Latent Markov Chain Monte Carlo

Frank Noé[1,2*]

December 21, 2018

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

Abstract

The recently introduced Boltzmann Generators are an approach to generate direct, independent samples of equilibrium states in many-body systems. This approach works by learning an invertible variable transformation to a so-called latent space in which the sought equilibrium distribution becomes a simple distribution, such as a Gaussian. The latent space is then sampled and samples in configuration space are generated. Since the generated configuration space distribution is not exactly the equilibrium distribution, it must be reweighted to the true equilibrium distribution. However, reweighting suffers from numerical and statistical problems when the generated and the equilibrium distribution have significant differences. Here we present an alternative approach: we construct a Markov-Chain Monte Carlo method in which independent proposals are generated by sampling the latent distribution and the equilibrium distribution is used for acceptance. We demonstrate that Latent MCMC can efficiently sample the equilibrium states in toy models and condensed matter systems.

1 Introduction

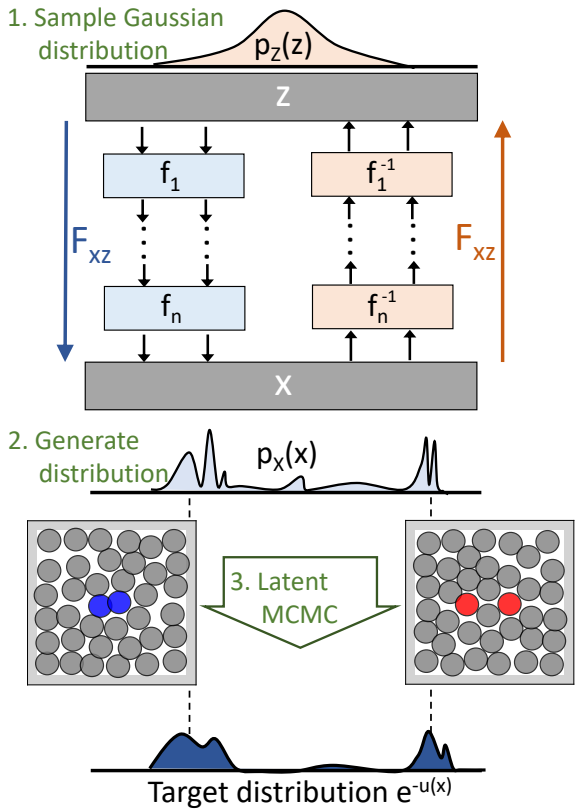

Figure 1: **Boltzmann Generators. a)** A Boltzmann Generator is trained by minimizing the difference between its generated distribution and the desired Boltzmann distribution. It is used by transforming samples from a simple (e.g., Gaussian) distribution to generated configurations. To compute thermodynamics, 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.

2 Latent Markov Chain Monte Carlo

2.1 Latent transformation

The basis of latent sampling is an invertible transformation between configurations $\mathbf{x}$ and latent variables $\mathbf{z}$ in which the sampling takes place:

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

Hence $T_{xz} = T_{zx}^{-1}$. Here, we implement these transformations with invertible deep neural networks, using volume-preserving [1] and non-volume preserving transformations [2].

Each transformation has a Jacobian matrix with the pairwise first derivatives of outputs with respect to inputs:

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

The absolute value of the Jacobian's determinant, $|\det \mathbf{J}_{zx}(\mathbf{z})|$, measures how much a volume element at $\mathbf{z}$ is scaled by the transformation. Forward and reverse transformation are related by $|\det \mathbf{J}_{zx}(\mathbf{z})| = |\det \mathbf{J}_{xz}(\mathbf{x})|^{-1}$, and respectively for $\mathbf{x}$ and $\mathbf{z}$ exchanged. Our main motivation to use invertible transformations is that they allow us to transform random variables as follows:

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

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

2.2 Sampling and reweighting

Suppose we want to sample from the target distribution $\mu_X(\mathbf{x})$ in configuration space (e.g., the Boltzmann distribution). To this end, we train the transformation $F_{zx}(\mathbf{z})$ such that sampling $\mathbf{z}$ from the prior distribution $\mu_Z(\mathbf{z})$ and transformation through F_{zx} will produce a distribution $q_X(\mathbf{x})$ is similar to the target $\mu_X(\mathbf{x})$:

$$\mu_Z(\mathbf{z}) \quad \xrightarrow{F_{zx}} \quad q_X(\mathbf{x}) \approx \mu_X(\mathbf{x})$$

For example, we may use the isotropic Gaussian prior as in [1, **?**]:

$$\mu_Z(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}) = Z_Z^{-1} \mathrm{e}^{-u_Z(\mathbf{x})}, \tag{3}$$

with normalization constant Z_Z and prior energy

$$u_Z(\mathbf{z}) = \frac{1}{2\sigma_k^2} \left\|\mathbf{z}\right\|^2, \tag{4}$$

which is equal to $-\log \mu_Z(\mathbf{z})$, up to a constant. As our aim is to sample with respect to the distribution $\mu_X(\mathbf{x})$, we need to modify the sampled distribution $q_X(\mathbf{x})$ accordingly. The simplest approach is reweighting [**cite**], where we compute the ratio of the two probability densities for every sample, and use this as a statistical weight for this sample. Using (1-2), we obtain:

$$\begin{aligned} w_X(\mathbf{x}) = \frac{\mu_X(\mathbf{x})}{q_X(\mathbf{x})} &= \frac{q_Z(\mathbf{z})}{\mu_Z(\mathbf{z})}. \\ &\propto \mathrm{e}^{-u_X(T_{zx}(\mathbf{z})) + u_Z(\mathbf{z}) + \log\left|\det \mathbf{J}_{zx}(\mathbf{z})\right|} \end{aligned} \tag{5}$$

This approach is followed in [**?**]. Using these weights, expectation values can be computed as

$$\mathbb{E}_{\mu_X}[O] = \mathbb{E}_{w_X p_X}[O] \approx \frac{\sum_{i=1}^N w_X(\mathbf{x}) O(\mathbf{x})}{\sum_{i=1}^N w_X(\mathbf{x})}. \tag{6}$$

The disadvantage of reweighting is that it requires that q_X is an excellent approximation of μ_X. If the two distributions are significantly different for some configurations, and that is hard to avoid in high dimensions, expectation values such as (6) may be dominated by a few samples. In this case, the method is stastistically inefficient, as only few

2.3 Latent MCMC

In MCMC methods, we generate a sequences $\{\mathbf{x}_t\}_{t=1,\dots,T}$ by steps

$$\begin{array}{ccccccc} & & \text{prop} & \mathbf{x}_t^* & & & \\ & \nearrow & & & \searrow & & \text{acc} \\ \mathbf{x}_t & \to & \to & \to & \to & \mathbf{x}_{t+1} \end{array} \tag{7}$$

where the proposal step generates a proposal $\mathbf{x}_t^*$ from a given configuration $\mathbf{x}_t$ with probability $p_{\text{prop}}(\mathbf{x}_t \to \mathbf{x}_t^*)$. With probability $p_{\text{acc}}(\mathbf{x}_t \to \mathbf{x}_t^*)$ this proposal is accepted, otherwise we stay at the old state. This is realized by generating a uniform random variable in $[0, 1)$, $r \sim \text{Uniform}(0, 1)$

$$\mathbf{x}_{t+1} = \begin{cases} \mathbf{x}_t^* & r < p_{\text{acc}}(\mathbf{x}_t \to \mathbf{x}_t^*) \\ \mathbf{x}_t & \text{else.} \end{cases}$$

The typical computational graph is thus as shown in (7): $\mathbf{x}_t^*$ depends on $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$ depends on both $\mathbf{x}_t, \mathbf{x}_t^*$. Standard MCMC is thus an inherently sequential algorithm.

We have a lot of flexibility in choosing proposal moves, as long as p_{prop} can be computed for every move and the move set is ergodic, i.e. any point of configuration space with nonzero equilibrium propobability can be reached. The MCMC method defines p_{acc} such that the MCMC chain will sample from the equilibrium distribution. The most common choice is the Metropolis function:

$$p_{\text{acc}} = \min\left\{1, \frac{\mu_X(\mathbf{x}_t^*)}{\mu_X(\mathbf{x}_t)} \frac{p_{\text{prop}}(\mathbf{x}_t^* \to \mathbf{x}_t)}{p_{\text{prop}}(\mathbf{x}_t \to \mathbf{x}_t^*)}\right\}. \tag{8}$$

Latent MCMC makes the following choice as a proposal step: Each proposal is generated by sampling the latent space distribution independently and transforming it via F_{zx}. As a result, the proposed configurations $\mathbf{x}_t^*$ are sampled from the generated distribution $q_X(\mathbf{x})$:

$$\begin{aligned} \mathbf{z}_t^* &\sim \mu_Z(\mathbf{z}) \\ \mathbf{x}_t^* = F(\mathbf{z}_t^*) &\sim q_X(\mathbf{x}) \end{aligned}$$

The Metropolis acceptance probability then becomes:

$$p_{\text{acc}} = \min\left\{1, \frac{\mu_X(\mathbf{x}_t^*)}{\mu_X(\mathbf{x}_t)} \frac{q_X(\mathbf{x}_t)}{q_X(\mathbf{x}_t^*)}\right\} = \min\left\{1, \frac{w_X(\mathbf{x}_t^*)}{w_X(\mathbf{x}_t)}\right\}, \tag{9}$$

where we have used the definition of the weights from (5). Using (5), we can further write:

$$Q(\mathbf{x}_t, \mathbf{x}_t^*) = -\log \frac{w_X(\mathbf{x}_t^*)}{w_X(\mathbf{x}_t)} = \Delta u_X(\mathbf{z}_t, \mathbf{z}_t^*) - \Delta u_Z(\mathbf{z}_t, \mathbf{z}_t^*) - \Delta R(\mathbf{z}_t, \mathbf{z}_t^*)$$

with

$$\begin{aligned} \Delta u_X(\mathbf{z}_t, \mathbf{z}_t^*) &= u_X\left(F_{zx}(\mathbf{z}_t^*)\right) - u_X\left(F_{zx}(\mathbf{z}_t)\right) \\ \Delta u_Z(\mathbf{z}_t, \mathbf{z}_t^*) &= u_Z(\mathbf{z}_t^*) - u_Z(\mathbf{z}_t) \\ \Delta R(\mathbf{z}_t, \mathbf{z}_t^*) &= \log\left|\det \mathbf{J}_{zx}(\mathbf{z}_t^*)\right| - \log\left|\det \mathbf{J}_{zx}(\mathbf{z})\right| \end{aligned}$$

If we use a Gaussian proposal distribution as in Eq. (3). Because proposals are independent, they can be made in parallel and our computational graphs simplifies to:

$$\begin{array}{ccccccc} & \text{prop} & \mathbf{x}_t^* & & \mathbf{x}_{t+1}^* & & \cdots \\ & \text{acc} & \downarrow & & \downarrow & & \\ \mathbf{x}_t & \to & \mathbf{x}_t & \to & \mathbf{x}_{t+1} & \to & \cdots \end{array} \tag{10}$$

This structure is important for computational efficiency: We can generate a batch of proposals $\{\mathbf{x}_1^*, ..., \mathbf{x}_B^*\}$ in parallel, transform them to configurations and compute their energies and Jacobians in parallel. We are then only left with a simple and fast serial step in which $Q(\mathbf{x}_t, \mathbf{x}_t^*)$ is computed from already-computed components, and $\mathbf{x}_{t+1}$ is selected, for all members of the batch. The algorithm using a Gaussian prior density is outlined in (**ref**).

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

2.4 Thermodynamics

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:

$$\mu_X(\mathbf{x}) = Z_X^{-1} \mathrm{e}^{-\beta U(\mathbf{x})} \tag{11}$$

Algorithm 1 Latent MCMC with isotropic Gaussian prior with variance σ^2.

LatentMCMC($\mathbf{x}_0$, σ, B)

1. $\mathbf{z}_0 = F_{xz}(\mathbf{x}_0)$,
2. Generate $\{\mathbf{z}_1^*, ..., \mathbf{z}_B^*\}$ with $\mathbf{z}_1^* \sim \mathcal{N}(0, \mathbf{I})$.
3. For $i = 0, ..., N$ compute

$$\begin{aligned} R_i &= \log |\det \mathbf{J}_{zx}(\mathbf{z}_i)| \\ u_{X,i} &= u_X(\mathbf{x}_i) \\ u_{Z,i} &= \frac{1}{2\sigma^2} \|\mathbf{z}_i\|^2 \end{aligned}$$

4. For $t = 1, ..., B$:
   (a) Sample $r \sim \text{Uniform}(0, 1)$
   (b) $Q = \log J(\mathbf{z}_1) - \log J(\mathbf{z}_2) + u(T_{zx}(\mathbf{z}_2)) - u(T_{zx}(\mathbf{z}_1)) + \frac{1}{2\sigma^2} \|\mathbf{z}_1\|^2 - \frac{1}{2\sigma^2} \|\mathbf{z}_2\|^2$
   (c) If $-\log r \geq Q(\mathbf{x}_t, \mathbf{x}_t^*)$: $\mathbf{x}_{t+1} = \mathbf{x}_t^*$
   (d) Else: $\mathbf{x}_{t+1} = \mathbf{x}_t$

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.

3 Results

3.1 Double well

3.2 Particle dimer

References

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

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

[3] Frank Noé and Hao Wu. Boltzmann generators - sampling equilibrium states of many-body systems with deep learning. *arXiv:1812.01729*, 2018.

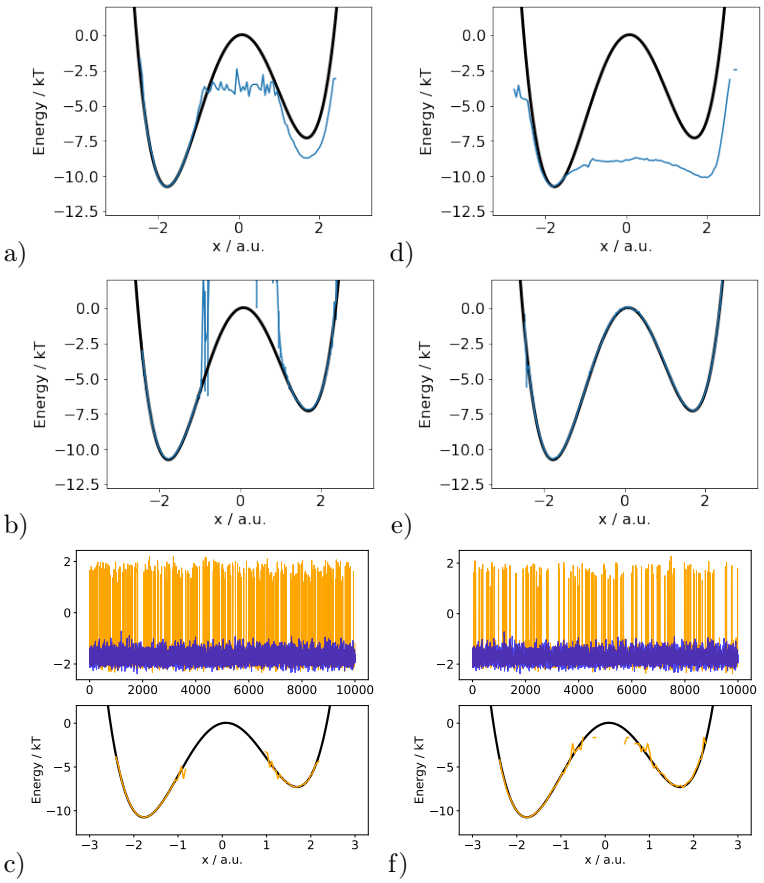

Figure 2: Boltzmann Generator for 2D double well described in [3]. Left **(a-c)**: Training with ML+KL loss. Right **(d-f)**: Training with ML+KL+RC loss. **(a,c)** Free energy corresponding to distribution generated by Boltzmann Generator. **(b,e)** Free energy after reweighting as in [3]. **(c,f)** Top: Trajectory with direct MCMC (blue) and latent MCMC (yellow). Bottom: Free energy from latent MCMC samples (yellow) compared to exact free energy (black).

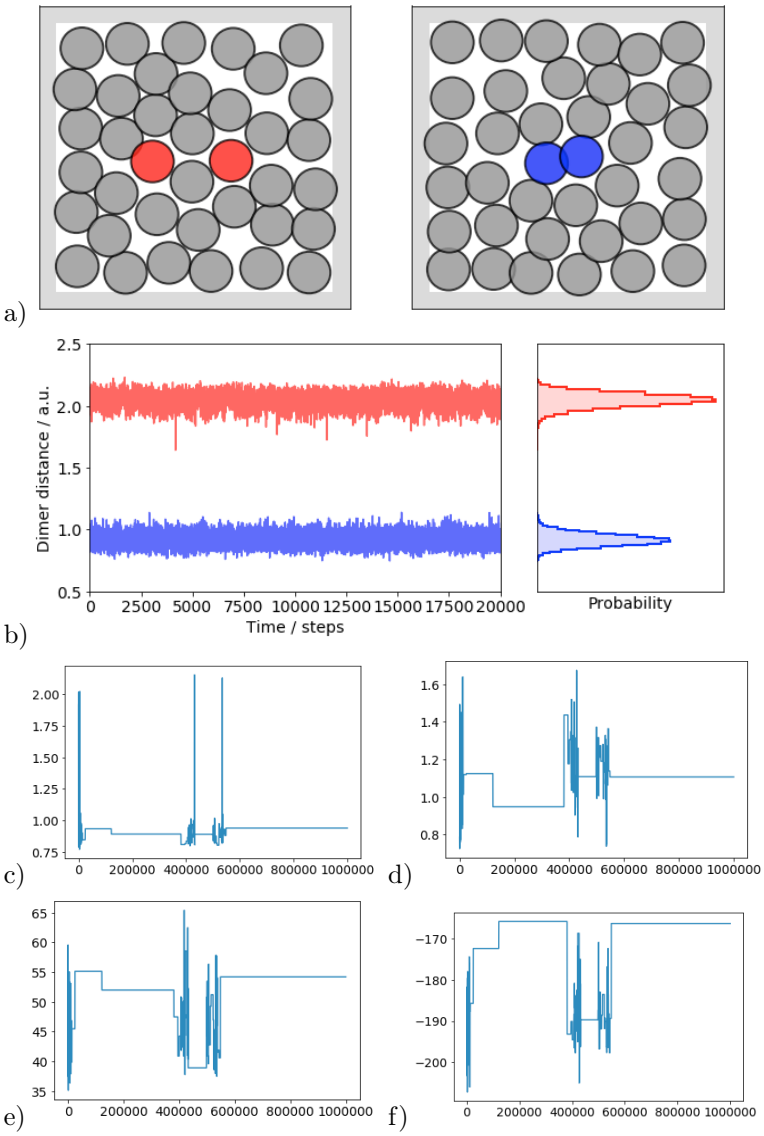

Figure 3: Boltzmann Generator for bistable repulsive particle system described in [3]. **a)** Open and closed states, dimer distance is governed by a double-well potential. **b)** Short MCMC trajectories in open and closed states. MCMC mixing time is extremely long (estimated on the order of 10^{12} time steps). **c-f)** Latent MCMC Simulation trajectory with 10^6 steps shows poor mixing due to kinetic traps. **c)** Dimer distance, **d)** Squared norm of latent vector, $\frac{1}{d}\mathbf{z}^\top\mathbf{z}$, where d is the dimension, **e)** Potential energy, **f)** log determinant of the Jacobian.