Efficient Sampling of Equilibrium States using Boltzmann Generators

Jeremy Binagia
Dept. of Chem. Eng.
Stanford University
jbinagia@stanford.edu

Sean Friedowitz
Dept. of Mat. Sci.
Stanford University
sfriedo@stanford.edu

Kevin J. Hou
Dept. of Chem. Eng.
Stanford University
kjhou@stanford.edu

Abstract

Generating independent configurations sampled from the physical Boltzmann distribution is an extremely difficult task, owing to the incredibly small volume in configuration space that high probability states (e.g. the folded state of a protein) occupy. Consequently, classical sampling of such Boltzmann distributions relies on the use of either Monte Carlo (MC) or Molecular Dynamics (MD) simulations to slowly propagate a valid initial configuration forward in time, which is a slow and computationally expensive process. We employ here a neural network based approach to generate statistically independent configurations of various physical models in hopes of improving sampling efficiency of such states. We first show that our trained Boltzmann generator model can quantitatively recreate an analytical solution for the free energy of a double-well potential/ We then proceed to apply our Boltzmann generator to a simple harmonic oscillator confined in a box, and show that the agreement between generated samples and the analytical result converges as training proceeds. We conclude our report with perspectives on ongoing and future work.

1 Introduction

Molecular simulations provide a uniquely useful tool for studying nanoscale systems. For example, simulating large biomolecules enables computational drug discovery; likewise, simulating novel polymers enables development of new materials, such as plastic electrolytes and fuel cell membranes. In practice, such simulations are difficult because we require an efficient way to sample equilibrium (low-energy) states of such systems, which often constitute only a vanishing small fraction of all possible configurations.

To illustrate this, consider a system of N particles. Each particle is described by its three spatial coordinates, leading to a system with $3N$ degrees of freedom and on the order of 10^{3N} possible configurations. For proteins or large polymers, it is not uncommon for $N > 1000$, such that sampling all configurations is computationally prohibitive. Instead, we typically initialize a known equilibrium state, and attempt to sample the distribution by making small, perturbative moves (e.g. Markov chain Monte Carlo). However, such simulations tend to get kinetically trapped; for most systems of interest, proper sampling remains computationally impossible.

Nevertheless, it is often possible to define a variable transformation which converts the coordinates of all molecules in a simulation into a set of 'reduced coordinates' upon which all interesting states lie. Such transformations typically encode physical intuition about the simulated system, such as a reaction pathway or order parameter, and enable efficient sampling of equilibrium states. There has been recent work applying deep learning to learn such transformations in arbitrary systems [4]. The network architectures and techniques presented could enable both faster simulations and provide new physical insights from molecular simulations.

We aim to further develop the technique of Boltzmann generators [4] from Noe *et al.* and to generalize their approach to more complex molecular simulations. A 'Boltzmann generator' is an invertible neural network which aims to learn a mapping between two probability distributions. The input to the model is coordinate data $\mathbf{x}$ from Monte Carlo (MC) simulations. This data is used to train the Boltzmann network, which we have structured analogously to the the layout in Noe *et al*. The 'output' of our model is, in effect, the trained model parameters which defines an invertible coordinate transform. For example, for a simulation tracking the coordinates of N particles, the network learns a bijection $f : \mathbb{R}^{3N} \leftrightarrow \mathbb{R}^{3N}$ which maps the coordinate space $\mathbf{x}$ to a latent space $\mathbf{z}$ which enables more efficient sampling than can be achieved with simple MC.

2 Related work

The application of deep learning we investigate in this report represents a particularly specific problem, so we defer this section to a general discussion of the use of deep learning in molecular simulations. Deep learning has natural application in the parameterization/evaluation of molecular force fields, i.e. the interactions between atoms and molecules. In a molecular simulation, the net force on a single particle is a complex function of the state (position, velocity, etc.) of all other particles. This complex function is specified by a physical model for the system, typically with adjustable parameters set so that simulation results are consistent with experimental data or quantum mechanical calculations. Such approaches have been integrated into notable simulation packages [8, 9] to approximate the parameterization process — physical models are encoded in the network architecture, and the loss function is designed to match simulation results with available data

Techniques from deep learning have also been applied to the Markov modeling approach for molecular simulations. One of the underlying physical challenges in molecular simulation is the disparity of timescales which must be simulated, ranging from femotoseconds for individual atomic motions to the order of microseconds to seconds for bulk properties such as viscosity. The Markov modeling approach employs the theory of Markov processes to address this problem — such techniques are loosely equivalent to a coarse-graining with respect to time, employing 'featurization' of short simulation trajectories (e.g. nanoseconds) which are then stitched together into a long-timescale process [6]. Traditionally, selecting features for short simulations is a manual process, with differences in feature design causing large variation in results. Recently, there has been efforts to abstract this featurization process into an overarching deep learning framework [3], eliminating the need for manual feature design. At this stage, the aforementioned work has achieved comparable performance to manual design, but it remains unclear how much of an advantage can be leveraged with deep learning.

Finally, we turn to the primary challenge addressed in this report — sampling rare events in MD simulation. Traditionally, this is challenging because such rare events have a vanishingly small probability of being visited in a simulation; sampling the full distribution thus requires special techniques (e.g. biasing, umbrella sampling [7]). The technique of Boltzmann generators, developed by Noe et. al. [4], uses deep learning to address this problem geometrically. The deep network described in this publication learns a coordinate transform which places high probability states 'close' (in the Euclidean sense) to one another in the transformed coordinate space. This enables efficient sampling of 'important' rare events which represent critical transitions between metastable states, e.g. stable intermediate structures in a protein folding pathway. This methodology is made possible by the recently-developed real NVP algorithm [1], which describes a network architecture that efficiently learns **n**on **v**olume-**p**reserving transformations between arbitrary probability distributions. This is a novel approach to the problem, and to the best of our knowledge the only published approach of its kind.

3 Dataset and Features

Our dataset(s) consist of the outputs of molecular systems of various systems, and are synthesized using Monte Carlo simulations. Each dataset consists of a number of states, with each state defined by a generalized state vector $\mathbf{x}$. In most cases, $\mathbf{x}$ is a $3N$-dimensional vector describing the spatial coordinates of N particles, though in general $\mathbf{x}$ may represent internal coordinates, bond angles, magnetic spin, etc. From this point, we will refer to individual data points $\mathbf{x}$ as 'configurations'. Each simulation configuration is characterized by a corresponding energy $H(\mathbf{x})$, which is in general a straightforward function of the state vector $\mathbf{x}$. The set of configurations $\mathbf{x}$ and their corresponding energies $H(\mathbf{x})$ will be used as inputs to train our neural network.

Datasets are synthesized using the Monte Carlo simulation code that we have implemented in the `models` sub-directory in the project repository. Each system that we study (e.g. the toy double well potential, the simple harmonic oscillator, etc.) is implemented as its own class, with methods for generating simulation output and wrapper functions for interfacing with our neural network code. The size of each data is simply the number of MC steps we run the simulation for, with the trade-off that larger data sets require longer simulation times. In practice, typically only short simulations/small datasets are required to train the Boltzmann generator.

4 Methods

In this project, we aim to reproduce and extend the method of 'Boltzmann generators' [4]. This method is an application of the real NVP algorithm [1] to molecular simulations. Real NVP defines a deep network architecture which efficiently learns invertible transformations between arbitrary probability distributions (c.f. Fig. 1). In particular, the network learns the function $\mathbf{z} = F_{xz}(\mathbf{x})$ and its inverse $\mathbf{x} = F_{zx}(\mathbf{z})$, such that sampling from a Gaussian distribution $p_z(\mathbf{z})$ is equivalent to sampling from the Boltzmann distribution $p_x(\mathbf{x}) = \exp(-H(\mathbf{x}))$, where the energy function $H(\mathbf{x})$ is defined by the system we are simulating. The network architecture is designed such that the Jacobian $R_{zx}(\mathbf{z}) = |\det \mathbf{J}_{zx}|$ (which measures how volume is scaled by the transformation) is cheap and easy to evaluate.

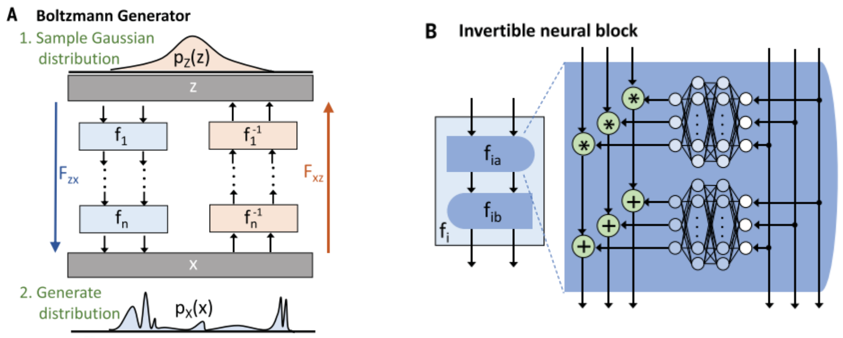

Figure 1: Schematic of the Boltzmann generator methodology, reproduced from Noe *et al.* (A) The algorithm learns a mapping (F_{zx}) from a simple latent space distribution (here a Gaussian) to the desired Boltzmann distribution through a series of stacked real NVP blocks. (B) Each real NVP block consists of two subunits that pass half the inputs through two fully-connected networks that respectively learn a scaling and translation of the remaining inputs. Adjacent sub-units alternate the inputs that are passed through the two networks.

Design of the loss function is critical to training the transformation network. We use a loss function consisting of two parts, $J = J_{\text{KL}} + \alpha J_{\text{ML}}$, where α is used to adjust the relative magnitude of each term. The first term is denoted the Kullback-Leibler loss, and has the form

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

The KL loss is used to train random (Gaussian) samples from the latent space $\mathbf{z}$. Minimizing the first term of J_{KL} results in F_{zx} mapping to low-energy configurations $\mathbf{x}$ in real space. The latter term is an entropic penalty which prevents the network from mapping to a single low-energy state in $\mathbf{x}$-space.

The second loss term is used to 'seed' the network with existing data from MC simulations, and has the form

$$J_{ML} = \mathbb{E}_{\mathbf{x}}\left[\frac{1}{2}\left\|F_{xz}(\mathbf{x})\right\|^2 - \log R_{xz}(\mathbf{x})\right] \tag{2}$$

In the above, $R_{xz}(\mathbf{x}) = |\det \mathbf{J}_{xz}(\mathbf{x})|$. The first term drives the network to map high-probability states $\mathbf{x}$ from training data to the center of the Gaussian $p_z(\mathbf{z})$. This ensures that the states in our training data are seen when sampling from the generator. Similarly to above, the second term is an entropic penalty that prevents the collapse of all low-energy states in $\mathbf{x}$ to the same point in $\mathbf{z}$.

For the following examples, our network architecture consists of four stacked real NVP blocks (c.f. Fig. 1B), each containing fully-connected networks with three hidden layers. We use ReLU and tanh activations in the scaling (*) and translation (+) networks respectively. Finally, we use the Adam optimizer [2] for training.

5 Experiments/Results/Discussion

5.1 Double well potential

Due to difficulties using the source code in the original publication [4], we wrote our own implementation of the network in PyTorch [5]. For validation, we tested our model on the double well potential introduced in Noe *et al*. This toy model has the functional form $H(x_1, x_2) = x_1^4 - x_1^2 + x_1 + x_2^2/2$. The energy landscape is illustrated in Fig. 2A — there are two low energy wells separated by a transition state at $x_1 = 0$.

We begin training using only the ML loss, J_{ML}. We use 1000 samples of low energy states (shown as dots in Fig. 3A) as a training set, and train until the loss converges. We then proceed by training with the KL loss J_{KL}. The training set in this case is 1000 samples from latent space $\mathbf{z}$ (i.e. Gaussian random variables), transformed into real space using the network to evaluate $\mathbf{x} = F_{\text{zx}}(\mathbf{z})$. In contrast with Noe *et al.*, we found that optimal performance was attained using 256 nodes per hidden layer in the invertible blocks (vs. 100 in the original publication). For training, we used a learning rate of 0.0001 and a batch size of 1000.

The fully trained model is visualized Fig. 3, which shows the transformation learned by the network. The two potential wells, shown in Fig. 3A, are mapped to the Gaussian distribution in Fig. 3B. The left-most deeper well occupies more space because it has a higher probability (lower energy) in real space. The trained network

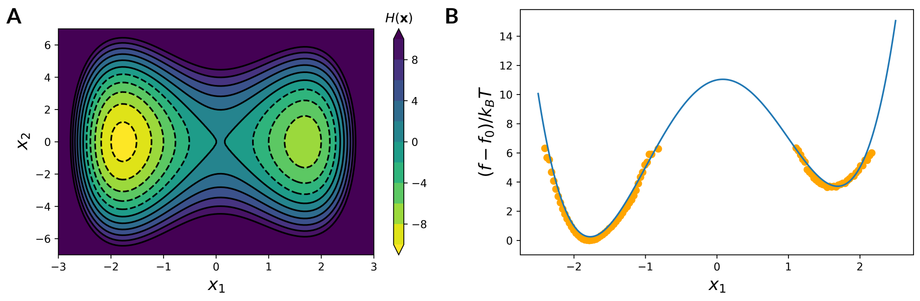

Figure 2: (A) Energy landscape and (B) Free energy as a function of x_1 for the double well potential.

can be used to sample the double well potential by generating Gaussian variable in $\mathbf{z}$-space (Fig. 3C) and transforming to real space with $F_{zx}(\mathbf{z})$ (Fig. 3D). The agreement between Figures A and D demonstrates our implementation is correct.

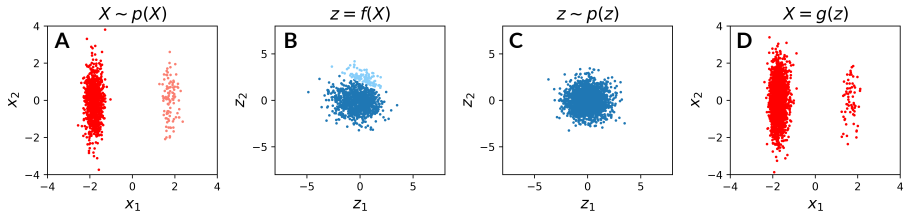

Figure 3: Application of Boltzmann generators to the double well potential (A) Sampling in real space (B) Transformation to latent space (C) Independent sampling in latent space (D) Inversion into real space.

We may now briefly elaborate on the benefits of using the Boltzmann generator approach. For more complex examples, it is often not possible to sample the source distribution $p(X)$ with conventional methods (Fig. 3A). After training, we replace the difficult problem of sampling $p(X)$ with the easy problem of sampling Gaussian random variables and transforming them back to real space. Sampling in this way lets us compute thermodynamic properties, such as the free energy w.r.t the reaction coordinate x_1 (Fig. 2B). Excellent agreement with the analytical solution for the free energy (shown in blue in Fig. 3) illustrates the utility of our approach.

5.2 Simple harmonic oscillator

Using our verified Boltzmann generator implementation, we further investigate our methodology using a system with a rigorous analytical solution. We consider a toy model with two particles confined in a 1D box, connected by a spring with force constant k. This simple harmonic oscillator is completely defined by the state vector $\mathbf{x} = [x_1, x_2]$, where x_1 and x_2 are the positions of each particle. The energy of this system is given by $H(\mathbf{x}) = k(x_2 - x_1)^2$, with constraint $0 \leq x_1 \leq L$. The Boltzmann distribution has the exact form

$$p(\mathbf{x}) = \sqrt{\frac{k}{\pi L^2}} \exp\left(-k(x_2 - x_1)^2\right) \quad , \quad 0 \leq x_1 \leq L \, . \tag{3}$$

The generator network attempts to learn the transformation $\mathbf{z} = F_{xz}(\mathbf{x})$ such that $p(z)dz = p(x)dx$ and $p(z)dz$ is a normal distribution. The transformation function has the analytical solution

$$\mathbf{z} = F_{zx}(\mathbf{x}) = \begin{bmatrix} \Phi^{-1}\left(\frac{x_1}{L}\right) \\ \sqrt{2k}\,|x_2 - x_1| \end{bmatrix} \, . \tag{4}$$

Above, Φ^{-1} is the inverse Gaussian cumulative distribution function. We may evaluate the accuracy of our network by comparing the learned transformation to this 'perfect' transformation.

In training this model, we use the Adam optimizer [2] with default parameters. We use mini-batches with size 128 and learning rate of 0.0001. Our training set consists of 3200 points sampled from the probability distribution above. Training by example yields the results seen in Fig. 5, while Fig. 4 illustrates the performance of the untrained network. From these figures we can clearly see the performance of the Boltzmann generator improve (i.e. from Fig. 4D to Fig. 5D) as the learned transformation converges to a Gaussian (c.f. Fig. 5B).

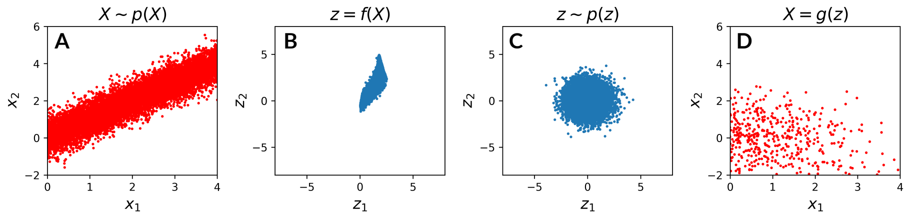

Figure 4: Prior to training, the network cannot recreate the starting distribution for the harmonic oscillator.

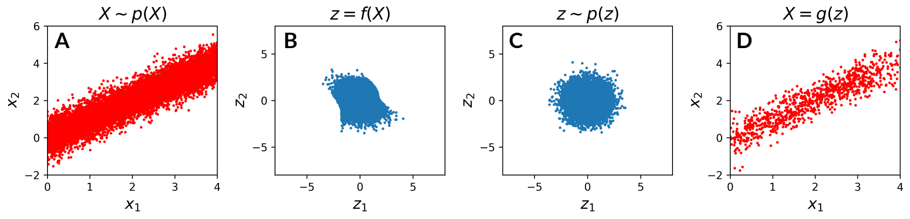

Figure 5: After training, the learned transformation is quite close to a Gaussian (Fig. 5B). Inverting samples from latent space (Fig. 5C) produces a distribution (Fig. 5D) closely resembling the true distribution (Fig. 5A).

6 Conclusion/Future Work

We have implemented, from scratch, the method of Boltzmann generators [4]. This framework allows us to efficiently sample equilibrium states of molecular systems using deep learning. By considering a simple energy landscape in two dimensions (the so-called double well potential), we demonstrate that our method can successfully generate low energy states of the system from direct sampling in a far simpler latent space distribution (here a Gaussian). We show that such one-shot sampling allows us to efficiently compute equilibrium properties requiring a large sample size such as the free energy of the system. With out method successfully validated, we then proceeded to demonstrate its use on a new system not considered by Noe *et al.*, that of two particles confined in a box and connected to one another with a spring. For this system, we again show that Boltzmann generators can be utilized to recapitulate the Boltzmann distribution. We also analytically solve for the transformation that the network is trying to learn and compare it to the learned transformation.

Mechanistically, the network architecture used in the Boltzmann generator is largely unchanged from the real NVP algorithm. The strength of the real NVP algorithm lies in generality — in principal, this network should be able to learn transformations between any two arbitrary probability distributions. That said, for the particular distributions encountered in molecular simulations, it should be possible to make substantial improvements to the network architecture by exploiting the properties of the Boltzmann and Gaussian distributions. Our justification lies in the observation that a 'perfect', minimum loss network must satisfy the equality $p_x(\mathbf{x})d\mathbf{x} = p_z(\mathbf{z})d\mathbf{z}$ where $\mathbf{z} = f(\mathbf{x})$, with the function f representing the trained network. The transformation function f must therefore satisfy the following differential equation:

$$\det\left(\frac{df(\mathbf{x})}{d\mathbf{x}}\right) = \frac{p_x(\mathbf{x})}{p_z(f(\mathbf{x}))}$$

Real NVP is designed such that the determinant on the left hand side above is always cheap and easy to evaluate given the network representation of $f(\mathbf{x})$. Thus, the algorithm generalizes well to arbitrary distributions p_x and p_z. However, for a Boltzmann-distributed p_x and Gaussian p_z in particular, this equation simplifies further:

$$\det\left(\frac{df(\mathbf{x})}{d\mathbf{x}}\right) = \frac{\exp(-H(\mathbf{x}))}{\exp(-|\mathbf{z}|^2/2)} = \exp\left(\frac{|f(\mathbf{x})|^2}{2} - H(\mathbf{x})\right)$$

We can see that because both the Gaussian and Boltzmann are exponential distributions, the differential equation describing the $f(\mathbf{x})$ the network must learn is substantially simpler than for the general case. In general, so long as the energy function $H(\mathbf{x})$ is well-behaved, $f(\mathbf{x})$ actually has an analytical solution, albeit one that is too cumbersome for practical use. It follows that there should exist some network architecture which takes advantage of this property of the two distributions, which should enable faster training of the generator. Though we have laid out some of the initial mathematical justification, significant architecture design and research into similar techniques would be necessary to generate ideas for a new architecture, and significant time for testing. While we find this an academically interesting endeavor, it is not particularly necessary since the out-of-the-box real NVP algorithm works quite well. Thus we have determined this is beyond the scope of the project for this quarter and have set it as a goal for future work.

7 Contributions

Jeremy wrote the code for training and visualizing the models (i.e. the notebooks `analytical_example.ipynb`, `ising_boltzmann_training.ipynb`, and `double_well.ipynb`). Kevin performed the analytical calculations and fixed several critical bugs in the aforementioned notebooks. Sean wrote the classes for the Ising model and other models not mentioned in this report, calculated the free energy for the double well potential, and provided useful discussion for concepts related to molecular simulation. Jeremy and Kevin wrote the final report with Sean editing. The code for this project may be found at `https://github.com/jbinagia/CS-230-Final-Project.git`.

References

[1] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. *CoRR*, abs/1605.08803, 2016.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] Andreas Mardt, Luca Pasquali, Hao Wu, and Frank Noé. Vampnets for deep learning of molecular kinetics. *Nature Communications*, 9(5), 2018.

[4] Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457), 2019.

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[6] Ch Schütte, A Fischer, W Huisinga, and P Deuflhard. A direct approach to conformational dynamics based on hybrid monte carlo. *Journal of Computational Physics*, 151(1):146 – 168, 1999.

[7] Glenn M Torrie and John P Valleau. Nonphysical sampling distributions in monte carlo free-energy estimation: Umbrella sampling. *Journal of Computational Physics*, 23(2):187–199, 1977.

[8] Han Wang, Linfeng Zhang, Jiequn Han, and Weinan E. Deepmd-kit: A deep learning package for many-body potential energy representation and molecular dynamics. *Computer Physics Communications*, 228:178 – 184, 2018.

[9] Linfeng Zhang, Jiequn Han, Han Wang, Roberto Car, and Weinan E. Deep potential molecular dynamics: A scalable model with the accuracy of quantum mechanics. *Phys. Rev. Lett.*, 120:143001, Apr 2018.