Boltzmann Generators

Frank Noé and Hao Wu

November 5, 2018

Abstract

Computing equilibrium structures and statistics of condensed-matter many-body systems such as spin systems or macromolecules in solution is a computationally intensive task that presently uses algorithms such as molecular dynamics (MD) simulation or Markov-chain Monte Carlo (MCMC) that need to take many small steps in configuration space until statistical estimates are converged. Here we develop Boltzmann Generators, a deep learning method that learns a variable transformation such that the equilibrium distribution of a given many-body system can be "one-shot" sampled, i.e. by sampling statistically independent random variables from a simple distribution in the transformed space. This is achieved by combining recent developments in directed generative neural networks with principles of equilibrium statistical mechanics.

1 Introduction

Equilibrium statistical mechanics is concerned with computing the average behavior of many copies of a physical system based on models of its microscopic constituents and their interactions. For example, what is the average magnetization in an Ising model of interacting magnetic spins in an external field, or what is the probability of a protein to be folded in an atomistic molecular model as a function of the temperature. Under a wide range of conditions, the equilibrium probability of a microscopic configuration $\mathbf{x}$ (setting of all spins, positions of all protein atoms, etc.) can be expressed by a Boltzmann-type distribution of the form

$$\mu(\mathbf{x}) = \frac{1}{Z}\mathrm{e}^{-u(\mathbf{x})},$$

where $u(\mathbf{x})$ is a dimensionless energy which contains the potential energy of the system, the temperature and optionally other thermodynamic quantities (SI).

Except for the simple model systems, such as the ideal gas or harmonic oscillators, we presently neither know exact solutions for Boltzmann-weighted averages, nor ways to sample $\mathbf{x}$ directly from the Boltzmann distribution. Therefore, the generic simulation methods that are currently in use are essentially all variants of Markov-Chain Monte Carlo (MCMC) or Molecular Dynamics (MD) simulations that make tiny changes to $\mathbf{x}$ in each simulation step and will, over a long time, generate simulation trajectories that sample from the Boltzmann distribution. These methods suffer from rare events, i.e., many simulation steps are needed to transition between different phases or states – for example, $10^9 - 10^{15}$ MD simulation steps are needed to fold or unfold a protein. As a result, these methods are extremely expensive and consume much of the worldwide supercomputing resources. Rare events transitions which can be traced by low-dimensional coordinates can be sped up by enhanced sampling methods [16, 6, 10], but the computational effort remains enourmous.

Ideally we would like to have a "Boltzmann Generator" machine that samples statistically independent configurations in "one-shot" from a known distribution $p_X(\mathbf{x})$ that is very similar to the Boltzmann distribution. If that could be achieved, we could use the ratio between $p_X(\mathbf{x})$ and the Boltzmann weight $\mathrm{e}^{-u(\mathbf{x})}$ to reweight each sample and compute any desired statistics. This enterprise seems, at first, hopeless for condensed-matter systems and complex polymers (e.g., Fig. 3a, Fig. 4k). In these systems, particles with strong repulsive interactions are densely packed in space, such that the number of "valid" configurations with a significant Boltzmann probability are vanishingly few compared to the number of possible ways to place particles.

Nonetheless, we directly address this problem in the present paper. Our strategy is: since constructing a Boltzmann Generator in configuration space is too difficult, can instead seek a coordinate transformation of $\mathbf{x}$ to a so-called "latent" representation $\mathbf{z}$, in which sampling is easy and every sample can be back-transformed to a configuration $\mathbf{x}$ with a relevant Boltzmann weight. To find such a transformation, we employ deep learning that has recently led to breakthroughs in pattern recognition, games and autonomous control [11].

2 Boltzmann Generators

Neural networks that can draw statistically independent samples from a desired distribution are called directed generative networks [5, 8]. Such generative networks have been demonstrated to draw photo-realistic images [7], to produce deceivingly realistic speech audio [14], and even to sample formulae of chemical compounds with certain physico-chemical properties [4]. In these domains, the exact target distribution is not known and must be inferred by observing many examples, which explains why the approach works best when large datasets are available. Here we are in the inverse situation, as we want to avoid having to train the Boltzmann Generator with samples $\mathbf{x}$ from the Boltzmann distribution, as these are difficult to generate. However, we have the great advantage that the Boltzmann distribution is known up to constant factor, such that the weight of each generated sample can be evaluated and used in the training procedure.

The idea of Boltzmann Generators is as follows: we want to learn a neural network transformation F_{zx} such that when sampling from a simple distribution in $\mathbf{z}$, such as a standard normal distribution, $F_{zx}(\mathbf{z})$ will provide a Boltzmann-distributed configuration x:

$$p_Z(\mathbf{z}) = \mathcal{N}(0, \tau\mathbf{I}) \overset{\text{sample}}{\sim} \mathbf{z} \xrightarrow{F_{zx}} \mathbf{x} \overset{\text{sample}}{\sim} p_X(\mathbf{x}) \propto \mathrm{e}^{-u(\mathbf{x})/\tau}.$$

Here, $\mathcal{N}(0, \tau\mathbf{I})$ indicates the normal distribution with variance τ and $\mathrm{e}^{-u(\mathbf{x})/\tau}$ is a Boltzmann distribution at temperature τ (SI).

In general, we will sample from the Boltzmann distribution exactly, and in some cases we may even want to deviate from that goal, for example in order to sample certain high-energy states with a higher probability. In order to both train the Boltzmann Generator and subsequently reweight its samples to the Boltzmann distribution, it is important that we can quantify the distribution $p_X(\mathbf{x})$ that is generated by the network at all times. This can be achieved when F_{zx} is invertible: For invertible transformations, we can not only back-compute $\mathbf{z} = F_{zx}^{-1}(\mathbf{x}) = F_{xz}(\mathbf{x})$, but we can also compute $p_X(\mathbf{x})$ from the known $p_Z(\mathbf{z})$ and F_{xz} (SI).

Invertible neural network transformations are similar to flows of a fluid – an incompressible fluid when using the NICE layers [2] (Fig. 1b) and a compressible fluid when using the RealNVP layers [9] (Fig. 1e). Invertibility is achieved by splitting the variables into two or more channels, performing only simple, invertible operations on each of them, such as translations (T) and scaling (S) operations (Fig. 1c,e). Nonlinear operations acting between the channels define the T/S operations. These are represented by general neural networks that do not need to be inverted themselves (Fig. 1a). The NICE or RealNVP layers can be stacked to form a deep neural network (Fig. 1f). The resulting flow transformation can be trained to represent any continuous nonlinear variable transformation, and it is yet fully invertible (Fig. 1f).

In other application areas, invertible networks are trained with maximum likelihood (ML) [2]. Following this idea, we would use some initial simulation data, e.g. short MD simulations in one or a few metastable states, that are not (yet) Boltzmann distributed as an input. Then we train F_{xz} such that z will be distributed according to a Gaussian. This can be done by sampling configurations $\mathbf{x}$ from the input data, transforming each of them to the latent space by $\mathbf{z} = F_{xz}(\mathbf{x})$, and minimizing the likelihood loss:

$$J_{ML} = \mathbb{E}_{\mathbf{x}}\left[u_Z\left(F_{xz}(\mathbf{x})\right) - S_{xz}(\mathbf{x})\right]. \tag{1}$$

Here, u_Z is the "energy" of the latent space distribution, which is simple a harmonic oscillator energy when using a Gaussian distribution in $\mathbf{z}$. S_{xz} quantifies how much the network scales the configuration

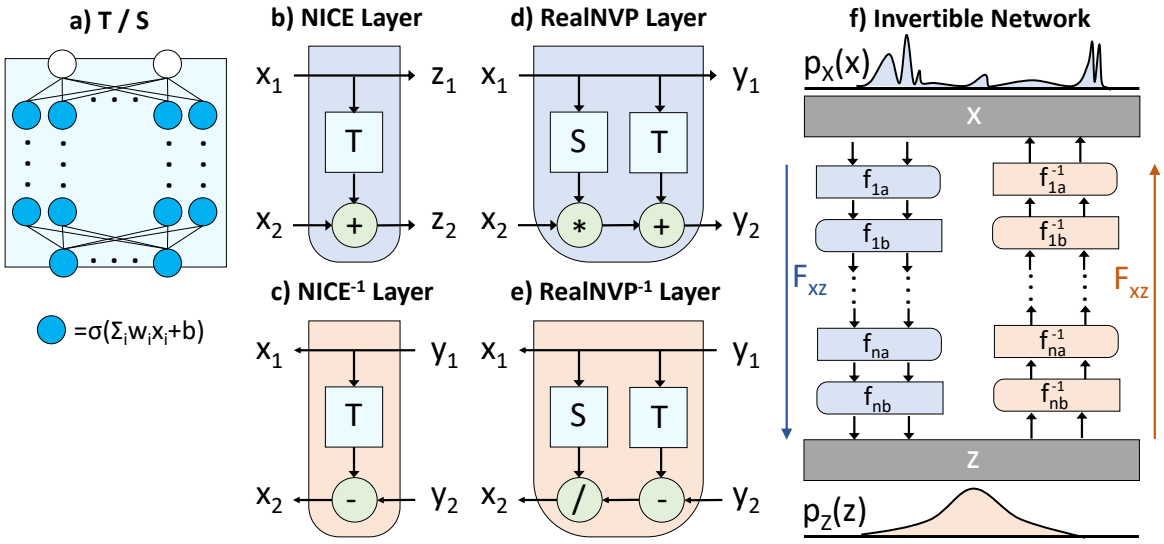

Figure 1: **Boltzmann Generator network architecture. a)** Nonlinear transformations T and S are built with multilayer neural networks. **b,c)** Volume-preserving NICE layer and its inverse. **d,e)** Non-volume-preserving RealNVP layer and its inverse. **f)** Stacking any sequence of these layers with channels exchanges produces the full Boltzmann Generator, and invertible network.

space volume at $\mathbf{x}$ and can be computed from the scaling transformations of the network (SI). S_{xz} is zero for volume-preserving NICE layers.

The unique capability of Boltzmann Generators is that they can also be trained in the reverse direction: We sample Gaussian normal variables $\mathbf{z}$, transform them through the network T_{zx} and try to minimize the difference between the generated distribution and the Boltzmann distribution. This leads to minimizing the following loss, which has the same principal form:

$$J_{KL} = \mathbb{E}_{\mathbf{z}\sim q_Z(\mathbf{z})}\left[u_X(F_{zx}(\mathbf{z})) - S_{zx}(\mathbf{z})\right] \tag{2}$$

In both (1-2), the first term is minimizing the mean potential energy, i.e. the enthalpy of the system, and the second term is maximizing the entropy of the transformation, which avoids that the network only learns to sample the minimum energy configuration. Still, minimizing only J_{KL} tends to focus sampling on the most stable metastable state (Suppl. Fig. 5,6). We found it beneficial to combine both losses, in particular it is useful to start the training by minimizing J_{ML} using a small set of simulation data in order to initialize T_{zx} to point to the relevant part of state space. Finally, it may not always be desired to sample according to the Boltzmann distribution, for example when we want to sample transition states with low probability. For this reason, we introduce an additional loss term that enhances sampling along specific coordinates called reaction coordinate (RC) loss and that will be used for certain applications (SI).

3 Results

We first illustrate Boltzmann Generators using a two-dimensional potential that has two stable states in x_1-direction separated by a high energy barrier, while it is a harmonic oscillator in x_2 (Fig. 2a). MD simulations will oscillate close to one of the stable state for a long time before a rare transition event occurs (Fig. 2b). Hence, the distribution in configuration space (x, y) is split into two modes (Fig. 2c, transition state ensemble is shown in yellow for clarity but not used for training). We are training Boltzmann Generators using the two short simulations shown in Fig. 2b as input, either by just minimizing (2-1) or with additional RC loss, which adds a term to promote sampling along x_1 (SI). Fig. 2d,e show the latent space learned by non-volume-preserving and a volume-preserving transformation, respectively. In both cases, the probability densities of the two states and the transition state are "repacked" so as to form a compact density close to the center.

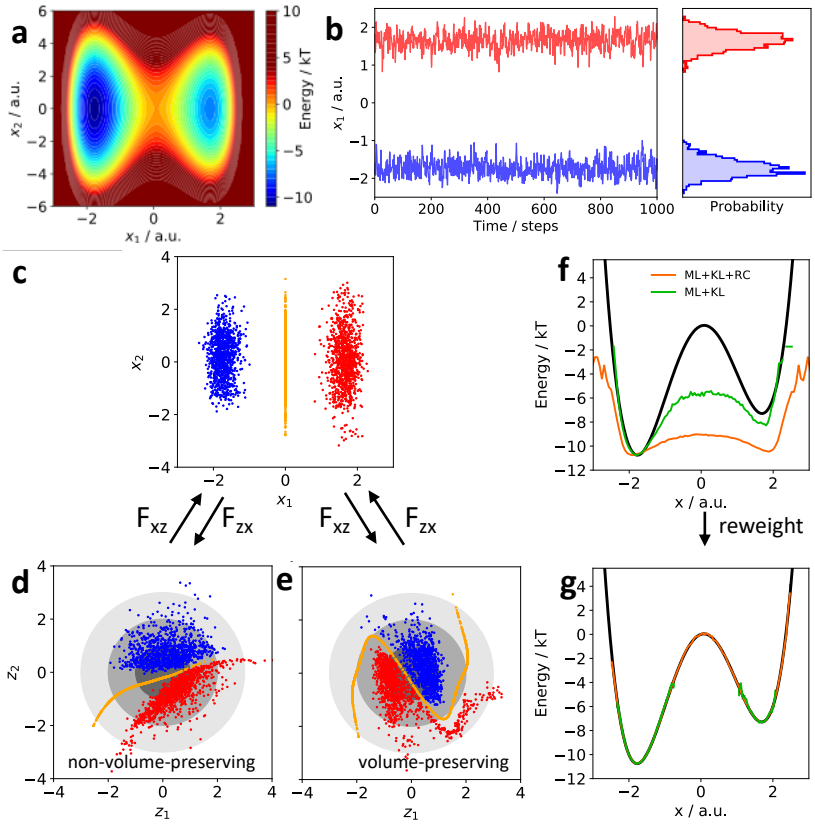

Figure 2: **Illustration of Boltzmann Generators for two-dimensional bistable system. a)** Two-dimensional potential, x_1 is the slow coordinate. **b)** Distribution of training data in x-space and in z-space after training. **c)** Distribution of trajectories of b) in configuration space (x_1, x_2). Transition state ensemble is shown (orange) but not used for training. **d,e)** Latent-space distribution of trajectories of b) when mapped through F_{xz} using non-volume preserving (d) and volume-preserving (e) transformations. **f)** Free energy corresponding to distribution sampled by Boltzmann Generators trained mainly with Eq. (2) (green) and using reaction coordinate training (orange). **g)** Free energy estimates obtained after reweighting, colors as in (f).

We use the Boltzmann generator by sampling from its latent space according to the Gaussian distribution. After transforming these variables via T_{zx}, this produces uncorrelated samples from both stable states without any sampling problem. A variety of training methods (2-1) succeed in sampling across the barrier such that the rare event nature of the system is eliminated (SI). Using simple reweighting, this procedure reproduces the precise free energy differences of the two metastable states (Fig. 2g green). By additionally training with the RC loss, the low-probability transition states are sampled (Fig. 2f orange), and reconstruct the full free energy profile along x_1 can be reconstructed with high precision (Fig. 2f,g orange).

As a second example, we demonstrate that Boltzmann Generators can sample high-probability structures and efficiently compute the thermodynamics in crowded condensed matter systems. For this we simulated a dense system of two-dimensional particles confined to a box as suggested in [13] (Fig. 3a). Particles repel each other with the 12^{th} power of their inverse distance when being close. A bistable particle dimer is immersed in the fluid. The open and closed states of the dimer are separated by a high barrier (Fig. 3a-c). Directly opening or closing the dimer is not possible due to the high density of the system and requires a concerted rearrangement of the solvent particles. As a result of this setup, this system's low-energy/high-probability states are organized on a complex and thin nonlinear manifold in the 76-dimensional configuration space - almost every vector of length 76 corresponds to a

physically unrealistic configuration as clashing particles immediately cause gigantic potential energies and thus vanishing probabilities. Designing a sampling method by hand that would simultaneously place all 38 particles and achieve low energies and appears unfeasible.

We train a Boltzmann Generator to sample valid one-shot configurations with low energy and use it in order to compute the free energy profiles of opening / closing the dimer. The training is initialized with separate, disconnected simulations of the open and closed states, but in later stages, mostly the KL-loss (2) is used, i.e. training is guided by the potential energy function that defines the system (SI). A restraint keeps the bistable particle dimer centered and aligned in the simulation box, therefore the x-position of each dimer particle indicates if we are in the open or closed state (Fig. 3d). The trained Boltzmann Generator has learned a transformation of the complex configuration space density to a compact, 76-dimensional ball in latent space (Fig. 3e). Direct sampling of from 76-dimensional Gaussian in latent space and transformation via T_{zx} not only generates realistically looking configurations where all particles are placed without significant clashes, but also samples potential energies that overlap with the energy distribution of MD trajectories started in open and closed states (Fig. 3f-h). The sampling procedure generates open and closed states without any rare event sampling problem and also produces realistic transition states that have not been included in any training data (Fig. 3g).

In order to demonstrate that thermodynamic quantities can be computed with Boltzmann Generators, we conduct the training using the KL loss (2) simultaneously to a range of temperatures between one fourth and four times the reference temperature. For this training we exploit that the temperature, which changes the configuration space distribution in a complex way, simply enters as a scaling factor in the width of the Gaussian in latent space (SI). Then, we sample the Boltzmann Generator for a range of temperatures and use simple reweighting to compute the free energies along the dimer distances. As shown in Fig. 3i, these temperature-dependent free energies agree precisely with extensive umbrella sampling simulations that employ bias potentials along the dimer distance [16].

Finally, we demonstrate that Boltzmann Generators can sample complex polymer structures. Cyclical polymers are especially challening, because for each change of a torsion angle, other torsion angles must be changed so as to maintain ring closure and all bond angle constraints. Sophisticated Monte Carlo moves have been designed for this purpose [3], but they are so complex that calculating the Jacobian of the transformation is likely untractable, and it is thus unknown how to construct an MCMC procedure that would sample the Boltzmann distribution. Sampling thus relies on standard methods, such as replica-exchange MD that is used here in order to generate initial and reference data.

With Boltzmann Generators, valid low-energy structures of cyclical hydrocarbons can be generated in one shot. Using a short replica-exchange MD simulation of cyclononane (C_9H_{18}) as input in which 4 distinct rotamers have been sampled, one-shot samples from the Boltzmann Generator produce structures with both known and new rotamers (Fig. 4a, e-j). Potential energies over generated structures have a high overlap with the potential energy sampled in the MD simulation (Fig. 4b). Note that all atoms – including hydrogens – are generated in one shot. Nontheless, the bond lengths and bond angles follow their equilibrium distribution closely (Fig. 4c,d).

Finally, one-shot samples were generated for bicyclo[4.4.4]tetradecane $C_{14}H_{26}$, a highly constrained and densely packed hydrocarbone with two interconnected ring systems and 120 dimensions (Fig. 4k,l and insets).

Boltzmann Generators are, as yet, the first approach that can sample the Boltzmann distribution and generate structures of condensed matter systems and complex polymers directly, i.e. by avoiding to make small MD or MCMC steps. We have demonstrated this for systems with up to about hundred dimensions. Although we expect the methodology to improve futher, we do not expect one-shot sampling to scale to systems of 100,000's of dimensions, such as atomistic models of proteins in solution. The present approach could still be applied to these systems by executing cluster Monte Carlo moves, i.e. by grouping the system into clusters of atoms and re-sample them according to their marginal distribution in the environment of the remaining system.

The present work shows that Boltzmann Generators can sample the Boltzmann distribution of complex systems in one shot and may offer a way out of the sampling problem in condensed matter systems. The limitation of the current work is that the transformation that achieves this needs to be trained using the system-specific energy. In order to make the approach general, it needs to be transferable across systems, and a promising route is the transferable machine learning practiced, e.g., in Quantum

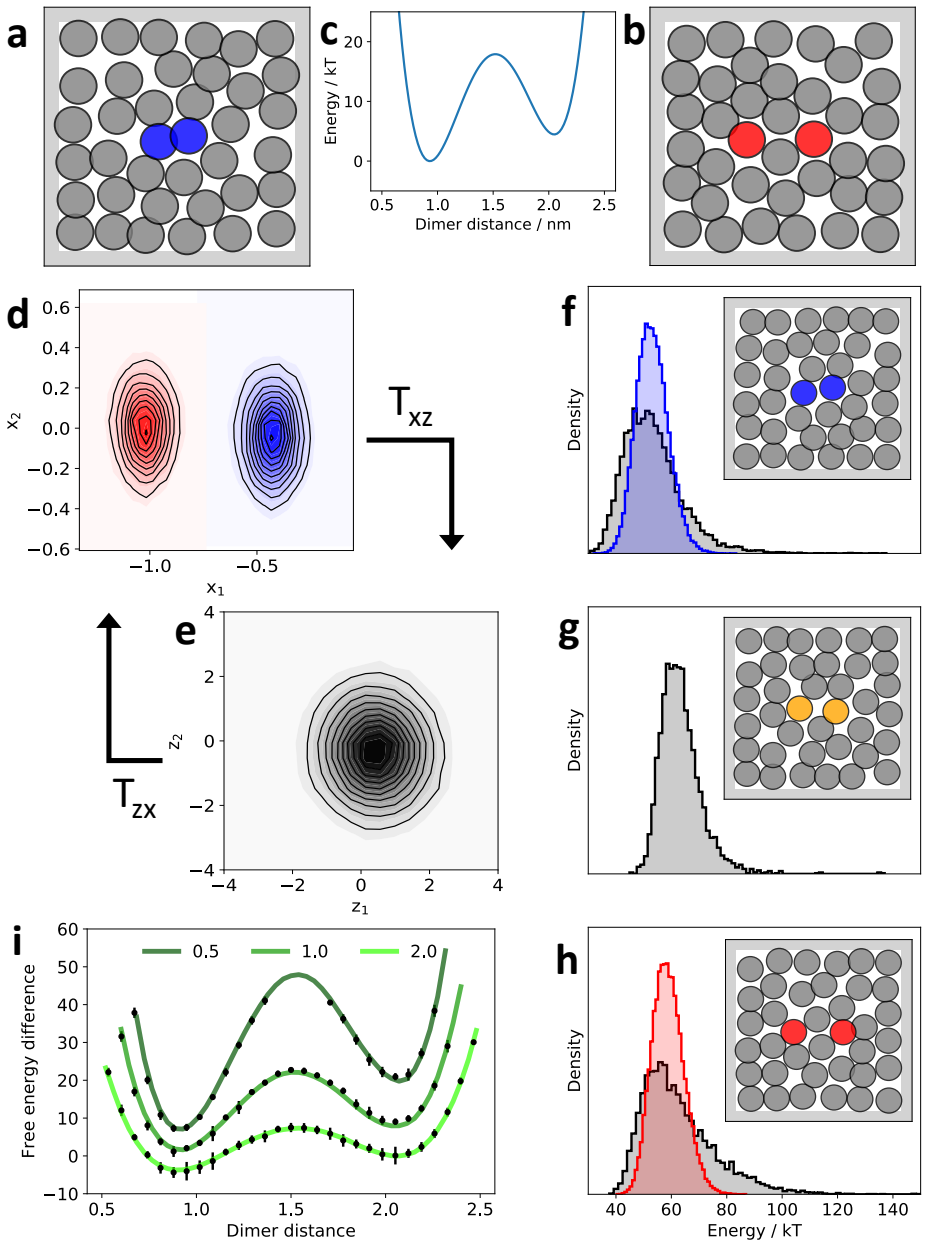

Figure 3: **Particle System with bistable dimer. a,b)** Closed (blue) and open (red) configurations from MD simulations (input data). **c)** Bistable dimer potential. **d)** Distribution of input data on x_1, x_2. e) Distribution of input data on z_1, z_2 after training. **f, g, h)** Potential energy distribution from MD (colored) and Boltzmann generator (grey) for closed (f), open (h) and transition configurations (g). Insets show one-shot samples from Boltzmann generator. **i)** Free energy differences as a function of dimer distance and temperature sampled with Boltzmann generators (one-shot generation and reweighting, bullets with error bars indicating one standard deviation) and umbrella sampling (green lines).

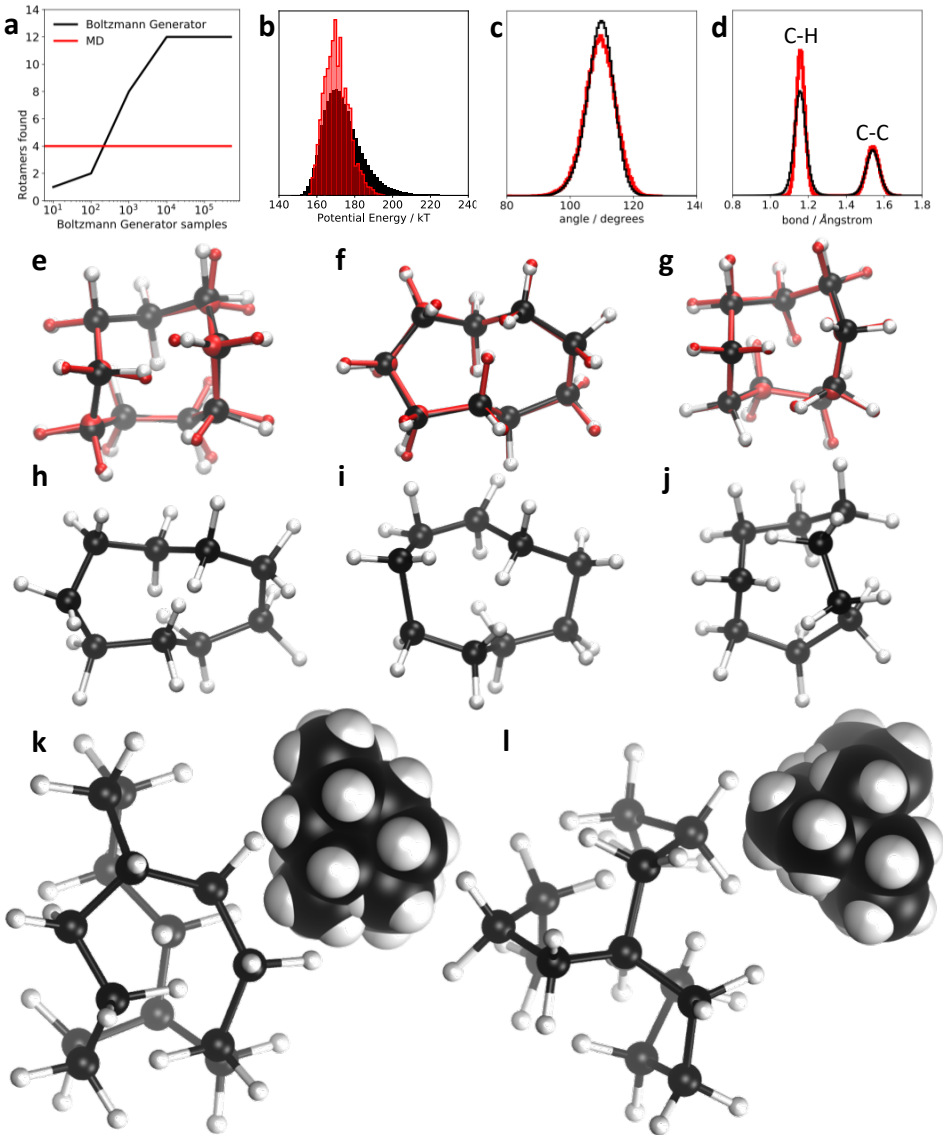

Figure 4: **One-shot sampling of cyclical molecule structures**: Cyclononane C_9H_{18} (a-j) and bicyclo[4.4.4]tetradecane $C_{14}H_{26}$ (k-l). **a)** Number of distinct rotamers samples with Boltzmann Generator that is initialized with MD data containing 12 rotamers. **b)** Potential energy distribution. **c-d)** Generated bond length and angle distribution compared to MD data. **e-g)** One-shot Boltzmann-generated structures and the most similar MD structures with free energies estimated from replica-exchange MD (red) and Boltzmann Generator (black). **h-j)** One-shot Boltzmann-generated structures that are not contained in the MD simulations, along with free energy estimates. k-l) One-shot Boltzmann-generated structures of bicyclo[4.4.4]tetradecane $C_{14}H_{26}$.

Machine Learning [1, 15].

References

[1] J. Behler and M. Parrinello. Generalized neural-network representation of high-dimensional potential-energy surfaces. *Phys. Rev. Lett.*, 98:146401, 2007.

[2] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Nonlinear independent components estimation. *ICLR*, page arXiv:1410.8516, 2015.

[3] N. Go and H. A. Scheraga. Ring closure and local conformational deformations of chain molecules. *Macromolecules*, 3:178–187, 1970.

[4] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules r gómez-bombarelli, jn wei, d duvenaud, jm hernández-lobato, ... *ACS Cent. Sci.*, 4:268–276, 2018.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and J. Bengio. Generative adversarial networks. *arXiv:1406.2661*, 2014.

[6] H. Grubmüller. Predicting slow structural transitions in macromolecular systems: conformational flooding. *Phys. Rev. E*, 52:2893, 1995.

[7] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *ICLR*, 2018.

[8] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR), arXiv:1312.6114*, 2014.

[9] S Bengio L Dinh, J Sohl-Dickstein. Density estimation using real nvp. *arXiv:1605.08803*, 2016.

[10] A. Laio and M. Parrinello. Escaping free energy minima. *Proc. Natl. Acad. Sci. USA*, 99:12562–12566, 2002.

[11] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[12] Alexei M. Nikitin, Yury V. Milchevskiy, and Alexander P. Lyubartsev. A new amber-compatible force field parameter set for alkanes. *J. Mol. Model.*, 20:2143, 2014.

[13] Jerome P. Nilmeier, Gavin E. Crooks, David D. L. Minh, and John D. Chodera. Nonequilibrium candidate monte carlo is an efficient tool for equilibrium simulation. *Proc. Natl. Acad. Sci. USA*, 108(E1009-E1018), 2011.

[14] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *SSW*, page 125, 2016.

[15] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. Von Lilienfeld. Fast and accurate modeling of molecular atomization energies with machine learning. *Phys. Rev. Lett.*, 108:058301, 2012.

[16] G. M. Torrie and J. P. Valleau. Nonphysical Sampling Distributions in Monte Carlo Free-Energy Estimation: Umbrella Sampling. *J. Comp. Phys.*, 23:187–199, 1977.

Supplementary Material

A. Invertible networks

We employ invertible networks in order to learn the transformation between the Gaussian random variables $\mathbf{z}$ and the Boltzmann-distributed random variables $\mathbf{x}$:

$$\begin{aligned}\mathbf{z} &= F_{xz}(\mathbf{x};\boldsymbol{\theta})\\ \mathbf{x} &= F_{zx}(\mathbf{z};\boldsymbol{\theta}).\end{aligned}$$

Hence $T_{xz} = T_{zx}^{-1}$. Note that the set of parameters $\boldsymbol{\theta}$ defining these transformations are identical (shared) between the forward and backward transformations. Each transformation has a Jacobian matrix with the pairwise first derivatives of outputs with respect to inputs:

$$\begin{aligned}\mathbf{J}_{zx}(\mathbf{z};\boldsymbol{\theta}) &= \left[\frac{\partial F_{zx}(\mathbf{z};\boldsymbol{\theta})}{\partial z_1}, ..., \frac{\partial F_{zx}(\mathbf{z};\boldsymbol{\theta})}{\partial z_n}\right]\\ \mathbf{J}_{xz}(\mathbf{x};\boldsymbol{\theta}) &= \left[\frac{dF_{xz}(\mathbf{x};\boldsymbol{\theta})}{dx_1}, ..., \frac{dF_{xz}(\mathbf{x};\boldsymbol{\theta})}{dx_n}\right]\end{aligned}$$

The absolute value of the Jacobian's determinant, $|\det \mathbf{J}_{zx}(\mathbf{z};\boldsymbol{\theta})|$, measures how much a volume element at $\mathbf{z}$ is scaled by the transformation. Forward and reverse transformation are related by $|\det \mathbf{J}_{zx}(\mathbf{z};\boldsymbol{\theta})| = |\det \mathbf{J}_{xz}(\mathbf{x})|^{-1}$, and respectively for $\mathbf{x}$ and $\mathbf{z}$ exchanged. As we frequently deal with logarithms of Jacobian determinants, we introduce the abbreviations:

$$\begin{aligned}S_{xz}(\mathbf{x}) &= \log|\det \mathbf{J}_{xz}(\mathbf{x})|\\ S_{zx}(\mathbf{z}) &= \log|\det \mathbf{J}_{zx}(\mathbf{z})|\,.\end{aligned}$$

Our main motivation to use invertible transformations is that they allow us to transform random variables as follows:

$$\begin{aligned}p_X(\mathbf{x}) &= p_Z(\mathbf{z})\,|\det \mathbf{J}_{zx}(\mathbf{z})|^{-1}\\ &= p_Z(T_{xz}(\mathbf{x}))\,|\det \mathbf{J}_{xz}(\mathbf{x})| \qquad (3)\\ p_Z(\mathbf{z}) &= p_X(\mathbf{x})\,|\det \mathbf{J}_{xz}(\mathbf{x})|^{-1}\\ &= p_X(T_{zx}(\mathbf{z}))\,|\det \mathbf{J}_{zx}(\mathbf{z})| \qquad (4)\end{aligned}$$

Here we employ the invertible network structures NICE [2] and RealNVP [9]. The main idea is to split the variables into two channels, $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ and $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2)$, do only trivially invertible operations on each channel, such as multiplication and addition, and use trainable, nonlinear neural network transformations between the channels to compute the value of these multiplication and addition transformations.

Table 1 summarizes the transformations employed here, their inverses and Jacobian determinant values. A single transformation of $(\mathbf{z}_1, \mathbf{z}_2) = f_{xz}(\mathbf{x}_1, \mathbf{x}_2)$, where T_{xz} is implement via NICE or RealNVP transforms only the second channel and leaves the first channel unchanged. In order to allow all variables to be transformed, we swap channels in the next transformation, and define a NICE or RealNVP block as:

$$\begin{aligned}(\mathbf{y}_1, \mathbf{y}_2) &= f_{xy}(\mathbf{x}_1, \mathbf{x}_2)\\ (\mathbf{z}_1, \mathbf{z}_2) &= f_{yz}(\mathbf{y}_2, \mathbf{y}_1)\end{aligned}$$

Boltzmann Generators are build by putting the forward and the inverse of such blocks in parallel (Fig. 1f). The forward and the inverse transformation in each layer share the same nonlinear transformation (T or S), and therefore the same parameters.

The NICE transformation is volume-preserving. As such, it also preserves the entropy $H_X = \int p(\mathbf{x}) \log p(\mathbf{x}) \mathrm{d}\mathbf{x}$. In order to be able to model probability distributions with arbitrary entropy, we need to insert at least one scaling layer into a Boltzmann Generator that otherwise only contains NICE layers.

Layer	f_{xz}	$\lvert\det \mathbf{J}_{xz}\rvert$	f_{zx}	$\lvert\det \mathbf{J}_{zx}\rvert$
NICE	$\mathbf{z}_1 = \mathbf{x}_1$ $\mathbf{z}_2 = \mathbf{x}_2 + T(\mathbf{x}_1;\boldsymbol{\theta})$	1	$\mathbf{x}_1 = \mathbf{z}_1$ $\mathbf{x}_2 = \mathbf{z}_2 - T(\mathbf{y}_1;\boldsymbol{\theta})$	1
Scaling, Exp	$\mathbf{z} = \mathrm{e}^{\mathbf{k}} \circ \mathbf{x}$	$\mathrm{e}^{\sum_i k_i}$	$\mathbf{x} = \mathrm{e}^{-\mathbf{k}} \circ \mathbf{z}$	$\mathrm{e}^{-\sum_i k_i}$
RealNVP	$\mathbf{z}_1 = \mathbf{x}_1$ $\mathbf{z}_2 = \mathbf{x}_2 \odot \exp(S(\mathbf{x}_1;\boldsymbol{\theta})) + T(\mathbf{x}_1;\boldsymbol{\theta})$	$\mathrm{e}^{\sum_i S_i(\mathbf{x}_1;\boldsymbol{\theta})}$	$\mathbf{x}_1 = \mathbf{z}_1$ $\mathbf{x}_2 = (\mathbf{z}_2 - T(\mathbf{x}_1;\boldsymbol{\theta})) \odot \exp(-S(\mathbf{z}_1;\boldsymbol{\theta}))$	$\mathrm{e}^{-\sum_i S_i(\mathbf{z}_1;\boldsymbol{\theta})}$

Table 1: Invertible network components.

B. Training Boltzmann Generators

The Boltzmann Generator is trained by minimizing a loss functional that has the following form:

$$J = w_{ML} J_{ML} + w_{KL} J_{KL} + w_{RC} J_{RC}.$$

where the terms represent maximum-likelihood (ML), Kullback-Leiber (KL) or relative entropy, and reaction-coordinate (RC) optimization and the w's control their weights. Below we will derive these terms in detail.

We call the prior distribution injected into the latent space $q_Z(\mathbf{z})$ and the Boltzmann distribution in the configuration space $\mu_X(\mathbf{x})$. The generated distributions are then called p:

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

Boltzmann distribution: We aim at sampling configurations $\mathbf{x}$ from the Boltzmann distribution

$$\mu_X(\mathbf{x}) = Z_X^{-1} \mathrm{e}^{-\beta U(\mathbf{x})} \tag{5}$$

where $\beta^{-1} = k_B T$ with Boltzmann constant k_B and temperature T. When we only have one temperature, we can simply subsume the constant into a reduced energy

$$u(\mathbf{x}) = \frac{U(\mathbf{x})}{k_B T}$$

In order to evaluate a set of temperatures $(T^1, ..., T^K)$, we can define a reference temperature T^0 and the respective reduced energy $u^0(\mathbf{x}) = U(\mathbf{x})/k_B T^0$ and we then obtain the reduced energies simply by scaling:

$$u^k(\mathbf{x}) = \frac{T^0}{T^k} u^0(\mathbf{x}) = \frac{u^0(\mathbf{x})}{\tau_k}$$

where τ_k is the relative temperature.

Prior distribution: We sample the input in $\mathbf{z}$ from the isotropic Gaussian distribution:

$$q_Z^k(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \sigma_k^2 \mathbf{I}) = Z_Z^{-1} \mathrm{e}^{-\frac{1}{2}\|\mathbf{z}\|^2/\sigma_k^2}, \tag{6}$$

with normalization constant Z_Z. The prior energy, i.e. the energy whose Boltzmann distribution is the prior distribution, is given by:

$$\begin{aligned} u_Z^k(\mathbf{z}) &= -\log q_Z^k(\mathbf{z}) \\ &= \frac{1}{2\sigma_k^2} \|\mathbf{z}\|^2 + \text{const.} \end{aligned} \tag{7}$$

Thus the variance takes the same role as the relative temperature. We define (arbitrarily) to set the variance equal 1 at the standard temperature, and obtain:

$$\sigma_k^2 = \tau_k.$$

Latent KL divergence The KL divergence between two distributions q and p is given by

$$\begin{aligned}\mathrm{KL}(q \parallel p) &= \int q(\mathbf{x}) \left[\log q(\mathbf{x}) - \log p(\mathbf{x})\right] \mathrm{d}\mathbf{x}, \\ &= H_q - \int q(\mathbf{x}) \log p(\mathbf{x}) \mathrm{d}\mathbf{x},\end{aligned}$$

where H_q is the entropy of the distribution q.

Here we use the KL divergences to minimize the difference between the probability densities predicted by the Boltzmann generator and the respective reference distribution. Using the variable transformations (3-4) and the Boltzmann distribution (5), we can express the KL divergence in latent space as:

$$\begin{aligned}\mathrm{KL}_{\boldsymbol{\theta}}\left[q_Z \parallel p_Z\right] &= H_Z - \int q_Z(\mathbf{z}) \log p_Z(\mathbf{z};\boldsymbol{\theta}) \mathrm{d}\mathbf{z}, \\ &= H_Z - \int q_Z(\mathbf{z}) \left[\log \mu_X(F_{zx}(\mathbf{z};\boldsymbol{\theta})) + \log\left|\det \mathbf{J}_{zx}(\mathbf{z};\boldsymbol{\theta})\right|\right] \mathrm{d}\mathbf{z}, \\ &= H_Z + \log Z_X + \mathbb{E}_{\mathbf{z}\sim q_Z(\mathbf{z})}\left[u(F_{zx}(\mathbf{z};\boldsymbol{\theta})) - \log\left|\det \mathbf{J}_{zx}(\mathbf{z};\boldsymbol{\theta})\right|\right]\end{aligned}$$

Here, $\boldsymbol{\theta}$ are the trainable neural network parameters. Since H_Z and Z_X are constants in $\boldsymbol{\theta}$, the KL loss is given by:

$$J_{KL} = \mathbb{E}_{\mathbf{z}\sim q_Z(\mathbf{z})}\left[u(F_{zx}(\mathbf{z};\boldsymbol{\theta})) - \log\left|\det \mathbf{J}_{zx}(\mathbf{z};\boldsymbol{\theta})\right|\right]. \tag{8}$$

Practically, each training batch samples points $\mathbf{z} \sim q_Z(\mathbf{z})$ from a normal distribution, transformes them via T_{zx}, and evaluates Eq. (8).

We can extend (8) to simultaneously train at multiple temperatures, obtaining:

$$J_{KL}^{T^1,\ldots,T^K} = \sum_{k=1}^{K} \mathbb{E}_{\mathbf{z}\sim q_Z^k(\mathbf{z})}\left[u^k(F_{zx}(\mathbf{z};\boldsymbol{\theta})) - \log\left|\det \mathbf{J}_{zx}(\mathbf{z};\boldsymbol{\theta})\right|\right].$$

Interpretation of latent KL as reweighting loss The KL loss (8) has an interesting thermodynamic interpretation. By transforming the prior distribution q_Z through the Boltzmann generator, we arrive at a proposal distribution p_X. We can now employ reweighting (see below) to turn this proposal distribution into a Boltzmann distribution. In reweighting, each point is assigned a weight

$$w_X(\mathbf{x} \mid \mathbf{z}) = \frac{\mu_X(\mathbf{x})}{p_X(\mathbf{x})} = \frac{p_Z(\mathbf{z})}{q_Z(\mathbf{z})}.$$

where the equivalence on the right hand side results from (3-4). The minimization of the latent KL divergence can be rewritten in terms of these weights:

$$\begin{aligned}\min \mathrm{KL}_{\boldsymbol{\theta}}\left[q_Z \parallel p_Z\right] &= \min \mathbb{E}_{\mathbf{z}\sim q_Z(\mathbf{z})}\left[\log q_Z(\mathbf{z}) - \log p_Z(\mathbf{z};\boldsymbol{\theta})\right] \\ &= \max \mathbb{E}_{\mathbf{z}\sim q_Z(\mathbf{z})}\left[\log w_X(\mathbf{x} \mid \mathbf{z})\right].\end{aligned}$$

Thus, the minimization of the latent KL divergence is equivalent to maximizing the expected log-weights of points, or equivalently the product of all weights, in a reweighting procedure. Indeed the maximum weights are achieved when the proposal distribution is identical to the Boltzmann distribution, resulting in $w_X(\mathbf{x}) \equiv 1$.

Interpretation of latent KL as free energy For invertible transformation F_{xz}, we additionally use the following relationship of the entropies of the two distributions:

$$\begin{aligned}
H_X &= -\int_{\mathbf{x}} p_X(\mathbf{x}) \log p_X(\mathbf{x}) \, \mathrm{d}\mathbf{x} \\
&= -\int_{\mathbf{x}} p_X(F_{zx}(\mathbf{z})) \log p_X(F_{zx}(\mathbf{z})) \left|\det \mathbf{J}_{zx}(\mathbf{z})\right| \, \mathrm{d}\mathbf{z} \\
&= -\int_{\mathbf{x}} p_Z(\mathbf{z}) \log p_X(F_{zx}(\mathbf{z})) \, \mathrm{d}\mathbf{z} \\
&= -\int_{\mathbf{x}} p_Z(\mathbf{z}) \log \left(p_Z(\mathbf{z}) \left|\det \mathbf{J}_{zx}(\mathbf{z})\right|^{-1} \right) \, \mathrm{d}\mathbf{z} \\
&= -\int_{\mathbf{x}} p_Z(\mathbf{z}) \log p_Z(\mathbf{z}) \, \mathrm{d}\mathbf{z} + \mathbb{E}_Z \log \left|\det \mathbf{J}_{zx}(\mathbf{z})\right| \, \mathrm{d}\mathbf{z} \\
&= H_Z + \mathbb{E}_Z \log \left|\det \mathbf{J}_{zx}(\mathbf{z})\right| \, \mathrm{d}\mathbf{z}
\end{aligned} \tag{9}$$

Hence we have:

$$\begin{aligned}
\mathrm{KL}_{\boldsymbol{\theta}} \left[q_Z \parallel p_Z\right] &= H_Z + \log Z_X + \mathbb{E}_{\mathbf{z}\sim q_Z(\mathbf{z})} \left[u(F_{zx}(\mathbf{z};\boldsymbol{\theta}))\right] - \mathbb{E}_{\mathbf{z}\sim q_Z(\mathbf{z})} \left[\log \left|\det \mathbf{J}_{zx}(\mathbf{z};\boldsymbol{\theta})\right|\right] \\
&= 2H_Z + \log Z_X + \mathbb{E}_{\mathbf{z}\sim q_Z(\mathbf{z})} \left[u(F_{zx}(\mathbf{z};\boldsymbol{\theta}))\right] - H_X
\end{aligned}$$

(**interpretation unclear**). The loss function becomes a free energy:

$$\begin{aligned}
J_{KL} &= \mathbb{E}_{\mathbf{z}\sim q_Z(\mathbf{z})} \left[u(F_{zx}(\mathbf{z};\boldsymbol{\theta})) - \log \left|\det \mathbf{J}_{zx}(\mathbf{z};\boldsymbol{\theta})\right|\right] \\
&= U_Z - H_{ZX}
\end{aligned}$$

with the transformation entropy:

$$H_{ZX} = H_X - H_Z$$

Configuration KL divergence Likewise, we can express the KL divergence in $\mathbf{x}$ space where we compare the generated distributions with a Boltzmann weight. Using (3-4) and the Gaussian prior density (6), this KL-divergences evaluates as:

$$\begin{aligned}
\mathrm{KL}_{\boldsymbol{\theta}}(\mu_X \parallel p_X) &= H_X - \int \mu_X(\mathbf{x}) \log p_X(\mathbf{x};\boldsymbol{\theta}) \mathrm{d}\mathbf{x} \\
&= H_X - \int \mu_X(\mathbf{x}) \left[\log q_Z(F_{xz}(\mathbf{x};\boldsymbol{\theta})) + \log \left|\mathbf{J}_{xz}(\mathbf{z};\boldsymbol{\theta})\right|\right] \mathrm{d}\mathbf{x}. \\
&= H_X + \log Z_Z + \mathbb{E}_{\mathbf{x}\sim\mu(\mathbf{x})} \left[\frac{1}{\sigma^2} \left\|F_{xz}(\mathbf{x};\boldsymbol{\theta})\right\|^2 - \log \left|\mathbf{J}_{xz}(\mathbf{x};\boldsymbol{\theta})\right|\right].
\end{aligned}$$

Although the constants H_X and Z_Z can be ignored during the training, this loss is difficult to evaluate because it needs to sample configurations according to $\mu(\mathbf{x})$, which is actually the problem we are trying to solve.

Maximum Likelihood However we can approximate the configuration KL divergence by starting from a sample $\rho(\mathbf{x})$ and using the loss:

$$\begin{aligned}
J_{LL} &= -\mathbb{E}_{\mathbf{x}\sim\rho(\mathbf{x})} \left[\log p_X(\mathbf{x};\boldsymbol{\theta})\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\rho(\mathbf{x})} \left[\frac{1}{\sigma^2} \left\|F_{xz}(\mathbf{x};\boldsymbol{\theta})\right\|^2 - \log \left|\mathbf{J}_{xz}(\mathbf{x};\boldsymbol{\theta})\right|\right]
\end{aligned}$$

This loss is the negative log-likelihood, i.e. minimizing $\mathrm{LL}_{\boldsymbol{\theta}}$ corresponds to maximizing the likelihood of the sample $\rho(\mathbf{x})$ in the Gaussian prior density.

Symmetric divergence The two KL divergences above can be naturally combined to the symmetric divergence

$$D_{JS} = \frac{1}{2} D_{KL}(p_X \parallel p_Z) + \frac{1}{2} D_{KL}(p_Z \parallel p_X),$$

which corresponds, up to an additive constant, to the Jensen-Shannon divergence which uses the geometric mean of $m = \sqrt{p_X p_Z}$ instead of the arithmetic mean.

Reaction coordinate loss In some applications we do not want to sample from the Boltzmann distribution but promote the sampling of high-energy states in a specific direction of configuration space, for example in order to compute a free energy profile along a predefined reaction coordinate $R(\mathbf{x})$ (Fig. 2g). This is achieved by adding the reaction-coordinate (RC) loss to the minimization problem:

$$\begin{aligned} J_{RC} &= \int p\left(R(\mathbf{x})\right) \log p\left(R(\mathbf{x})\right) \, \mathrm{d}R(\mathbf{x}) \\ &= \mathbb{E}_{\mathbf{x}\sim p_X(\mathbf{x})} \log p\left(R(\mathbf{x})\right). \end{aligned}$$

To implement this loss, the function R is a user input, minimum and maximum bounds are given, and $p\left(R(\mathbf{x})\right)$ is computed as a batchwise kernel density estimate along between the bounds.

We can approximate the symmetrized divergence by combining the maximum likelihood and the latent KL-divergence

$$\mathrm{D}_{JS} \approx J_{LL} + J_{KL}.$$

This approximation becomes exact in the limit where we sample configurations $\mathbf{x}$ from the Boltzmann distribution.

4 Sampling the Boltzmann Density

A trained Boltzmann generator will generally sample from the Boltzmann density only approximately. The more significant problem is that the Boltzmann generator may not cover the complete configuration space. Below we describe a number of algorithms to correct the proposal density of the Boltzmann generator and to embed it into a sampling algorithm that may asymptotically sample the whole configuration space.

Reweighting The most direct way to compute quantitative statistics using Boltzmann generators is to employ reweighting of probability densities. In this framework, we assign to each generated configuration $\mathbf{x}$ the statistical weight:

$$\begin{aligned} w_X(\mathbf{x} \mid \mathbf{z}) = \frac{\mu_X(\mathbf{x})}{p_X(\mathbf{x})} &= \frac{p_Z(\mathbf{z})}{q_Z(\mathbf{z})}. \\ &\propto \mathrm{e}^{-u_X(T_{zx}(\mathbf{z})) + u_Z(\mathbf{z}) + \log\left|\det(\mathbf{J}_{zx}(\mathbf{z}))\right|} \end{aligned} \tag{10}$$

Using these weights, expectation values can be computed as

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

Latent MCMC Consider that we always propose $\mathbf{z}$ samples from the prior distribution

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}) \propto \exp\left(-\frac{1}{2\sigma^2} \left\|\mathbf{z}\right\|^2\right).$$

Our aim is to sample $\mu(\mathbf{x})$. The MCMC acceptance probability should be:

$$p_{\mathrm{acc}}(\mathbf{z}_1 \to \mathbf{z}_2) = \min\left\{1, \frac{p_Z(\mathbf{z}_2)}{p_Z(\mathbf{z}_1)} \frac{p_{\mathrm{prop}}(\mathbf{z}_2 \to \mathbf{z}_1)}{p_{\mathrm{prop}}(\mathbf{z}_1 \to \mathbf{z}_2)}\right\}$$

Using

$$p_Z(z) = J(z) p_X(F_{zx}(z))$$

with

$$J(z) = \left|\det\left(\frac{dF_{zx}}{dz}\right)\right|(z)$$

we have:

$$\begin{aligned}
p_{\mathrm{acc}}(\mathbf{z}_1 \to \mathbf{z}_2) &= \min\left\{1, \frac{J(\mathbf{z}_2)\mu(F_{zx}(\mathbf{z}_2))}{J(\mathbf{z}_1)\mu(F_{zx}(\mathbf{z}_1))} \frac{p_{\mathrm{prop}}(\mathbf{z}_1)}{p_{\mathrm{prop}}(\mathbf{z}_2)}\right\} \\
&= \min\left\{1, \frac{\mathrm{e}^{\log J(\mathbf{z}_2) - u(F_{zx}(\mathbf{z}_2))}}{\mathrm{e}^{\log J(\mathbf{z}_1) - u(F_{zx}(\mathbf{z}_1))}} \frac{\mathrm{e}^{-\frac{1}{2\sigma^2}\|\mathbf{z}_1\|^2}}{\mathrm{e}^{-\frac{1}{2\sigma^2}\|\mathbf{z}_2\|^2}}\right\} \\
&= \min\left\{1, \mathrm{e}^{\log J(\mathbf{z}_2) - u(F_{zx}(\mathbf{z}_2)) + \frac{1}{2\sigma^2}\|\mathbf{z}_2\|^2 - \log J(\mathbf{z}_1) + u(F_{zx}(\mathbf{z}_1)) - \frac{1}{2\sigma^2}\|\mathbf{z}_1\|^2}\right\}
\end{aligned}$$

In line 2 and 3 we have cancelled equal prefactors: the prefactor involved in variable transformation, e.g. $p_z(z) = |\det(S)|\,\mu(T_{zx}(z))$ for the scaled NICER network, and the constant prefactor of the Gaussian densities. This results in the check:

$$\begin{aligned}
r &\le p_{\mathrm{acc}}(\mathbf{z}_1 \to \mathbf{z}_2) \\
-\log r &\ge \log J(\mathbf{z}_1) - \log J(\mathbf{z}_2) + u(F_{zx}(\mathbf{z}_2)) - u(F_{zx}(\mathbf{z}_1)) + \frac{1}{2\sigma^2}\|\mathbf{z}_1\|^2 - \frac{1}{2\sigma^2}\|\mathbf{z}_2\|^2
\end{aligned}$$

C. Systems and Hyperparameters

Double well We define a two-dimensional toy model which is bistable in x-direction and harmonic in y-direction:

$$E(x, y) = \frac{1}{4}ax^4 - \frac{1}{2}bx^2 + cx + \frac{1}{2}dy^2$$

with $a = c = d = 1$ and $b = 6$ – see Fig. 2 for the potential in x-direction.

Network	epochs ML	epochs KL
$N_{10}S$	200	1000
R_{10}	200	1000

Particle dimer Here we simulate two-dimensional system of a particle dimer in a bath of $n_s = 36$ solvent particles. The configuration vector is defined by alternating $x-$ and $y-$ positions and starting with the two dimer particles:

$$\mathbf{x} = \left[\mathbf{x}_{1x}, \mathbf{x}_{1y}, \mathbf{x}_{2x}, \mathbf{x}_{2y}, ..., \mathbf{x}_{(n_s+2)x}, \mathbf{x}_{(n_s+2)y}\right].$$

Defining the dimer distance $d = \|\mathbf{x}_1 - \mathbf{x}_2\|$, and the Heavyside step function h, we use the potential energy:

$$
\begin{aligned}
U(\mathbf{x}) = {} & k_d(\mathbf{x}_{1x} + \mathbf{x}_{2x})^2 + k_d\mathbf{x}_{1y}^2 + k_d\mathbf{x}_{2y}^2 \\
& + \frac{1}{4}a(d-d_0)^4 - \frac{1}{2}b(d-d_0)^2 + c(d-d_0)^4 \\
& + \sum_{i=1}^{n+2} h(-\mathbf{x}_{1x} - l_{\text{box}})k_{\text{box}}(-\mathbf{x}_{1x} - l_{\text{box}})^2 + \sum_{i=1}^{n+2} h(\mathbf{x}_{1x} - l_{\text{box}})k_{\text{box}}(\mathbf{x}_{1x} - l_{\text{box}})^2 \\
& + \sum_{i=1}^{n+2} h(-\mathbf{x}_{1y} - l_{\text{box}})k_{\text{box}}(-\mathbf{x}_{1y} - l_{\text{box}})^2 + \sum_{i=1}^{n+2} h(\mathbf{x}_{1y} - l_{\text{box}})k_{\text{box}}(\mathbf{x}_{1y} - l_{\text{box}})^2 \\
& + \epsilon \sum_{i=1}^{n+1} \sum_{j=i+1, j\neq 2}^{n+2} \left(\frac{\sigma}{\|\mathbf{x}_i - \mathbf{x}_j\|} \right)^{12}
\end{aligned}
$$

where the five rows correspond to: (1) Constraints for the center and the y-position of the particle dimer, (2) particle dimer interaction, (3,4) box constraints in $x-$ and $y-$direction, (5) particle repulsion. The following parameter values were used:

Parameter	ϵ	σ	k_d	d_0	a	b	c	l_{box}	k_{box}
Value	1.0	1.1	20.0	1.5	25.0	10.0	-0.5	3.0	100.0

Boltzmann Generator training was done with $w_{KL} = 1$ and decreasing w_{ML} according to the following schedule:

Epochs	200	300	300	1000	2000
w_{ML}	100	100	100	20	*0.01*
w_{KL}	1	1	1	1	1
w_{RC}	1	5	10	10	*10*

The italic values in the last row were treated as variable hyperparameters. We then did a hyperparameter search as indicated in the table below. The hyperparameters were chosen by minimizing the estimator variance for the free energy profile along dimer distance d. Each trained network makes predictions for the free energy profile shown in Fig. 3i. Using bootstrapping the standard error over all free energies along the profile between $d = [0.5, 2.5]$ are computed, resulting in $(\epsilon_{0.5}, \epsilon_{1.0}, \epsilon_{2.0})$ for the three temperatures and $\bar{\epsilon} = \sqrt{\epsilon_{0.5}^2 + \epsilon_{1.0}^2 + \epsilon_{2.0}^2}$ as a total estimator error.

Hydrocarbons A simple molecular mechanics model including bond, angle, torsion and Lennard-Jones interactions (between all pairs) was implemented in TensorFlow. We use the parameters in [12] to modeling the alkanes shown in Fig. 4: (1) cyclononane C_9H_{18}, i.e. a ring of nine CH_2 groups, and (2) bicyclo[4.4.4]tetradecane $C_{14}H_{26}$, connected as follows:

$$
\begin{array}{ccccccccccccc}
 & & & & -CH_2- & CH_2- & CH_2- & CH_2- & & & & & \\
 & & & / & & & & & \backslash & & & & \\
CH- & CH_2- & CH_2- & CH_2- & & & & & & CH_2- & CH_2- & CH_2- & CH \\
 & & & \backslash & & & & & & & & / & \\
 & & & & -CH_2- & CH_2- & CH_2- & CH_2- & & & & &
\end{array}
$$

Architecture	nl_{layers}	nl_{hidden}	w_{ML}	w_{RC}	$\epsilon_{0.5}$	ϵ_1	ϵ_2	$\sqrt{\sum \epsilon^2}$
R_8	4	200	0.1	10.0	1.62	2.07	2.04	3.33
R_4	·	·	·	·	2.23	1.83	1.53	3.27
R_6	·	·	·	·	1.69	1.64	2.29	3.28
R_{12}	·	·	·	·	1.49	1.85	2.0	3.10
$(RN)_2$	·	·	·	·	2.01	1.59	2.56	3.62
$(RN)_7$	·	·	·	·	1.76	2.29	1.39	3.20
$(RN)_{12}$	·	·	·	·	1.44	1.66	1.99	2.96
R_8	3	200	0.1	10.0	1.51	1.97	1.64	2.97
R_4	·	·	·	·	1.41	1.59	1.78	2.77
R_6	·	·	·	·	1.49	1.73	1.76	2.88
R_{12}	·	·	·	·	1.84	1.28	2.24	3.17
$(RN)_2$	·	·	·	·	1.84	1.29	2.25	3.18
$(RN)_7$	·	·	·	·	1.65	1.72	2.07	3.16
$(RN)_{12}$	·	·	·	·	2.87	1.80	1.58	3.74
R_8	2	·	·	·	1.85	1.58	2.50	3.48
·	4	·	·	·	1.69	1.51	1.52	2.73
·	3	50	·	·	1.32	1.71	2.11	3.02
·	·	100	·	·	2.85	2.05	2.16	4.12
·	·	**200**	**0.01**	·	**1.58**	**1.33**	**1.33**	**2.45**
·	·	·	1.0	·	1.87	1.93	1.63	3.15
·	·	·	0.1	1.0	1.66	1.83	1.75	3.02
·	·	·	·	5.0	1.73	1.72	1.81	3.03
·	·	·	·	20.0	1.88	2.06	1.84	3.34

Table 2: Hyperparameter selection for the particle dimer. In the architecture, R corresponds to a RealNVP and N to a NICE double layer with channel swaps (Fig. 1). The subscript indicates the number of repetitions, e.g. R_4 = RRRR, corresponding to eight single layers. All nonlinear transformations (T, S) used dense networks with tanh activation using the given number of layers (nl_{layers}) and hidden nodes (nl_{hidden}). All networks were trained on the following range of relative temperatures: $\tau \in [0.1, 0.25, 0.5, 0.75, 1, 1.5, 2, 3, 4]$ and used $w_{KL} = 1.0$.

D. Supplementary Figures

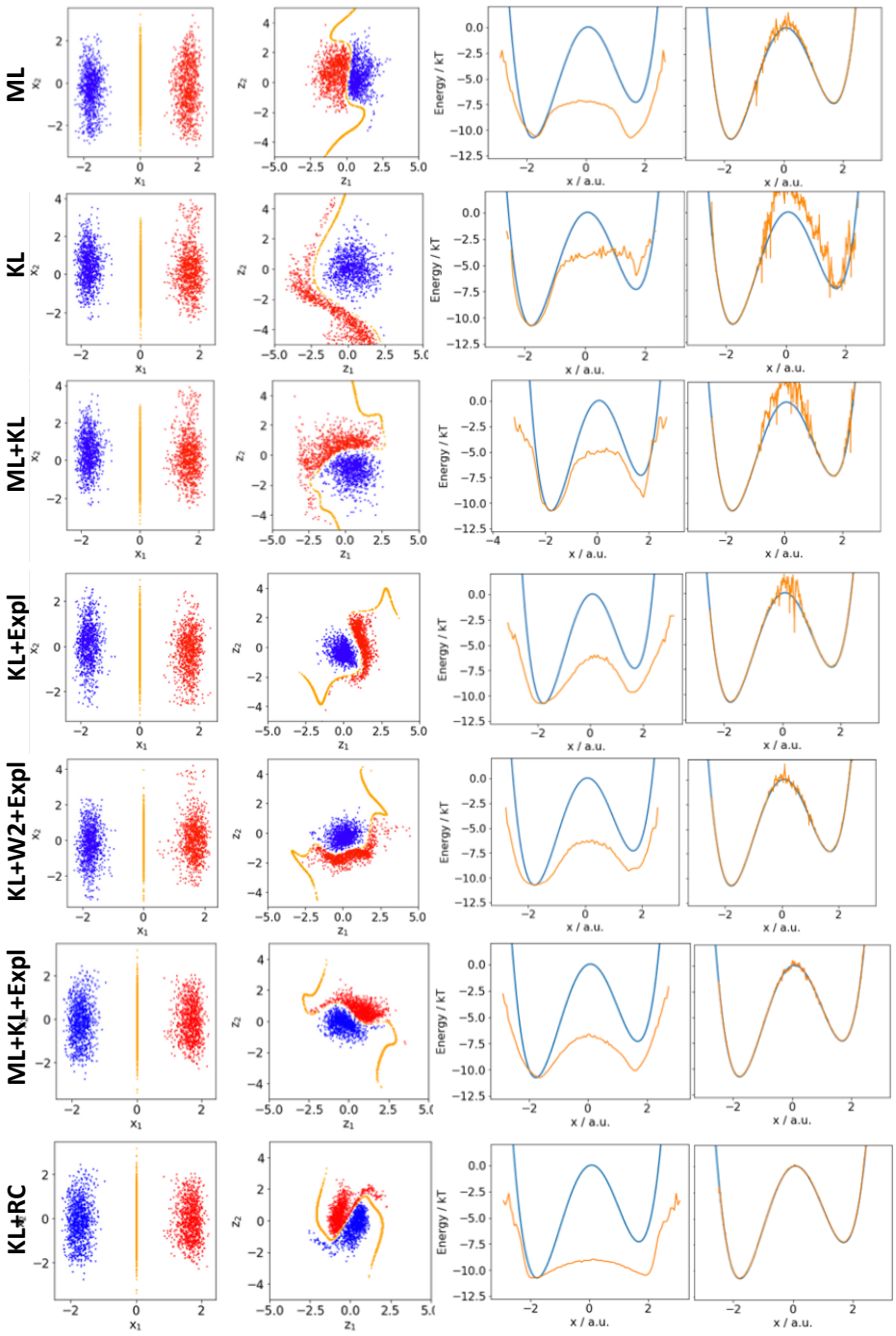

Figure 5: Different methods for training Boltzmann Generators with NICE layers using the double well example shown in Fig. 2. 1^{st} column: distribution in configuration space $\mathbf{x}$, 1^{nd} column: distribution in latent space $\mathbf{z}$, 3^{rd} column: free energy of Boltzmann Generator output $p_X(\mathbf{x})$ along x_1, 4^{th} column: free energy after reweighting. ML

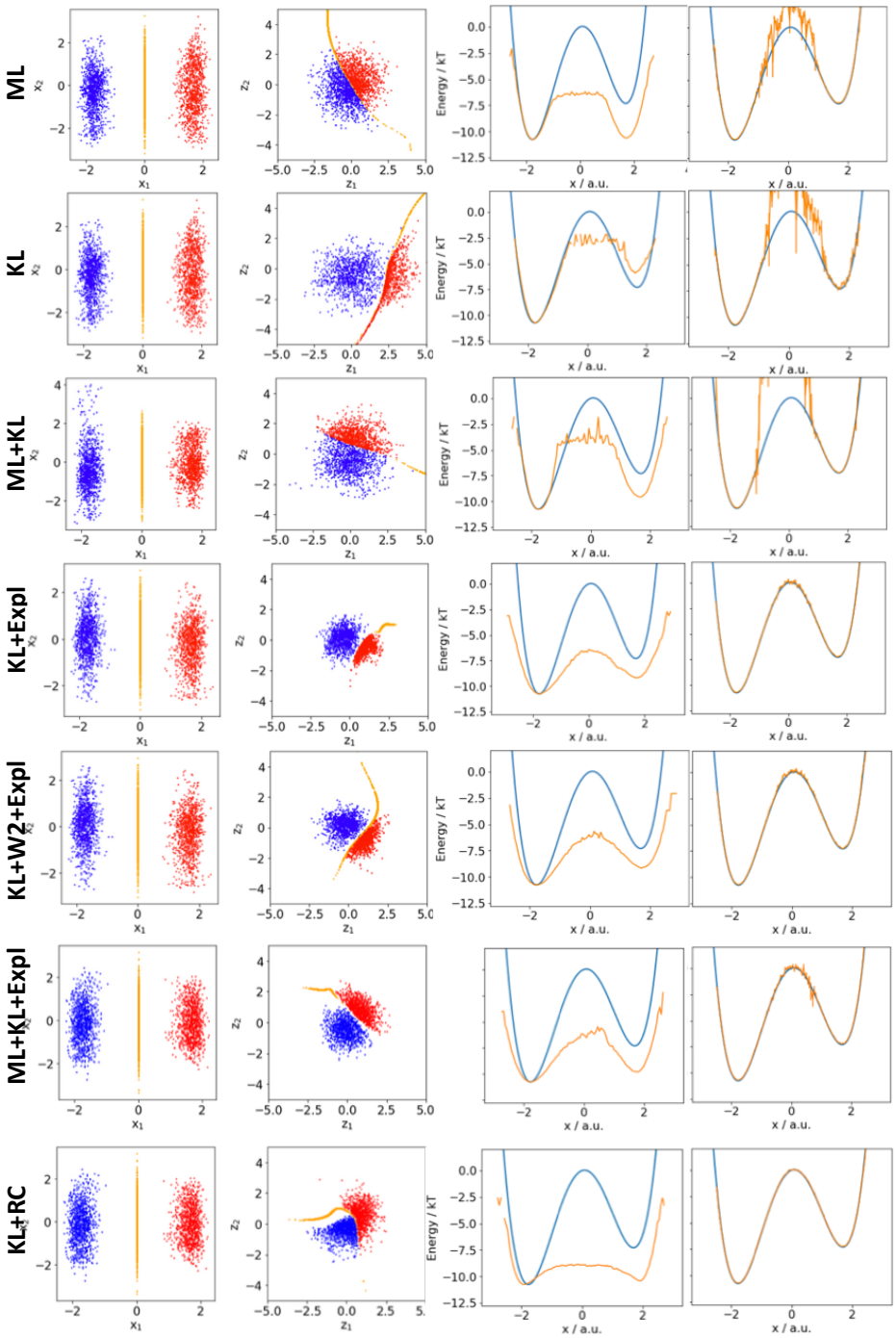

Figure 6: Same as Fig. 5 but for Boltzmann Generators using RealNVP layers.