Deep Boltzmann Generators

October 21, 2018

1 Introduction

Equilibrium statistical mechanics is concerned with computing the statistical properties of an ensemble, i.e. infinitely many copies, of a microscopic physical system. A classical example is the Ising magnetization model, where interesting quantities are which fraction of spins are "up" and "down" for a given external field, or spatial properties, such as the typical size of contiguous clusters of equal spins (Fig. 1a). A second example is protein biophysics – for a protein system that can exist in two or more macroscopic states (active or inactive, folded or unfolded, bound or dissociated), what is the probability of finding the protein in either of these states, and does their population depend on external factors, such as temperature, illumination or electric fields (Fig. 1b).

A common concept to approach these problems is to assign to each possible configuration $\mathbf{x}$ (the setting of all spins, the position of all protein atoms, etc.) a dimensionless energy, $u(\mathbf{x})$ whose contributions depend on the thermodynamic ensemble of interest (e.g., constant particle number, constant volume, etc. – see SI). Then, each configuration has the equilibrium probability:

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

Now we would like to compute expectation values of relevant observables weighted by this probability distribution, such as the probability of finding the Ising spins "up" or the protein in the active state. However, following this idea is fraught with difficulties that inspire much of the research done in statistical mechanics. Even if $u(\mathbf{x})$ is exactly known, it if often very expensive to evaluate as it contains all microscopic interactions between spins or atoms, possible involving millions of terms. The normalization factor, Z, is an integral of $\mathrm{e}^{-u(\mathbf{x})}$ of all possible configurations $\mathbf{x}$, and generally considered to be impossible to compute for large systems with nontrivial interactions.

The only known strategies to tackle this problem are Markov-Chain Monte Carlo (MCMC) simulations where we propose changes to $\mathbf{x}$ (e.g., flipping a spin) and accepting or rejecting according to how the energy changes, or Molecular Dynamics (MD) simulations where we change $\mathbf{x}$ by a tiny step that involves the derivatives of the energy with respect to $\mathbf{x}$ that ensure that $\mu(\mathbf{x})$ will be sampled. These methods are generally extremely expensive and much of the worldwide supercomputing resources are used for MCMC or MD simulations. This expense is due to (1) evaluating $u(\mathbf{x})$ or its gradient which may involve computing millions of interaction terms that make every step expensive, and (2) computing expectation values according to $\mu(\mathbf{x})$ involves sampling back and forth between phases or states that need and extremely large number of steps (e.g. $10^9 - 10^{15}$ steps in a typical MD simulation to fold or unfold a protein). Only for some systems we know specifically designed MCMC moves which make large changes in an efficient way (e.g. cluster moves in Ising models or implicit protein models [**cite**]). Speeding up the transition with enhanced sampling methods is possible if we can "drive" the system along a few collective coordinates which must describe all the slow transitions of the system and sampling the remaining fast motions [2][**cite metadynamics etc**], but this approach breaks down in systems where the number of slow processes is large.

Ideally, we would like to have a machine that samples $\mathbf{x}$ directly from the distribution $\mu(\mathbf{x})$, or at least something very close to it. This problem is probably impossible to solve in configuration space, because for a large dimension, the subvolume of low-energy configurations is vanishingly small compared to the full configuration space and has a complex and unknown shape. Thus, generating $\mathbf{x}$ by simply generating random configurations is bound to fail – e.g. generating random atom positions in a box

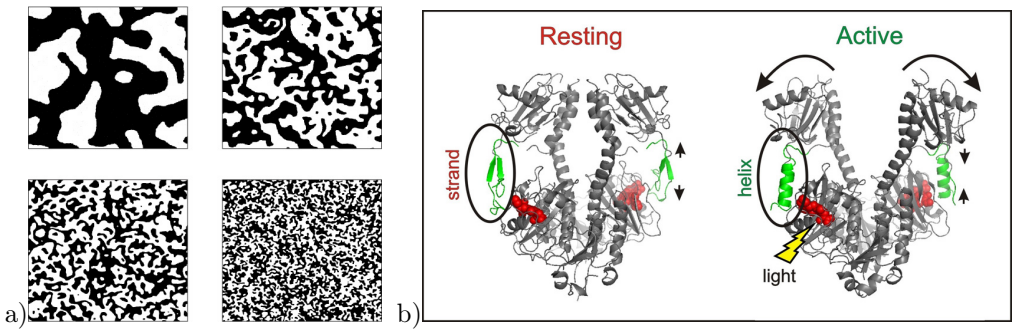


Figure 1: State- and phase transitions in complex metastable systems

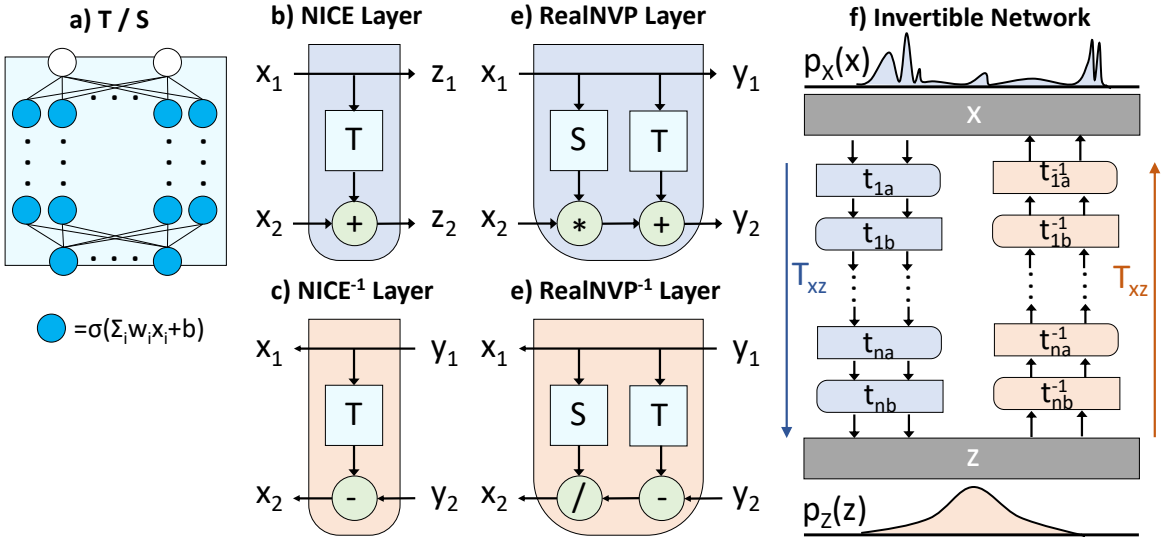


Figure 2: Network architecture

will have almost zero probability to generate a configuration that corresponds to a protein (Fig. 1b), and almost always result in numerically infinite energies that contribute nothing to $\mu(\mathbf{x})$.

Nonetheless, we directly address this problem in the present paper. Our strategy is: since sampling $\mu(\mathbf{x})$ in configuration space is too difficult, can we instead find a coordinate transformation of $\mathbf{x}$ to another representation $\mathbf{z}$, in which sampling is easy and every sample can be back-transformed to a relevant configuration $\mathbf{x}$ that contributes to $\mu(\mathbf{x})$?

2 Boltzmann Generators

To find such a transformation, we employ machine learning, specifically deep learning that has recently led to breakthroughs in pattern recognition, games and autonomous control [**cite**]. The key idea is to sample a complicated distribution $p_X(\mathbf{x}) \propto \mathrm{e}^{-u(\mathbf{x})}$ by learning a reversible transformation to a latent space, $\mathbf{z} = T_{xz}(\mathbf{x})$, such that $p_Z(\mathbf{z}) = p_Z\left(T_{xz}(\mathbf{x})\right)$ is simple. Specifically, we want to make the distribution in z a standard normal distribution:

$$p_Z(\mathbf{z}) = \mathcal{N}\left(0, \mathbf{I}\right)$$

2.1 Invertible Neural Network Transformation

We first use the volume-preserving transformation proposed for nonlinear independent components estimation (NICE, Fig. 2) [1]. For this transformation one defines two groups of variables, $\mathbf{x}_1$ and $\mathbf{x}_2$,

and employs a nonlinear transformation, P, to transform only $\mathbf{x}_2$, while $\mathbf{x}_1$ is unchanged. Independent of the choice of P, this transformation is easily invertible:

$$\begin{aligned} \mathbf{y}_1 &= \mathbf{x}_1 & \mathbf{x}_1 &= \mathbf{y}_1 \\ \mathbf{y}_2 &= \mathbf{x}_2 + P(\mathbf{x}_1) & \mathbf{x}_2 &= \mathbf{y}_2 - P(\mathbf{y}_1) \end{aligned}$$

This transformation has a Jacobian determinant of 1, $\det(\partial\mathbf{y}/\partial\mathbf{x}) = 1$, which makes the transformation volume-preserving. In physics, such transformations are found in symplectic integrators, and in incompressible fluid flows. In order to transform both channels, we define the NICER layer (Fig. 2) with involves a second transformation Q. By concatenating many of these layers, we obtain the deep NICER network T_{xz} is reversible $T_{zx} = T_{xz}^{-1}$ and volume-preserving as well (Fig. 2). Here we use two-layer perceptrons with 100 hidden neurons and rectified linear units [**cite**] for each P and Q and ten NICER layers.

A particularly nice property of volume-preserving reversible transformations is that they allows easy transformation of probability densities:

$$p_X(\mathbf{x}) = \left|\det\left(\frac{\partial T_{xz}}{\partial \mathbf{x}}\right)\right| p_Z(T_{xz}(\mathbf{z})) = p_Z(T_{xz}(\mathbf{z}))$$

2.2 Training

Basic idea: Minimize the difference between the

$$J_{KL} = \mathbb{E}_{\mathbf{z}\sim q_Z(\mathbf{z})}\left[u(T_{zx}(\mathbf{z};\boldsymbol{\theta})) - \log|\mathbf{J}_{zx}(\mathbf{z};\boldsymbol{\theta})|\right] \tag{1}$$

and

$$J_{ML} = \mathbb{E}_{\mathbf{x}\sim\rho(\mathbf{x})}\left[\frac{1}{\sigma^2}\left\|T_{xz}(\mathbf{x};\boldsymbol{\theta})\right\|^2 - \log|\mathbf{J}_{xz}(\mathbf{x};\boldsymbol{\theta})|\right] \tag{2}$$

3 Results

We first illustrate Boltzmann generators using a two-dimensional potential that has two stable states in x-direction separated by a high energy barrier, while it is a harmonic oscillator in y-direction (Fig. 3a). MD simulations will oscillate close to one of the stable state for a long time before a rare transition event occurs (Fig. 3b). Hence, the distribution in configuration space (x, y) is split into two modes (Fig. 3c, transition state ensemble is shown in yellow for clarity but not used for training). We are training Boltzmann generators by minimizing (1-2) using the two short simulations shown in Fig. 3b as input and with an additional term to promote sampling along x (SI). Fig. 3d,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. The flow field shown in Fig. 3f shows how the volume-preserving transformation T_{zx} "unpacks" the latent density into the two-state configuration density (Fig. 3e→c).

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 that populate both stable states. A variety of training methods (1-2) succeed in sampling across the barrier such that the rare event nature of the system is eliminated. In order to also sample rare transition states, we additionaly have to add a training term that enhances the sampling of rare states (**refer to SI**, Fig. 3g). Using simple reweighting, this procedure reproduces the precise free energy along the x-coordinate without any sampling problem (Fig. 3h).

Finally, we demonstrate that Boltzmann Generators can sample high-probability structures and efficiently compute the thermodynamics in high-dimensional condensed matter systems. For this we simulated a dense system of two-dimensional particles confined to a box borrowed from [3] (Fig. 4a). Particles repel each other with the 12^{th} power of their distance when being close. A bistable particle

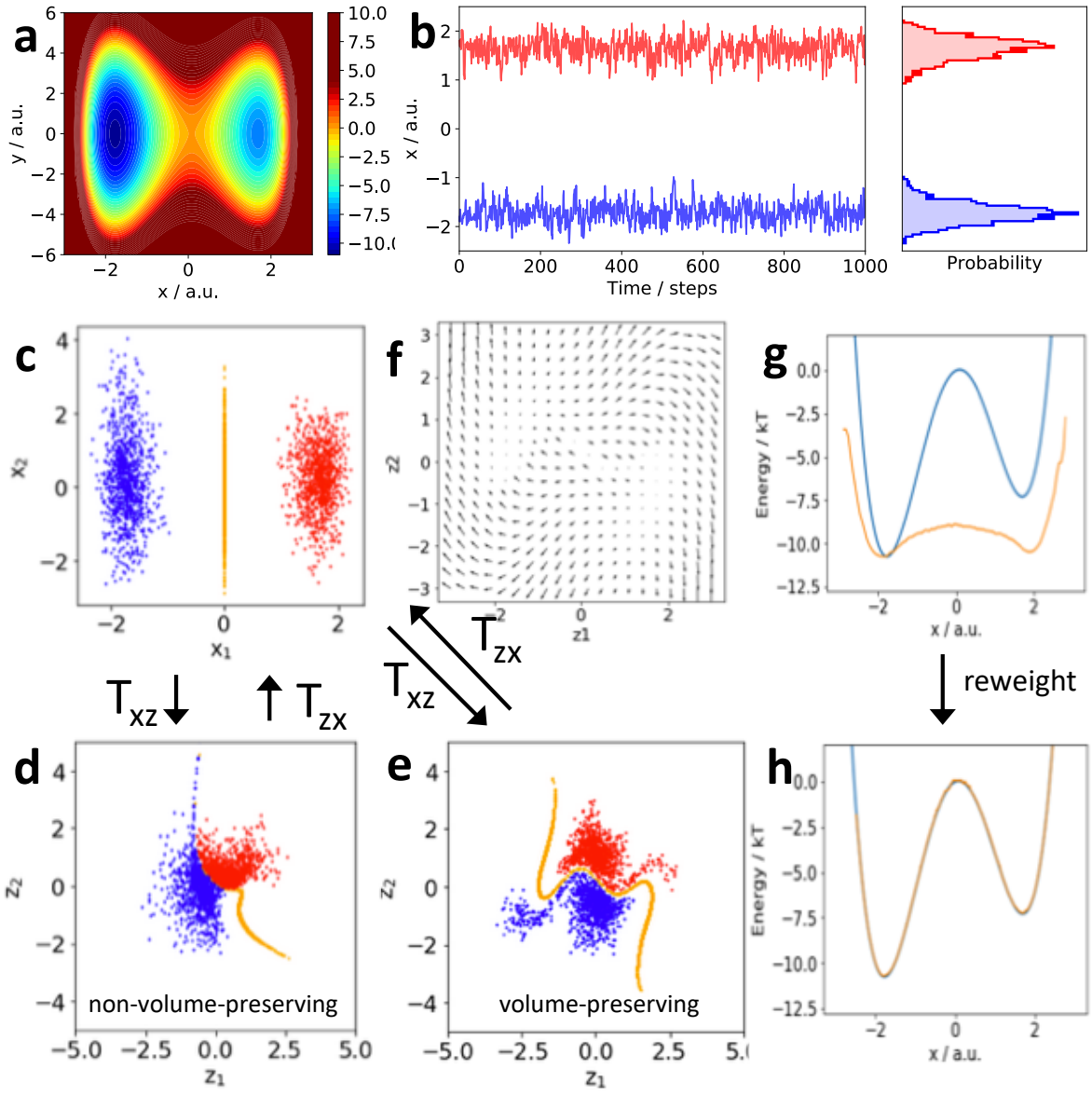


Figure 3: Double well results. a) Potential in the slow coordinate x_1. b) Distribution of training data in x-space and in z-space after training. c) Transformation flow of the neural network T_{zx}. d) Energies sampled by Latent Monte Carlo in z-space. e) Free energy obtained from Latent Monte Carlo and reweighting. f) Direct MCMC simulation in x-space. g) Latent MCMC simulation. h) Free energy obtained from Latent MCMC simulation.

dimer is immersed in the fluid. The open and closed states of the dimer are separated by a high barrier (Fig. 4a-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.

A Boltzmann Generator is trained to sample open, closed, and intermediate states. We initialize the training procedure with separate, disconnected simulations of the open and closed states, but in later stages, only the KL-loss 1 is used, i.e. training is guided by the potential energy function that defines the system (SI for details). The low-energy/high-probability states of this system 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. 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. 4d). The trained Boltzmann Generator has learned a transformation of the complex configuration space density to a compact, 76-dimensional ball in latent space (Fig. 4e). 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 have a similar distribution as the MD trajectories started in open and closed states (Fig. 4f-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. 4g).

In order to demonstrate that thermodynamic quantities can be computed with Boltzmann Generators, we conduct the training using the KL loss (1) simultaneously to a range of temperatures between one fourth and four times the reference temperature. For this training we exploit that the temperature, which has a complicated effect in configuration space, simply enters as a scaling factor in the width of the Gaussian in latent space. 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. 4i, these temperature-dependent free energies agree precisely with extensive Umbrella Sampling simulations that employ bias potentials along the dimer distance [4].

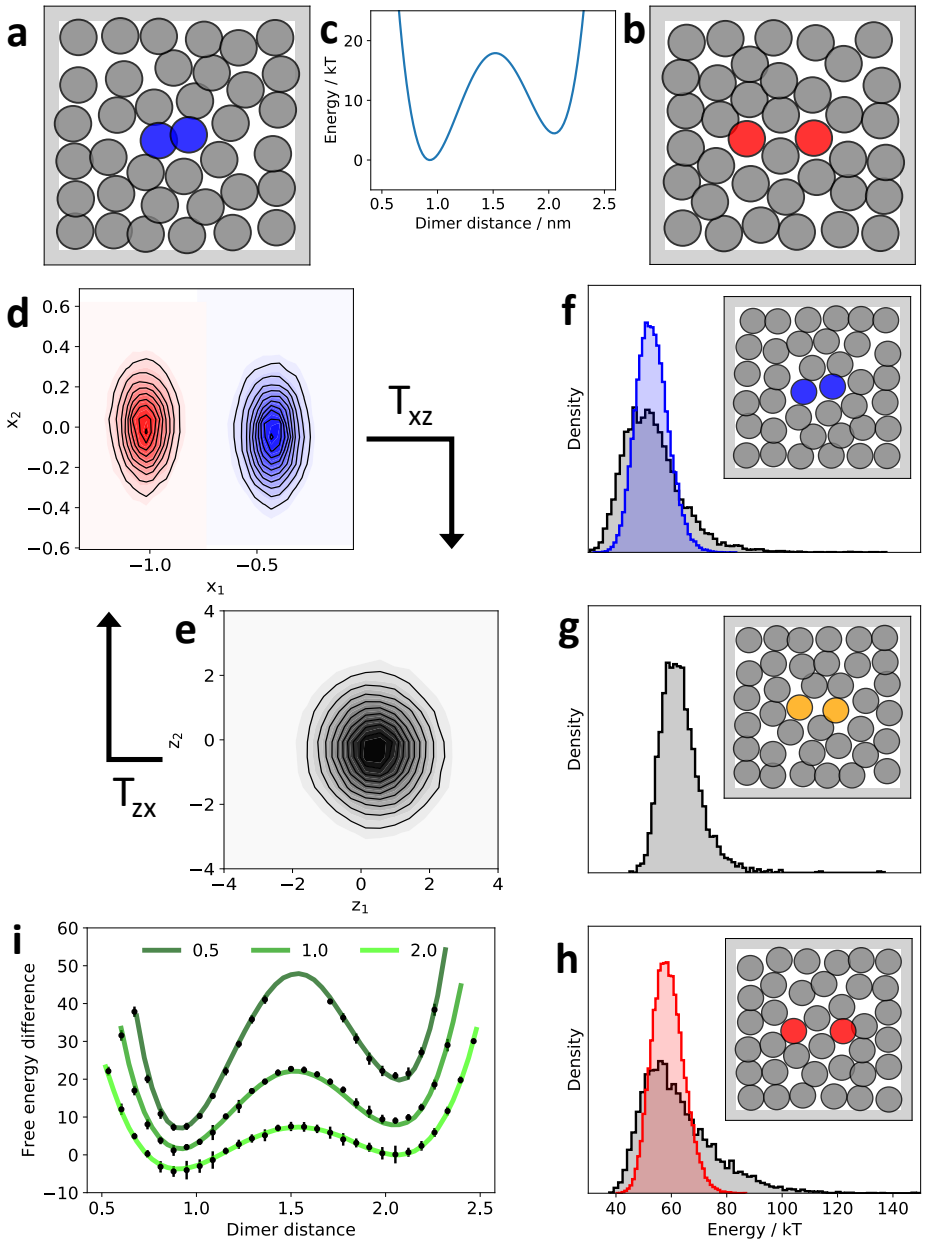


Figure 4: Particle System with bistable dimer. a,b) Closed (blue) and open (red) configurations from MD simulations (input data). c) Bistable dimer potential. d) Distribution of input data on x_1, x_2. e) Distribution of input data on z_1, z_2 after training. f, g, h) Potential energy distribution from 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) .

Supplementary Material

A. Invertible networks

$$\mathbf{J}_{zx} = \frac{dT_{zx}(\mathbf{z})}{d\mathbf{z}}$$
$$\mathbf{J}_{xz} = \frac{dT_{xz}(\mathbf{x})}{d\mathbf{x}}$$

Transformation of variables:

$$p_X(\mathbf{x}) = p_Z(\mathbf{z}) \left|\mathbf{J}_{zx}(\mathbf{z})\right|^{-1} \tag{3}$$
$$p_Z(\mathbf{z}) = p_X(\mathbf{x}) \left|\mathbf{J}_{xz}(\mathbf{x})\right|^{-1} \tag{4}$$

Layer	T_{xz}	$\lvert\det \mathbf{J}_{xz}\rvert$	T_{zx}	$\lvert\det \mathbf{J}_{zx}\rvert$
NICE	$\mathbf{z}_1 = \mathbf{x}_1$ $\mathbf{z}_2 = \mathbf{x}_2 + P(\mathbf{x}_1)$	1	$\mathbf{x}_1 = \mathbf{z}_1$ $\mathbf{x}_2 = \mathbf{z}_2 - P(\mathbf{y}_1)$	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	$\mathbf{y}_1 = \mathbf{x}_1$ $\mathbf{y}_2 = \mathbf{x}_2 \odot \exp\left(S(\mathbf{x}_1)\right) + T(\mathbf{x}_1)$	$\mathrm{e}^{\sum_i S_i(\mathbf{x}_1)}$	$\mathbf{x}_1 = \mathbf{y}_1$ $\mathbf{x}_2 = (\mathbf{y}_2 - T(\mathbf{x}_1)) \odot \exp\left(-S(\mathbf{y}_1)\right)$	$\mathrm{e}^{-\sum_i S_i(\mathbf{y}_1)}$

B. Training Boltzmann Generators

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:

$$q_Z(\mathbf{z}) \xrightarrow{T_{zx}} p_X(\mathbf{x})$$
$$\mu_X(\mathbf{x}) \xrightarrow{T_{xz}} p_Z(\mathbf{z})$$

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

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

with normalization constant Z_Z. We also define the prior energy as

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

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

$$\mu_X(\mathbf{x}) = Z_X^{-1}\mathrm{e}^{-\beta U(\mathbf{x})} \tag{7}$$

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:

$$u_k(\mathbf{x}) = \frac{T_0}{T_k} u_0(\mathbf{x})$$

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:

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

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

$$q_Z(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}) = Z_Z^{-1} \mathrm{e}^{-\frac{1}{2}\|\mathbf{z}\|^2/\sigma^2}, \tag{8}$$

with normalization constant Z_Z. We also define the prior energy as

$$\begin{aligned} u_Z(\mathbf{z}) &= -\log q_Z(\mathbf{z}) \\ &= \frac{1}{2\sigma^2} \|\mathbf{z}\|^2 + \text{const.} \end{aligned} \tag{9}$$

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

$$\mu_X(\mathbf{x}) = Z_X^{-1} \mathrm{e}^{-\beta U(\mathbf{x})} \tag{10}$$

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:

$$u_k(\mathbf{x}) = \frac{T_0}{T_k} u_0(\mathbf{x})$$

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 (10), we can express the KL divergence in latent space as:

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

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:

$$J_{KL} = \mathbb{E}_{\mathbf{z} \sim q_Z(\mathbf{z})} \left[u(T_{zx}(\mathbf{z}; \boldsymbol{\theta})) - \log \left|\mathbf{J}_{zx}(\mathbf{z}; \boldsymbol{\theta})\right|\right]. \tag{11}$$

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. (11).

Interpretation of latent KL as reweighting loss The KL loss (11) 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 $w_X(\mathbf{x}) \equiv 1$.

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 (8), this KL-divergences evaluates as:

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

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:

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

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. Likelihood maximization is used in the NICE [**cite**] and RealNVP methods [**cite**].

Jensen-Shannon divergence The two KL divergences above can be naturally combined to the Jensen-Shannon divergence

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

which can be approximated by:

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

C.

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:

$$\begin{aligned} w_X(\mathbf{x} \mid \mathbf{z}) = \frac{\mu_X(\mathbf{x})}{p_X(\mathbf{x})} &= \frac{p_Z(\mathbf{z})}{q_Z(\mathbf{z})}. \\ &\propto \mathrm{e}^{-u_X(T_{zx}(\mathbf{z}))+u_Z(\mathbf{z})+\log\lvert\det(\mathbf{J}_{zx}(\mathbf{z}))\rvert} \end{aligned} \tag{12}$$

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 MC 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

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

with

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

we have:

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

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

$$\begin{aligned} r &\leq p_{\mathrm{acc}}(\mathbf{z}_1 \to \mathbf{z}_2) \\ -\log r &\geq \log J(\mathbf{z}_1) - \log J(\mathbf{z}_2) + u(T_{zx}(\mathbf{z}_2)) - u(T_{zx}(\mathbf{z}_1)) + \frac{1}{2\sigma^2}\left\|\mathbf{z}_1\right\|^2 - \frac{1}{2\sigma^2}\left\|\mathbf{z}_2\right\|^2 \end{aligned}$$

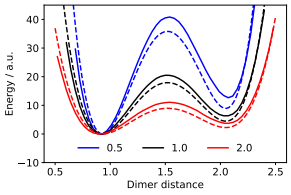
A. Systems

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. 3 for the potential in x-direction.

Particle dimer Free energy (umbrella sampling) vs dimer potential :



Hyperparameter search

z compactness

Old hyperparameter search in NICER networks. This is out of date because we now use ML and KL, but it gives an indication about capacities of network types.

	20	200	2000	f_low	min_E
nlayers=10,nl_layers=3,relu	0.0295	0.0130	0.0033	0.92	37.64
nlayers=10,nl_layers=4,relu	0.0241	0.0106	0.0023		
nlayers=10,nl_layers=4, nhidden=50, relu	0.0337	0.0145			
nlayers=10,nl_layers=2,relu	0.1222	0.0335	0.0060	0.78	38.26
nlayers=10, nl_layers=3, nhidden=10, relu	0.0925	0.0352	0.0200	?	?
nlayers=10,nl_layers=3,tanh	0.0496	? unstable	? unstable	X	X
nlayers=10,nl_layers=3,soft+	0.1363	0.0410	?	?	?
nlayers=10,nl_layers=3,elu	0.2500	0.1100	?	?	?

KL+ML training

Old code (NICER network with one **z** scaling constant layer. Shape training without maximum likelihood)

		L^{KL}_{train}	L^{KL}_{test}		$\sigma^{\mathbf{z}}_{\text{train}}$	$\sigma^{\mathbf{z}}_{\text{test}}$	N_{trans}	ACF		
default		236.30	239.02		1.14	1.41	92	0.98	0.92	0.46
nlayers (10)	2	244.95	246.31		1.32	1.40	34	0.87	0.41	-0.01
	5	238.46	240.45		1.19	1.33	138	0.78	0.39	0.00
	20	234.62	238.73		1.09	1.52	294	0.93	0.79	0.63
nl_layers (4)	1	9646.06	9646.06		0.00	0.00	0	1.00	0.99	0.95
	2	237.51	239.28		1.20	1.29	176	0.92	0.74	0.22
	6	237.29	239.42		1.15	1.55	54	0.94	0.85	0.51
nl_hidden (100)	10	259.11	261.43		1.53	1.57	2	0.93	0.66	0.32
	50	238.88	240.81		1.21	1.33	188	0.92	0.68	0.39
	200	234.22	244.20		1.00	0.99	13	1.00	0.99	0.95
activation (**ReLU**)	soft+	237.93	240.50		1.21	1.46	20	0.85	0.42	0.03
	tanh	3702.28	3919.81		1.76	1.77	0	1.00	0.98	0.84
weight_shape (10)	1	234.35	236.88		1.28	1.51	28	0.69	0.19	0.04
	25	237.70	240.61		**1.10**	**1.36**	436	0.77	0.27	0.04
	100	239.58	245.77		1.04	1.24	154	0.98	0.95	0.90
	250	242.34	247.58		1.02	1.24	226	0.90	0.60	0.14
	1000	247.28	253.88		1.00	1.24	28	0.98	0.87	0.38

...

		KL training		MC training				
		$E_{\min}/N_{\text{low}}$	$\sigma^{\mathbf{z}}_{\text{train}}$, $\sigma^{\mathbf{z}}_{\text{test}}$	$E_{\min}/N_{\text{low}}$	$\sigma^{\mathbf{z}}_{\text{train}}$, $\sigma^{\mathbf{z}}_{\text{test}}$	$\log p_{\text{acc}}$	N_{trans}	
default (RRRR)		43 / 60 K	1.04, 1.16	39 / 77 K	1.07, 1.19	13.09, 13.54	87	0.9
layer types (R_4)	$(RN)_3R$	43 / 64 K	1.02, 1.18	40 / 81 K	1.06, 1.22	12.74, 13.46	69	0.9
	$(RNNN)_3R$	41 / 66 K	1.02, 1.23	38 / 83 K	1.06, 1.26	12.42, 13.55	181	0.9
	R_6	43 / 67 K	1.01, 1.18	38 / 85 K	1.06, 1.22	12.32, 13.37	109	0.9
	$\mathbf{R}_{10}$	38 / 71 K	0.99, 1.24	34 / 89 K	1.05, 1.30	**11.30, 13.34**	**264**	0.9
	$(NR)_{10}N$	41 / 70 K	1.00, 1.29	34 / 89 K	1.05, 1.34	11.35, 13.53	93	0.9
nl_layers (4)	3	43 / 54 K	1.06, 1.15	39 / 71 K	1.07, 1.16	13.49, 13.74	35	0.9
	5	42 / 62 K	1.02, 1.18	40 / 79 K	1.06, 1.21	16.90, 16.90	7	1.0
nl_hidden (100)	**200**	43 / 71 K	0.98, 1.29	35 / 89 K	1.05, 1.34	**11.08, 13.50**	**178**	0.9
$\sigma_{\text{ref,train}}$ (0.8)	0.6	39 / 90 K	0.63, 0.69	33 / 96 K	0.61, 0.67	13.86, 13.93	0	1.0
	0.7	47 / 44 K	0.96, 1.07	45 / 69 K	0.96, 1.07	13.55, 13.83	22	0.9
	0.9	43 / 74 K	1.10, 1.23	37 / 84 K	1.13, 1.26	12.92, 13.45	0	1.0
	1.0	40 / 85 K	1.16, 1.31	39 / 90 K	1.18, 1.33	12.59, 13.32	201	0.9
reg_Jxz (0.25)	0.0	35 / 88 K	0.93, 1.03	31 / 90 K	0.96, 1.07	12.79, 13.30	0 (**check**)	1.0
	0.1	43 / 68 K	1.02, 1.15	39 / 82 K	1.04, 1.16	12.92, 13.46	104	0.9
	0.5	43 / 55 K	1.04, 1.17	44 / 71 K	1.08, 1.20	13.23, 13.61	18	0.9
	1.0	45 / 54 K	1.04, 1.17	38 / 68 K	1.08, 1.21	13.20, 13.69	61	0.9
	2.0	46 / 51 K	1.04, 1.18	42 / 64 K	1.08, 1.22	13.40, 13.76	50	0.9

References

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

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

[3] 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.

[4] 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.