more resampled configurations. Here, the correlations are overlapping perfectly and the violin plot shows no irregularities.
We can conclude that the network successfully produces uncorrelated configurations and a resampling algorithm can be applied without problems in order to get the exact probability density of each configuration according to Boltzmann statistics. The less smooth violin plot of the resampled distribution compared to the unresampled one leads us to the introduction of the accuracy or the relative effective sample size:

$$RESS = \frac{(\sum_i \omega_i)^2}{N \sum_i \omega_i^2} \tag{3.6}$$

Here, ω_is are the weights attributed to every configuration defined in equation 2.37. The $RESS$ multiplied by N, the number of produced configurations gives the effective sample defined in the appendices of [23]. With a $RESS = 0.58 \pm 0.14$ the bumpy violin plot can be explained as an effect of insufficient many samples. Given enough samples are produced, the effective sample size is large enough to represent observables of the Boltzmann distribution correctly.

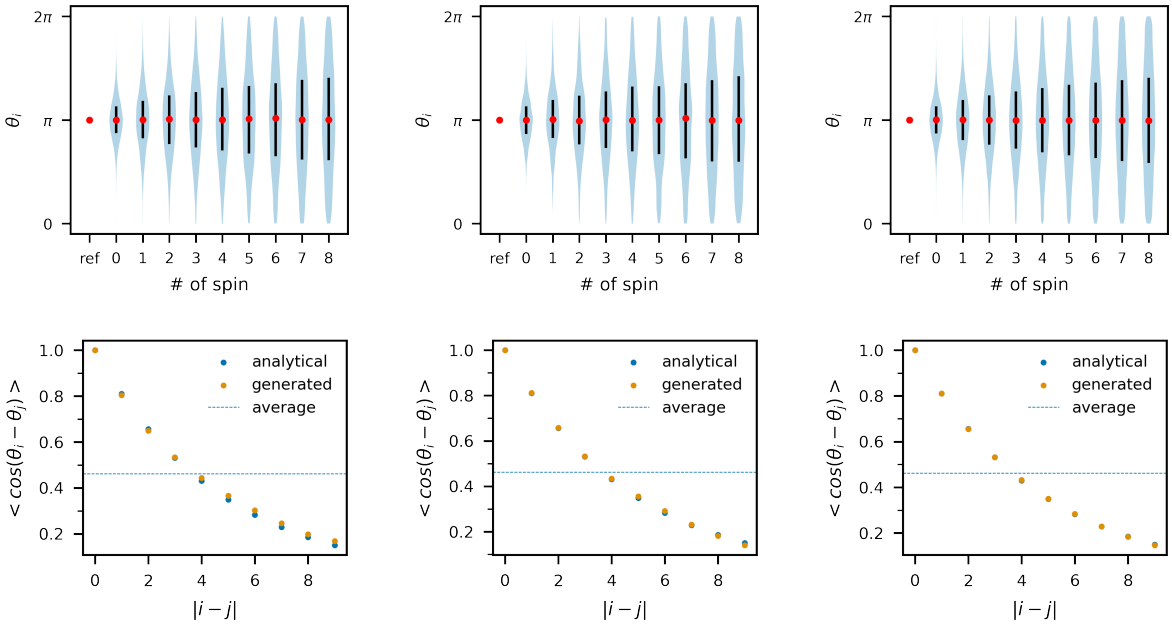

Figure 3.6.: BG trained on the XY-model at $\beta J = 3$. Top: Violin plots showing the distribution of each spin given that the first spin is fixed at π. The red dot indicates the mean and the black bar is the interval containing 50 % of the spins. Bottom: Comparison between the analytical and neural network-generated spin-spin correlation function. The blue line shows the average of the analytical correlations. Left: Unresampled (10k samples), middle: Resampled (10k samples), right: Resampled (100k samples).

Comparison Of BGs with Different Network And System Parameters

To establish a solid baseline of the performance of these networks, we test them with different sets of parameters. In the following graphs, the performance of the BGs is indicated by the learning curves showing the target energy fraction indicating the fraction of generated configurations $\mathbf{x}_{gen}$ within the first and 99th percentile $E_{0.01}^{\{\mathbf{x}_{train}\}}$ and $E_{0.99}^{\{\mathbf{x}_{train}\}}$ of the training data energy distribution RESS $= P(E_{0.01}^{\{\mathbf{x}_{train}\}} < E(\mathbf{x}_{gen}) < E_{0.99}^{\{\mathbf{x}_{train}\}})$. In one epoch, the network is trained once on every available training configuration batched together randomly. In the next epoch, the configurations are assigned to new batches randomly. Besides the previously introduces $RESS$ and $D_{KL} = KL$, we additionally look at the KL-divergence, which can be calculated in one direction as:

$$\begin{aligned} D_{KL}(F_{xz}(\mathbf{x})||\mathbf{z}) &= \int q_Z(F_{xz}(\mathbf{x}))\frac{q_Z(F_{xz}(\mathbf{x}))}{p_Z(\mathbf{z})} \\ &= \mathbb{E}_{\mathbf{z}\sim p_Z(\mathbf{z})}[-u(\mathbf{x}) - \log Z_{\mathbf{x}\sim p_X}(\mathbf{x}) + \log R_{zx}(\mathbf{z}) - \log p_Z(\mathbf{z})] \end{aligned} \tag{3.7}$$

where $\mathbf{x} = F_{zx}(\mathbf{z})$ are generated configurations. Since the partition function $Z_{\mathbf{x}\sim p_X}(\mathbf{x})$ is known for the XY-model, this KL divergence can be approximated well and used as a performance indicator. The network generates 10.000 configurations 100 times, from which we calculate the average and the standard deviation of the KL-divergence and the $RESS$, which are shown in the following plots.

Below in figure 3.7 the performance of networks with different amounts of hidden nodes per layer in each FNN is shown. Here one can conclude, that the more hidden nodes, the better. The worse performance of the network with 16 nodes compared to the one with 8 nodes is within statistical fluctuations. Additionally, the convergence of the training is achieved faster for networks with more nodes.

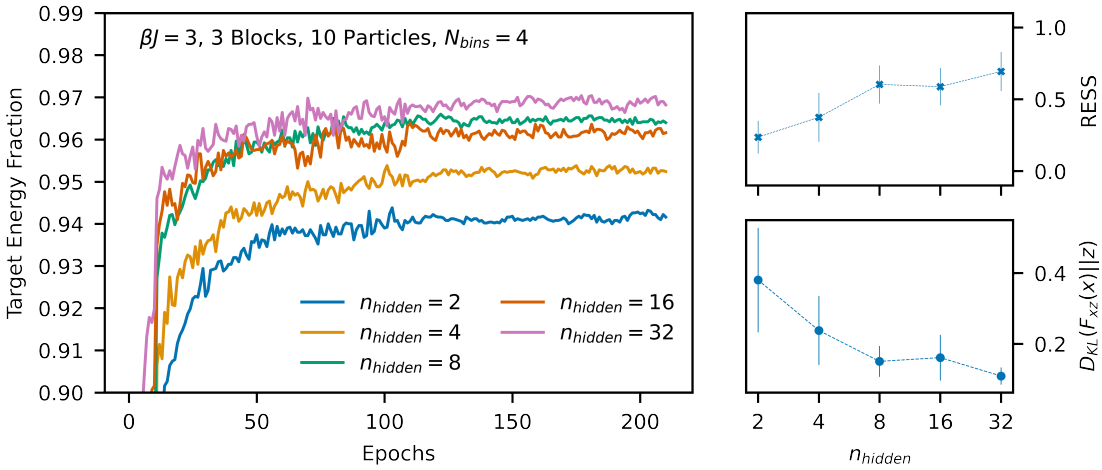

Figure 3.7.: Comparison of learning curves, $RESS$, and relative entropies D_{KL} for BGs trained with different numbers of nodes in the hidden layers.

In figure 3.8 we compare networks with a different number of coupling blocks. Here one also notices that the performance increases with the number of blocks, but the increase per block from the third block onwards becomes rather small. This leads to diminishing returns since the memory requirements and backpropagation time increases with the number of blocks.

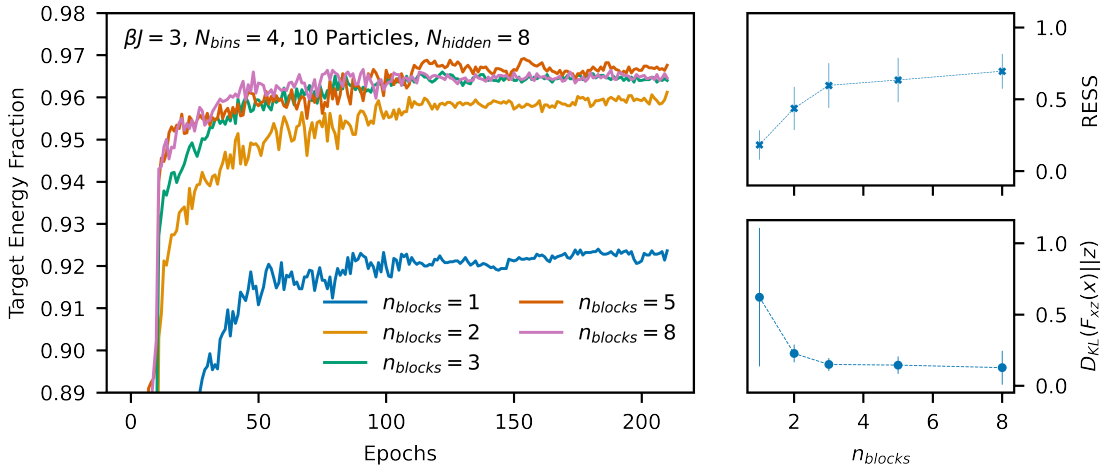

Figure 3.8.: Comparison of generators with increasing number of split-coupling blocks.

When comparing networks with a different number of bins (fig. 3.9), one notices that here the saying "the more, the better" does not hold. For these model and network parameters, the optimal bin number ranges from four to eight. This optimal number is probably specific to the system and needs to be adjusted for each network-system combination. The difference in performance here is less strong compared to the number of hidden nodes as shown in figure 3.7.

In figure 3.10, performances of networks trained for the XY-model with different interaction strengths are shown. The network performs best for either very weak or very strong interaction, so when randomness or order dominates. When the average correlation between spins $\langle\langle\cos(\theta_i - \theta_j)\rangle\rangle_{|i-j|\in[1,..,8]} \approx 0.5$ the network performs worse, so when neither randomness nor order dominates. Another trend is that the standard deviations increase both with worse performance and with stronger interaction. The reason for the latter probably is that the difference in internal energies $u(\mathbf{z})$ is larger for stronger interactions. When looking at equation 2.37 one can see that the difference between weights is higher for larger differences in $u(\mathbf{z})$.

We can conclude, that the BG performs better, the more hidden nodes and blocks are used, that there is a sweet spot for the number of bins in the spline functions, and that those systems that are neither very ordered with the average correlation $\langle\langle\cos(\theta_i - \theta_j)\rangle\rangle_{|i-j|\in[1,..,8]} \approx 1$ nor very random with $\langle\langle\cos(\theta_i - \theta_j)\rangle\rangle_{|i-j|\in[1,..,8]} \approx 0$ pose the most difficulties to a network, but the ones with $\langle\langle\cos(\theta_i - \theta_j)\rangle\rangle_{|i-j|\in[1,..,8]} \approx 0.5$.

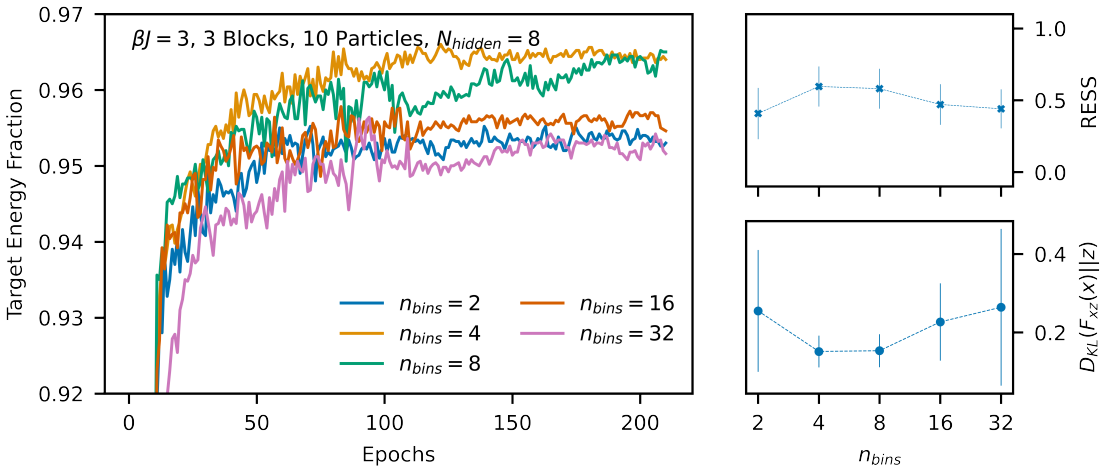

Figure 3.9.: Performance of BGs with a different number of spline knots n_{bins} in the split-coupling flow.

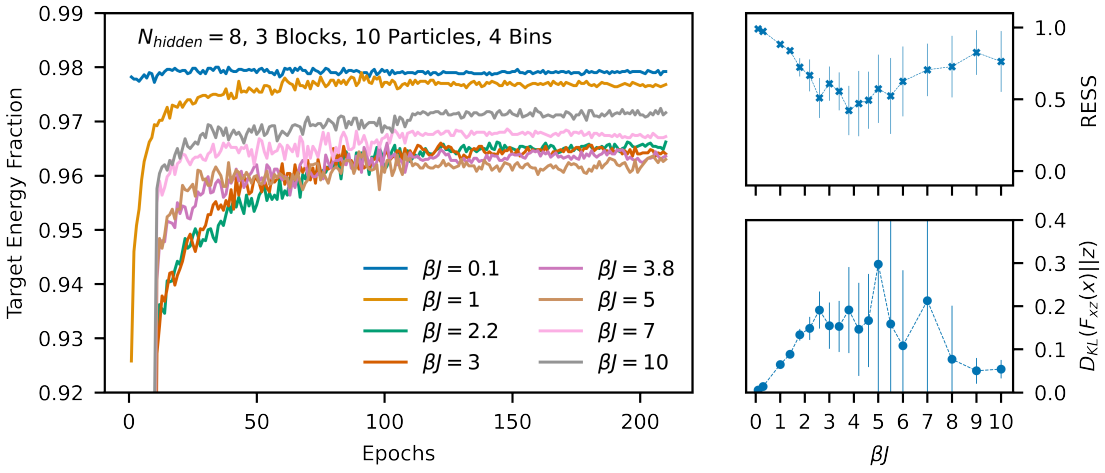

Figure 3.10.: Comparison of performances of BGs trained for different interaction strengths βJ. A higher βJ indicates a stronger interaction between neighboring spins.

Mixed Latent Vector Results

We try to improve the learning process and increase the resulting performance of BGs by substituting some uniform distributions with a Gaussian mixture model. In the following, we compare the performances of BGs with different priors.
We begin by comparing BGs with purely uniform priors to ones, where the prior distribution transformed to the first three spins is a Gaussian mixture model consisting of one three-dimensional Gaussian fitted to the first three spins of some train data passed through the transformation layer (fig. 3.11). While the target energy fraction curves indicate an equal to better performance and faster learning for all GMM-supported BG's, the $RESS$ and D_{KL} show, that supported networks are worse with the exception of strong interactions $\beta J = 10$.

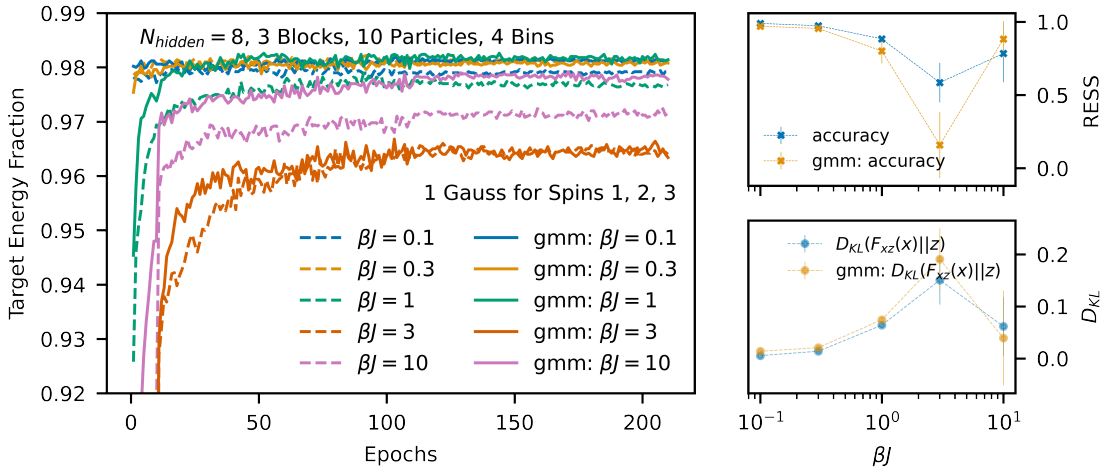

Figure 3.11.: Comparison of BGs with uniform (dashed lines) and gauss-uniform-mixed latent vector distributions (gmm, solid lines) for different interaction strengths βJ.

We can draw the following conclusions from figure 3.11. While the learning curves show, that GMM-supported BGs converge faster, and in most cases to a higher limit, the other performance indicators tell different stories. For all cases except the strong interaction/low-temperature ones, the GMM-support impairs the networks. Also, the $RESS$ is more sensible to worse network performance than the KL-divergence (fig. 3.11). The support only helping for strong interactions is probably due to the fact, that a Gaussian mixture model is not able to describe the periodicity of the problem, since it is defined on in this case $\mathbb{R}^3$ instead of the periodic interval $\mathbb{I}^3 =]0; 2\pi]^3$. Only for strong interactions, this problem becomes less important, since the probability density of configurations is very low at the boundaries.
In the graphs seen in figure 3.12 the performance of networks with a different number of Gaussians fitted to the first three spins are compared. The performance difference

between all networks apart from the four-Gaussian-supported Boltzmann generators is negligible, we attribute the exception to statistical fluctuations in the precision of the GMM. Therefore, we conclude, that the chosen number of components does in the here present case not play any role.

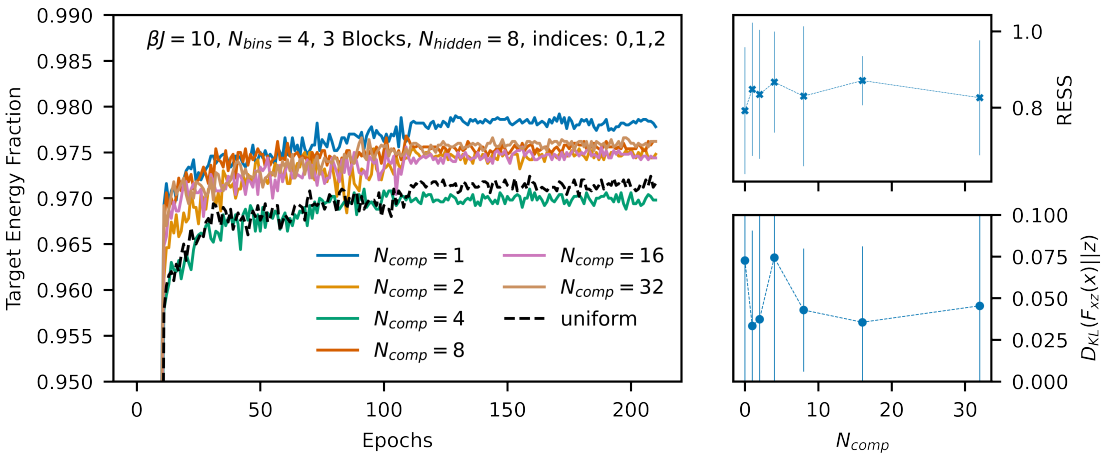

Figure 3.12.: Performance comparison of BGs for different numbers of Gaussians N_{comp} fitted to the distribution of the first three spins. The dashed line indicates the baseline performance using a uniform latent distribution. Solid lines show training runs with different numbers of Gaussian components.

In figure 3.13 we compare networks with GMM support for different spins. Here again, the Target Energy Fraction and the other indicators tell different tales. While the learning curve indicates, that the BG with a GMM for all indices performs best and learns much quicker than the others, according to the *RESS* it performs worst. We achieve the best results with GMM support for the first three indices. According to the KL-divergence and the *RESS*, this is also the only tested choice of indices improving the performance compared to the unsupported network.

To calculate the KL-divergences, we need to know $p_X(\mathbf{x})$, which we can only calculate analytically up to a constant factor. For the exact probability we use a Monte Carlo integration technique provided by [78], which for higher dimensions is prone to errors. This might lead to the per definition impossible negative KL-divergence for all indices in figure 3.13.

A reason for the worse performance of networks supported by a GMM in 5 or more dimensions (fig. 3.13) can be found when again looking at equation 2.37. While in the uniform prior case, $u_X(F_{zx}(\mathbf{z}))$ is always constant, in the high-dimensional GMM case, this term is fluctuating strongly. Therefore, the network must work with a wide range of different prior probabilities making the network more prone to over- or underestimating the probabilities of certain regions in phase space.

In figure 3.14 a violin plot and correlations of network-generated configurations are shown,

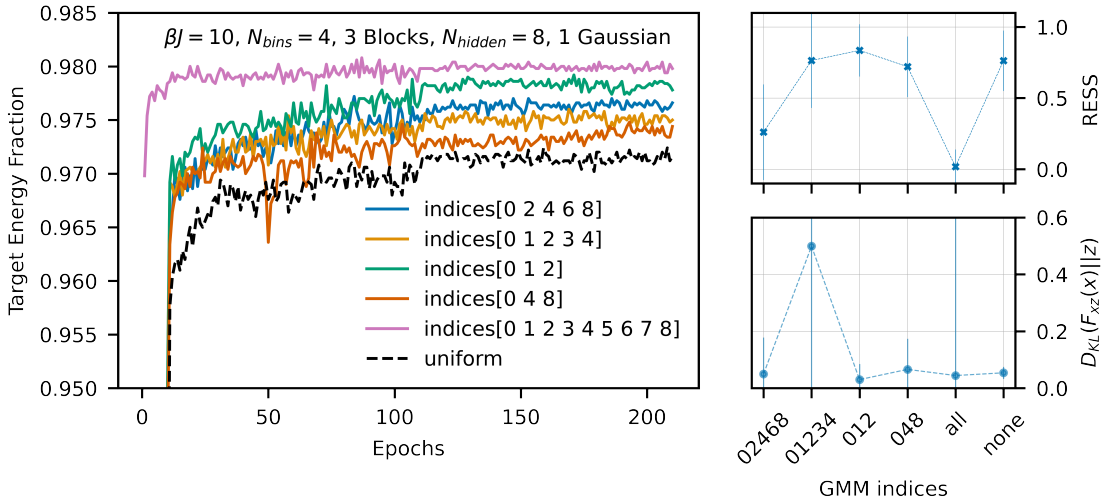

Figure 3.13.: Performance of BGs with one Gaussian component fitted on different spins. The dashed line indicates the baseline performance using a uniform latent distribution. The indices indicate the spin distribution with which the Gaussian was parameterized.

unresampled and resampled. While the unresampled configurations look perfectly fine, the resampled ones do not. This is due to a relatively large weight being assigned to a small selection of configurations, leading to the resampling breaking down, also seen in figure 3.14.
This is likely due to one of two reasons. One reason may be numerical errors appearing when the network has to deal with very low prior distribution probabilities. The other reason can be explained as follows; We think of a multivariate Gaussian fitted over the spin numbers $1-6$. The configuration(s) dominating the violin plot probably has a very low probability density under this Gaussian distribution. Assuming the Boltzmann generator is "lazy" and maps samples $\mathbf{z}$ drawn from the Gaussian to similar configurations $\mathbf{x}$, the term $-\log p_Z(\mathbf{z})$ in equation 2.37 becomes very large. In the apparent case, the corresponding energy $u(F_{zx}(\mathbf{z}))$ is not large enough, the weight of the sample becomes relatively large when compared to other weights.
In other words, according to the weights, the high-energy configuration(s) dominating the violin plot are underrepresented. This might be due to the corresponding latent vectors being drawn from the Gaussian by a very small chance, and the high energy not being able to compete with the very small chance. The network should have mapped these latent vectors to higher energy configurations, but due to their low probability density, it might have not visited this latent space region during learning sufficiently.
We can conclude:

- We successfully used BGs to produce uncorrelated configurations of a 1D XY-model.

- We tested the performance and learning sensitivities of the BGs for different network parameters.
- The choice of the prior influences the performance of the network.
- Due to the symmetry of the model, the GMM only helps in strong interaction cases, where the boundary does not play a large role.
- GMM support can impair the network performance, but it leads to a faster learning process.
- Using higher dimensional multivariate Gaussians as priors make for a strongly fluctuating term in equation 2.37, which is constant for uniform prior distributions, leading to comparatively higher weights.
- In extreme cases, the above or numerical errors lead to the resampling breaking down.

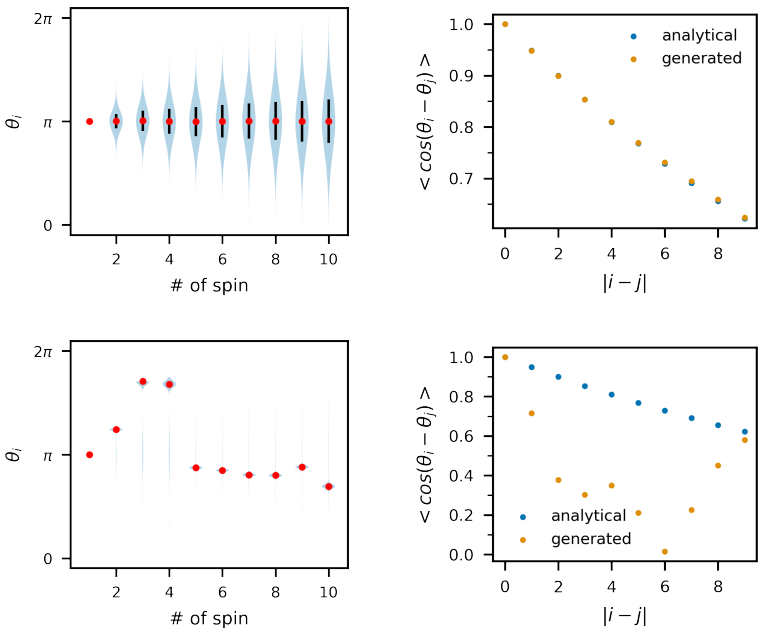

Figure 3.14.: Distributions (left) and neighbor-correlations (right) are shown for a $\beta J = 10$ model with a GMM latent distribution for the indices $0-4$. The top and bottom rows show the unresampled and resampled data respectively.

3.2. The Alanine Dipeptide Model

Even though many enhanced sampling algorithms were developed over the years, MD simulations of systems with rare transitions remain a challenging task. Say, for example, a molecule changes from one set of states to another on average once every millisecond. Since the simulation timestep is bounded by the fastest motion of the system, a 10^{12} timestep simulation is necessary to get maybe one transition and one needs many transitions to get a reliable statistic.
Boltzmann generators can produce uncorrelated configurations and therefore do not need to undergo rare transitions to get reliable equilibrium statistics of a system. However, as stated before, BGs do not scale well with increasing DOF, so in this thesis, we try to modify the latent space of the network to increase its performance.
For molecular systems we achieve that by combining three ideas: In the work by Coretti et al. [27], BGs are used to map configurations of a liquid system with certain interactions to a system with different interactions. The DOFs of both liquids are identical in this work. Already in the work by Dinh et al. [65], a multiscale architecture is proposed, where harder-to-learn vector components were transformed by more coupling blocks. We here propose to use a CG MD simulation to generate configurations representing the difficult-to-learn degrees of freedom efficiently. Those are then, together with a set of Gaussian distributions to model missing DOF, mapped to all-atom configurations.
To study this approach we aim to generate independent configurations of Alanine Dipeptide in implicit solvent (fig. 3.15). It undergoes a rare transition along the dihedral angle ϕ, which is defined by four atoms in a chain as the angle between planes spanned by atoms C^1, N^2, C^2_α and C^2 and N^2, C^2_α, C^2 and N^3 respectively, but is otherwise a rather simple molecule. It therefore has been studied rigorously [81], also using BGs [24].

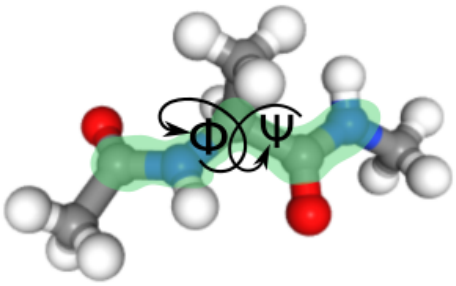

Figure 3.15.: Alanine Dipeptide. Ball colors correspond to the following elements: White to H, gray to C, blue to N, and red to O. ϕ and ψ are two dihedral angles.

3.2.1. MD-Simulation for the Generation of Reference Data

For the network to learn to generate configurations, reference data is required to initiate the training by example. This data we generate via an MD-simulation using OpenMM [82] and the Amber99 forcefield [83, 84]. To analyze the resulting configurations, the

Python package MDAnalysis [85, 86] is used. Further details on the implementation can be found in the appendix A.1.4
In the plots in figure 3.16 the results of a 10^6 timestep simulation at 450 K are shown. To the left one can see the free energy as a function of the backbone dihedral angles ϕ and ψ estimated from the simulation. At around $\phi = 0$, traces of density indicate one of the rare transitions that can occur in this system. The white regions have not been visited even once in the MD simulation and therefore we have no information on the free energy linked to those phase space regions.
To the right, the root mean square deviation between the minimal energy configuration and the visited configuration at a certain timestep is shown. Only once after ~ 0.76 ns the difference is increased for about 0.08 ns during which the simulation explored states with $\phi > 0$. Repeated 10^6 timestep simulations at 450 K often showed no transition at all.

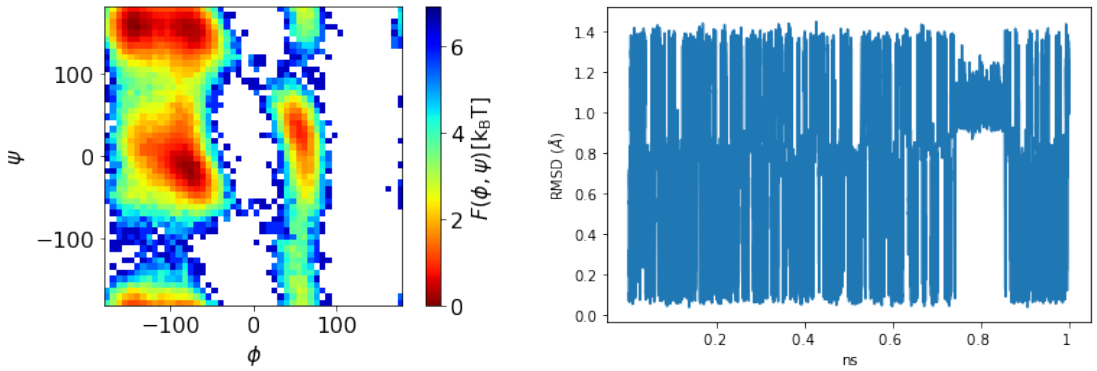

Figure 3.16.: Left: Free energy of the configurations with ϕ and ψ according to a 1 million time step MD-simulation at 450 Kelvin. $F(\phi, \psi) = -log(\rho(\phi, \psi))$ with F the free energy as a function of ϕ and ψ and ρ the probability density of configurations in the ϕ, ψ plane. Right: Root mean square deviation of configurations to minimal energy configuration plotted against time.

At 450 K or below regular MD simulations do not give us train data that reliably explore the configuration space according to the weights indicated by the Boltzmann distribution. In Falkner et al. [26], the authors combined BGs with transition path sampling to get reliable statistics of state transitions and Noé and coworkers [24] introduced temperature steerable flows to explore lower-temperature ensembles. We here mostly focus on 600 K systems, where MD-simulations show enough transitions and later try modeling a 450 K by mapping high-temperature training data to a lower temperature for the generated distribution.

3.2.2. Latent Vector - Coarse Graining

We want a coarse-grained representation of alanine dipeptide that contains the difficult-to-learn central dihedral angles ϕ and ψ and is tunable in its fastness and complexity. We

therefore use Boltzmann-inversion (eq. 2.2.3) to obtain an effective potential definition, the procedure is illustrated in figure 3.17.

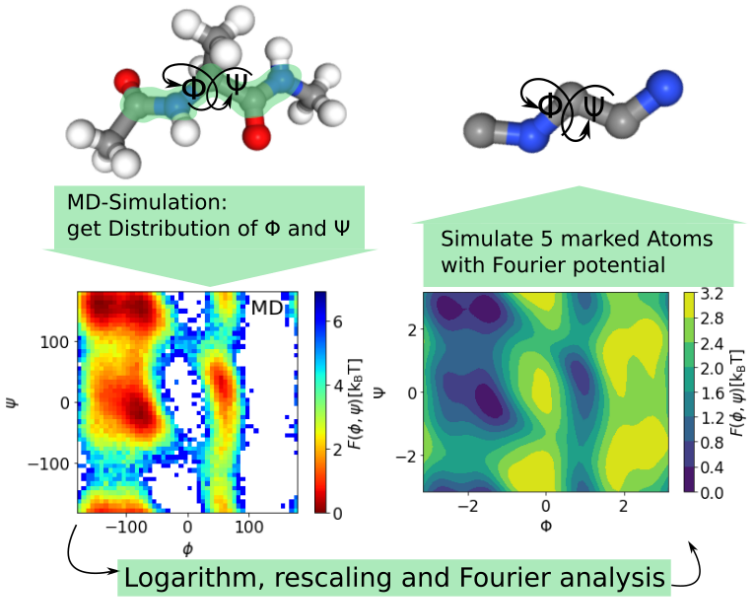

Figure 3.17.: The applied coarse-graining procedure: All but the 5 central Atoms are cut away, and only hard-sphere, bonded, and angle interactions are kept. A 600K AA-simulation is run. The resulting dihedral angle distribution is used for a Boltzmann-inversion. The resulting free energies are then Fourier-transformed, and an arbitrary amount of Fourier components (in this case 6) are then used to describe the free energy landscape, which is used as a dihedral angle interaction potential for a CG-MD-simulation.

We begin by omitting all atoms but the five ϕ and ψ defining ones, which we from now will denote as the backbone. The other atoms we refer to as side chain atoms. The bonded, angular, and hard-sphere interaction potentials are identical to the all-atom definitions. As established in equation 2.2.3, we approximate the dihedral angle interaction to be independent of other interactions and approximate it using Boltzmann-inversion of the probability density along the backbone dihedral angles. To obtain a continuously differentiable and periodic representation of the potential definition, we perform a Fast-Fourier transformation (FFT) of these free energies. The first couple of Fourier components are then used to define a periodic potential coupling ϕ and ψ via a reverse Fourier transformation. This leads to a regularized potential energy while the degree of regularization depends on the number of included Fourier components.
For the FFT to work, every value in the free energy histogram needs to be well-defined. Therefore a negligibly small value was added to the distribution histogram, mapping a constant high free energy to all white areas in the histogram. Details of implementation

and CG-MD-simulation details can be found in the appendix A.1.5.

3.2.3. Transformation Layers

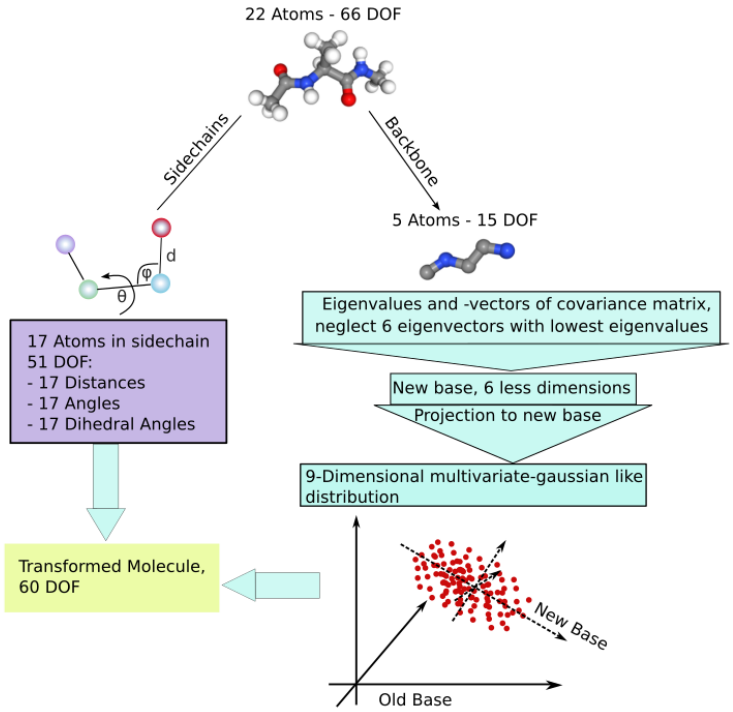

Figure 3.18.: Alanine dipeptide mixed coordinate transformation [22]. The Cartesian coordinate representation is mapped to a mixture of an internal coordinate representation and a principal component analysis representation. In the process degrees of freedom corresponding to the translational and rotational ones are removed.

A decisive feature of BGs for molecular systems is a mixed transformation layer introduced by [22]. It changes the coordinate system such that symmetries can be removed and all distribution components become more independent and more Gaussian-like. Additionally, the transformation and its inverse become more stable and less prone to error propagation. This is achieved by combining internal coordinate transformations with a principal component analysis-derived transformation.
In our case, as seen in figure 3.18, we describe the sidechains using internal coordinates and the backbone using coordinates defined by the principal components of the backbone coordinates. Sidechain coordinates are represented by distances, angles, and dihedral angles with respect to three priori-defined reference atoms. The sidechain coordinates are then also normalized.

For the principal component analysis, we first normalize the configuration distributions and then compute a covariance matrix thereof. Along the eigenvectors with the highest eigenvalues, the distribution has by definition the highest variance. Due to the prior alignment of the molecules to a reference structure, the six eigenvectors with the lowest eigenvalues correspond to the six rotational and translational degrees of freedom of the molecule and can therefore be discarded. The other eigenvectors span a new coordinate base for the backbone coordinates. This mixed transformation allows for handy symmetry removal and does not pose problems like error propagation.

3.2.4. Training the Boltzmann generator on the Alanine Dipeptide

We incorporate the coarse-grained latent model and the mixed coordinate transform of the all-atom model into a single network as depicted in figure 3.19.
This Boltzmann generator, like many others, consists of a coordinate transformation layer and affine coupling blocks. The prior distribution used in this thesis is composed of a coarse-grained simulation and Gaussian distributions instead of solely Gaussians. The CG configurations are transformed using the same principal component transformation used on the all-atom side. The probability density of a sample according to the prior is proportional to the product of the probability densities of the Gaussian distributions and the exponential of the negative coarse-grained potential. The coupling blocks and the transformation layers transform samples from the all-atom configurations. As a result, the main task of the BG is to adjust the backbone coordinates and transform the Gaussian latent vectors to the distribution of sidechain coordinates.
We hypothesize that this task is easier to learn and accomplish for a neural network than generating configurations from a Gaussian latent space. This would mean that, compared to regular BGs with only Gaussian distributions in the prior, this network's performance increases.
Mapping CG configurations back to all-atom configurations using machine learning has been researched and is a common tool for obtaining atomistic resolution during post-processing [87, 88], but to our knowledge not using BGs and also not explicitly considering Boltzmann statistics.
As briefly mentioned before, when training a network to generate 450 K configurations instead of the here usual 600 K, there is the problem of rare transitions in the train data and, when using our approach, also rare transitions when generating reference data for the coarse-grained simulation. To circumvent the latter problem, we tried the following approach: We multiplied the dihedral angle interaction potential by a factor $\frac{T_2}{T_1} = \frac{450\,\mathrm{K}}{600\,\mathrm{K}}$. One could interpret this as trying to map 600 K CG configurations to 400 K all-atom ones. We will therefore dub this approach the "High-Temperature Coarse-Grained Support" method.

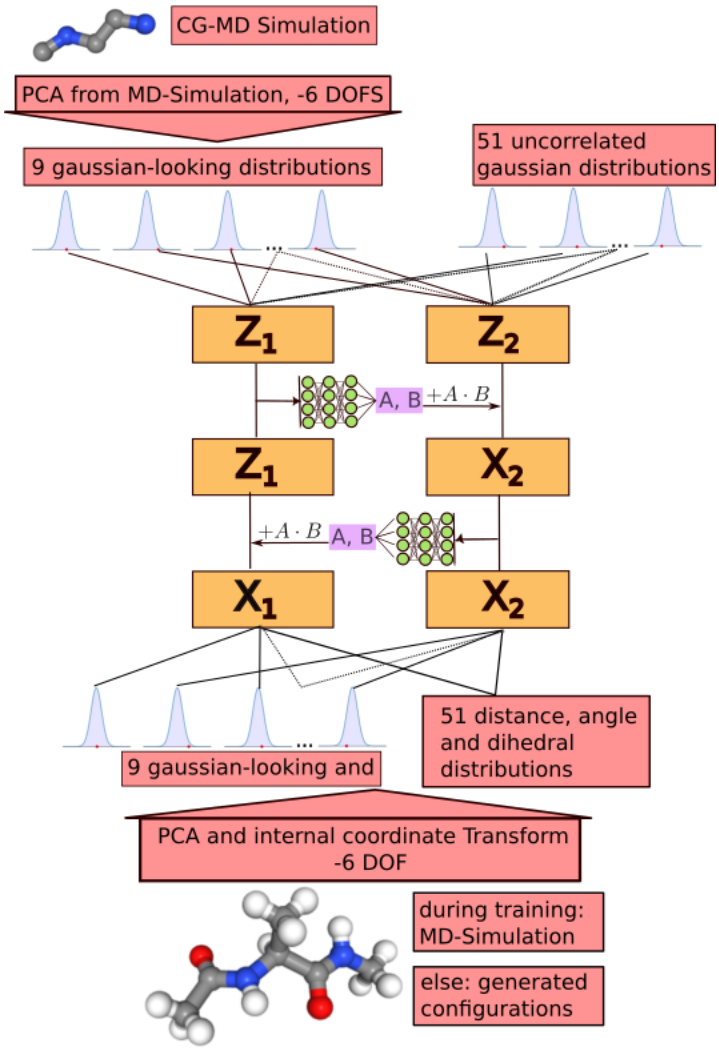

Figure 3.19.: BG used for alanine dipeptide. The bottom block corresponds to the transformation seen in figure 3.18. At the top, the CG-MD simulation is transformed by the same PCA-inspired transformation used at the bottom. The resulting 9-component vector is combined with 51 samples from Gaussian distributions. In between is a standard BG middle part. Same as before, both the top and bottom parts can give samples or receive samples returning the corresponding energies or probabilities.

3.3. Performance with Coarse Graining in Latent Distribution

3.3.1. Proof Of Concept

In an attempt to overcome the limitations of Boltzmann generators, we support it with coarse-grained molecular configurations. The first question is, does this approach work at all, can this network structure be used to sort of back map CG configurations?
After training the above sketched architecture to convergence, we see that produced configurations show no particle overlap or unrealistic geometries (fig. 3.20). Furthermore, the backbone atoms of the atomistic representations follow the coordinates of the coarse-grained representation. This indicates that the BG successfully maps coarse-grained configurations augmented with Gaussians to an atomistic representation.

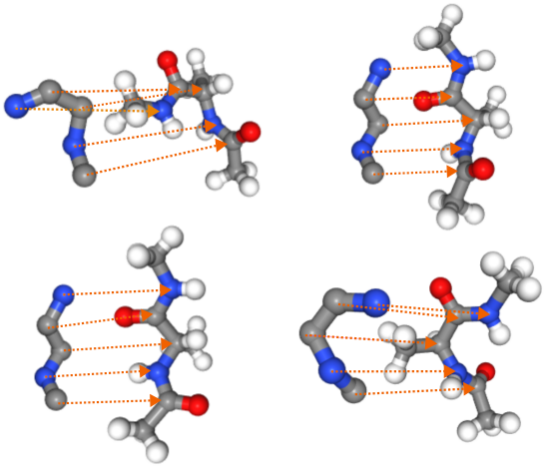

Figure 3.20.: Coarse-grained configurations are mapped to all-atom configurations via a BG-based learned mapping. Four configurations are shown with the coarse-grained representation on the left and the mapped all-atom configuration on the right. Arrows indicate equivalent atoms.

Since a BG without support is likewise capable of generating good-looking configurations, in the following section, we look at the dihedral angle distributions, the learning curves, and the relative effective sample size (eq. 3.6) of different networks to compare performances.

3.3.2. Performance of Boltzmann Generators with Coarse Graining

To learn more about the behavior of BGs with coarse-grained support we compare networks with the same parameters but different amounts of Fourier components describing the coarse-grained dihedral angle interaction potential. An increasing amount of Fourier components results in a more accurate yet sometimes more noisy representation of the effective backbone potential. As a baseline, we compare the results to the performance of a standard BG with a Gaussian latent space.

In the following figure 3.21 one can see the dihedral angle-dependent free energies of ensembles stemming from those generators. Our initial expectation was that more Fourier components (FCs) lead to better results since the backbone potential will follow more closely the effective potential from Boltzmann inversion. The results show that, indeed, CG support parametrized using just one or two FCs harms the performance, but three or more FC-CG support improve the generated angle distribution.

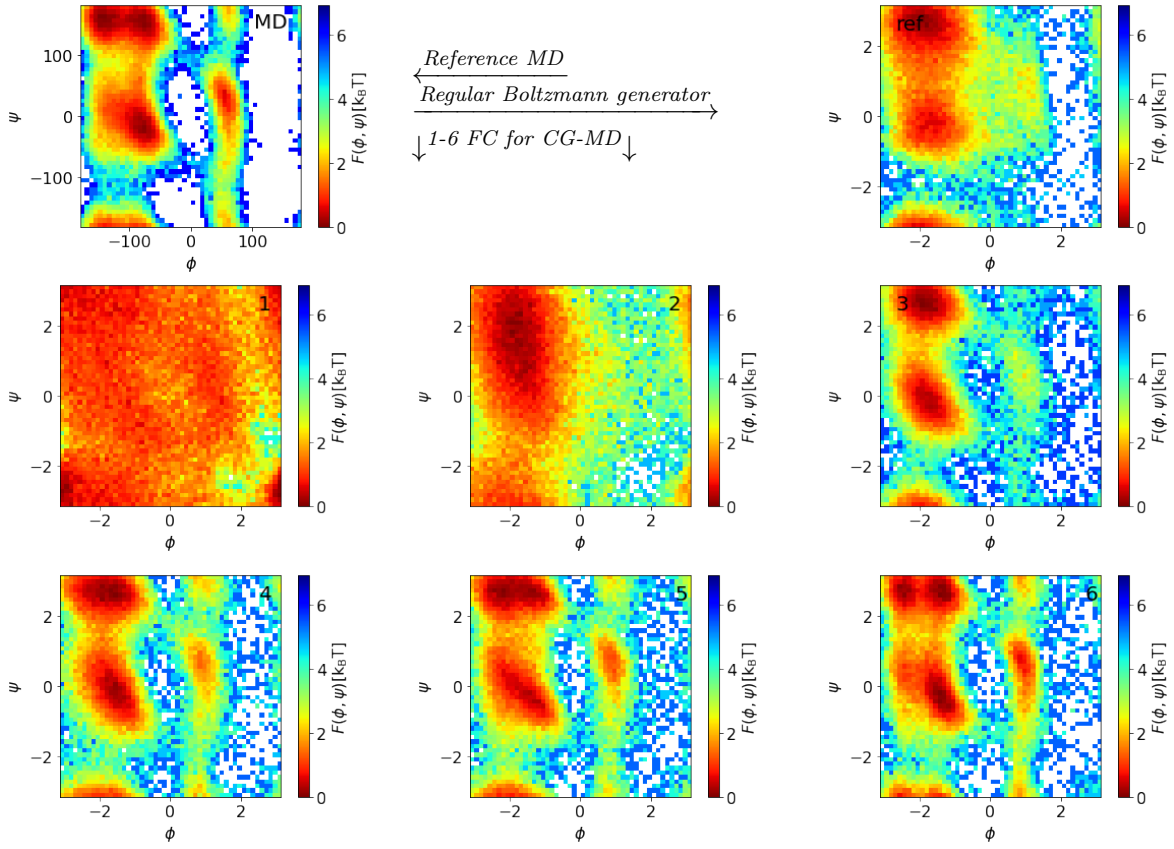

Figure 3.21.: The free energy of configuration ensembles with fixed ϕ and ψ of alanine dipeptide. Top left: Molecular dynamics simulation results. Top right: 5×10^4 Samples from BG with 60 Gaussian distributions as latent vector space. This result is used as a reference. In the lower two rows results from BG trained with CG configurations in the latent vectors are shown. The numbers in the corners correspond to the number of Fourier components used. $F(\phi, \psi) = -\log(\rho(\phi, \psi))$ with F the free energy as a function of ϕ and ψ and ρ the probability density of configurations in the ϕ, ψ plane.

These results indicate that the supported networks perform better at predicting the dihedral angle distribution with increasing components, but are the statistics concerning

the other degrees of freedom also correct? To work this out, we first look at the learning curves seen in figure 3.22.

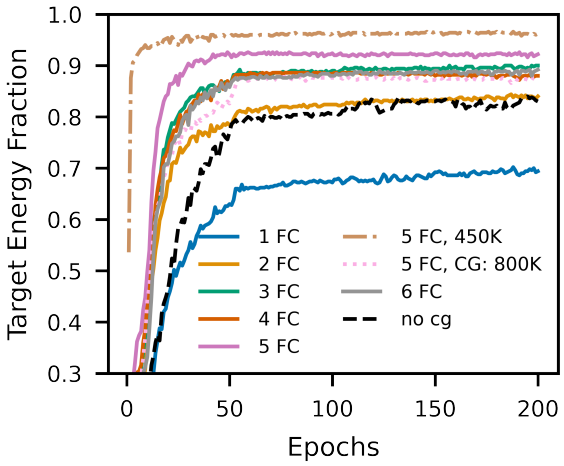

Figure 3.22.: Comparison of the learning curves of different networks with different numbers of Fourier components. Here, 1-6 FC stands for different numbers of Fourier components. The reference (dashed line) is a training run with a Gaussian latent distribution shown also in figure 3.21. The dash-dotted and the dotted lines show the training of networks mapping 600 K and 800 K CG configurations to 450 K and 600 K AA molecules respectively (sec. 3.3.4).

The by far worst learning curve is from the network with 1 FC-CG support, CG support without dihedral interaction potentials impairs the performance. Equal in energy fraction are the two FCs and the no-support networks, whereby the supported BG learns faster. Support with three, four, or six FCs leads to very similar learning curves, but the BG supported by five FCs performs best out of the ones generating 600 K configurations. This might be because the sixth FC carries detailed aspects of the angle distribution that are not fully converged. By that, the network has to ”unlearn” weights given to some regions by the prior, which appears to be difficult. So in this case it is better to give the network rather a regularized prior distribution than to provide it with incorrect details.

Surprisingly, the network mapping of 600 K CG samples to 450 K configurations learns the fastest and achieves the highest target energy fraction. To test, if the High Temperature CG Support method generally performs better, we tried mapping 800 K CG configurations to 600 K all-atom samples using five Fourier components. This led to a decrease in the performance compared to the five FCs support at the same temperature. We therefore hypothesize that the 450 K configuration space is smaller and easier to explore than the 600 K one.

3.3.3. Reweighting And Resampling the Alanine Dipeptide Configurations

A Boltzmann generator is capable of mapping a probability density to every produced sample, in our case up to a constant. When we reweight the produced configurations, we retrieve the exact Boltzmann distribution. While doing so, we can measure the *RESS*, an indicator of the performance telling us how many configurations we effectively drew from the actual Boltzmann distribution.

In the following plots, we first establish the baseline in the form of the performance of an unsupported BG at 600 K as shown in figure 3.23. Plots showing the free energies as a function of the dihedral angles can be seen. Resampling and reweighting differ in the following way: Weights are estimated for all configurations, and when resampling, those configurations are drawn randomly according to their weights with replacement. If a configuration has a relatively high weight, it will hence be drawn more often. Reweighting means, that we use weighted histograms according to equation 2.38.

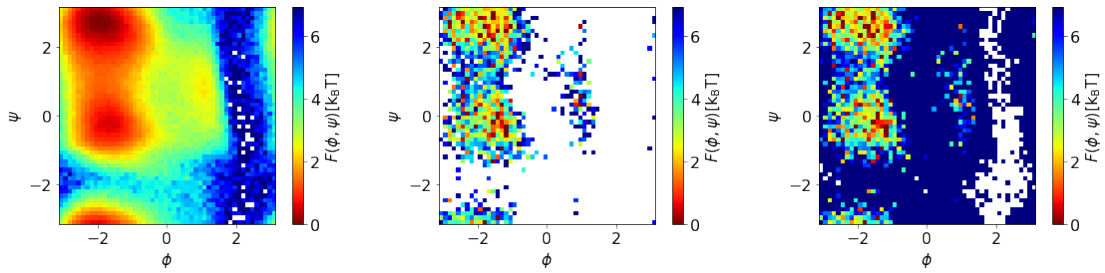

Figure 3.23.: Free energies of 5×10^5 configurations generated by a BG without CG assistance. Left: unreweighted, middle: resampled, right: Free energy of the average of ten reweighted distributions, each containing 5×10^4 samples. Again, $F(\phi, \psi) = -\log(\rho(\phi, \psi))$ with F the free energy of the configurations with ϕ and ψ and ρ the probability density of configurations in the ϕ, ψ plane. The $RESS = 0.0002$ with an $std = 0.0001$

The *RESS* tells us, that we effectively have only ~ 100 effective configurations in the pool of 5×10^5 generated configurations, which is the reason for the way the resampled and reweighted histograms look. It is clear that 100 samples are not enough to obtain an accurate histogram.

In the next plots (fig. 3.24), we can see the same histogram for a BG that received coarse-grained support with five Fourier components describing the dihedral angle interaction potential. The unresampled histogram looks much more similar to the actual free energy landscape than the histogram of the unsupported BG. We observe a 2.5 times higher *RESS* which leads to an effective sample size of 250 configurations leading to the resampled and reweighted histograms showing more detail of the shape of the actual free energy landscape.

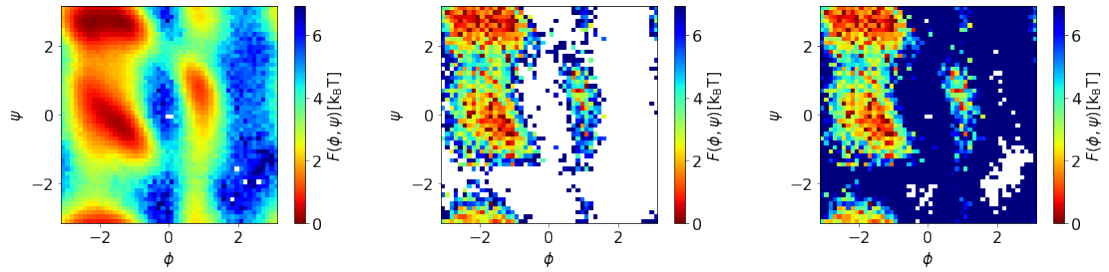

Figure 3.24.: Free energies of 5×10^5 configurations generated by a Network with 5 FC CG assistance. Left: unreweighted, middle: resampled, right: Free energy of the average of ten reweighted distributions, each containing 5×10^4 samples. The $RESS = 0.0005$ with an $std = 0.0002$

3.3.4. Temperature Exchange

When going to lower temperatures the problem of rare transitions becomes more evident since thermal fluctuations decrease in magnitude. We try circumventing this problem using the High Temperature CG Support method meaning we train the network at a lower temperature with example data from a higher temperature, where rare transitions occur frequent enough. This method might also up to a certain degree compensate for a lack of train data in some phase space regions. It might also help explore phase space regions with low probability densities due to a higher visiting frequency during training. While this approach teaches the network to visit different regions in phase space at the wrong frequency since $\exp\left(\frac{\mathcal{H}(\mathbf{x}_1)}{k_\mathrm{B}T_1}\right) / \exp\left(\frac{\mathcal{H}(\mathbf{x}_2)}{k_\mathrm{B}T_1}\right) \neq \exp\left(\exp\left(\frac{\mathcal{H}(\mathbf{x}_1)}{k_\mathrm{B}T_2}\right)\right) / \exp\left(\frac{\mathcal{H}(\mathbf{x}_2)}{k_\mathrm{B}T_2}\right)$, where $\mathcal{H}$ is the Hamiltonian, this error can later be mended by reweighting. And if other more severe problems can be handled with this approach it is worth the cost in $RESS$.

Below in figure 3.25 the same histograms as before for a BG using the High Temperature CG Support method can be seen. A total of 1000 effective configurations lead to resampled and reweighted histograms that capture the shape of the free energy landscape already quite well. As mentioned before, the five times higher $RESS$ compared to the 600 K BG probably stems from the fact, that at 450 K, the phase space is less complicated.

To gather more information about the workings of this method, we train a 310 K system with a 600 K inspired CG prior. However, the problem with this approach is that during training not a single rare transition occurred in the MD simulation. The training by energy using samples from the prior tells the network that it should look for configurations at $\phi > 0$, but the train data for training by example objects. Histograms of configurations produced during the training and histograms of the disagreeing train data and prior can be seen in figure 3.26.

To minimize the loss when training by example, the originally $\phi > 0$ configurations get pulled more and more to the left. But the network seems unable to "unlearn" the information by the prior. Even after long training, the phase space regions do not merge. One can conclude that if the training data do not visit some remote regions in phase

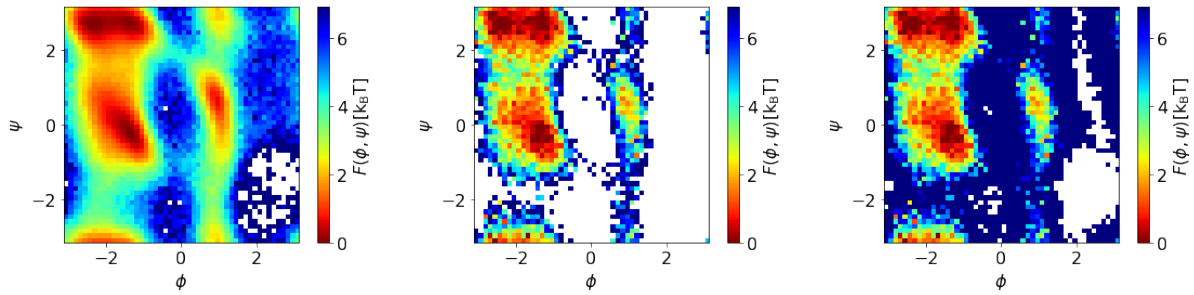

Figure 3.25.: Free energies of 5×10^5 configurations generated by a BG trained at 450 K with 5 FC CG assistance at 600 K. Left: unreweighted, middle: resampled, right: Free energy of the average of ten reweighted distributions, each containing 5×10^4 samples. The $RESS = 0.004$ with an $std = 0.002$

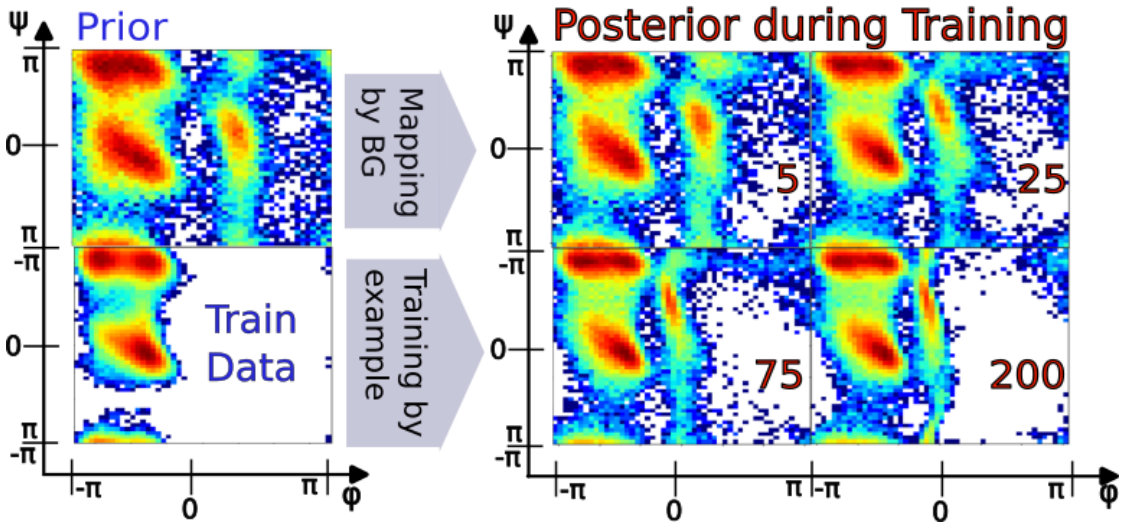

Figure 3.26.: The learning process of a Boltzmann generator at 310K with a 600 K CG simulation in the prior. Top left: Dihedral angle distribution of the coarse-grained 600 K simulation used in the prior. Bottom Left: All-atom 310 K dihedral angle distribution used as train data. Right: Dihedral angle distributions after five, 25, 75, and 200 training epochs. In one training epoch, the BG learns all train data once grouped in a certain way. In each epoch, the training configurations are regrouped randomly.

space, a BG and the here proposed method cannot help, although it might help if the regions are small and close. If the High-Temperature CG Support method does not help,

it leads to a breakdown of the BG.
We conclude that giving CG support to a Boltzmann generator does help, our hypothesis has been confirmed. We must add that with this approach our samples again become correlated since the CG-MD simulation-produced configurations are correlated. In the discussion, we will describe a possible approach to tackling this problem.

4. Discussion

Exploring equilibrium probability density functions of different systems is a much-researched problem in many scientific fields. Different approaches such as Markov Chain Monte Carlo- and Molecular Dynamics simulations deal with this problem but have difficulties such as correlations between samples. Normalizing Flow Neural Networks called Boltzmann-Generators have been developed to help with this task [22]. Those Boltzmann-Generators in turn have the problem of not scaling well with increasing degrees of freedom.
For this thesis, we research supporting BGs, instead of letting them shape configurations all on their own from a Gaussian Prior, we provide them samples resembling the desired configurations in some DOFs. An example treated in this work is mapping coarse-grained configurations to all-atom ones. Our goal was to test if BGs are capable of this task and if so, if the quality of generated configurations, when compared to standard BGs, increases. As probably best seen when comparing the figures 3.23 and 3.24, Boltzmann-Generators can be used for said task and the performance increases with support, but only if the support is carefully constructed.
We have here established a method capable of mapping less detailed descriptions of a system to a more detailed one in a probabilistic way. We find that looking at the respective DOFs, the detailed generated configuration distribution usually is similar to the distribution of undetailed samples and although whole high-density regions can be shifted and deformed, the total probability of a region is not changed in the transformation.
Using a Gaussian Mixture Model to support BGs generating XY-model configurations impairs the network unless the model is in a very ordered low-temperature state. This is on the one hand due to a non-periodic Gaussian being a bad fit for a periodic system. On the other hand, numerical issues may play a role when using many Gaussians and multiplying the respective probabilities together in a joint distribution. Also, the network seems to perform especially badly at learning to transform samples with a low probability under the GMM, overestimating the respective weights. Using a GMM to support a BG was shown to enhance performance in one case, it may be useful also in other situations, but it must be constructed very sensibly.
The XY-model was especially difficult to learn when the average correlation was around 0.5, which can be interpreted as the temperature of maximal multi-modality, a state neither dominated by randomness nor order. We conclude that systems with higher multi-modality are more difficult to learn for BGs.
We could also see the networks correctly learning all the low-energy configurations but having problems with higher-energy ones. This is probably due to the finite size of the networks. They first concentrate on important low-energy configurations and may not have enough resources left to learn higher-energy ones after finishing the first task. When

some phase space areas are not represented because of the network's finite size, we see a so-called "Mode Collapse". Also, the fact that the network trains on free energies of batches may play a role, since small batch sizes might inhibit the network from more thoroughly exploring higher energy configurations. When testing the BGs we noticed that different performance measures sometimes contradict each other. Especially the target free energy is a very rough, inaccurate measure, not matching with the KL-divergence and relatively effective sample size. Those two in turn mostly concur, but not their standard deviations. Another observation is that the BG is unable to learn phase space regions missing in the train data, no matter the support that is given to it.
Since the Coarse-grained support worked better than the GMM support, we recommend future researchers focus on the first method. It is questionable if our test system Alanine-Dipeptide represents complex molecule behaviors well since it only really has two multi-modal degrees of freedom ϕ and ψ. Thus the next step would be to test this method on larger systems with more multi-modal degrees of freedom, for example, a polymer model like in [26] or the protein bovine pancreatic trypsin inhibitor like [22]. Other CG models might have to be found for those since the Fourier-based analysis may become infeasible. We recommend using existing CG models like MARTINI [6], which represents four heavy atoms and all attached hydrogens to one interaction center, although one then needs to find a suitable transformation layer. Similar to the here used transformation layer, one could use internal coordinates defined by distances, angles, and dihedral angles to the coarse-grained interaction centers and a PCA transformation on those CG beads. To lessen the correlations appearing when using a coarse-grained latent space, one could consider putting BGs in series similar to [27], but with increasing numbers of freedom like in [65]. The first BG learns to generate CG configurations, the next one maps those to all-atom configurations. This may result in a network architecture needing fewer parameters than a usual BG, but in turn needing train data not only from all-atom-, but also CG simulations.
A still unanswered problem is how to treat explicit solvent. So far we and others have used implicit solvents, which do not capture local contributions to the free energy of a molecule. Explicit solvents would change the free energy landscape more drastically by stabilizing certain configurations. Also here, using coarse graining such as monoatomic water models as a latent distribution could bring the field closer to modeling processes important physics, chemistry and biology.

Bibliography

[1] Daan Frenkel and Berend Smit. *Understanding molecular simulation: From algorithms to applications*. Academic Press, 1996.

[2] Glenn M. Torrie and John P. Valleau. Monte carlo free energy estimates using non-boltzmann sampling: Application to the sub-critical lennard-jones fluid. *Chemical Physics Letters*, 28:578–581, 1974.

[3] Alessandro Laio and Michele Parrinello. Escaping free-energy minima. *Proceedings of the National Academy of Sciences*, 99:12562–12566, 2002.

[4] Yuji Sugita and Yuko Okamoto. Replica-exchange molecular dynamics method for protein folding. *Chemical Physics Letters*, 314:141–151, 1999.

[5] Soumil Y. Joshi and Sanket A. Deshmukh. A review of advancements in coarse-grained molecular dynamics simulations. *Molecular Simulation*, 47, 2021.

[6] Jan A. Stevens, Fabian Grünewald, P. A. Marco van Tilburg, Melanie König, Benjamin R. Gilbert, Troy A. Brier, Zane R. Thornburg, Zaida Luthey-Schulten, and Siewert J. Marrink. Molecular dynamics simulation of an entire cell. *Frontiers in Chemistry*, 11, 2023.

[7] G.M. Torrie and J.P. Valleau. Nonphysical sampling distributions in monte carlo free-energy estimation: Umbrella sampling. *Journal of Computational Physics*, 23:187–199, 1977.

[8] Alessandro Barducci, Giovanni Bussi, and Michele Parrinello. Well-tempered metadynamics: A smoothly converging and tunable free-energy method. *Physical Review Letters*, 100:020603, 2008.

[9] Alireza Seif, Mohammad Hafezi, and Christopher Jarzynski. Machine learning the thermodynamic arrow of time. *Nature Physics*, 17:105–113, 2021.

[10] Jiang Wang, Simon Olsson, Christoph Wehmeyer, Adrià Pérez, Nicholas E. Charron, Gianni De Fabritiis, Frank Noé, and Cecilia Clementi. Machine learning of coarse-grained molecular dynamics force fields. *ACS Central Science*, 5, 2019.

[11] Dan Guest, Kyle Cranmer, and Daniel Whiteson. Deep learning and its application to lhc physics. *Annual Review of Nuclear and Particle Science*, 68:161–181, 2018.

[12] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583–589, 2021.

[13] E. M. Dogo, O. J. Afolabi, N. I. Nwulu, B. Twala, and C. O. Aigbavboa. A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks. In *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*, pages 92–99. IEEE, 2018.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84–90, 2017.

[15] Tomas Mikolov, Anoop Deoras, Daniel Povey, Lukas Burget, and Jan Cernocky. Strategies for training large scale neural network language models. In *2011 IEEE Workshop on Automatic Speech Recognition I& Understanding*, pages 196–201. IEEE, 2011.

[16] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12:2493–2537, 2011.

[17] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[18] David Foster and an O'Reilly Media Company. Safari. *Generative deep learning : teaching machines to paint, write, compose, and play.* O'Reilly Media, Inc., 2019.

[19] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1462–1471, Lille, France, 2015. PMLR.

[20] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv*, 2303.12712, 2023.

[21] Esteban G. Tabak and Eric Vanden-Eijnden. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8, 2010.

[22] Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365, 2019.

[23] Peter Wirnsberger, George Papamakarios, Borja Ibarz, Sébastien Racanière, Andrew J. Ballard, Alexander Pritzel, and Charles Blundell. Normalizing flows for atomic solids. *Machine Learning: Science and Technology*, 3, 2022.

[24] Manuel Dibak, Leon Klein, Andreas Krämer, and Frank Noé. Temperature steerable flows and boltzmann generators. *Physical Review Research*, 4, 2022.

[25] Jonas Köhler, Andreas Krämer, and Frank Noe. Smooth normalizing flows. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 2796–2809. Curran Associates, Inc., 2021.

[26] Sebastian Falkner, Alessandro Coretti, Salvatore Romano, Phillip Geissler, and Christoph Dellago. Conditioning normalizing flows for rare event sampling. *arXiv*, 2207.14530, 2022.

[27] Alessandro Coretti, Sebastian Falkner, Phillip Geissler, and Christoph Dellago. Learning mappings between equilibrium states of liquid systems using normalizing flows. *arXiv*, 2208.10420, 2022.

[28] Steyn van Leeuwen, Alberto Pérez de Alba Ortíz, and Marjolein Dijkstra. A boltzmann generator for the isobaric-isothermal ensemble. *arXiv*, 2305.08483, 2023.

[29] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21:1087–1092, 1953.

[30] Loup Verlet. Computer "experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical Review*, 159:98–103, 1967.

[31] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[32] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[33] Varun Kumar Ojha, Ajith Abraham, and Václav Snášel. Metaheuristic design of feedforward neural networks: A review of two decades of research. *Engineering Applications of Artificial Intelligence*, 60:97–116, 2017.

[34] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

[35] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[36] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[37] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1:1–23, 1993.

[38] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[39] Mohamed Alloghani, Dhiya Al-Jumeily, Jamila Mustafina, Abir Hussain, and Ahmed J. Aljaaf. *A Systematic Review on Supervised and Unsupervised Machine Learning Algorithms for Data Science*, pages 3–21. Springer International Publishing, Cham, 2020.

[40] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[41] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv*, 2204.06125, 2022.

[42] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[43] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. Deep learning techniques for music generation – a survey. *arXiv*, 1709.01620, 2017.

[44] Ping-Huan Kuo, Ssu-Ting Lin, and Jun Hu. Dnae-gan: Noise-free acoustic signal generator by integrating autoencoder and generative adversarial network. *International Journal of Distributed Sensor Networks*, 16:155014772092352, 2020.

[45] Pourya Hoseini, Liang Zhao, and Amarda Shehu. Generative deep learning for macromolecular structure and dynamics. *Current Opinion in Structural Biology*, 67:170–177, 2021.

[46] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[47] David Ha and Jürgen Schmidhuber. World models. *arXiv*, 1803.10122, 2018.

[48] Xiaojie Guo, Yuanqi Du, Sivani Tadepalli, Liang Zhao, and Amarda Shehu. Generating tertiary protein structures via interpretable graph variational autoencoders. *Bioinformatics Advances*, 1(1):vbab036, 2021.

[49] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63:139–144, 2020.

[50] Jie Gui, Zhenan Sun, Yonggang Wen, Dacheng Tao, and Jieping Ye. A review on generative adversarial networks: Algorithms, theory, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 35:3313–3332, 2023.

[51] Hang Yang, Minghui Wang, Zhenhua Yu, Xing-Ming Zhao, and Ao Li. Gancon: Protein contact map prediction with deep generative adversarial network. *IEEE Access*, 8:80899–80907, 2020.

[52] Zhaoyu Li, Son P. Nguyen, Dong Xu, and Yi Shang. Protein loop modeling using deep generative adversarial network. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1085–1091. IEEE, 2017.

[53] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *33rd International Conference on Machine Learning, ICML 2016*, volume 4, 2016.

[54] Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G. Willcocks. Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44:7327–7347, 2022.

[55] Ruslan Salakhutdinov and Hugo Larochelle. Efficient learning of deep boltzmann machines. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 693–700, Chia Laguna Resort, Sardinia, Italy, 2010. PMLR.

[56] Yusuke Nomura, Nobuyuki Yoshioka, and Franco Nori. Purifying deep boltzmann machines for thermal quantum states. *Physical Review Letters*, 127:060601, 2021.

[57] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *32nd International Conference on Machine Learning, ICML 2015*, volume 3, 2015.

[58] Lvmin Zhang and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. *arXiv*, 2302.05543, 2023.

[59] E. G. Tabak and Cristina V. Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66:145–164, 2013.

[60] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 1530–1538. JMLR.org, 2015.

[61] Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43, 2021.

[62] He Kaiming, Zhang Xiangyu, Ren Shaoqing, and Sun Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification kaiming. *Biochemical and Biophysical Research Communications*, 498, 2018.

[63] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[64] Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4743–4751, Red Hook, NY, USA, 2016. Curran Associates Inc.

[65] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[66] Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, 2016.

[67] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[68] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-December, 2016.

[69] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[70] Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 2018.

[71] J. A.K. Suykens, H. Verrelst, and J. Vandewalle. On-line learning fokker-planck machine. *Neural Processing Letters*, 7, 1998.

[72] Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, 2011.

[73] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[74] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.

[75] J. A. Gregory and R. Delbourgo. Piecewise rational quadratic interpolation to monotonic data. *IMA Journal of Numerical Analysis*, 2, 1982.

[76] Elliott Lieb, Theodore Schultz, and Daniel Mattis. Two soluble models of an antiferromagnetic chain. *Annals of Physics*, 16, 1961.

[77] M. R. B. Clarke, Richard O. Duda, and Peter E. Hart. Pattern classification and scene analysis. *Journal of the Royal Statistical Society. Series A (General)*, 137, 1974.

[78] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12, 2011.

[79] Suchuan Dong and Naxian Ni. A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks. *Journal of Computational Physics*, 435, 2021.

[80] Danilo Jimenez Rezende, George Papamakarios, Sebastien Racaniere, Michael Albergo, Gurtej Kanwar, Phiala Shanahan, and Kyle Cranmer. Normalizing flows on tori and spheres. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8083–8092. PMLR, 2020.

[81] Vladimir Mironov, Yuri Alexeev, Vikram Khipple Mulligan, and Dmitri G. Fedorov. A systematic study of minima in alanine dipeptide. *Journal of Computational Chemistry*, 40:297–309, 2019.

[82] Peter Eastman, Jason Swails, John D. Chodera, Robert T. McGibbon, Yutong Zhao, Kyle A. Beauchamp, Lee-Ping Wang, Andrew C. Simmonett, Matthew P. Harrigan, Chaya D. Stern, Rafal P. Wiewiora, Bernard R. Brooks, and Vijay S. Pande. Openmm 7: Rapid development of high performance algorithms for molecular dynamics. *PLOS Computational Biology*, 13:e1005659, 2017.

[83] Junmei Wang, Piotr Cieplak, and Peter A. Kollman. How well does a restrained electrostatic potential (resp) model perform in calculating conformational energies of organic and biological molecules? *Journal of Computational Chemistry*, 21:1049–1074, 2000.

[84] Aleksandar Spasic, John Serafini, and David H. Mathews. The amber ff99 force field predicts relative free energy changes for rna helix formation. *Journal of Chemical Theory and Computation*, 8:2497–2505, 2012.

[85] Naveen Michaud-Agrawal, Elizabeth J. Denning, Thomas B. Woolf, and Oliver Beckstein. Mdanalysis: A toolkit for the analysis of molecular dynamics simulations. *Journal of Computational Chemistry*, 32, 2011.

[86] Richard Gowers, Max Linke, Jonathan Barnoud, Tyler Reddy, Manuel Melo, Sean Seyler, Jan Domański, David Dotson, Sébastien Buchoux, Ian Kenney, and Oliver Beckstein. Mdanalysis: A python package for the rapid analysis of molecular dynamics simulations. In *Proceedings of the 15th Python in Science Conference*, 2016.

[87] Wei Li, Craig Burkhart, Patrycja Polińska, Vagelis Harmandaris, and Manolis Doxastakis. Backmapping coarse-grained macromolecules: An efficient and versatile machine learning approach. *The Journal of Chemical Physics*, 153:041101, 2020.

[88] Yaxin An and Sanket A. Deshmukh. Machine learning approach for accurate backmapping of coarse-grained models to all-atom models. *Chemical Communications*, 56:9312–9315, 2020.

[89] David A. Sivak, John D. Chodera, and Gavin E. Crooks. Using nonequilibrium fluctuation theorems to understand and correct errors in equilibrium and nonequilibrium simulations of discrete langevin dynamics. *Physical Review X*, 3, 2013.

List of Figures

2.1. Scheme of a feedforward neural network (FNN). An n component vector is used as input $\{x_i\}_{i\in[0,1,2,...,n]}$. With m biases, $n \cdot m$ weights and these inputs $\{y_j\}_{j\in[1,2,...,m]}$ are calculated. The activation function f is applied on those giving the values of the nodes in the first hidden layer. This procedure is repeated for all hidden layers and finally once more resulting in the output $\{z_i\}_{i\in[1,2,...,k]}$. 10

2.2. Scheme of a normalizing flow. A bijective function F_{zx} maps every point z of an easy-to-sample from probability distribution p_z to a point x in some physical configuration space. The function can be altered by changing the components of the parameter vector θ. θ is optimised by minimising the Kullback-Leibler losses L_{KL}. In the formulas $\beta = \frac{1}{k_B T}$, U is the potential energy function of the physical system and $J_{zx} = J_{xz}^{-1}$ are the Jacobian matrices of transformations from $\{z\}$ to the configuration space. 14

2.3. Scheme of a Boltzmann generator. From a simple latent vector distribution, uncorrelated samples are drawn. The block also provides probabilities for each vector. The uneven components become part of the vector Z_1, the even ones part of Z_2. Z_1 is used as input values of a FNN giving the vectors **A** and **B**. Those then transform Z_2 to X_2. The same procedure gives values transforming Z_1 to X_1. This block is repeated n times. The final X_1 and X_2 are recombined to a vector fed to a mixed transformation layer, an invertible function transforming configurations into more well-behaved distributions and vice versa. The bottom box provides configurations from some other simulation and maps the energy to every vector. 18

2.4. The weights (in log scale) and energies of configurations produced by a BG. 21

3.1. Depiction of two configurations of the XY-model with different interaction strengths. 23

3.2. Left: Exponentially decreasing correlation of direction of arrows with $\beta J = 1$, formula inset. Right: Correlation of neighboring arrows and heat capacity per site in the infinite arrows limit both plotted against the interaction strength divided by $k_B T = \frac{1}{\beta}$. 24

3.3. Left: Violin plot of distributions of spin angles θ. Right: Correlation between spin pairs at distance $|i-j|$, analytically calculated and estimated from samples. Samples for plots were generated with an MC-simulation. . 25

3.4. Left: Uniform latent vector distribution. Right: Joint Gaussian distributions mixed with uniform distributions as latent vector distribution. If in the top most distribution the red dot is sampled, the second distribution becomes the orange one. Right and left the dots stand for possible samples with corresponding arrows. 26

3.5. The architecture of the BG used for the XY-model. A latent vector drawn from the latent distribution (fig. 3.4) is evenly split into two vectors. The sine and cosine of one vector components are used as inputs of an FNN, which outputs the parameters of a bijective spline function transforming the other vector. This is repeated in the other direction and this whole block can be repeated arbitrarily often. The configurations of an MC-simulation are rotated, so the first spin points in direction π. Then π is subtracted from every value and the first arrow is removed. The topmost block both provides samples and, if receiving a sample, provides its probability of occurring. Similarly, the XY-model provides samples from an MC-simulation and gives the energy of any configuration it receives. 27

3.6. BG trained on the XY-model at $\beta J = 3$. Top: Violin plots showing the distribution of each spin given that the first spin is fixed at π. The red dot indicates the mean and the black bar is the interval containing 50 % of the spins. Bottom: Comparison between the analytical and neural network-generated spin-spin correlation function. The blue line shows the average of the analytical correlations. Left: Unresampled (10k samples), middle: Resampled (10k samples), right: Resampled (100k samples). . . . 29

3.7. Comparison of learning curves, $RESS$, and relative entropies D_{KL} for BGs trained with different numbers of nodes in the hidden layers. 30

3.8. Comparison of generators with increasing number of split-coupling blocks. 31

3.9. Performance of BGs with a different number of spline knots n_{bins} in the split-coupling flow. 32

3.10. Comparison of performances of BGs trained for different interaction strengths βJ. A higher βJ indicates a stronger interaction between neighboring spins. 32

3.11. Comparison of BGs with uniform (dashed lines) and gauss-uniform-mixed latent vector distributions (gmm, solid lines) for different interaction strengths βJ. 33

3.12. Performance comparison of BGs for different numbers of Gaussians N_{comp} fitted to the distribution of the first three spins. The dashed line indicates the baseline performance using a uniform latent distribution. Solid lines show training runs with different numbers of Gaussian components. 34

3.13. Performance of BGs with one Gaussian component fitted on different spins. The dashed line indicates the baseline performance using a uniform latent distribution. The indices indicate the spin distribution with which the Gaussian was parameterized. 35

3.14. Distributions (left) and neighbor-correlations (right) are shown for a $\beta J = 10$ model with a GMM latent distribution for the indices $0 - 4$. The top and bottom rows show the unresampled and resampled data respectively. . 36

3.15. Alanine Dipeptide. Ball colors correspond to the following elements: White to H, gray to C, blue to N, and red to O. ϕ and ψ are two dihedral angles. 37

3.16. Left: Free energy of the configurations with ϕ and ψ according to a 1 million time step MD-simulation at 450 Kelvin. $F(\phi, \psi) = -log(\rho(\phi, \psi))$ with F the free energy as a function of ϕ and ψ and ρ the probability density of configurations in the ϕ, ψ plane. Right: Root mean square deviation of configurations to minimal energy configuration plotted against time. 38

3.17. The applied coarse-graining procedure: All but the 5 central Atoms are cut away, and only hard-sphere, bonded, and angle interactions are kept. A 600K AA-simulation is run. The resulting dihedral angle distribution is used for a Boltzmann-inversion. The resulting free energies are then Fourier-transformed, and an arbitrary amount of Fourier components (in this case 6) are then used to describe the free energy landscape, which is used as a dihedral angle interaction potential for a CG-MD-simulation. . 39

3.18. Alanine dipeptide mixed coordinate transformation [22]. The Cartesian coordinate representation is mapped to a mixture of an internal coordinate representation and a principal component analysis representation. In the process degrees of freedom corresponding to the translational and rotational ones are removed. 40

3.19. BG used for alanine dipeptide. The bottom block corresponds to the transformation seen in figure 3.18. At the top, the CG-MD simulation is transformed by the same PCA-inspired transformation used at the bottom. The resulting 9-component vector is combined with 51 samples from Gaussian distributions. In between is a standard BG middle part. Same as before, both the top and bottom parts can give samples or receive samples returning the corresponding energies or probabilities. 42

3.20. Coarse-grained configurations are mapped to all-atom configurations via a BG-based learned mapping. Four configurations are shown with the coarse-grained representation on the left and the mapped all-atom configuration on the right. Arrows indicate equivalent atoms. 43

3.21. The free energy of configuration ensembles with fixed ϕ and ψ of alanine dipeptide. Top left: Molecular dynamics simulation results. Top right: 5×10^4 Samples from BG with 60 Gaussian distributions as latent vector space. This result is used as a reference. In the lower two rows results from BG trained with CG configurations in the latent vectors are shown. The numbers in the corners correspond to the number of Fourier components used. $F(\phi, \psi) = -\log(\rho(\phi, \psi))$ with F the free energy as a function of ϕ and ψ and ρ the probability density of configurations in the ϕ, ψ plane. . 44

3.22. Comparison of the learning curves of different networks with different numbers of Fourier components. Here, 1-6 FC stands for different numbers of Fourier components. The reference (dashed line) is a training run with a Gaussian latent distribution shown also in figure 3.21. The dash-dotted and the dotted lines show the training of networks mapping 600 K and 800 K CG configurations to 450 K and 600 K AA molecules respectively (sec. 3.3.4). 45
3.23. Free energies of 5×10^5 configurations generated by a BG without CG assistance. Left: unreweighted, middle: resampled, right: Free energy of the average of ten reweighted distributions, each containing 5×10^4 samples. Again, $F(\phi, \psi) = -\log(\rho(\phi, \psi))$ with F the free energy of the configurations with ϕ and ψ and ρ the probability density of configurations in the ϕ, ψ plane. The $RESS = 0.0002$ with an $std = 0.0001$ 46
3.24. Free energies of 5×10^5 configurations generated by a Network with 5 FC CG assistance. Left: unreweighted, middle: resampled, right: Free energy of the average of ten reweighted distributions, each containing 5×10^4 samples. The $RESS = 0.0005$ with an $std = 0.0002$ 47
3.25. Free energies of 5×10^5 configurations generated by a BG trained at 450 K with 5 FC CG assistance at 600 K. Left: unreweighted, middle: resampled, right: Free energy of the average of ten reweighted distributions, each containing 5×10^4 samples. The $RESS = 0.004$ with an $std = 0.002$ . . . 48
3.26. The learning process of a Boltzmann generator at 310K with a 600 K CG simulation in the prior. Top left: Dihedral angle distribution of the coarse-grained 600 K simulation used in the prior. Bottom Left: All-atom 310 K dihedral angle distribution used as train data. Right: Dihedral angle distributions after five, 25, 75, and 200 training epochs. In one training epoch, the BG learns all train data once grouped in a certain way. In each epoch, the training configurations are regrouped randomly. 48

A. Appendix

A.1. Algorithms

A.1.1. MC-Simulation of the XY-Model

The Markov Chain Monte Carlo simulation starts with a configuration of random spins (between 0 and 2π). In a simulation step, the angle of a randomly chosen spin is changed with a maximum step size of 0.25. If a new spin is larger than 2π, 2π is subtracted from it to impose periodicity. If one is smaller than 0, 2π is added. The new configuration is then either rejected or accepted based on the old and new energies. For equilibration, 5000 steps are taken before recording training data in order to start at reasonable configurations.

A.1.2. Gaussian Mixture Model

The sklearn.mixture package [78] is used to find a Gaussian Mixture Model fit for provided configurations. First of all, each configuration is rotated until the first spin is π, then the configurations are normalized by subtracting π from every spin and finally, the first spin is removed. The Mixture model is initialized with the following code:
"gmm = GaussianMixture(n_components = gauss_components, covariance_type = 'full', tol = 0.0001, max_iter = 10000, ini_params = 'random_from_data')"
The integer "gauss_components" stands for the number of Gaussians used, different numbers are compared in 3.12. With the command "gmm.fit(train_data)" we then find the weights, means, and covariance matrices.

A.1.3. XY-Model Network

For each set of interaction strength and system size tested, 10^7 configurations are generated in advance. 50000 randomly drawn samples are used as train data, whereby 5000 of those are used as test configurations. The GMM is established and fitted to 10000 configurations. All the networks learn using the same training parameters:

- Ten epochs with a learning rate of 0.001, only training by example, and a batch size of 256.
- 100 epochs with a learning rate of 0.001, equally weighted training by example and energy, and a batch size of 1024
- 100 epochs with a learning rate of 0.0001, equally weighted training by example and energy, and a batch size of 1024

In all cases, energies are clamped at 100. For backpropagation, the Adam algorithm is used with the parameters $\beta_1 = 0.9$, $\beta_1 = 0.999$, $\epsilon = 10^{-8}$, and a weight decay of 0.001.

A.1.4. Alanine Dipeptide MD-Simulation

We used tools provided by OpenMM [82], MDAnalysis [85] [86] and Amber forcefields[83] [84].
We used the "amber99sbildn" forcefields for the molecule parametrization and "amber99_obc" for modeling the implicit solvent. MD-simulations were conducted, unless otherwise stated, at a temperature of 600 K, with a collision rate of 1 ps and time steps of 1 fs. As integrator we used the Velocity-Verlet-Integrator with velocity randomization [89].

A.1.5. Coarse Grained Simulation

In order to derive a CG dihedral angle interaction potential depicted as Fourier Components, we first make a histogram of the probabilities in the $\phi - \psi$-plane according to a 1 ns MD-simulation. We then add a constant of $3 \cdot 10^-7$ to fill empty bins, representing a very small probability. The histogram consists of $50 \cdot 50$ bins. Then we take the negative logarithm and factor out the temperature to arrive at a free energy function given as a histogram. A Fast-Fourier-Transform then gives us Fourier-Components, which define our CG potential. The distance- and angle interactions are again parametrized using the "amber99sbildn" forcefield.
While a higher timestep might be possible for the CG model, we use 1 fs to match with the all-atom simulation. The collision rate was 1 ps. Again we used a Velocity-Verlet-Integrator with velocity randomization. During training, the equilibration time is 3 ps and the decorrelation time is 50 fs. When generating and testing, those times are 5 ps and 100 fs respectively.

A.1.6. Alanine Dipeptide Network

The BGs here used consists of eight affine coupling blocks with 200 nodes in each of the three hidden layers in each FNN. From a 1 ns simulation, 10000 randomly drawn configurations are used as train data.
The training starts with 50 epochs of learning by example and a batch size of 128. Then there are again 50 epochs with a $1 : 10^{-5}$ ratio between learning by example and learning by energy. The batch size is 2500. For the last 100 epochs the batch size stays the same, and the ratio increases to $1 : 10^{-3}$. For all epochs, the energies are clamped at 10^4. When training by energy, an angle loss is introduced to prevent angle generation outside of $[-\pi, \pi)$. The other training parameters such as the optimizer parameters are the same as for the XY-model network.