Boltzmann Generators for Efficient Molecular Simulations

https://github.com/jbinagia/CS-230-Final-Project.git

Jeremy Binagia (jbinagia), Sean Friedowitz (sfriedo), Kevin Hou (kjhou)

November 8, 2019

1 Introduction/Motivation

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 challenge, consider a simulation 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 some well known equilibrium state, and attempt to sample the distribution by making small, perturbative moves using techniques such as 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 [2]. The network architectures and techniques presented in this study have the potential to enable both faster simulations, and to provide new physical insights from molecular simulations. Noe *et al.* present promising early successes with simple systems. In this work, we aim to further develop the technique of Boltzmann generators and generalize their approach to more complex molecular simulations.

2 Dataset and Methodology

Our dataset(s) consist of the outputs of molecular systems of various systems, described by a generalized state vector $\vec{x}$. This may represent particle coordinates, bond angles, bond lengths, etc. From this point, we will refer to individual data points $\vec{x}$ as 'configurations'. Each simulation configuration is characterized by a corresponding energy $H(\vec{x})$, which is in general a straightforward function of the state vector $\vec{x}$. The set of configurations $\vec{x}$ and their corresponding energies $H(\vec{x})$ will be used as input to our neural network. Our datasets are synthesized using MC-Monte Carlo simulations, which are implemented in the `models` sub-directory in our project repository. Each molecular system we study (e.g. nematic liquid crystals, polymer chains in solvent, Ising model, etc.) is implemented as its own class, with methods for generating simulation output and wrapper functions for interfacing with our neural networks.

In this project, we aim to reproduce and extend the method of 'Boltzmann generators' in Noe *et al.* (available online). This work is an application of the RealNVP algorithm[1], which describes a neural network architecture for transforming between probability distributions, to molecular simulations. In particular, it is an application of RealNVP to transform arbitrary

Boltzmann distributions of the form $\exp(-H(\vec{x}))$ onto a multivariate Gaussian prior. Conceptually, 'Boltzmann generators' attempt to learn a coordinate transform $\vec{z} = f(\vec{x})$ such that the Boltzmann distribution in coordinate space $\exp(-H(\vec{x}))$ becomes approximately Gaussian in $\vec{z}$ space. Such transformations enable efficient sampling, and can provide physical insight by illuminating reaction pathways or identifying important folded configurations of a protein.

We propose to extend the analysis of Noe *et al.* to two systems. First, we will apply this technique to the well-known Ising model. This system is simple to simulate, and has a well-known analytical solution we can use for validation. Next, we will apply the methodology to nematic liquid crystals. This system has a similar known analytical solution, but is substantially more challenging to simulate, thus providing a benchmark to evaluate the efficacy of this technique.

Finally, time permitting, we aim to extend this methodology to Gibbs ensemble simulations to study phase coexistence. Here, two Monte-Carlo simulations are run in parallel with some information exchange. We should be able to reproduce this construction using Boltzmann generators by training parallel networks on each of the two coupled simulations, with careful modifications to the loss function. This 'stretch goal' is still in a preliminary stage, but would be a significant contribution to this field of research.

At this stage, we have not made substantial modifications to the methods presented in Noe *et al.* — network architecture and implementation details can be found in their manuscript. As we iterate on the architecture, we will document these changes in greater detail.

3 Progress

We have completed the simulation code needed to generate training data (`models, mcmc`), as well as a working implementation of the Boltzmann generator deep network and training losses from Noe *et al.* (`networks, training`). In this section, we present preliminary results obtained from applying these networks to the simulation to a simple 'toy model'. A more detailed presentation of this model can be found in our project repository (`analytical_example.ipynb`).

For our toy model, we consider two particles confined in a 1D box, connected by a spring with force constant k. This system is completely defined by the state vector $\vec{x} = [x_1, x_2]$, where x_1 and x_2 are the positions of each particle. The energy of this system is simply:

$$H(\vec{x}) = \begin{cases} k(x_2 - x_1)^2 & : \text{ if } 0 \le x_1 \le L \\ \infty & : \text{ otherwise} \end{cases}$$

We choose this potential for our model because its probability distribution can be solved exactly:

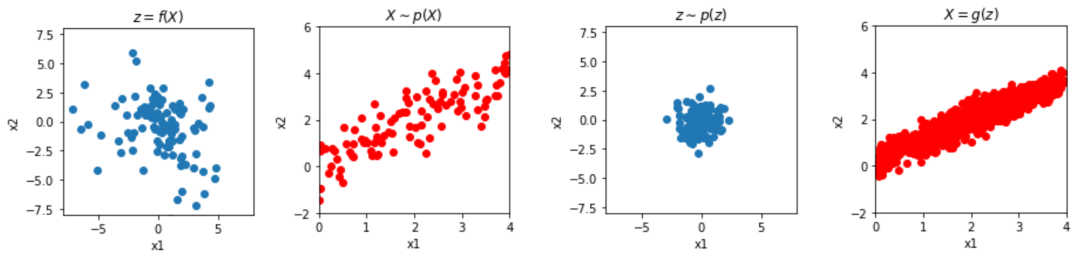
$$p(\vec{x}) = \sqrt{\frac{k}{\pi L^2}} \exp\left(-k(x_2 - x_1)^2\right) \quad , \quad 0 \le x_1 \le L$$

Recall the Boltzmann generator network is attempting to learn a transformation $\vec{z} = f(\vec{x})$ such that $p(z)dz = p(x)dx$ and $p(z)dz$ is a normal distribution. In that sense, the deep network is trying to approximate the function $f(\vec{x})$. Construction of this example is helpful because for such a simple system, we can analytically solve for this $f(\vec{x})$.

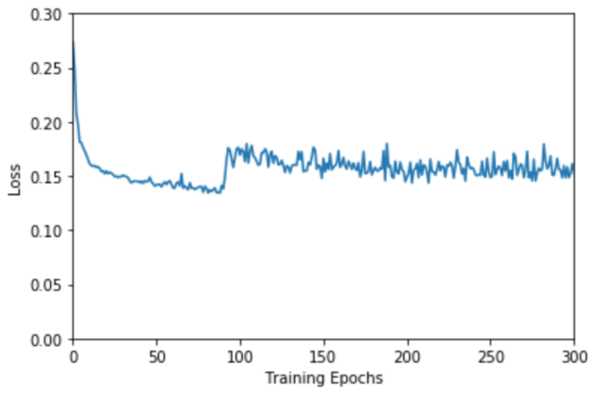
$$f(x_1, x_2) = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} \Phi^{-1}\left(\frac{x_1}{L}\right) \\ \sqrt{2k}\,|x_2 - x_1| \end{bmatrix}$$

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.

As outlined above, the Boltzmann generator network learns a variable transform $\vec{z} = f(\vec{x})$ such that a Gaussian distribution in $\vec{z}$-space maps to the Boltzmann distribution in $\vec{x}$-space. Thus, after the network has been trained, we can more efficiently sample the original $\vec{x}$-space by generating normally distributed random coordinates in $\vec{z}$ space and applying the reverse transformation (i.e. use the trained network) $\vec{z} = f^{-1}(\vec{x})$. In the below, we present the results of training a Boltzmann generator on the toy model described above.



The plot above visualizes the transformation $\vec{z} = f(\vec{x})$ learned by the network. The first two sub-figures shows the training data; the latter is original training data, in real ($\vec{x}$) space, and the former shows the training data transformed onto $\vec{z}$ space. If the network learns perfectly, the first plot should sample a normal distribution. The third and fourth sub-figures shows the results of sampling from the Boltzmann generator. The third image shows points sampled from a Gaussian distribution, and the fourth image shows those points inverse-transformed onto real ($\vec{x}$) space. If the network learned perfectly, the final image should reproduce the training distribution exactly. The network partially reproduces the source distribution, but predicts a distribution with much less variance. To understand this mismatch, we also show the network training loss (using batch gradient descent) plotted against the number of epochs:



As seen above, our network loss function appears to blow up at longer training times. Determining why this occurs is our top priority. We are currently focused on adjusting the network architecture and loss functions to better mach the source and target distribution. Our intuition is that, without being able to obtain a near-exact solution for such a simple model, we should not expect this methodology to generalize to more complicated systems. In more complex simulations, the 'exact' variable transformation $f(\vec{x})$ will in general be some complex, non-analytical function which is likely harder for the network to learn.

4 Next Steps

We are aiming to understand what variable transformations the network is actually learning, and how changing the network architecture and loss functions affects these transformations. Focusing on this simple toy model allows us to develop more intuition about this methodology, gives us a simple and quick-to-train system for testing changes to the architecture, and allows us to debug errors in our implementation. At this stage, the simulation (data-synthesis) code for our other systems of interest is complete, so our remaining work for this project will lie in adjusting the network architecture and training these deep networks on new systems.

References

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

[2] 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.