Title: Boltzmann Generators – Sampling Equilibrium States of Many-Body Systems with Deep Learning

One Sentence Summary: Combining deep learning and statistical mechanics, we develop neural networks that sample the equilibrium distribution of complex many-body systems.

Authors: Frank Noé[1,2*] and Hao Wu[1,3]

Affiliations:

1: FU Berlin, Department of Mathematics and Computer Science, Arnimallee 6, 14195 Berlin

2: Rice University, Department of Chemistry, Houston, Texas 77005, United States

3: Tongji University, School of Mathematical Sciences, Shanghai, 200092, P.R. China

*: Correspondance to: frank.noe@fu-berlin.de

Abstract: Computing equilibrium states in condensed-matter many-body systems, such as solvated proteins, is a long-standing challenge. Lacking "one-shot" methods for generating statistically independent equilibrium samples, vast computational effort is invested for simulating these system in small steps, e.g., using Molecular Dynamics. Here we combine deep learning and statistical mechanics to develop Boltzmann Generators, that are shown to achieve "one-shot" sampling for equilibrium states of representative condensed matter systems and complex polymers. Boltzmann Generators use neural networks to learn a coordinate transformation of the complex configurational equilibrium distribution to a distribution that can be easily sampled. Accurate computation of free energy differences, and discovery of new system states are demonstrated, providing a new statistical mechanics tool that performs orders of magnitude faster than standard simulation methods.

Main Text:

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.) is proportional to $\mathrm{e}^{-u(\mathbf{x})}$, for example, the famous Boltzmann distribution. The dimensionless energy $u(\mathbf{x})$ contains the potential energy of the system, the temperature and optionally other thermodynamic quantities (SI).

Except for simple model systems, we presently have no approach to directly draw statistically independent samples $\mathbf{x}$ from Boltzmann-type distributions in order to compute statistics of the system, such as free energy differences. Therefore, one currently relies on trajectory methods, such as Markov-Chain Monte Carlo (MCMC) or Molecular Dynamics (MD) simulations that make tiny changes to $\mathbf{x}$ in each simulation step. These methods sample from the Boltzmann distribution, but many simulation steps are needed to produce a statistically independent sample. This is because complex systems often have metastable (long-lived) phases or states and the transitions between them are rare events – for example, $10^9 - 10^{15}$ MD simulation steps are needed to fold or unfold a protein. As a result, MCMC and MD methods are extremely expensive and consume much of the worldwide supercomputing resources. In specific cases, where low-dimensional coordinates can be identified that trace the rare event transitions, these can be sped up using enhanced sampling methods [1, 2, 3], but the computational effort remains enormous.

Here we set out to develop a "Boltzmann Generator" machine that is trained on a given energy function $u(\mathbf{x})$ and then produces statistically independent samples from $\mathrm{e}^{-u(\mathbf{x})}$, circumventing the sampling problem. At first sight, this enterprise seems 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 low-energy configurations are vanishingly few compared to the number of possible ways to place particles.

Key to the solution is combining the strengths of deep machine learning [4] and statistical mechanics (Fig. 1a): We train a deep invertible neural network, to learn a coordinate trans-

formation from $\mathbf{x}$ 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 high Boltzmann probability. We can improve the ability to find relevant parts of configuration space by "learning from example", where we feed the Boltzmann Generator not only with the potential energy $u(\mathbf{x})$, but also relevant samples $\mathbf{x}$, e.g., from the folded or unfolded state of a protein, but without knowing the probabilities of these states. Then we employ statistical mechanics which offers a rich set of tools to generate the target distribution $\mathrm{e}^{-u(\mathbf{x})}$ when the proposal distribution is sufficiently similar.

This paper demonstrates that Boltzmann Generators can be trained to sample low-energy structures of condensed-matter systems and complex polymer structures in "one-shot". When the Boltzmann Generator is initialized with a few structures from different metastable states, it can generate independent samples from these states and can compute the free energy difference between them without suffering from rare events. We also demonstrate that the Boltzmann Generator has a chance of generating new, previously unseen states. Exploiting this property, an "iterative discovery" procedure is constructed in which the Boltzmann Generator gradually explores the state space.

Neural networks that can draw statistically independent samples from a desired distribution are called directed generative networks [5, 6]. Such generative networks have been demonstrated to draw photorealistic images [7], to produce deceivingly realistic speech audio [8], and even to sample formulae of chemical compounds with certain physico-chemical properties [9]. In these domains, the exact target distribution is not known and the network is "trained by example" using large databases of images, audio or molecules. Here we are in the inverse situation, as we can compute the Boltzmann weight of each generated sample $\mathbf{x}$, but we do not have samples from the Boltzmann distribution *a priori*. The idea of Boltzmann Generators is as follows:

1. We learn a neural network transformation F_{zx} such that when sampling from a simple distribution in $\mathbf{z}$, such as a Gaussian normal distribution, $F_{zx}(\mathbf{z})$ will provide a configuration $\mathbf{x}$ which has a high Boltzmann weight, i.e. is coming from a distribution $p_X(\mathbf{x})$ that is similar to the target Boltzmann distribution (Fig. 1).

2. To compute Boltzmann-weighted averages, we reweight the generated distribution $p_X(\mathbf{x})$ to the Boltzmann distribution $\mathrm{e}^{-u(\mathbf{x})}$. This can be achieved with various algorithms;

here the simplest one is used: assign the statistical weight $w(\mathbf{x}) = \mathrm{e}^{-u(\mathbf{x})}/p_X(\mathbf{x})$ to every sample $\mathbf{x}$ and then compute desired statistics, such as free energy differences using this weight.

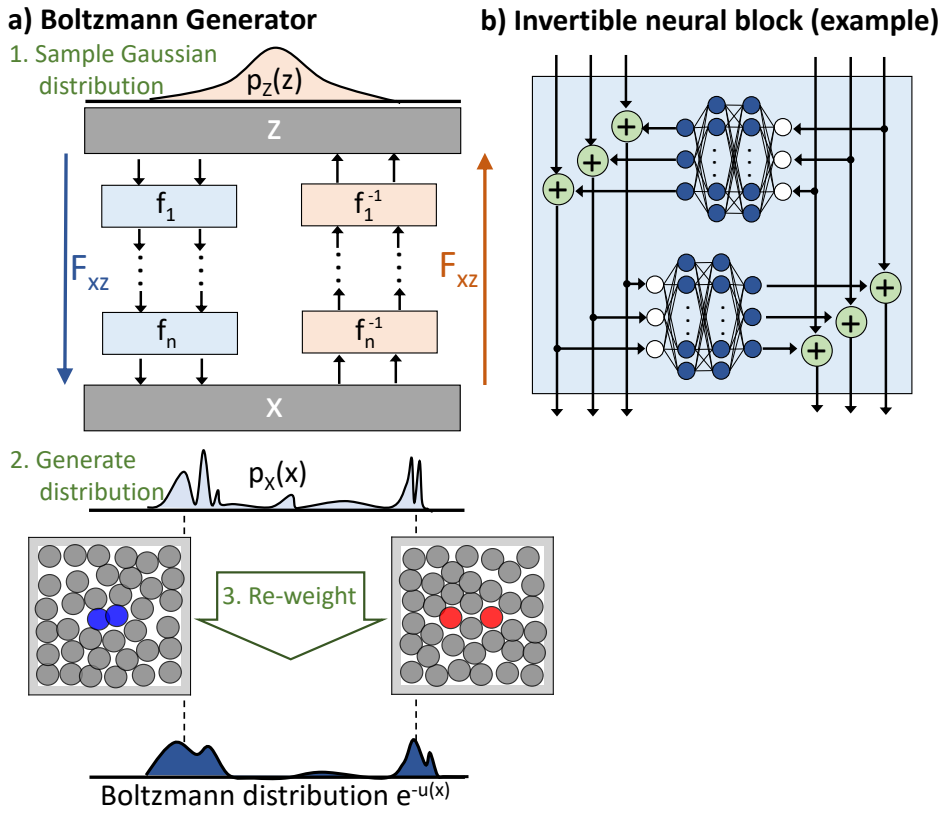


Figure 1: **Boltzmann Generators. a)** A Boltzmann Generator is trained by minimizing the difference between its generated distribution and the desired Boltzmann distribution. It is used by transforming samples from a simple (e.g., Gaussian) distribution to generated configurations. To compute thermodynamics, such as configurational free energies, the samples must be reweighted to the Boltzmann distribution. **b)** The Boltzmann Generator is composed of invertible neural network blocks. Here, a volume-preserving block is shown as an example.

For both, training and reweighting, it is important that we can compute the probability $p_X(\mathbf{x})$ of generating a configuration $\mathbf{x}$. This can be achieved when F_{zx} is an invertible transformation, for which we can compute $p_X(\mathbf{x})$ from the known $p_Z(\mathbf{z})$ (Fig. 1, SI). Invertible neural network transformations are similar to flows of a fluid that transform the probability density from configuration space to latent space, or backwards. Here we consider invertible neural network blocks that are volume-preserving (as in incompressible fluids) [10], and non-volume preserving (as in compressible fluids) [11] (Suppl. Fig. 1b-e). Invertibility is achieved by special neural network architectures (Fig. 1b, Suppl. Fig. 1c,e; see SI for details). Invertible blocks can be

stacked in various configurations to form a deep invertible neural network (Fig. 1a, Suppl. Fig. 1f). At least one non-volume preserving layer must be included so that the network is able to represent distributions with arbitrary "widths", or entropies.

Boltzmann Generators are trained with a combination of two modes: *training by energy* and *training by example*. Training by energy is the main principle behind Boltzmann Generators, and proceeds as follows: We generate random vectors $\mathbf{z}$ sampled from a Gaussian distribution, and then transform them through the neural network to proposal configurations, $\mathbf{x} = F_{zx}(\mathbf{x})$. In this way, the Boltzmann Generator will generate configurations from a proposal distribution $p_X(\mathbf{x})$, which, initially will be very different from the Boltzmann distribution, and include structures with very high energies. Next we compute the difference between the generated distribution $p_X(\mathbf{x})$ from $\mathrm{e}^{-u(\mathbf{x})}$, which is – up to a constant – equal to the distribution we want to generate. For Boltzmann Generators, a natural way to compute this difference is the relative Entropy, also known as Kullback-Leibler (KL) divergence. As derived in the SI, the KL divergence can be computed as the following expectation value over samples $\mathbf{z}$:

$$J_{KL} = \mathbb{E}_{\mathbf{z}} \left[u(F_{zx}(\mathbf{z})) - \log R_{zx}(\mathbf{z}) \right] \tag{1}$$

Here, $u_X(F_{zx}(\mathbf{z}))$ is the energy of the generated configuration. R_{zx} measures how much the network scales the configuration space volume at $\mathbf{z}$, and therefore equals one for volume-preserving network blocks, while it can be easily computed for non-volume-preserving network blocks (SI). In order to train the Boltzmann Generator, we approximate J_{KL} using a few thousand samples, and then change the neural network parameters so as to decrease J_{KL}. A few hundred or thousand such iterations are required to train the Boltzmann Generator for the examples in this paper. The resulting few million computations of the potential energy in Eq. (1) are the main computational investment to train the Boltzmann Generator and take several minutes for each system studied here.

As shown in the SI, minimizing the KL divergence (1) is equivalent to minimizing the free energy of the generated distribution: The first term $\mathbb{E}\left[u(F_{zx}(\mathbf{z}))\right]$ is the mean potential energy, i.e. the enthalpy of the system. The second term $\mathbb{E}\left[\log R_{zx}(\mathbf{z})\right]$ can be shown to be equal to the entropic contribution to the free energy at the chosen temperature, plus a constant factor. The terms in (1) counter-play in an interesting way: the first term tries to minimize the energy,

and therefore trains the Boltzmann Generator to sample low-energy structures. The second term tries to maximize the entropy of the generated distribution, and therefore prevents the Boltzmann Generator from the so-called mode-collapse, i.e. the repetitive sampling of a single minimum-energy configuration which would minimize the first term.

Despite the entropy term in (1), training by energy alone is not sufficient as it tends to focus sampling on the most stable metastable state (Suppl. Fig. 2,3). We therefore additionally employ training by example, which is the standard training method used in other Machine Learning applications. In training by example, we initialize the Boltzmann Generator with some "valid" configurations $\mathbf{x}$, e.g., from short initial MD simulations, and train it by feeding them through F_{xz} and maximizing their likelihood in the Gaussian distribution [10]. Training by example is especially used in the early stages of training, at it helps to train F_{zx} to point to relevant parts of state space.

By combining training by energy and training by example, we can sample configurations that have high probabilities and low free energies. However, sometimes we want to generate certain states with a low probability, for example the transition states along a certain reaction coordinate (RC) along which we want to compute the free energy profile. Standard sampling methods, such as MD and MCMC, can be combined with Umbrella Sampling [1], Metadynamics [3] or Flooding [3, 2] in order to bias the sampled distribution to be more uniform along a chosen RC. For the same purpose, we introduce an RC loss that can optionally be used to enhance the sampling of a Boltzmann Generator along a chosen RC (SI).

We first illustrate Boltzmann Generators using a two-dimensional potential that has two metastable states separated by a high energy barrier in x_1-direction, while it is a harmonic oscillator in x_2 (Fig. 2a). MD simulations will in one metastable 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 and disconnected simulations shown in Fig. 2b as example. 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 around the origin.

We use the Boltzmann generator by sampling from its latent space according to the Gaussian

distribution. After transforming these variables via F_{zx}, this produces uncorrelated samples from both stable states without any sampling problem. A variety of training methods succeed in sampling across the barrier such that the rare event nature of the system is eliminated (Suppl. Figs. 2,3). Combining a Boltzmann Generator trained by energy and by example with simple reweighting reproduces the precise free energy differences of the two metastable states (Fig. 2g, green). By additionally training with the RC loss to promote sampling along x_1, the low-probability transition states are sampled (Fig. 2f, orange), and the full free energy profile along x_1 can be reconstructed with high precision (Fig. 2f,g, orange).

For the double-well system, the unbiased MD simulation needs on average $4 \cdot 10^6$ MD steps for a single return trip between the two states (SI), and about 100 such crossings are required to compute the free energy difference with the same precision as the Boltzmann Generator results shown in (Fig. 2g). The total effort of training the Boltzmann Generator (including generating the initial simulation data) corresponds to about 10^6 steps, but once this is done, statistically independent samples can be generated at no significant cost. For this simple system, the Boltzmann Generator is therefore about a factor 100 more efficient than direct simulation, but much more extreme savings can be observed for complex systems, as shown below.

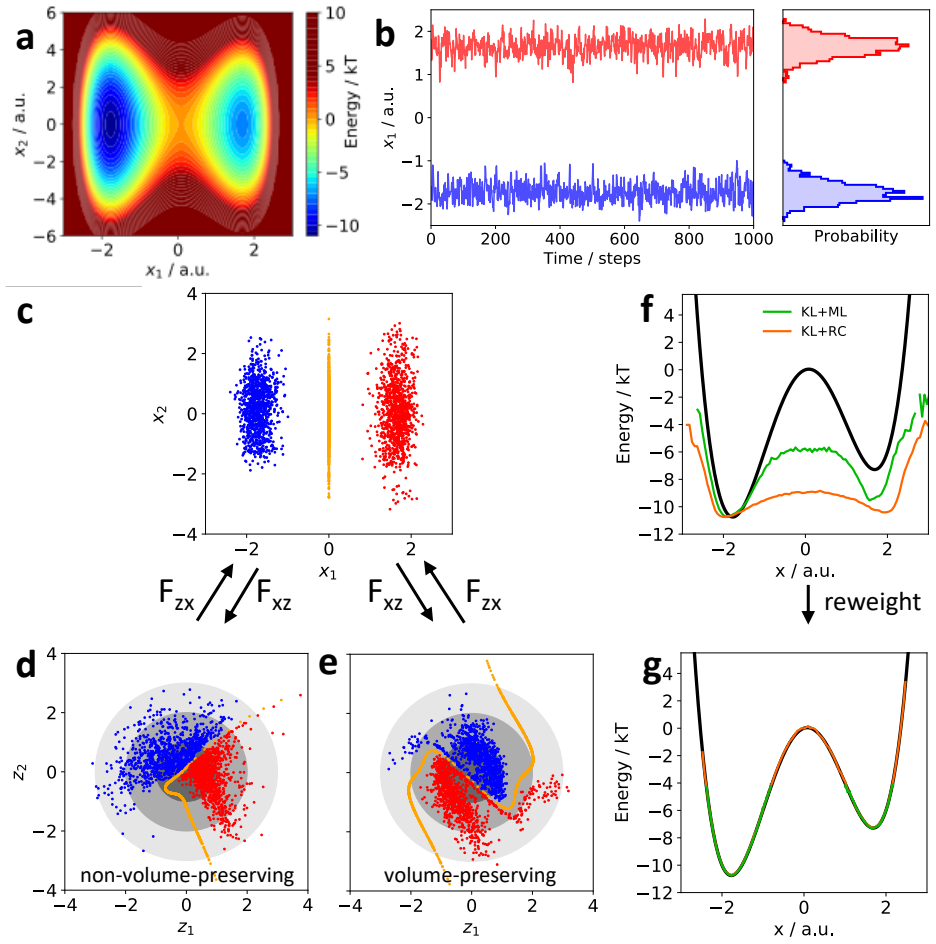


Figure 2: **Illustration of Boltzmann Generators for two-dimensional bistable system. a)** Two-dimensional potential, x_1 is the slow coordinate. **b)** Two short simulation trajectories that stay in their metastable states without crossing. **c)** Distribution of trajectories of b) in configuration 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 trained F_{xz} using transformations that are (d) non-volume preserving and (e) and volume-preserving with one global scaling factor. **f)** Free energy corresponding to distribution sampled by Boltzmann Generators trained by energy and by example (KL+ML, green) and using reaction coordinate training (KL+RC, orange). **g)** Free energy estimates obtained after reweighting, colors as in (f).

As a second example, we demonstrate that Boltzmann Generators can sample high-probability structures and efficiently compute the thermodynamics in crowded condensed matter systems. We simulated a dense system of two-dimensional particles confined to a box as suggested in [12] (Fig. 3a). Immersed in the fluid is a bistable particle dimer whose open and closed states are separated by a high barrier (Fig. 3a-c). Opening or closing the dimer directly is not

possible due to the high density of the system, but rather requires a concerted rearrangement of the solvent particles. Particles repel each other with the 12^{th} power of their inverse distance when being close. As a result, the fraction of low-energy configurations is vanishingly small, and manually designing a sampling method that simultaneously places all 38 particles and achieves low energies appears unfeasible.

We train a Boltzmann Generator to sample low-energy configurations in "one-shot" and use it in order to compute the free energy profiles of opening / closing the dimer. The training is initialized with examples from separate, disconnected simulations of the open and closed states, but in later stages, mostly training by energy (1) is used. 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 F_{zx} generates configurations where all particles are placed without significant clashes, and potential energies that overlap with the energy distribution of the unbiased MD trajectories (Fig. 3f-h). Also, realistic transition states that have not been included in any training data are sampled (Fig. 3g).

We estimate that the MD simulation needs at least 10^{12} steps to spontaneously see a single transition from closed to open state and back (SI), and about 100 such transitions would be needed to compute free energy differences with the precision of Boltzmann Generators shown in Fig. 3i. The total effort to train the Boltzmann generator is about $3 \cdot 10^7$ energy evaluations, but then statistically independent samples can be drawn at the entire temperature range trained at, resulting in about 7 orders of magnitude speedup compared to MD.

To demonstrate that thermodynamic quantities can be computed with Boltzmann Generators, we perform the training by energy (1) simultaneously to a range of temperatures between one fourth and four times the reference temperature (SI). Here, 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 $q_Z(\mathbf{z})$ (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 [1].

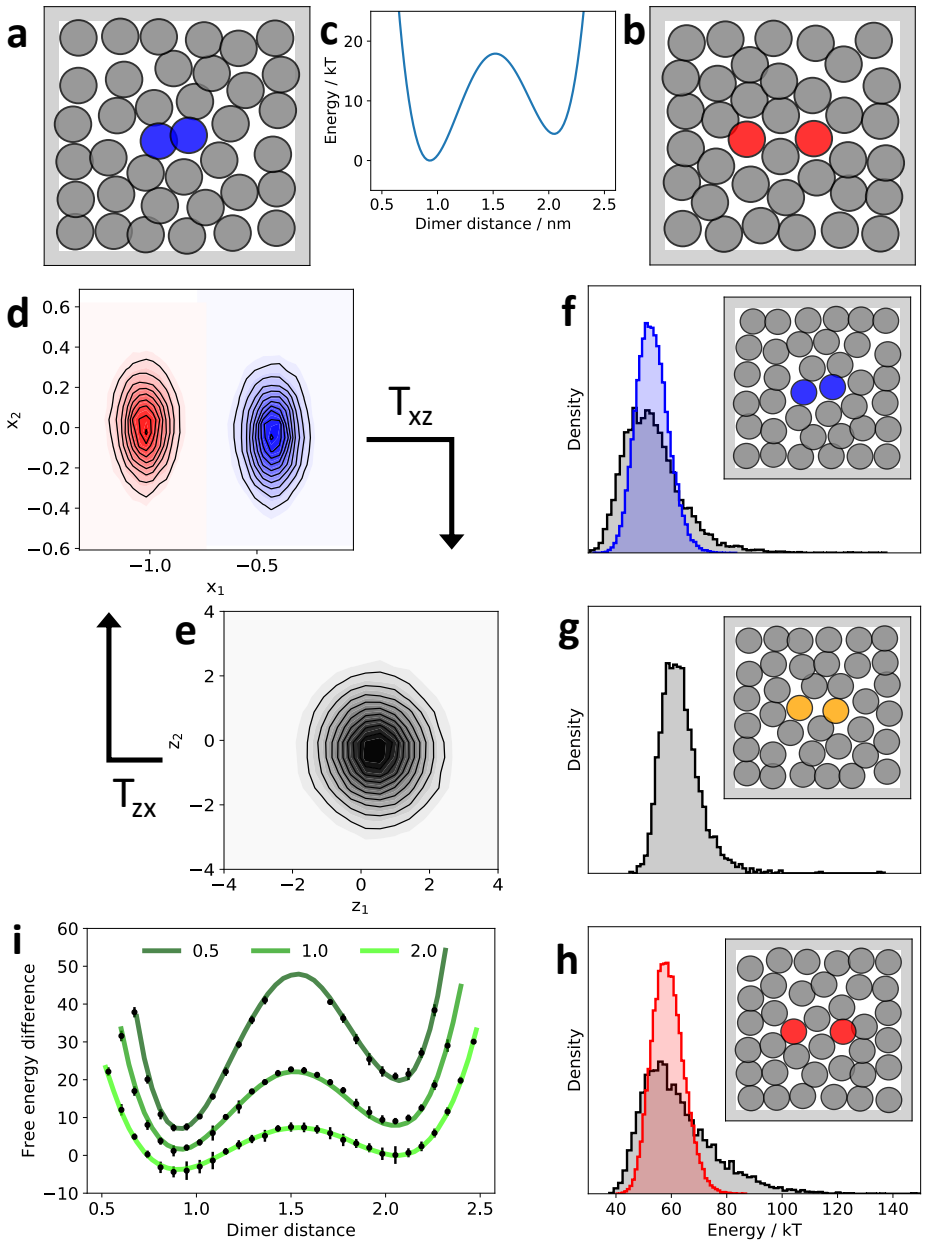


Figure 3: **Repulsive particle System with bistable dimer. a,b)** Closed (blue) and open (red) configurations from MD simulations (input data). **c)** Bistable dimer potential. **d)** Distribution of MD simulation data on x_1, x_2. e) Distribution of MD simulation data in latent space coordinates z_1, z_2 after training Boltzmann Generator. **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 relative temperature sampled with Boltzmann generators (one-shot generation and reweighting, bullets with error bars indicating one standard deviation) and umbrella sampling (green lines).

Finally, we demonstrate that Boltzmann Generators can sample complex polymer structures

in "one shot" that belong to known or new metastable states. Cyclical polymers are especially challenging, because the main degrees of freedom are torsion angles, but for each change of a torsion, other torsions must be changed concurrently so as to maintain ring closure and all bond angle constraints. Sophisticated Monte Carlo moves have been designed for this purpose [13], but they generally do not yield MCMC procedures that sample the Boltzmann distribution.

Here we using cyclical hydrocarbons as example. Each hydrocarbon torsion angle has three rotamers (around $-60°$, $60°$, $180°$), and we use "rotamer state" to denote the setting of all rotamers in the polymer. The cycle constraints stabilize some otherwise unstable conformations but generally reduce the total number of accessible rotamer states. For cyclononane (C_9H_{18}), we used a combination of training by energy and training by example, the latter initialized with a short replica-exchange MD simulation in which 4 distinct rotamer states have been sampled. We then use a Boltzmann Generator for iterative discovery: In each iteration, the Boltzmann generator samples structures from known rotamer states, and also a small fraction of structures from new rotamer states. We sample an equal number of configurations from each rotamer state found, and re-insert these samples for training by example in the next round (SI). A so-trained Boltzmann Generator quickly produces structures not included in the initial MD data (Fig. 4a, e-j). Potential energies of 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. Nonetheless, 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 hydrocarbon with two interconnected ring systems and 120 dimensions (Fig. 4k,l and insets).

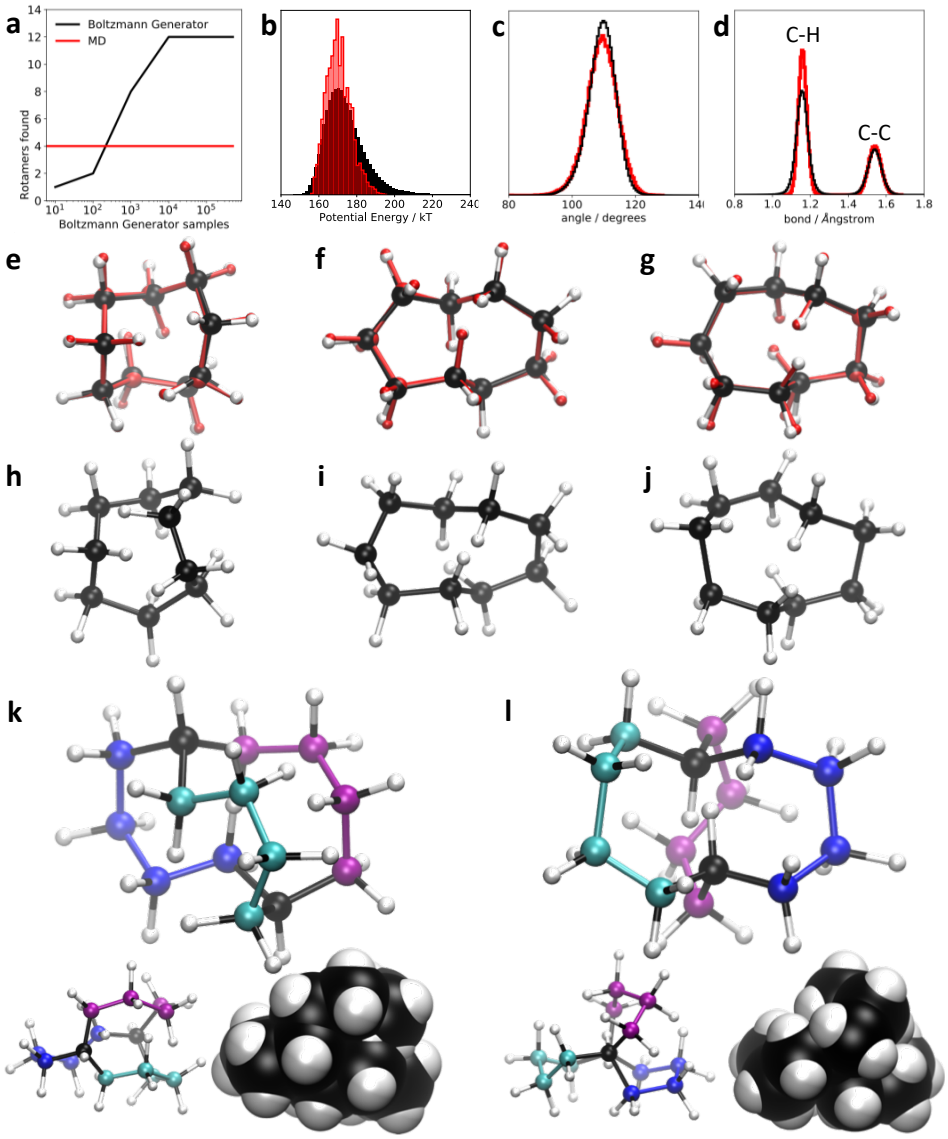


Figure 4: **Exploration of new states and one-shot sampling of cyclical molecule structures**: Cyclononane C_9H_{18} (a-j) and bicyclo[4.4.4]tetradecane $C_{14}H_{26}$ (k-l). **a)** Number of distinct rotamers sampled with Boltzmann Generator that is initialized with MD data containing 4 rotamer states. **b)** Potential energy distribution. **c-d)** Generated bond length and angle distribution compared to MD data. **e-g)** One-shot Boltzmann-generated structures (black) and the most similar structures from replica-exchange MD (red). **h-j)** One-shot Boltzmann-generated structures that are not contained in the MD simulations. **k-l)** One-shot Boltzmann-generated structures of bicyclo[4.4.4]tetradecane $C_{14}H_{26}$ – side-view is shown in large, top view below as ball+stick and space-filling representation. The three chains are colored for better visibility, the connecting carbons are kept dark.

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 around 100 dimensions. Although we expect the methodology to improve rapidly, we

believe that for very high-dimensional systems, such as solvated atomistic protein models with 100,000's of dimensions, the best strategy is to employ Boltzmann Generators for so-called cluster Monte Carlo moves. In each iteration of this approach, one would re-sample the positions of a cluster of atoms using the sum of potential energies between cluster atoms and all system atoms. With such a strategy, Boltzmann Generators can be naturally combined with existing local sampling methods.

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 become transferrable across systems, and a promising route is to employ transferrable featurization methods developed in the context of Machine Learning for Quantum Mechanics [14, 15].

References and Notes:

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

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

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

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

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, J. Bengio. Generative adversarial networks. *NIPS'14 Proceedings of the 27th International Conference on Neural Information Processing Systems*, wolumen 2, strony 2672–2680, MA, USA, 2014. MIT Press Cambridge.

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

[7] T. Karras, T. Aila, S. Laine, J. Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *ICLR*, 2018.

[8] A. van den Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. van den Driessche, E. Lockhart, L. C. Cobo, F. Stimberg, N. Casagrande, D. Grewe, S. Noury, S. Dieleman, E. Elsen, N. Kalchbrenner, H. Zen, A. Graves, H. King, T. Walters, D. Belov, D. Hassabis. Parallel WaveNet: Fast High-Fidelity Speech Synthesis. Jennifer Dy, Andreas Krause, redaktorzy, *Proceedings of the 35th International Conference on Machine Learning*, wolumen 80 serii *Proceedings of Machine Learning Research*, strony 3918–3926, Stockholmsmässan, Stockholm Sweden, 2018. PMLR.

[9] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D Hirzel, R. P Adams, A. Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Cent. Sci.*, 4:268–276, 2018.

[10] L. Dinh, D. Krueger, Y. Bengio. Nice: Nonlinear independent components estimation. *arXiv:1410.8516*, 2015.

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

[12] Jerome P. Nilmeier, Gavin E. Crooks, David D. L. Minh, John D. Chodera. Nonequilibrium candidate Monte Carlo is an efficient tool for equilibrium simulation. *Proc. Natl. Acad. Sci. USA*, 108:E1009–E1018, 2011.

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

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

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

[16] D. P. Kingma, J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[17] H. W. Kuhn. The hungarian method for the assignment problem. *Nav. Res. Logist. Quart.*, 2:83–97, 1955.

[18] A. M. Nikitin, Y. V. Milchevskiy, A. P. Lyubartsev. A new AMBER-compatible force field parameter set for alkanes. *J. Mol. Model.*, 20:2143, 2014.

Acknowledgements: FN acknowledges funding from European Commission (ERC CoG 772230 "ScaleCell"), Deutsche Forschungsgemeinschaft (CRC1114/A04), and the MATH$^+$ research cluster (AA1x8, EF1x2). We are grateful to Cecilia Clementi (Rice) and Brooke Husic (FU Berlin) for valuable comments and discussions.

Data and materials availability:

The data and code for generating the results of this paper are available at https://github.com/noegroup/paper_deep_boltzmann. A more general codebase implementing the present and other methods is available at https://github.com/noegroup/deep_boltzmann.

Supplementary Materials:

Materials and Methods

Supplementary Tables 1-2

Supplementary Figures 1-3

References 16-18

Supplementary Materials

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 $\mathbf{x}$:

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

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:

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

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 Jacobian determinants, we introduce the abbreviations:

$$R_{xz}(\mathbf{x}) = |\det \mathbf{J}_{xz}(\mathbf{x})|$$
$$R_{zx}(\mathbf{z}) = |\det \mathbf{J}_{zx}(\mathbf{z})|\,.$$

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

$$p_X(\mathbf{x}) = p_Z(\mathbf{z}) R_{zx}(\mathbf{z})^{-1} = p_Z(T_{xz}(\mathbf{x})) R_{xz}(\mathbf{x}) \tag{2}$$
$$p_Z(\mathbf{z}) = p_X(\mathbf{x}) R_{xz}(\mathbf{x})^{-1} = p_X(T_{zx}(\mathbf{z})) R_{zx}(\mathbf{z}) \tag{3}$$

Here we employ the invertible network structures NICE [10] and RealNVP [11]. 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

Layer	f_{xz}	R_{xz}	f_{zx}	R_{zx}
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	$\mathbf{z}_1 = \mathbf{x}_1$ $\mathbf{z}_2 = \mathbf{x}_2 \odot \exp\left(S(\mathbf{x}_1;\boldsymbol{\theta})\right) + T(\mathbf{x}_1;\boldsymbol{\theta})$	$\mathrm{e}^{\sum_i S_i(\mathbf{x}_1;\boldsymbol{\theta})}$	$\mathbf{x}_1 = \mathbf{z}_1$ $\mathbf{x}_2 = (\mathbf{z}_2 - T(\mathbf{x}_1;\boldsymbol{\theta})) \odot \exp\left(-S(\mathbf{z}_1;\boldsymbol{\theta})\right)$	$\mathrm{e}^{-\sum_i S_i(\mathbf{z}_1;\boldsymbol{\theta})}$

Supplementary Table 1: Invertible network components. f_{xz} and $f_{zx} = f_{xz}^{-1}$ are the forward and inverse transformations. R_{xz} and R_{zx} are the Jacobian determinants.

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 (Suppl. Fig. 1b-e).

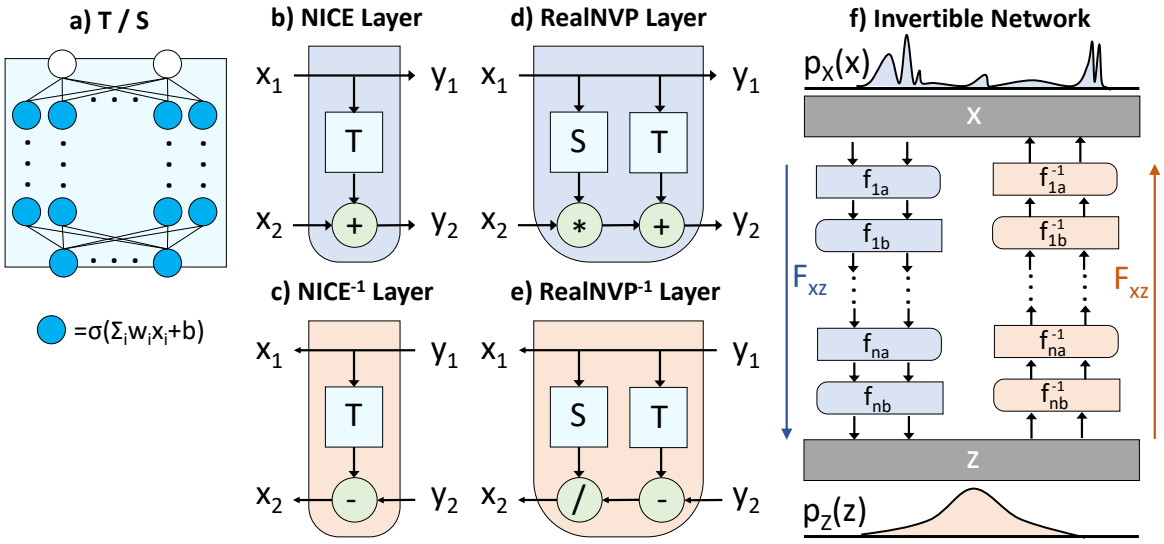
Table 1 summarizes the transformations employed here, their inverses and Jacobian determinant values. A single transformation of $(\mathbf{z}_1, \mathbf{z}_2) = f_{xz}(\mathbf{x}_1, \mathbf{x}_2)$, where T_{xz} is implement via NICE or RealNVP transforms only the second channel and leaves the first channel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

$$(\mathbf{y}_1, \mathbf{y}_2) = f_{xy}(\mathbf{x}_1, \mathbf{x}_2)$$

$$(\mathbf{z}_1, \mathbf{z}_2) = f_{yz}(\mathbf{y}_2, \mathbf{y}_1)$$

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.



Supplementary 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.

B. Training and using 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, "training by example"), Kullback-Leiber (KL, "training by energy"), and reaction-coordinate (RC) optimization and the w's control their weights. Below we will derive these terms in detail.

We call the "exact" distributions μ and the generated distributions q. In particular, $\mu_Z(\mathbf{z})$ is the Gaussian prior distribution injected into the latent space and $q_X(\mathbf{x})$ is the distribution that results from the network transformation F_{zx}. Likewise, $\mu_X(\mathbf{x}) \propto \exp(-u(\mathbf{x}))$ is the Boltzmann distribution in configuration space and $q_Z(\mathbf{z})$ is the distribution that results from the network transformation F_{xz}:

$$\mu_Z(\mathbf{z}) \xrightarrow{F_{zx}} q_X(\mathbf{x})$$
$$\mu_X(\mathbf{x}) \xrightarrow{F_{xz}} q_Z(\mathbf{z})$$

Boltzmann distribution: A special case is to use Boltzmann Generators to sample from the Boltzmann distribution of the canonical ensemble. This distribution has the form:

$$\mu_X(\mathbf{x}) = Z_X^{-1} \mathrm{e}^{-\beta U(\mathbf{x})} \tag{4}$$

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:

$$\mu_Z^k(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \sigma_k^2 \mathbf{I}) = Z_Z^{-1} \mathrm{e}^{-\frac{1}{2}\|\mathbf{z}\|^2/\sigma_k^2}, \tag{5}$$

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

$$\begin{aligned} u_Z^k(\mathbf{z}) &= -\log \mu_Z^k(\mathbf{z}) \\ &= \frac{1}{2\sigma_k^2} \|\mathbf{z}\|^2 + \text{const.} \end{aligned} \tag{6}$$

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 KL divergences to minimize the difference between the probability densities predicted by the Boltzmann generator and the respective reference distribution. Using the variable transformations (2-3) and the Boltzmann distribution (4), we can express the KL divergence in latent space as:

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

This is equivalent to the KL divergence expressed in configuration space:

$$\begin{aligned}\mathrm{KL}_{\boldsymbol{\theta}} \left[q_X \parallel \mu_X\right] &= \int q_X(\mathbf{x}; \boldsymbol{\theta}) \left[\log q_X(\mathbf{x}; \boldsymbol{\theta}) + \log Z_X + u(\mathbf{x})\right] \mathrm{d}\mathbf{x} \\ &= \int \mu_Z(\mathbf{z}) \left[\log \mu_Z(\mathbf{z}) - \log R_{zx}(\mathbf{z}; \boldsymbol{\theta}) + \log Z_X + u(F_{zx}(\mathbf{z}; \boldsymbol{\theta}))\right] \mathrm{d}\mathbf{z} \\ &= -H_Z + \log Z_X + \mathbb{E}_{\mathbf{z} \sim \mu_Z(\mathbf{z})} \left[u(F_{zx}(\mathbf{z}; \boldsymbol{\theta})) - \log R_{zx}(\mathbf{z}; \boldsymbol{\theta})\right]\end{aligned}$$

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:

$$J_{KL} = \mathbb{E}_{\mathbf{z} \sim \mu_Z(\mathbf{z})} \left[u(F_{zx}(\mathbf{z}; \boldsymbol{\theta})) - \log R_{zx}(\mathbf{z}; \boldsymbol{\theta})\right]. \tag{7}$$

Practically, each training batch samples points $\mathbf{z} \sim q_Z(\mathbf{z})$ from a normal distribution, transforms them via T_{zx}, and evaluates Eq. (7).

We can extend (7) to simultaneously train at multiple temperatures, obtaining:

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

The KL divergence $\mathrm{KL}_{\boldsymbol{\theta}}\left[\mu_Z \parallel q_Z\right]$ is also maximized in *probability density distillation* used in the training of recent audio generation networks [8]. Here, the reference distribution is defined by a teacher network that is used to help training a student network. However, the resulting expressions are different because the target distribution is not defined by a physical energy as here.

Reweighting and interpretation of latent KL as reweighting loss 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:

$$\begin{aligned} w_X(\mathbf{x}) = \frac{\mu_X(\mathbf{x})}{q_X(\mathbf{x})} &= \frac{q_Z(\mathbf{z})}{\mu_Z(\mathbf{z})}. \\ &\propto \mathrm{e}^{-u_X(T_{zx}(\mathbf{z}))+u_Z(\mathbf{z})+\log R_{zx}(\mathbf{z};\boldsymbol{\theta})} \end{aligned} \tag{8}$$

where the equivalence on the right hand side results from (2-3). Using these weights, expectation values can be computed as

$$\mathbb{E}[O] \approx \frac{\sum_{i=1}^{N} w_X(\mathbf{x}) O(\mathbf{x})}{\sum_{i=1}^{N} w_X(\mathbf{x})}. \tag{9}$$

All free energy profiles shown in Figs. 2, 3 and Suppl. Figs. 2, 2 were computed by $-k_B T \log p(R(\mathbf{x}))$ where $p(R(\mathbf{x}))$ is a probability density computed from a weighted histogram of the coordinate $R(\mathbf{x})$ using the weighted expectation (9). All histogram weights that have weights worth less than 0.01 samples are discarded to avoid making unreliable predictions.

With the reweighting (8), the KL loss (7) has an interesting thermodynamic interpretation.

The minimization of the latent KL divergence can be rewritten in terms of these weights:

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

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:

$$\begin{aligned}H_X &= -\int_{\mathbf{x}} q_X(\mathbf{x}) \log q_X(\mathbf{x})\, \mathrm{d}\mathbf{x} \\ &= -\int_{\mathbf{z}} q_X(F_{zx}(\mathbf{z})) \log \left(q_X(F_{zx}(\mathbf{z}))\, R_{zx}(\mathbf{z})\right)\, \mathrm{d}\mathbf{z} \\ &= -\int_{\mathbf{z}} \mu_Z(\mathbf{z}) \log q_X(F_{zx}(\mathbf{z}))\, \mathrm{d}\mathbf{z} \\ &= -\int_{\mathbf{z}} \mu_Z(\mathbf{z}) \log \left(\mu_Z(\mathbf{z}) R_{zx}(\mathbf{z})^{-1}\right)\, \mathrm{d}\mathbf{z} \\ &= -\int_{\mathbf{z}} \mu_Z(\mathbf{z}) \log \mu_Z(\mathbf{z})\, \mathrm{d}\mathbf{z} + \mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})} \log R_{zx}(\mathbf{z})\, \mathrm{d}\mathbf{z} \\ &= H_Z + \mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}\left[\log R_{zx}(\mathbf{z})\right] \qquad (10)\end{aligned}$$

Hence we have:

$$\begin{aligned}\mathrm{KL}_{\boldsymbol{\theta}}\left[\mu_Z \parallel q_Z\right] &= -H_Z + \log Z_X + \mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}\left[u(F_{zx}(\mathbf{z};\boldsymbol{\theta}))\right] - \mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}\left[\log R_{zx}(\mathbf{z};\boldsymbol{\theta})\right] \\ &= -H_X + \log Z_X + \mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}\left[u(F_{zx}(\mathbf{z};\boldsymbol{\theta}))\right] \\ &= -H_X + \log Z_X + \mathbb{E}_{\mathbf{x}\sim\mu_X(\mathbf{x};\boldsymbol{\theta})}\left[u(\mathbf{x})\right] \\ &= \mathrm{KL}_{\boldsymbol{\theta}}\left[q_X \parallel \mu_X\right].\end{aligned}$$

The loss function becomes:

$$\begin{aligned} J_{KL} &= \mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}\left[u(F_{zx}(\mathbf{z};\boldsymbol{\theta})) - \log R_{zx}(\mathbf{z};\boldsymbol{\theta})\right] \\ &= U - H_X + H_Z \end{aligned}$$

which is, up to the constant H_Z equal to the free energy of the generated distribution with enthalpy U and entropic factor H_X. Note that this entropic factor is taken at the temperature used for generating the distribution, a temperature dependence will enter when training the Boltzmann Generator at multiple temperatures.

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 (2-3) and the Gaussian prior density (5), this KL-divergences evaluates as:

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

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:

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

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

$$\mathrm{KL}_{\mathrm{sym}} = \frac{1}{2}\mathrm{KL}\left[\mu_X \parallel q_X\right] + \frac{1}{2}\mathrm{KL}\left[\mu_Z \parallel q_Z\right]$$

which corresponds, up to an additive constant, to the Jensen-Shannon divergence which uses the geometric mean of $m = \sqrt{q_X q_Z}$ 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:

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

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 batch-wise kernel density estimate along between the bounds.

C. Systems and Hyper-parameters

The "MD" simulations of the systems below are not using actual dynamics, but are emulated with Metropolis Monte Carlo with small local steps. In each step, a random vector from an isotropic Gaussian distribution with a system-dependent standard deviation σ_{Metro} is added to the present configuration. This proposed configuration is accepted or rejected with a standard Metropolis acceptance criterion.

All Boltzmann Generator networks are build of invertible blocks using NICE or RealNVP layers. Each block contains two such layers to make sure that all dimensions are subject to a nonlinear transformation, as described in SI Section A. Each configuration $\mathbf{x}$ or latent vector $\mathbf{z}$ is split into a channel of "even" and "odd" dimensions, defining the pairs $(\mathbf{x}_1, \mathbf{x}_2)$ and

$(\mathbf{z}_1, \mathbf{z}_2)$, respectively. To describe the network architecture used, we use N, R and S to denote NICE block, RealNVP block and Scaling layer, respectively. A subscript is used to denote the number of repetitions of a motif, e.g. N_{10} are ten stacked NICE blocks, i.e. 20 layers total, $(NR)_4$ are four repetitions of a NICE and a RealNVP block, i.e. 8 layers total.

All networks are trained using the Adam adaptive stochastic gradient descent method [16]. Other choices and hyper-parameters are described below.

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

$$E(x, y) = \frac{1}{4}ax^4 - \frac{1}{2}bx^2 + cx + \frac{1}{2}dy^2 \tag{11}$$

with $a = c = d = 1$ and $b = 6$ – see Fig. 2 for the potential in x-direction. The system is simulated with a Metropolis step of $\sigma_{\text{Metro}} = 0.1$. To estimate the average time needed for a return trip between both states, we construct another systems with $a = 0.25$ and $b = 1.5$ that has the same position of minima and the same energy difference between them, but a much smaller barrier. For the "flat" systems frequent transitions between the two end-states are observed. The return-trip time of the original system is then estimated by $t = t_{\text{flat}} \exp(B - B_{\text{flat}})$, where B, B_{flat} are the energy barriers for the original and the "flat" system from either one of the two minima, and t, t_{flat} are the times taken for a round-trip between the states. This results in an estimate of $t = 4 \cdot 10^6$ simulation steps for a return trip in the system shown in Fig. 2.

In Fig. 2 and Suppl. Figs. 2,3 we use NICE and RealNVP networks defined as below. All nonlinear transformations (T, S) used dense networks with tanh activation and two hidden layers with 100 hidden nodes. Networks are trained in two steps: in a first ML phase we only minimize J_{KL}. Subsequently we minimize $J_{\text{KL}} + J_{\text{ML}}$ with equal weights $w_{\text{ML}} = w_{\text{KL}} = 1$, unless otherwise noted.

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

Bistable particle dimer in a Lennard-Jones fluid Here we simulate two-dimensional system of a bistable particle dimer in a dense bath of $n_s = 36$ solvent particles with Lennard-Jones repulsion. A similar system has been proposed in [12]. 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 Heaviside step function h, we use the potential energy:

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

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 (all in reduced units):

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

To initialize training, we run Metropolis Monte Carlo simulations with a Metropolis step length of $\sigma_{\text{Metro}} = 0.02\sqrt{\tau}$, where τ is the relative temperature. To estimate the time taken for a return-trip between open and closed dimer states, we take the same approach as for the double-well system above: We conduct a simulation with 10^6 simulation steps for a system with maximally flattened energy ($a = 2.5$ and $b = 1.0$). Still no transition from closed to open states occur, we thus estimate the *lower bound* for the return trip to be $t = 10^6 \exp(B - B_{\text{flat}}) \approx 1.2 \cdot 10^{12}$ where B, B_{flat} are the intrinsic barrier heights for the unchanged and flattened system.

For validation of the free energy profiles predicted in Fig. 3i, we perform Umbrella Sampling simulations [1] for each relative temperature $(0.5, 1.0, 2.0)$ using 35 Umbrella potentials on the dimer distance between values of 0.5 and 2.5 and with a force constant of 500 (reduced units). Each umbrella simulation was $50,000$ steps, and to avoid hysteresis effects, we ran the umbrella sequence forward and backward, resulting in a total of $3 \cdot 70 \cdot 50,000 = 10.5$ million simulation steps.

For initializing the training by example (ML), 10^5 simulation steps are stored for the "open" and "closed" dimer states. No transitions between these states occur in the simulations. In order to not have to learn the permutational invariance of the diffusing solvent particles from the data, we remove this invariance by relabeling solvent particles using the Hungarian algorithm [17].

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

Epochs	20	200	300	300	1000	2000
w_{ML}	1	100	100	100	20	*0.01*
w_{KL}	0	1	1	1	1	1
w_{RC}	0	1	5	10	10	*10*

The italic values in the last row were treated as variable hyper-parameters. We then did a hyper-parameter search as indicated in the table below. The hyper-parameters 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.

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

Supplementary Table 2: Hyper-parameter 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 $w_{KL} = 1.0$.

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 [18] 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:

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

To generate some initial structures for training by example, we conducted a short ($14,000$ steps) replica-exchange simulation with relative temperatures $\tau = (1, 1.5, 2, 2.5, 3, 3.5, 4)$ where 300 Kelvin is the standard temperature. We used Metropolis-Monte Carlo step length $\sigma_{\text{Metro}} = 0.01\,\text{Å}$. The main aim of this simulation is to equilibrate the structure which is started from a random placement of atoms in space. The first $10,000$ steps were discarded and the last $4,000$ steps were retained for training.

The Boltzmann Generator used R_{10} (20 layers) as an architecture, All nonlinear transformations (T, S) used dense networks with tanh activation and four hidden layers with 100 hidden nodes. The Boltzmann Generator is trained using iterative discovery. In each of 50 iterations, we use $4,000$ configurations for training by example, and we train using the following schedule:

	First Iteration			Next Iterations
Epochs	300	300	300	300
w_{ML}	1	1	1	1
w_{KL}	0	0.01	0.1	0.1

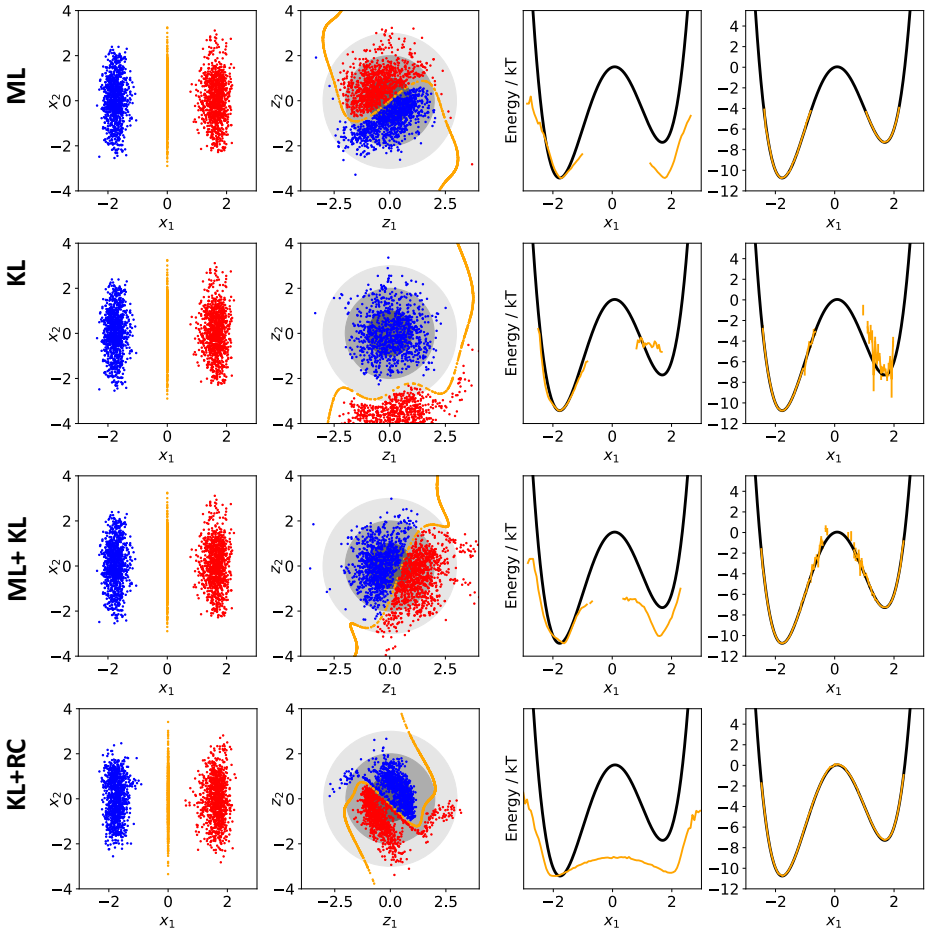
At the end of each iteration, we use the Boltzmann Generator to sample 10^6 structures and make a list of all rotamer conformations that have been generated. A rotamer is defined by discretizing each torsion as follows:

Rotamer	1	2	3
Angle	-120°...0°	0°...120°	120°...-120°

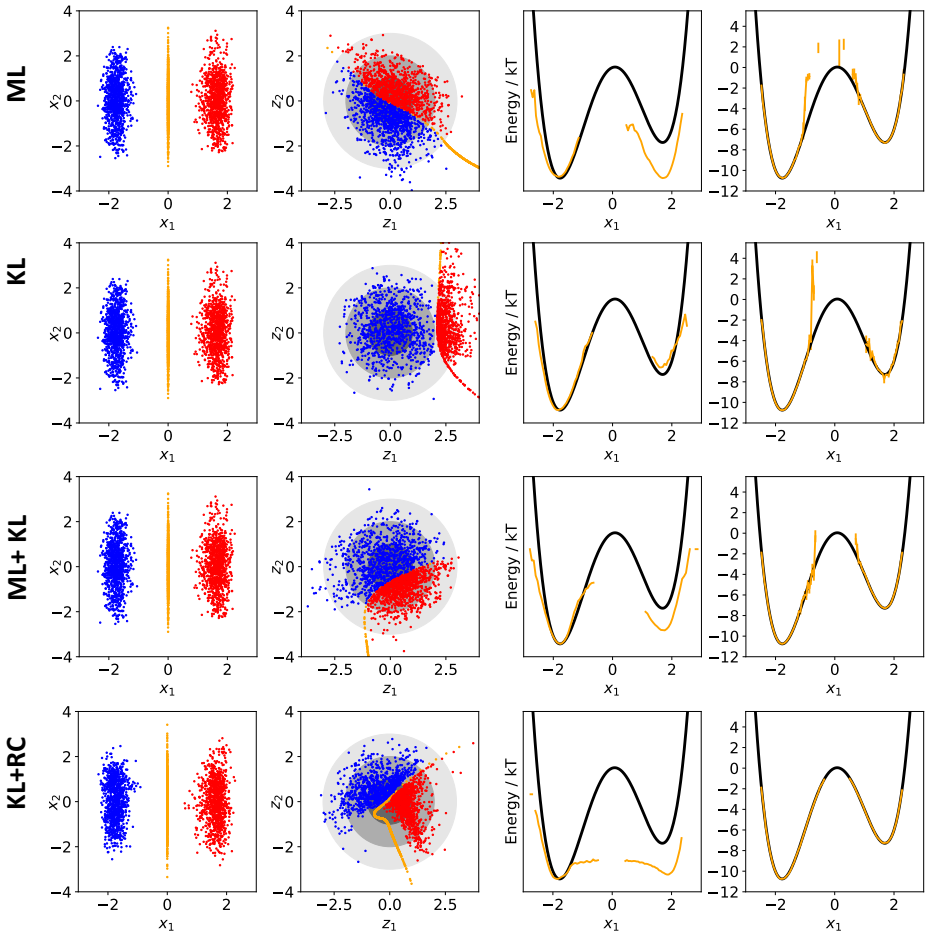
And the rotamer conformation is the combination of all rotamers, e.g. “123211211”. We remove permutational symmetries, in cycloalkanes this is achieved by setting all rotamer conformations equal that can be transformed by mirroring or cyclical permutation. After listing all conformations, we resample a list of $4,000$ configurations, but now each rotamer conformation

found has an equal probability of being sampled. This list is inserted into the next iteration for training by example.

D. Additional Figures



Supplementary Figure 2: Different methods for training Boltzmann Generators with NICE layers using the double well example shown in Fig. 2. Columns show: (1) distribution in configuration space $\mathbf{x}$, (2) distribution in latent space $\mathbf{z}$, (3) free energy of Boltzmann Generator output $p_X(\mathbf{x})$ along x_1, (4) free energy after reweighting, vertical vars show uncertainties (one standard deviation, 68% percentile). Training uses 200 epochs of ML and then 500 epochs of the method given in the rows, using equal weights for these modes. Training by example (ML) only reproduces the distribution of the training data, which can be reweighted to the Boltzmann distribution in this low-dimensional example but reweighting from the ML-generated distribution fails for high-dimensional examples. Training by energy (KL) alone tends to collapse to a single metastable state. ML+KL combined samples closer to the Boltzmann distribution than ML and avoids metastable state collapse, but samples high-energy transition states with low probability. KL+RC performs best in this example.



Supplementary Figure 3: Same as Suppl. Fig. 2 but for Boltzmann Generators using RealNVP layers.