Boltzmann Generators

October 20, 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 [**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

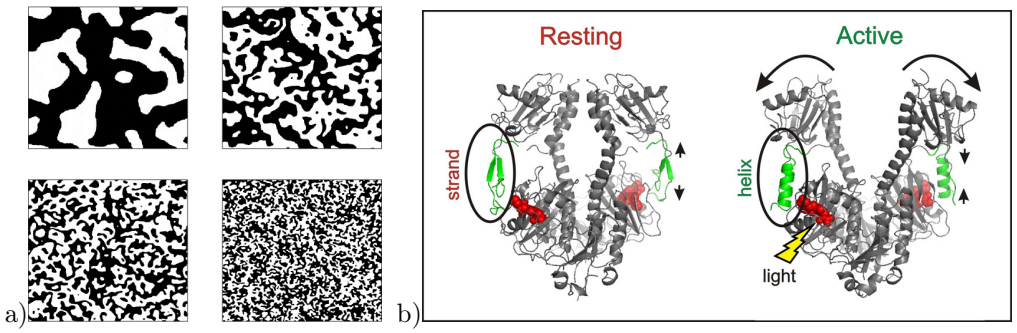

Figure 1: State- and phase transitions in complex metastable systems

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 here. 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 Invertible Networks

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

We call the transformation from configuration to latent space T_{xz} and the inverse transformation $T_{zx} = T_{xz}^{-1}$. In general, these transformations are not volume preserving and we thus keep record of the Jacobian of the transformation. We use the notation:

$$\mathbf{J}_{zx} = \frac{\partial T_{zx}(\mathbf{z})}{\partial \mathbf{z}^\top}$$
$$\mathbf{J}_{xz} = \frac{\partial T_{xz}(\mathbf{x})}{\partial \mathbf{x}^\top}$$

Random variables are transformed according to:

$$p_X(\mathbf{x}) = p_Z(\mathbf{z}) \left|\mathbf{J}_{zx}(\mathbf{z})\right|^{-1} \tag{1}$$
$$p_Z(\mathbf{z}) = p_X(\mathbf{x}) \left|\mathbf{J}_{xz}(\mathbf{x})\right|^{-1} \tag{2}$$

2.1 NICE and NICER

We first use the volume-preserving transformation proposed for nonlinear independent components estimation (NICE, Fig. 2) [**?**]. 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 the following Jacobian:

$$\mathbf{J}_{xy} = \begin{pmatrix} 1 & 0 \\ \frac{\partial P(x_1)}{\partial x_1} & 1 \end{pmatrix},$$

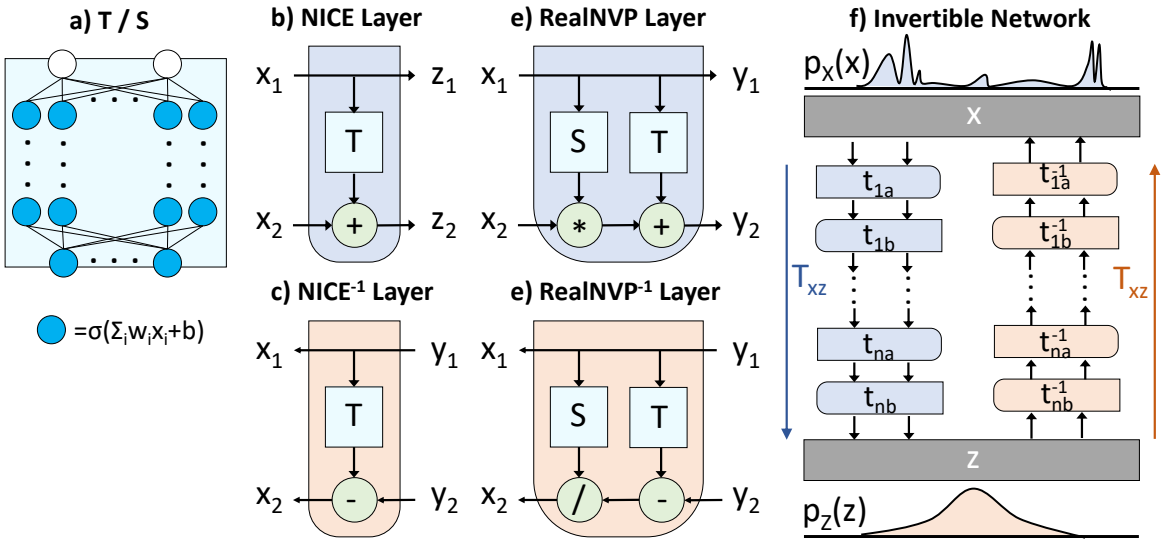

Figure 2: Network architecture

and, as a result:

$$|\det(\mathbf{J}_{xy})| = 1$$
$$|\det(\mathbf{J}_{yx})| = 1$$

This 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:

$$\mathbf{z}_2 = \mathbf{y}_2 \qquad \mathbf{y}_2 = \mathbf{z}_2$$
$$\mathbf{z}_1 = \mathbf{y}_1 + Q(\mathbf{y}_2) \qquad \mathbf{y}_1 = \mathbf{z}_1 - Q(\mathbf{z}_2)$$

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.

As a result of volume preservation, the transformation of probability densities is trivial:

$$\log p_X(x) = \log p_Z(T_{xz}(\mathbf{x}))$$
$$\log p_Z(z) = \log p_X(T_{zx}(\mathbf{z}))$$

2.2 Scaling layer

We generalize the transformation by adding a scaling layer:

$$\mathbf{z} = T_{xz}(\mathbf{x}) = \mathbf{s} \circ \mathbf{x}$$
$$\mathbf{x} = T_{zx}(\mathbf{z}) = \mathbf{s}^{-1} \circ \mathbf{z}$$

where $\mathbf{s} = (s_1, ..., s_n)$ are the scaling factors and $\mathbf{s}^{-1} = (s_1^{-1}, ..., s_n^{-1})^T$. The Jacobians of this transformation are:

$$|\det(\mathbf{J}_{xz})| = |\det(\mathrm{diag}(\mathbf{s}))| = \left|\prod_i s_i\right|$$
$$|\det(\mathbf{J}_{zx})| = \left|\det\left(\mathrm{diag}(\mathbf{s}^{-1})\right)\right| = \left|\prod_i s_i^{-1}\right|$$

Unless a better initialization for the problem at hand is available, we recomment to initialize the network in a regime with low condition number by choosing

$$\mathbf{s}^{(0)} = \text{diag}(1, ..., 1).$$

The scaling layer transforms logarithmized probability distributions as follows:

$$\log p_X(x) = \left| \sum_i \log s_i \right| + \log p_Z(\mathbf{s} \circ \mathbf{x})$$
$$\log p_Z(z) = \left| -\sum_i \log s_i \right| + \log p_X(\mathbf{s}^{-1} \circ \mathbf{z})$$

2.3 Exponential Scaling Layer

When scaling is just used to stretch or compress space and it is not desired to change signs, we can choose the following parametrization of $\mathbf{S}$ which enforces nonnegativity of the scaling factors:

$$\mathbf{S} = \text{diag}\left(\exp(k_1), ..., \exp(k_1)\right),$$

where k_i are the trainable parameters. With this formulation, the Jacobians become:

$$|\det\left(\mathbf{J}_{xz}\right)| = \exp\left(\sum_i k_i\right)$$
$$|\det\left(\mathbf{J}_{zx}\right)| = \exp\left(-\sum_i k_i\right)$$

Note that the absolute value operator is no longer needed as the value of the exponential function is always nonnegative. The exponential scaling layer transforms logarithmized probability distributions as:

$$\log p_X(x) = \sum_i k_i + \log p_Z(\exp(\mathbf{k}) \circ \mathbf{x})$$
$$\log p_Z(z) = -\sum_i k_i + \log p_X(\exp(-\mathbf{k}) \circ \mathbf{z})$$

$$p_X(\mathbf{x}) = |\det\left(S\right)|^{-1} p_Z(T_{xz}(\mathbf{x}))$$

If we simply decide to sample z from a normal distribution and transform to x, the network represents the density

$$p_X(\mathbf{x}) = |\det\left(S\right)|^{-1} \mathcal{N}(0, \mathbf{I}). \tag{3}$$

2.4 RealNVP

Forward transformation: $\mathbf{x} \to \mathbf{y} \to \mathbf{z}$ with the first step:

$$\mathbf{y}_1 = \mathbf{x}_1$$
$$\mathbf{y}_2 = \mathbf{x}_2 \odot \exp\left(S(\mathbf{x}_1)\right) + T(\mathbf{x}_1)$$

and the Jacobian:

$$\mathbf{J}_{xy} = \begin{bmatrix} I & 0 \\ \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_1} & \operatorname{diag}\left[\exp\left(S(\mathbf{x}_1)\right)\right] \end{bmatrix}$$
$$\left|\det\left(\mathbf{J}_{xy}\right)\right| = \mathrm{e}^{\sum_i S_i(\mathbf{x}_1)}$$

Likelihood:

$$p_X(\mathbf{x}) = J(\mathbf{x}) p_Z(T_{xz}(\mathbf{x}))$$

with

$$J(\mathbf{x}) = \left|\det\left(\frac{dT_{xz}}{dx}\right)\right|$$

The Log Likelihood of a Gaussian in $\mathbf{z}$ with mean 0 and std σ is given by:

$$\begin{aligned} L(\mathbf{x}) &= \log J(\mathbf{x}) - n \log(\sigma) - \frac{1}{2\sigma^2}\mathbf{z}^\top \mathbf{z} + \text{const} \\ &= \log J(\mathbf{x}) - \frac{1}{2\sigma^2}\mathbf{z}^\top \mathbf{z} + \text{const} \end{aligned}$$

For multiple trajectories that should each be standartized, if we sum their log-likelihoods we will get the same loss function except for a constant pre-factor. If we instead sum their likelihoods, we get:

$$\begin{aligned} \mathrm{e}^{L(\mathbf{X})} &= \mathrm{e}^{\sum_{s=1}^{N_1} \log J(\mathbf{x}_s) - \frac{1}{2\sigma^2}\mathbf{z}_s^\top \mathbf{z}_s} + \mathrm{e}^{\sum_{s=1}^{N_2} \log J(\mathbf{x}_t) - \frac{1}{2\sigma^2}\mathbf{z}_t^\top \mathbf{z}_t} \\ L(\mathbf{X}) &= \log\left(\mathrm{e}^{\sum_{s=1}^{N_1} \log J(\mathbf{x}_s) - \frac{1}{2\sigma^2}\mathbf{z}_s^\top \mathbf{z}_s} + \mathrm{e}^{\sum_{s=1}^{N_2} \log J(\mathbf{x}_t) - \frac{1}{2\sigma^2}\mathbf{z}_t^\top \mathbf{z}_t}\right) \\ &= \operatorname{logsumexp}\left(\sum_{s=1}^{N_1} \log J(\mathbf{x}_s) - \frac{1}{2\sigma^2}\mathbf{z}_s^\top \mathbf{z}_s, \sum_{s=1}^{N_2} \log J(\mathbf{x}_s) - \frac{1}{2\sigma^2}\mathbf{z}_s^\top \mathbf{z}_s\right) \end{aligned}$$

$$L(\mathbf{X}) = \sum_{s=1}^{N_1}\left(\frac{1}{N_1}\log J(\mathbf{x}_s) - \frac{1}{N_1}\frac{1}{2\sigma^2}\mathbf{z}_s^\top \mathbf{z}_s\right) + \sum_{t=1}^{N_2}\left(\frac{1}{N_2}\log J(\mathbf{x}_t) - \frac{1}{N_2}\frac{1}{2\sigma^2}\mathbf{z}_t^\top \mathbf{z}_t\right)$$

If we have equally many trajectories ($N/2$) in each batch, we can concatenate them and write

$$\begin{aligned} L(\mathbf{X}) &= \sum_{t=1}^{N/2}\left(\frac{2}{N}\log J(\mathbf{x}_s) - \frac{2}{N}\frac{1}{2\sigma^2}\mathbf{z}_s^\top \mathbf{z}_s\right) + \sum_{t=N/2}^{N}\left(\frac{2}{N}\log J(\mathbf{x}_t) - \frac{2}{N}\frac{1}{2\sigma^2}\mathbf{z}_t^\top \mathbf{z}_t\right) \\ &= \frac{2}{N}\sum_{t=1}^{N}\log J(\mathbf{x}_s) - \frac{2}{N}\sum_{t=1}^{N}\frac{1}{2\sigma^2}\mathbf{z}_s^\top \mathbf{z}_s \end{aligned}$$

Reverse transformation:

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{y}_1 \\ \mathbf{x}_2 &= (\mathbf{y}_2 - T(\mathbf{x}_1)) \odot \exp\left(-S(\mathbf{y}_1)\right) \end{aligned}$$

and the Jacobian:

$$\frac{\partial \mathbf{x}}{\partial \mathbf{y}} = \begin{bmatrix} I & 0 \\ \frac{\partial \mathbf{x}_2}{\partial \mathbf{y}_1} & \operatorname{diag}\left[\exp\left(-S(\mathbf{y}_1)\right)\right] \end{bmatrix}$$
$$\det\left(\frac{\partial \mathbf{x}}{\partial \mathbf{y}}\right) = \mathrm{e}^{-\sum_i S_i(\mathbf{y}_1)}$$

Note that if we use scaling layers with nonnegative output, the minus disappears.
As above, we can minimize the KL divergence, resulting in

$$\begin{aligned} J &= \log\left|\det\left(\frac{dT_{xz}}{dx}\right)\right| + \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,I)}\left[u(T_{zx}(\mathbf{z}))\right] \\ &= -\sum_i S_i + \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,I)}\left[u(T_{zx}(\mathbf{z}))\right] \end{aligned}$$

Where i runs over **all** scaling units in the network. Note that in the reverse transformation S enters with a negative sign, i.e. the role of the first term in J is the same as in the NICE transformation: minimizing the first term smaller increases the scaling from $\mathbf{z}$ to $\mathbf{x}$, corresponding to trying to include more configuration space, while it is beneficial to reduce the scaling from $\mathbf{z}$ to $\mathbf{x}$ to minimizing the mean energy (second term).

We can condatenate two RealNVP layers with swapped channels in order to transform all variables:

$$\begin{aligned}\mathbf{z}_2 &= \mathbf{y}_2\\ \mathbf{z}_1 &= \mathbf{y}_1 \odot \exp\left(S(\mathbf{y}_2)\right) + T(\mathbf{y}_2)\end{aligned}$$

Layer	T_{xz}	$\lvert\det \mathbf{J}_{xz}\rvert$	T_{zx}	$\lvert\det \mathbf{J}_{zx}\rvert$
NICE	$\begin{aligned}\mathbf{z}_1 &= \mathbf{x}_1\\ \mathbf{z}_2 &= \mathbf{x}_2 + P(\mathbf{x}_1)\end{aligned}$	1	$\begin{aligned}\mathbf{x}_1 &= \mathbf{z}_1\\ \mathbf{x}_2 &= \mathbf{z}_2 - P(\mathbf{y}_1)\end{aligned}$	1
NICER	$\begin{aligned}\mathbf{z}_2 &= \mathbf{x}_2 + P(\mathbf{x}_1)\\ \mathbf{z}_1 &= \mathbf{x}_1 + Q(\mathbf{x}_2 + P(\mathbf{x}_1))\end{aligned}$	1	$\begin{aligned}\mathbf{x}_1 &= \mathbf{z}_1 - Q(\mathbf{z}_2)\\ \mathbf{x}_2 &= \mathbf{z}_2 - P(\mathbf{z}_1 - Q(\mathbf{z}_2))\end{aligned}$	1
Scaling	$\mathbf{z} = \mathbf{s} \circ \mathbf{x}$	$\lvert\prod_i s_i\rvert$	$\mathbf{x} = \mathbf{s}^{-1} \circ \mathbf{z}$	$\left\lvert\prod_i s_i^{-1}\right\rvert$
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	$\begin{aligned}\mathbf{y}_1 &= \mathbf{x}_1\\ \mathbf{y}_2 &= \mathbf{x}_2 \odot \exp\left(S(\mathbf{x}_1)\right)\\ &\quad + T(\mathbf{x}_1)\end{aligned}$	$\mathrm{e}^{\sum_i S_i(\mathbf{x}_1)}$	$\begin{aligned}\mathbf{x}_1 &= \mathbf{y}_1\\ \mathbf{x}_2 &= (\mathbf{y}_2 - T(\mathbf{x}_1))\\ &\quad \odot \exp\left(-S(\mathbf{y}_1)\right)\end{aligned}$	$\mathrm{e}^{-\sum_i S_i(\mathbf{y}_1)}$

3 Training

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:

$$q_Z(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}) = Z_Z^{-1}\mathrm{e}^{-\frac{1}{2}\|\mathbf{z}\|^2/\sigma^2}, \tag{4}$$

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

$$\begin{aligned}u_Z(\mathbf{z}) &= -\log q_Z(\mathbf{z})\\ &= \frac{1}{2\sigma^2}\|\mathbf{z}\|^2 + \text{const.}\end{aligned} \tag{5}$$

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

$$\mu_X(\mathbf{x}) = Z_X^{-1}\mathrm{e}^{-\beta U(\mathbf{x})} \tag{6}$$

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:

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

3.1 Latent KL divergence and reweighting loss

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 (1-2) and the Boltzmann distribution (6), 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:

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

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. (7).

The KL loss (7) 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 (1-2). 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 $w_X(\mathbf{x}) \equiv 1$.

3.2 Configuration KL divergence and Maximum Likelihood

Likewise, we can express the KL divergence in $\mathbf{x}$ space where we compare the generated distributions with a Boltzmann weight. Using (1-2) and the Gaussian prior density (4), 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.

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**].

3.3 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:

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

3.4 Training Latent MCMC Acceptance

Using Barker Dynamics and Latent MCMC we can express the acceptance probability of $\mathbf{z}_2$ after $\mathbf{z}_1$ as:

$$\begin{aligned} \log p_{\mathrm{acc}} &= -\log\left(1+\mathrm{e}^{F(\mathbf{z}_2)-F(\mathbf{z}_1)}\right) \\ F(\mathbf{z}) &= u(T_{zx}(\mathbf{z})) - \log J_{zx}(\mathbf{z}) - \frac{1}{2\sigma^2}\left\|\mathbf{z}\right\|^2 \\ &= u(T_{zx}(\mathbf{z})) + \log J_{xz}(\mathbf{x}(\mathbf{z})) - \frac{1}{2\sigma^2}\left\|\mathbf{z}\right\|^2 \end{aligned}$$

Now we use the network in a parallel fashion, i.e.:

$$\begin{aligned} \mathbf{x}_{in} &\rightarrow \mathbf{z}_{out} \\ \mathbf{z}_{in} &\rightarrow \mathbf{x}_{out} \end{aligned}$$

where $\mathbf{x}$ is training data and $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$. We now consider the following Latent MCMC move:

$$\mathbf{z}_{out} \rightarrow \mathbf{z}_{in}$$

That means we start at a $\mathbf{z}$ value corresponding to a training configuration and we want to maximize the MCMC efficiency when considering a single step. We have

$$\begin{aligned} F(\mathbf{z}_1) &= u(\mathbf{x}_{in}) + \log J_{xz}(\mathbf{x}_{in}) - \frac{1}{2\sigma^2}\left\|\mathbf{z}_{out}\right\|^2 \\ F(\mathbf{z}_2) &= u(\mathbf{x}_{out}) - \log J_{zx}(\mathbf{z}_{in}) - \frac{1}{2\sigma^2}\left\|\mathbf{z}_{in}\right\|^2 \end{aligned}$$

Note that we have to use the Jacobian in both directions in order to work with available input data. Now we maximize:

$$\log p_{\mathrm{acc}} + \log\left\|\mathbf{x}_{in} - \mathbf{x}_{out}\right\|^2$$

3.5 Symmetric acceptance

We want to maximize the acceptance probability forward and backward. In Barker dynamics the forward-and-backward probabilities sum up to one, therefore we use the product:

$$\begin{aligned} p_{\rightleftarrows} &= p_{\mathrm{acc}}(\mathbf{z}_1 \to \mathbf{z}_2) p_{\mathrm{acc}}(\mathbf{z}_2 \to \mathbf{z}_1) \\ &= \frac{1}{1+\mathrm{e}^{F(\mathbf{z}_2)-F(\mathbf{z}_1)}} \frac{1}{1+\mathrm{e}^{-(F(\mathbf{z}_2)-F(\mathbf{z}_1))}} \\ &= \frac{1}{2+\mathrm{e}^{F(\mathbf{z}_2)-F(\mathbf{z}_1)}+\mathrm{e}^{-(F(\mathbf{z}_2)-F(\mathbf{z}_1))}}. \end{aligned}$$

3.6 Barker dynamics efficiency

Barker dynamics is given by

$$p_{\mathrm{acc}}(\mathbf{z}_1 \to \mathbf{z}_2) = \frac{1}{1+\exp\left(u(\mathbf{z}_2)-u(\mathbf{z}_1)+\log p_{\mathrm{prop}}(\mathbf{z}_1 \to \mathbf{z}_2)-\log p_{\mathrm{prop}}(\mathbf{z}_2 \to \mathbf{z}_1)\right)}$$

Using latent MCMC this is given by:

$$\begin{aligned} p_{\mathrm{acc}}(\mathbf{z}_1 \to \mathbf{z}_2) &= \frac{1}{1+\exp\left(u(\mathbf{z}_2)-\frac{1}{2\sigma^2}\left\|\mathbf{z}_2\right\|^2-u(\mathbf{z}_1)+\frac{1}{2\sigma^2}\left\|\mathbf{z}_1\right\|^2\right)} \\ &= \frac{1}{1+\exp\left(g(\mathbf{z}_2)-g(\mathbf{z}_1)\right)} \end{aligned}$$

with $g(\mathbf{z}) = u(\mathbf{z}) - \frac{1}{2\sigma^2}\left\|\mathbf{z}\right\|^2$, and typically we use $\sigma = 1$.
The log efficiency is

$$\begin{aligned} \log\left(p_{\mathrm{acc}}(\mathbf{z}_1 \to \mathbf{z}_2)\left\|\mathbf{x}_2-\mathbf{x}_1\right\|_2^2\right) &= \log p_{\mathrm{acc}}(\mathbf{z}_1 \to \mathbf{z}_2) + \log\left\|\mathbf{x}_2-\mathbf{x}_1\right\|_2^2 \\ &= -\log\left[1+\exp\left(g(\mathbf{z}_2)-g(\mathbf{z}_1)\right)\right] + \log\left\|\mathbf{x}_2-\mathbf{x}_1\right\|_2^2 \end{aligned}$$

Then we minimize

$$J = \log\left[1+\exp\left(g(\mathbf{z}_2)-g(\mathbf{z}_1)\right)\right] - \log\left\|\mathbf{x}_2-\mathbf{x}_1\right\|_2^2$$

batchwise

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.

4.1 Reweighting / Free Energy Perturbation

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

$$\begin{aligned} w_X(\mathbf{x} \mid \mathbf{z}) = \frac{\mu_X(\mathbf{x})}{p_X(\mathbf{x})} &= \frac{p_Z(\mathbf{z})}{q_Z(\mathbf{z})}. \\ &\propto \mathrm{e}^{-u_X(T_{zx}(\mathbf{z}))+u_Z(\mathbf{z})+\log\left|\det(\mathbf{J}_{zx}(\mathbf{z}))\right|} \end{aligned} \tag{8}$$

This principle can be directly used in order to compute statistical quantities. For example, the free energy difference two substates A and B is then given by free energy perturbation:

$$F_B - F_A = -\log \frac{\langle w_X(\mathbf{x})\rangle_B}{\langle w_X(\mathbf{x})\rangle_A}.$$

Expectation values can be computed as

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

etc. However direct reweighting is numerically unstable because the distribution of weights (8) contains a long tail of very high but very rare weights.

4.2 Free energy differences

A more robust way to compute free energy differences is to use samples at the two states whose free energy difference we want to compute and use Bennett's acceptance ratio [**?**] to relate them.

$$F_B - F_A = -\log \frac{Z_B}{Z_A} = -\log \frac{\langle M(u_B - u_A)\rangle_A}{\langle M(u_A - u_B)\rangle_B}$$

where we can use the Metropolis function $M(x) = \min(e^{-x}, 1)$. Normally, BAR is used to compute free energy differences between different thermodynamic states A and B and the assumption is that each thermodynamic state is sampled in equilibrium. Here we have the problem that we have samples at two configuration states A and B that are in the same thermodynamic state, but do not overlap in x-space, and we therefore cannot get an equilibrium sample. However, A and B do overlap in z-space. We can there do the following trick of definition three thermodynamic states:

1. Configuration A with energy $u_A(\mathbf{x}) = u(\mathbf{x})$ in configuration A and $u_A(\mathbf{x}) = \infty$ otherwise
2. Configuration B with energy $u_B(\mathbf{x}) = u(\mathbf{x})$ in configuration B and $u_B(\mathbf{x}) = \infty$ otherwise
3. State Z with energy $u_Z(\mathbf{z})$ given in (**??**).

Then, the resulting BAR ratio is given by:

$$F_B - F_A = F_B - F_Z + F_Z - F_A = -\log \frac{\langle M(u_B - u_Z)\rangle_Z}{\langle M(u_Z - u_B)\rangle_B} \frac{\langle M(u_Z - u_A)\rangle_A}{\langle M(u_A - u_Z)\rangle_Z}$$

4.3 Latent MC

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

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

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

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

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:

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

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

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

5 Applications

5.1 Double Well

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

$$E(x, y) = E(x) + \frac{1}{2}dy^2$$

with

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

defining the double well.

In order to estimate transition rates, we want to have a parametric transition state. However changing b alone also changes the position of minima. Taking the derivative leads to:

$$\frac{\partial E(x)}{\partial x} = ax^3 - bx + c$$

Setting this to 0 lead to very complicated solutions. However for the case $c = 0$, we have $0 = x\left(ax^2 - b\right)$ with the trivial stationary point $x = 0$ (transition state) and the minima at:

$$x = \pm\sqrt{\frac{b}{a}}$$

Then if we set the minima to ± 1 we obtain

$$b = a$$

removing one parameter in $E(x)$.

For $c = 0$, the minima energies are: $E(x = \pm 1) = -\frac{1}{4}a$, which is equal to the energy barrier.

5.2 Bistable Dimer in a Lennard-Jones Bath

5.3 Molecular Mechanics System