Boltzmann generators

Sampling equilibrium states of many-body systems with deep learning

Author: Noé, F., Olsson, S., Köhler, J. and Wu, H.

Presenter: Yihao Liang, Jiahao Qiu
Nov 2, 2023

Outline

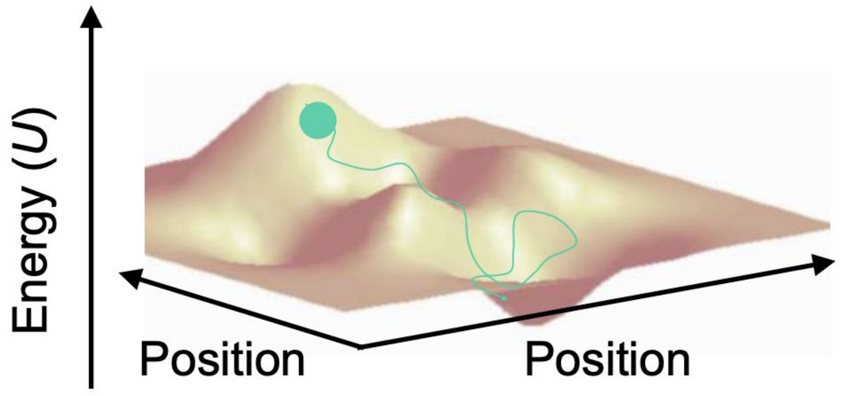
Outline - Background

- **MD Simulation**
    - Basic Idea
    - Relationship between energy and force
    - Basic algorithm
- **Example: understanding the process of protein folding**
- **Two Questions:**
    - How to sample from from the Boltzmann distribution?
    - Why MD needs to sample from the Boltzmann distribution in many steps?
- **Introduction to Boltzmann Generator**
    - Ideas behind Boltzmann Generator
    - Comparison with traditional Machine Learning

Molecular dynamics - Basic idea

Mimic what atoms do in real life, assuming a given potential energy function

- The energy function allows us to calculate the force experienced by any atom, given the positions of the other atoms
- Newton’s laws tell us how those forces will affect the motions of the atoms



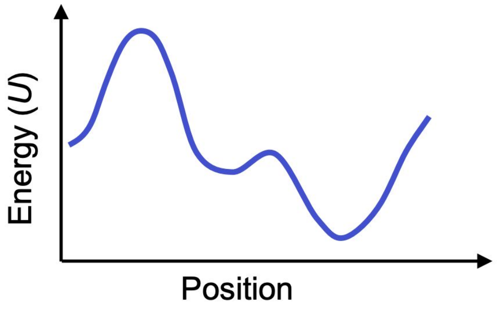
Reference: https://web.stanford.edu/class/cs279/lectures/lecture4.pdf

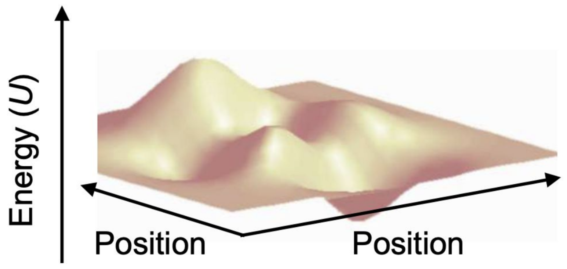
Relationship between energy and force

- A potential energy function U(x) specifies the total potential energy of a system of atoms as a function of all their positions (x)
- Force on atom i is given by derivatives of U with respect to the atom's coordinates xi, yi, and zi

$$F(x) = -\nabla U(x)$$

- At local minima of the energy U, all forces are zero





Reference: https://web.stanford.edu/class/cs279/lectures/lecture4.pdf

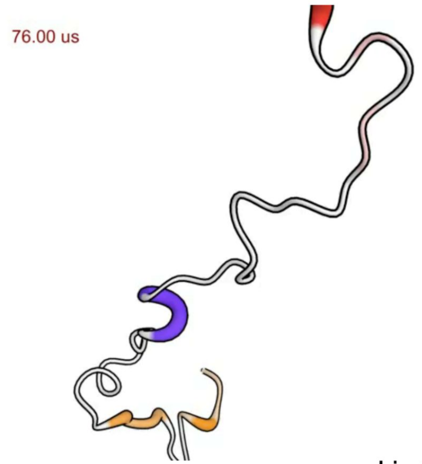
Molecular dynamics - Basic algorithm

- Divide time into discrete time steps, no more than a few femtoseconds each (1 fs = 10–15 s)
- At each time step:
  1. Compute the forces acting on each atom, using a molecular mechanics force field
  2. Move the atoms a little bit: update position and velocity of each atom using Newton’s laws of motion

Reference: https://web.stanford.edu/class/cs279/lectures/lecture4.pdf

Understanding the process of protein folding

- **How does the protein get from its unfolded state to its folded state (i.e., how does it “find” its folded structure)?**



Lindorff-Larsen et al., *Science* 2011

Reference: https://web.stanford.edu/class/cs279/lectures/lecture4.pdf

What is the probability of a protein to be folded as a function of the temperature?

Molecular Dynamics:
1. Initialization: The initial positions and velocities of all the atoms in the protein are set. The velocities are usually assigned randomly according to the Boltzmann distribution at a given temperature.
2. Force Calculation: The forces acting on each atom are calculated using a potential energy function, which describes the interactions between the atoms.
3. Integration: The equations of motion are integrated over a small time step to update the positions and velocities of the atoms.
4. Iteration: Steps 2 and 3 are repeated many times to simulate the dynamics of the protein over a certain period of time.
5. Analysis: The trajectory of the protein (i.e., the sequence of its configurations over time) is analyzed to calculate the probability of the protein being in a folded state as a function of temperature. This can be done by counting the number of times the protein is in a folded state and dividing by the total number of configurations.

Note: Generated by GPT4

How to sample from the Boltzmann distribution?

The Boltzmann distribution:
gives the probability of a system being in a particular state as a function of its energy and temperature

Metropolis Monte Carlo:

- A random move is proposed (e.g., changing the position of an atom), and the energy difference between the new and old states is calculated.
- If the energy is lower in the new state, the move is accepted.
- If the energy is higher, the move is accepted with a probability given by the Boltzmann factor, $\exp(-\Delta E/kT)$, where ΔE is the energy difference, k is the Boltzmann constant, and T is the temperature.

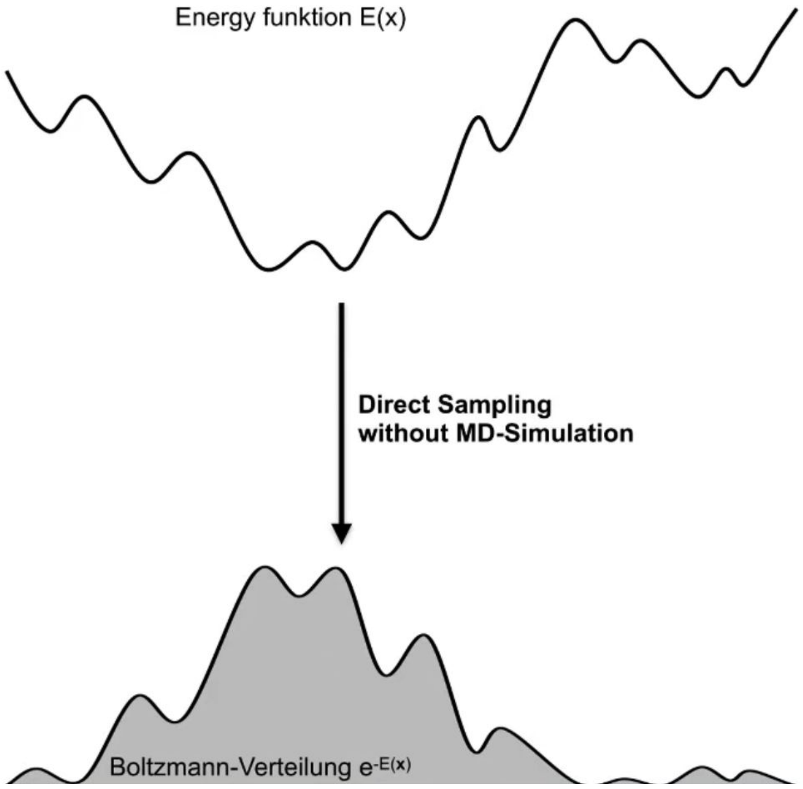
Why MD needs to sample from the Boltzmann distribution in many steps?

Complex systems like proteins have many metastable states, which are states that the system can stay in for a long time before transitioning to another state.

These transitions are rare events, so many sampling steps are needed to accurately capture the dynamics of the system.

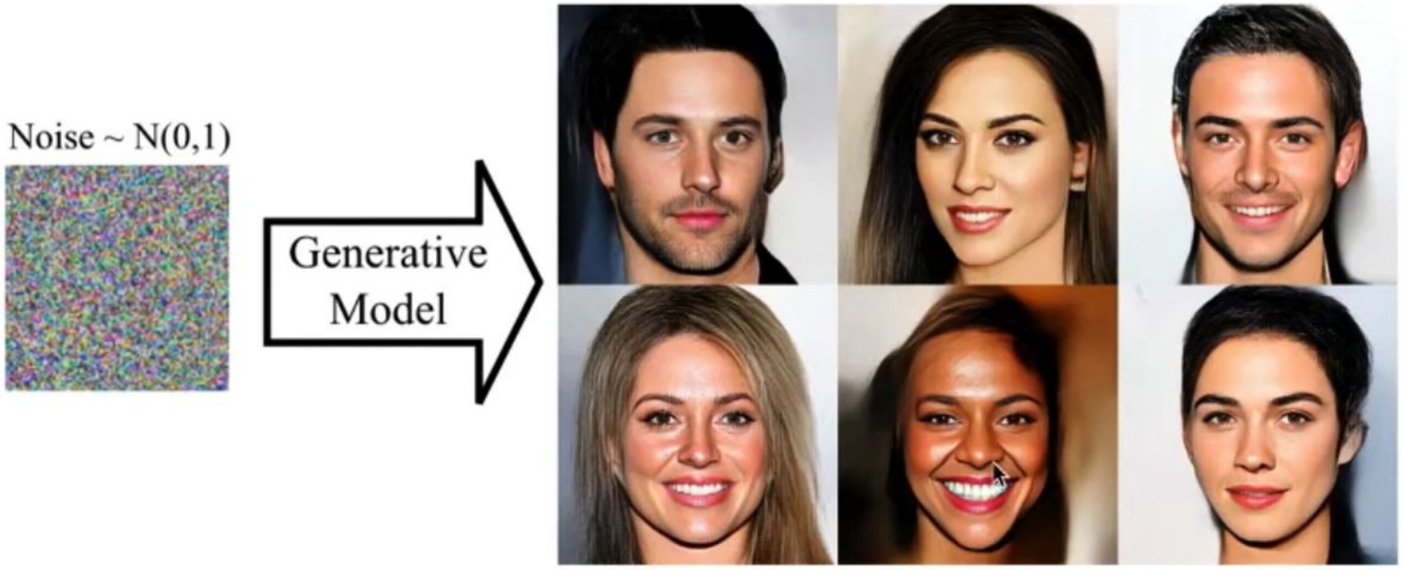
Remark: 10^9 to 10^15 MD simulation steps are needed to fold or unfold a protein. MCMC and MD methods are extremely expensive and consume much of the worldwide supercomputing resources.

Idea: Learn to sample directly



Reference: https://www.youtube.com/watch?v=WuXJRswYIaA&t=1169s

Glow: Generative Flow with Invertible 1x1 Convolutions



Reference: https://arxiv.org/abs/1807.03039

Comparison with traditional Machine Learning

- The focus is not on learning the probability distribution from given data points, but rather on generating a distribution that closely approximates the Boltzmann distribution.
- There is a re-weighting step involved in the process to ensure that the generated distribution is unbiased and suitable for sampling unbiased expectation values in physics.

Reference: https://www.youtube.com/watch?v=WuXJRswYIaA&t=1169s

Outline - Method

- Invertible Neural Networks
- Math Background
  - Jacobian matrix and determinant
  - Change of Variable Theorem
- Flow-based Generative Model
  - Generative Models & Generator
  - Flow-based model
  - Normalizing Flow
- Boltzmann Generator

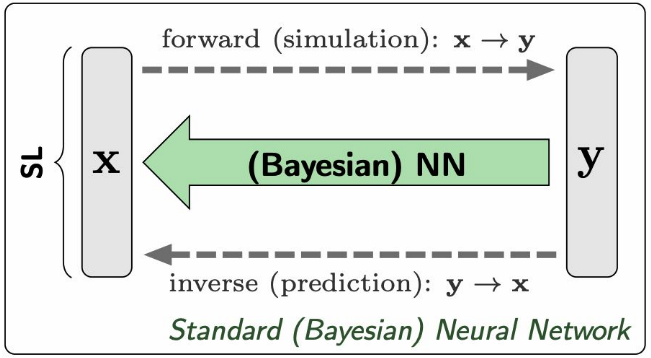
Invertible Neural Networks (INN)

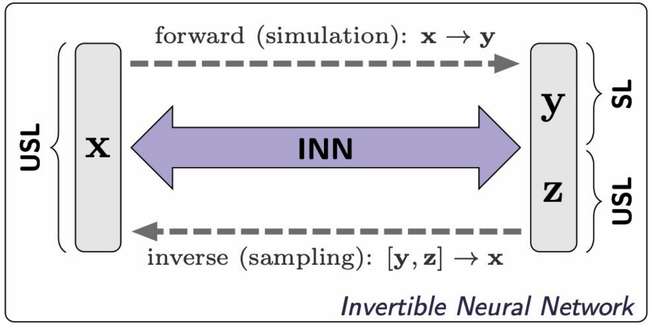
Inverse: predict the latent variables based on measured values.

The inverse process is a one-to-one mapping.

The features of INN:

- The shape of input (C × H × W) matches the output in the network.
- The Jacobian determinant is not equal to zero in the network.
- The mapping from the input to the output is a bijection.


forward (simulation): $\mathbf{x} \to \mathbf{y}$
SL
$\mathbf{x}$
(Bayesian) NN
$\mathbf{y}$
inverse (prediction): $\mathbf{y} \to \mathbf{x}$
Standard (Bayesian) Neural Network


forward (simulation): $\mathbf{x} \to \mathbf{y}$
USL
$\mathbf{x}$
INN
$\mathbf{y}$
$\mathbf{z}$
SL
USL
inverse (sampling): $[\mathbf{y}, \mathbf{z}] \to \mathbf{x}$
Invertible Neural Network

Reference: https://arxiv.org/abs/1808.04730

Invertible Neural Networks (INN)

Advantages:

- The model is lightweight because encoding and decoding use the same parameters.
- It remains the detailed information of the input data because the invertible network is information lossless.
- INN use a constant amount of memory to compute the gradient regardless of the depth of the network.
- **Reduce memory consumption**

Reference: https://arxiv.org/abs/1707.04585 The Reversible Residual Network: Backpropagation Without Storing Activations

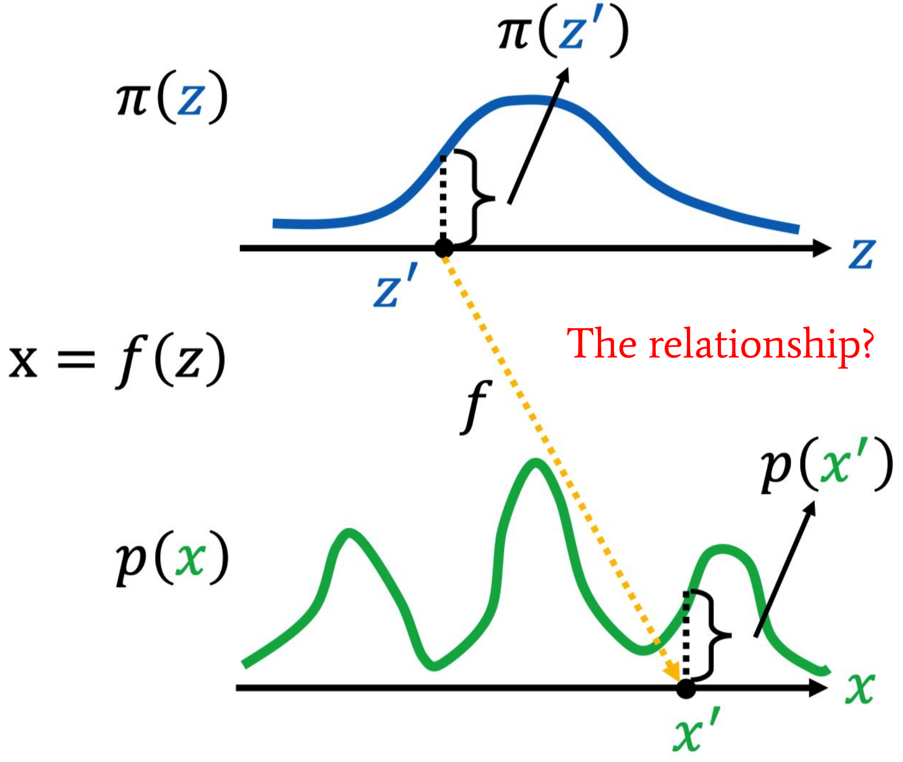
Jacobian matrix and determinant

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$x = f(z) \qquad z = f^{-1}(x)$$

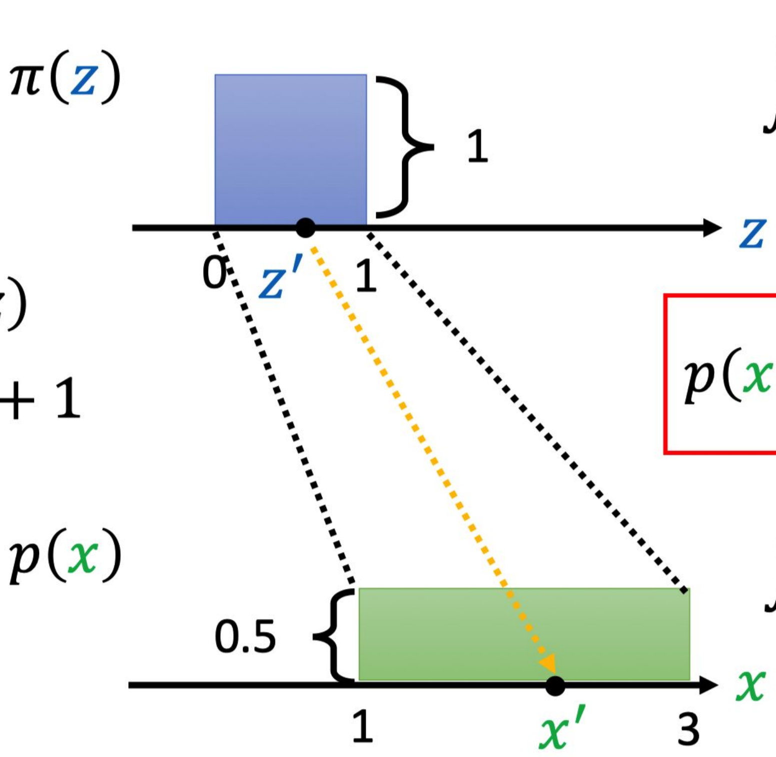
$$J_f = \overbrace{\begin{bmatrix} \partial x_1/\partial z_1 & \partial x_1/\partial z_2 \\ \partial x_2/\partial z_1 & \partial x_2/\partial z_2 \end{bmatrix}}^{\text{input}} \Big| \text{output}$$

$$J_{f^{-1}} = \begin{bmatrix} \partial z_1/\partial x_1 & \partial z_1/\partial x_2 \\ \partial z_2/\partial x_1 & \partial z_2/\partial x_2 \end{bmatrix}$$



Reference: https://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2019/Lecture/FLOW%20(v7).pdf

Change of Variable Theorem



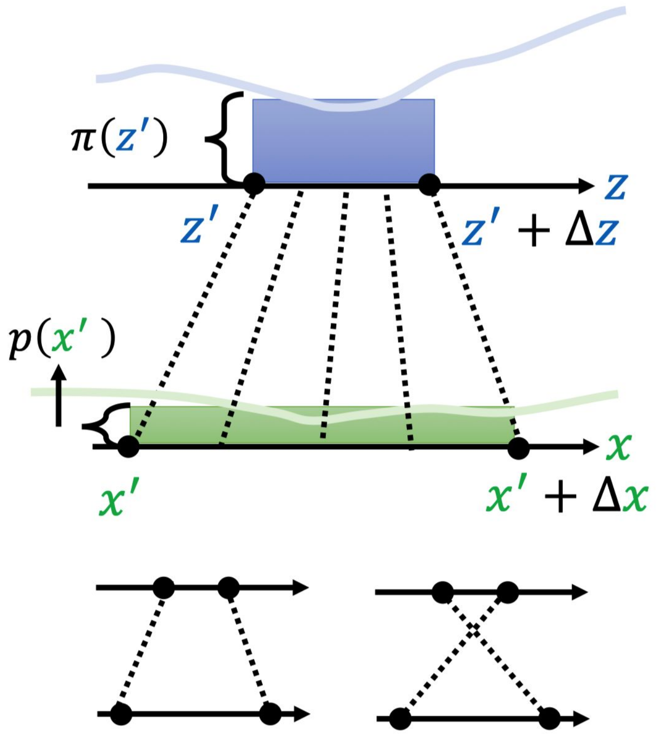
$$\int \pi(z)dz = 1$$

$$\mathrm{x} = f(z)$$
$$= 2z + 1$$

$$\boxed{p(x') = \frac{1}{2}\pi(z')}$$

$$\int p(x)dx = 1$$

Change of Variable Theorem

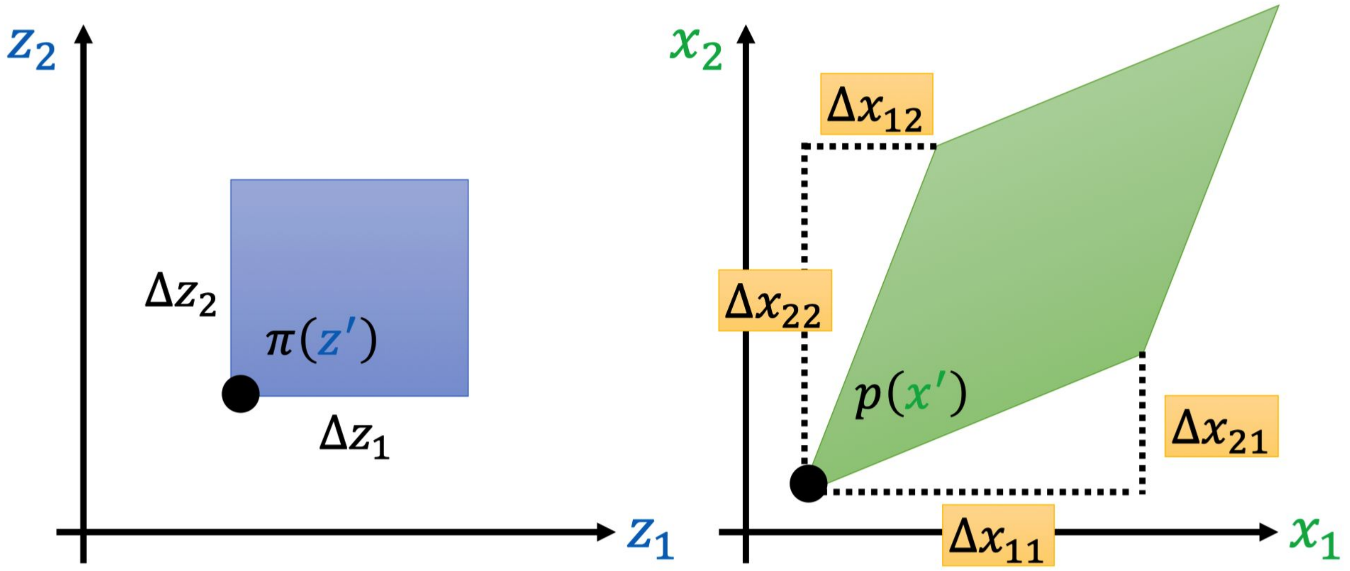


$$p(x')\Delta x = \pi(z')\Delta z$$

$$p(x') = \pi(z')\frac{\Delta z}{\Delta x}$$

$$p(x') = \pi(z')\left|\frac{dz}{dx}\right|$$

Change of Variable Theorem



$$p(x')\left|det\begin{bmatrix}\Delta x_{11} & \Delta x_{21}\\ \Delta x_{12} & \Delta x_{22}\end{bmatrix}\right| = \pi(z')\Delta z_1 \Delta z_2$$

Change of Variable Theorem

$$p(x')\left|det\begin{bmatrix}\partial x_1/\partial z_1 & \partial x_1/\partial z_2\\ \partial x_2/\partial z_1 & \partial x_2/\partial z_2\end{bmatrix}\right| = \pi(z')$$

$$x = f(z)$$

$$p(x')\left|det(J_f)\right| = \pi(z')$$

$$p(x') = \pi(z')\left|det(J_{f^{-1}})\right|$$

$$p(x') = \pi(z')\left|\frac{1}{det(J_f)}\right|$$

$$\det(M^{-1}) = (\det(M))^{-1}$$

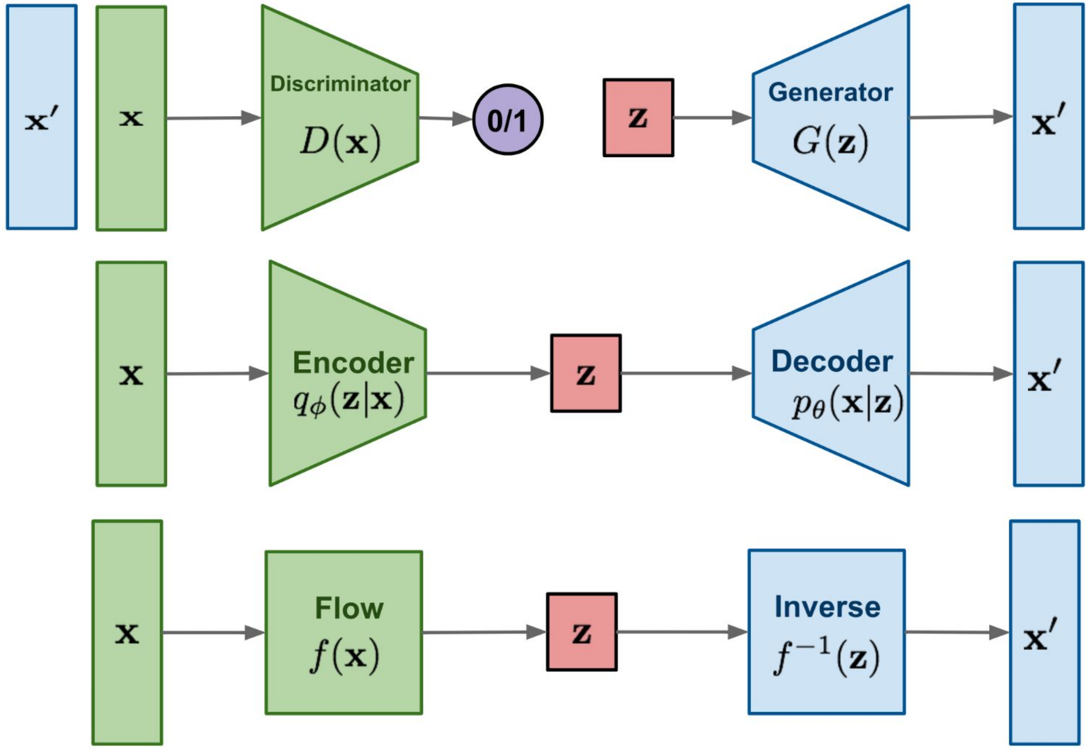
The Jacobian determinant is not equal to zero in the network.

Generative Models & Generator

GAN: minimax the classification error loss.

VAE: maximize ELBO.

Flow-based generative models: minimize the negative log-likelihood

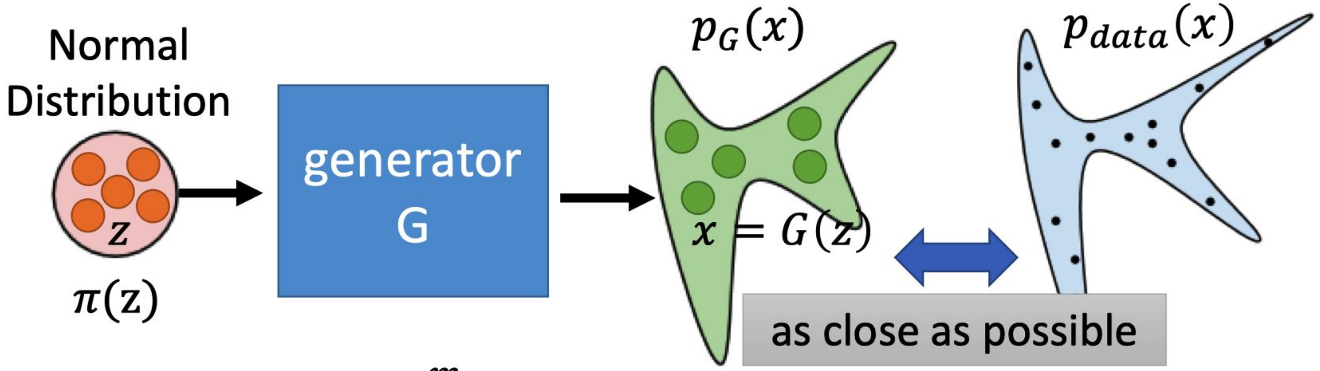


Generative Models & Generator

Auto-regressive Model: Slow generation

GAN (Generative Adversarial Network): Unstable training

VAE (Variational Auto-encoder): Optimizing a lower bound



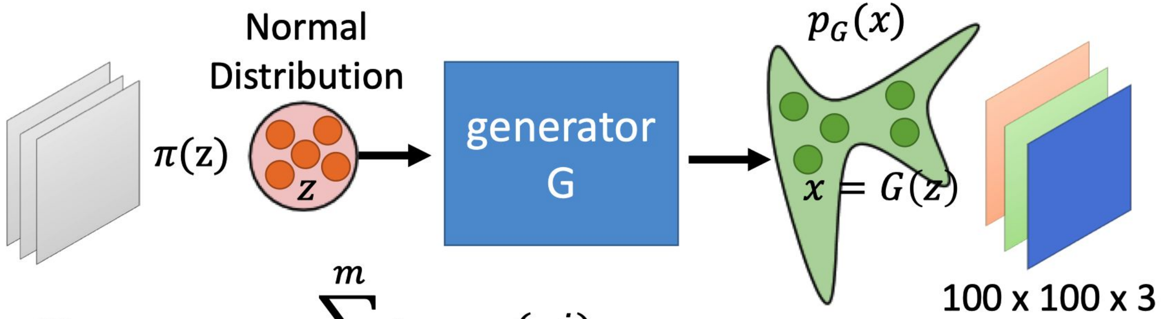
$$G^* = arg\max_G \sum_{i=1}^{m} log P_G(x^i)$$

$\{x^1, x^2, \dots, x^m\}$ from $P_{data}(x)$

Flow-based model

Directly optimizes the objective function

$$p(x')|det(J_f)| = \pi(z')$$
$$p(x') = \pi(z')|det(J_{f^{-1}})|$$



$$G^* = arg\max_G \sum_{i=1}^{m} log p_G(x^i)$$

$$p_G(x^i) = \pi(z^i)|det(J_{G^{-1}})|$$
$$z^i = G^{-1}(x^i)$$

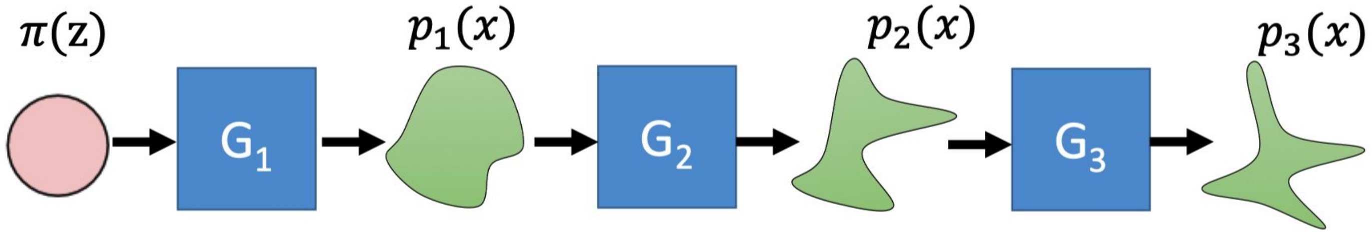
$\Rightarrow$ You can compute $det(J_G)$

$\Rightarrow$ You know G^{-1}

$\Uparrow$ G has limitation

$$log p_G(x^i) = log\pi\left(G^{-1}(x^i)\right) + log|det(J_{G^{-1}})|$$

Flow-based model



$$p_1(x^i) = \pi(z^i)\left(\left|det\left(J_{G_1^{-1}}\right)\right|\right)$$

$$z^i = G_1^{-1}\left(\cdots G_K^{-1}(x^i)\right)$$

$$p_2(x^i) = \pi(z^i)\left(\left|det\left(J_{G_1^{-1}}\right)\right|\right)\left(\left|det\left(J_{G_2^{-1}}\right)\right|\right)$$

$$\vdots$$

$$p_K(x^i) = \pi(z^i)\left(\left|det\left(J_{G_1^{-1}}\right)\right|\right)\cdots\left(\left|det\left(J_{G_K^{-1}}\right)\right|\right)$$

$$log p_K(x^i) = log\pi(z^i) + \sum_{h=1}^{K} log\left|det\left(J_{G_K^{-1}}\right)\right|$$ Maximize

Flow-based model

Max $$\log p_G(x^i) = \log\pi\left(G^{-1}(x^i)\right) + \log|det(J_{G^{-1}})|$$

Actually, we train G^{-1} , but we use G for generation.

- NICE
- RealNVP
- Glow

Unbiased estimate method:

- FFJORD
- Residual Flow

Normalizing Flow

Gaussian distribution is often used in latent variable generative models, even though most of real world distributions are much more complicated than Gaussian.

A normalizing flow **transforms a simple distribution into a complex one by applying a sequence of invertible transformation functions.**

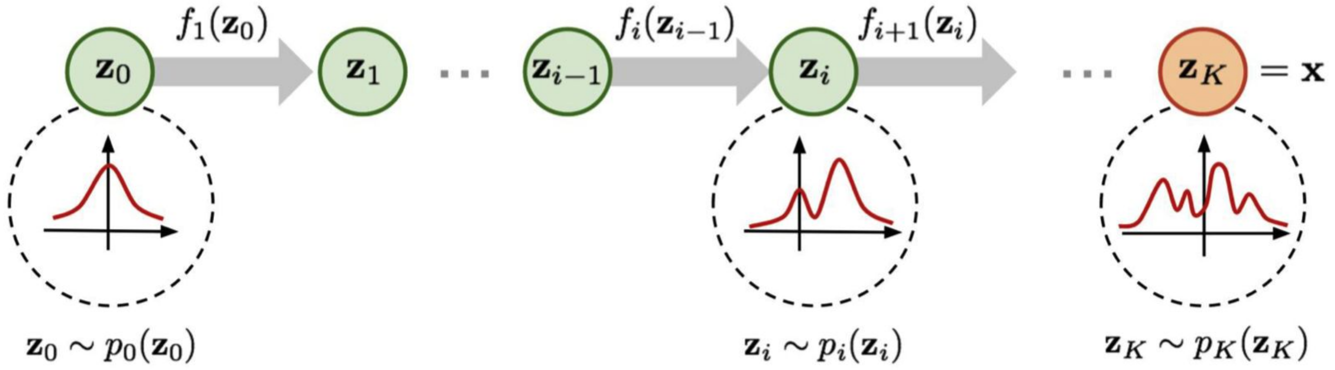
Flowing through a chain of transformations, we repeatedly substitute the variable for the new one according to the change of variables theorem and eventually obtain a probability distribution of the final target variable.

Reference: https://lilianweng.github.io/posts/2018-10-13-flow-models/

Normalizing Flow

Normalizing : the change of variables gives a normalized density after applying an invertible transformation.

Flow : the invertible transformations can be composed with each other to create more complex invertible transformations.



Normalizing Flow

$$\begin{aligned}
\mathbf{z}_{i-1} &\sim p_{i-1}(\mathbf{z}_{i-1}) \\
\mathbf{z}_i &= f_i(\mathbf{z}_{i-1}), \text{ thus } \mathbf{z}_{i-1} = f_i^{-1}(\mathbf{z}_i) \\
p_i(\mathbf{z}_i) &= p_{i-1}(f_i^{-1}(\mathbf{z}_i)) \left| \det \frac{df_i^{-1}}{d\mathbf{z}_i} \right| \\
&= p_{i-1}(\mathbf{z}_{i-1}) \left| \det \left(\frac{df_i}{d\mathbf{z}_{i-1}} \right)^{-1} \right| \\
&= p_{i-1}(\mathbf{z}_{i-1}) \left| \det \frac{df_i}{d\mathbf{z}_{i-1}} \right|^{-1} \\
\log p_i(\mathbf{z}_i) &= \log p_{i-1}(\mathbf{z}_{i-1}) - \log \left| \det \frac{df_i}{d\mathbf{z}_{i-1}} \right|
\end{aligned}$$

$$p(x')\left|det\left(J_f\right)\right| = \pi(z')$$

$$p(x') = \pi(z')\left|det\left(J_{f^{-1}}\right)\right|$$

$$\frac{df^{-1}(y)}{dy} = \frac{dx}{dy} = \left(\frac{dy}{dx}\right)^{-1} = \left(\frac{df(x)}{dx}\right)^{-1}$$

Normalizing Flow

$$\begin{aligned}
\mathbf{x} = \mathbf{z}_K &= f_K \circ f_{K-1} \circ \cdots \circ f_1(\mathbf{z}_0) \\
\log p(\mathbf{x}) = \log \pi_K(\mathbf{z}_K) &= \log \pi_{K-1}(\mathbf{z}_{K-1}) - \log \left| \det \frac{d f_K}{d \mathbf{z}_{K-1}} \right| \\
&= \log \pi_{K-2}(\mathbf{z}_{K-2}) - \log \left| \det \frac{d f_{K-1}}{d \mathbf{z}_{K-2}} \right| - \log \left| \det \frac{d f_K}{d \mathbf{z}_{K-1}} \right| \\
&= \dots \\
&= \log \pi_0(\mathbf{z}_0) - \sum_{i=1}^{K} \log \left| \det \frac{d f_i}{d \mathbf{z}_{i-1}} \right|
\end{aligned}$$

Example

$$\begin{tikzcd} x = (x_1, x_2) \arrow[d, "g_i"'] \\ x' = (x'_1, x'_2) \end{tikzcd}$$

$$x'_1 = e^{s(x_2)} x_1 + t(x_1)$$
$$x'_2 = x_2$$

invertible:

$$x_1 = e^{-s(x'_2)} x'_1 - t(x_1)$$
$$x_2 = x'_2$$

$$\frac{\partial g(x_1, x_2)}{\partial x} = \left(\begin{array}{ccc|ccc} e^{[s(x_2)]_1} & & & \cdot & \cdot & \cdot \\ & e^{[s(x_2)]_2} & & \cdot & \cdot & \cdot \\ & & \ddots & \cdot & \cdot & \cdot \\ \hline & & & 1 & & \\ & 0 & & & 1 & \\ & & & & & \ddots \end{array} \right)$$

$$J(x) = \left| \det_{kl} \frac{\partial [g(x_1, x_2)]_k}{\partial x_l} \right| = \prod_k e^{[s(x_2)]_k}$$

$$J_{\text{reverse}}(x') = 1/J(x') = \prod_k e^{[-s(x'_2)]_k}$$

Reference: https://arxiv.org/abs/2101.08176

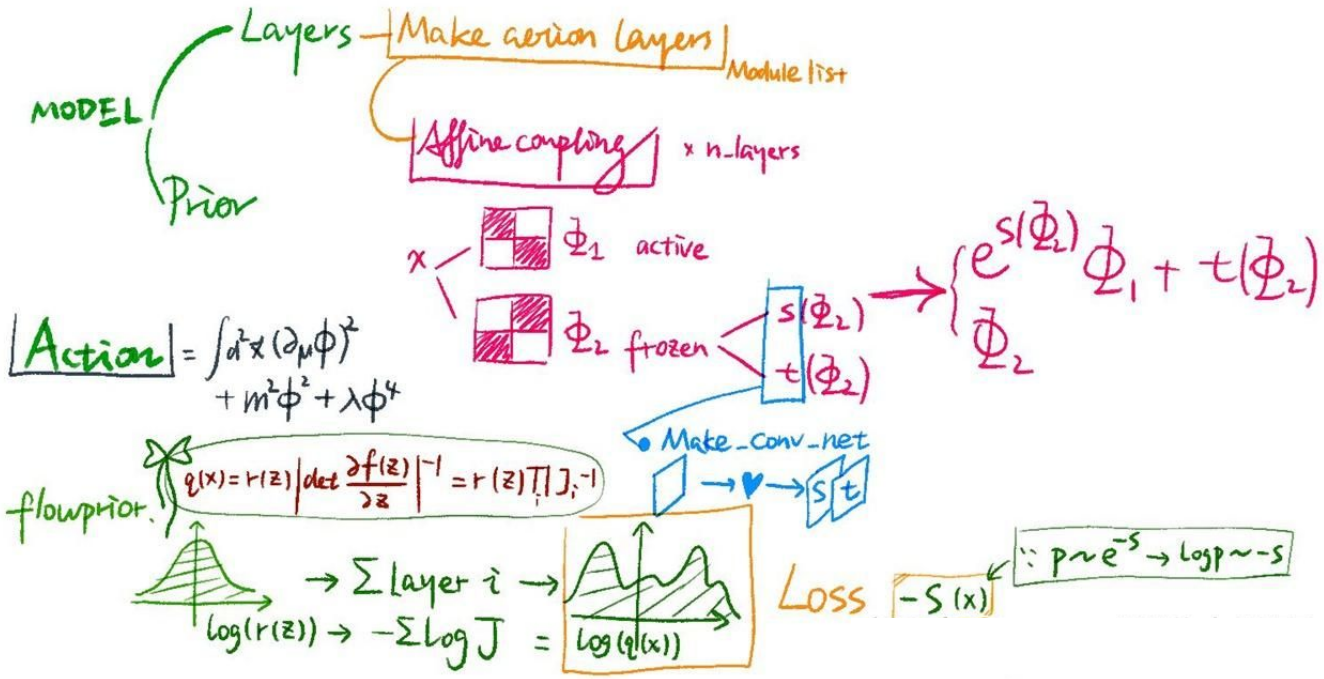
Example

Prior density $r(z)$

Model density $q(x)$

Target density $p(x)$

$$q(x) = r(z)[J(z)]^{-1} = r(z)\left|\det_{kl}\frac{\partial f_k(z)}{\partial z_l}\right|^{-1}$$


MODEL
Layers
Make action layers
Module list
Affine coupling
x n_layers
Prior
x
Φ_1 active
Φ_2 frozen
$s(\Phi_2)$
$t(\Phi_2)$
$e^{s(\Phi_2)}\Phi_1 + t(\Phi_2)$
Φ_2
Action $= \int d^2x\,(\partial_\mu\phi)^2 + m^2\phi^2 + \lambda\phi^4$
Make_conv_net
s t
flowprior
$q(x) = r(z)\left|\det\frac{\partial f(z)}{\partial z}\right|^{-1} = r(z)\prod_i |J_i|^{-1}$
→ Σ layer i →
log(r(z)) → −Σ log J = log(q(x))
Loss
−S(x)
∵ p ~ e^{−S} → log p ~ −S

Example[1]

$$\text{loss} = \mathbb{E}\log(q) - \log(p)$$

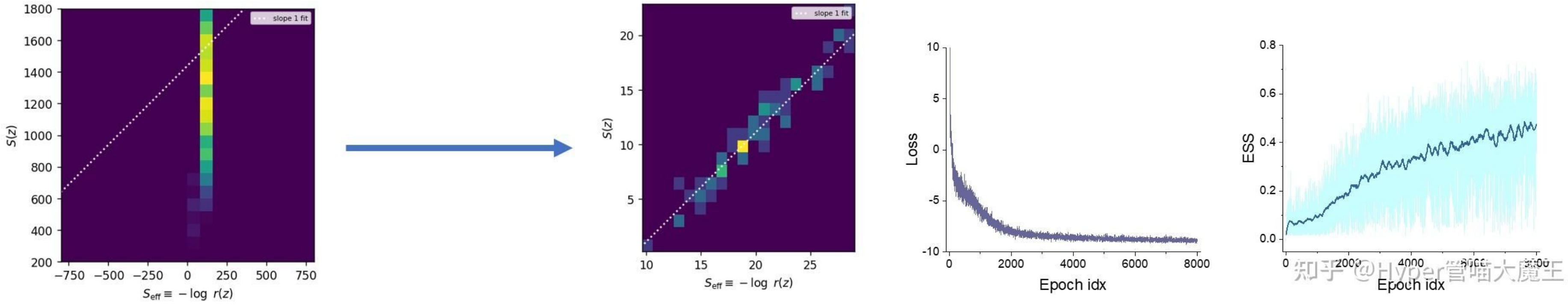
S could be any action, like energy

$$\log(p) = -S(x) = -energy(x(z))$$

$$\log(q) = \log(q(x)) = \log(r(z)) - \sum \log(J_{g_i})$$

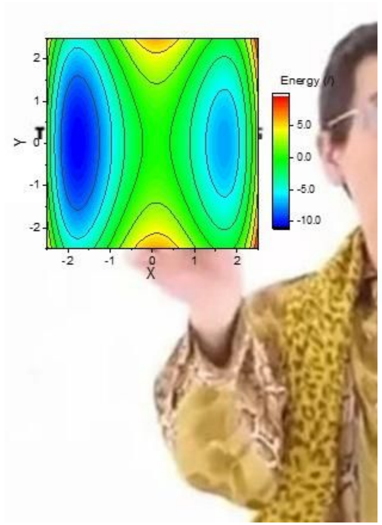
$$q(x) = r(z)[J(z)]^{-1} = r(z)\left|\det_{kl} \frac{\partial f_k(z)}{\partial z_l}\right|^{-1}$$

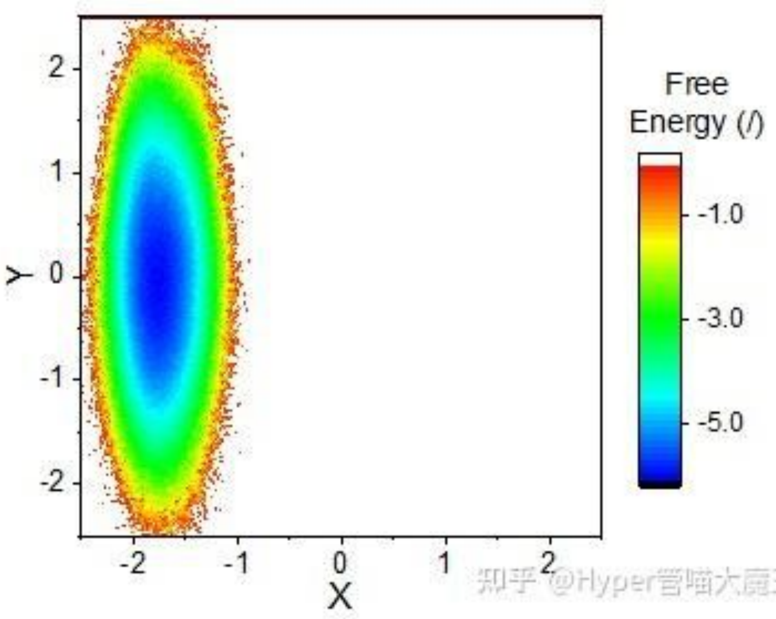
$$\text{loss} = \mathbb{E}_z\, energy(x(z)) - \sum \log(J_{g_i})$$

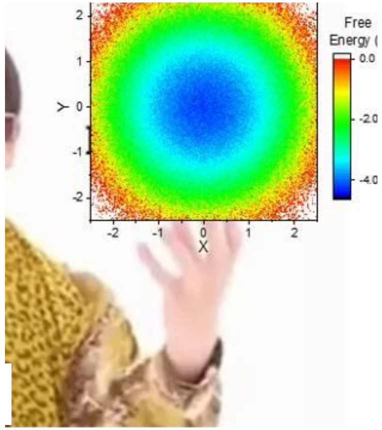


[1] Introduction to Normalizing Flows for Lattice Field Theory, https://arxiv.org/abs/2101.08176

Problem in training







aussian Prior

How to solve the problem?

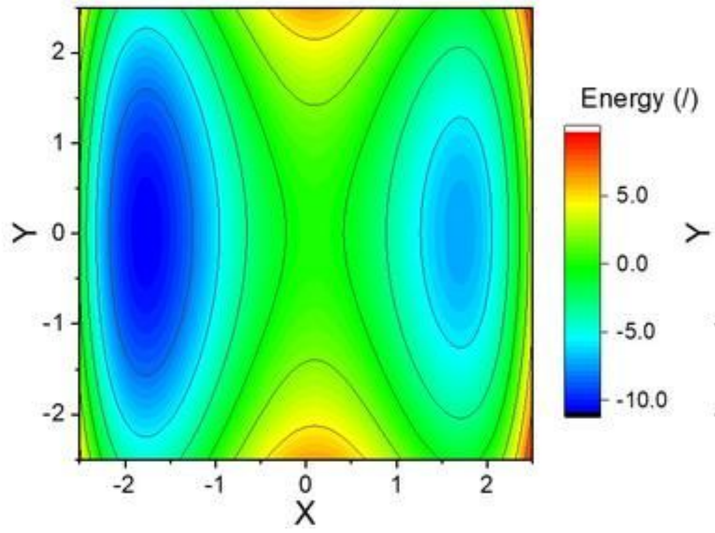
Reweight

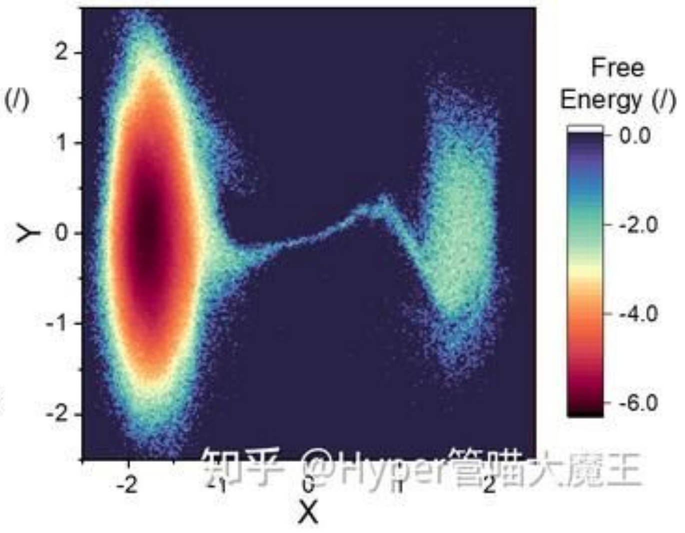
$$\text{loss} = \mathbb{E}_z \boxed{energy(x(z))} - \boxed{\sum \log(J_{g_i})}$$

potential energy

entropic contribution to the free energy

$$\text{loss} = \mathbb{E}_z\, energy(x(z)) - 5 * \sum \log(J_{g_i})$$





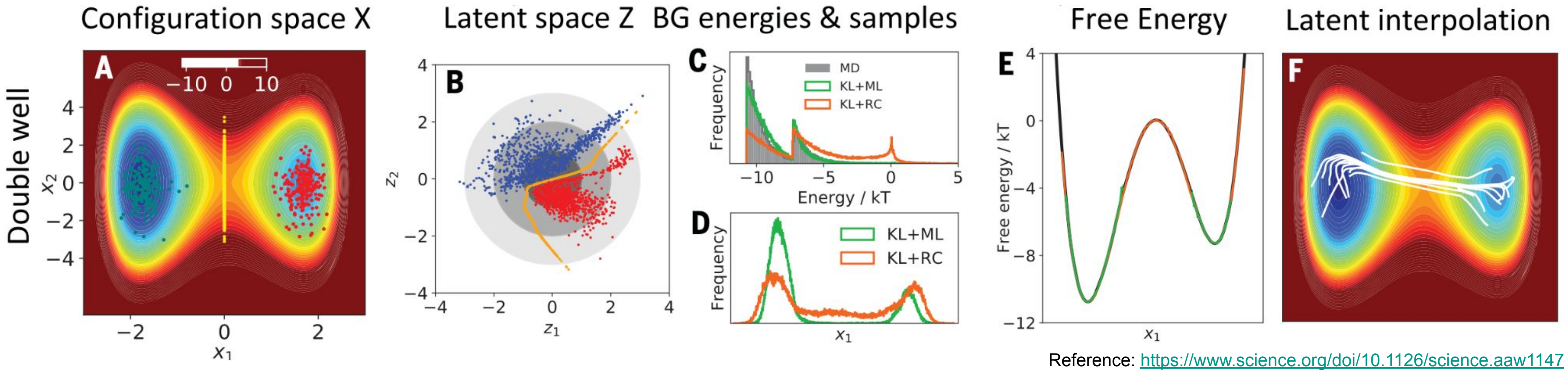
Training

Train by energy

KL loss: $J_{KL} = \mathbb{E}_{\mathbf{z}}\left[u\left(F_{zx}\left(\mathbf{z}\right)\right) - \log R_{zx}\left(\mathbf{z}\right)\right]$

free-energy difference of transforming the Gaussian prior distribution to the generated distribution

! training by energy alone tends to focus sampling on the most stable metastable state



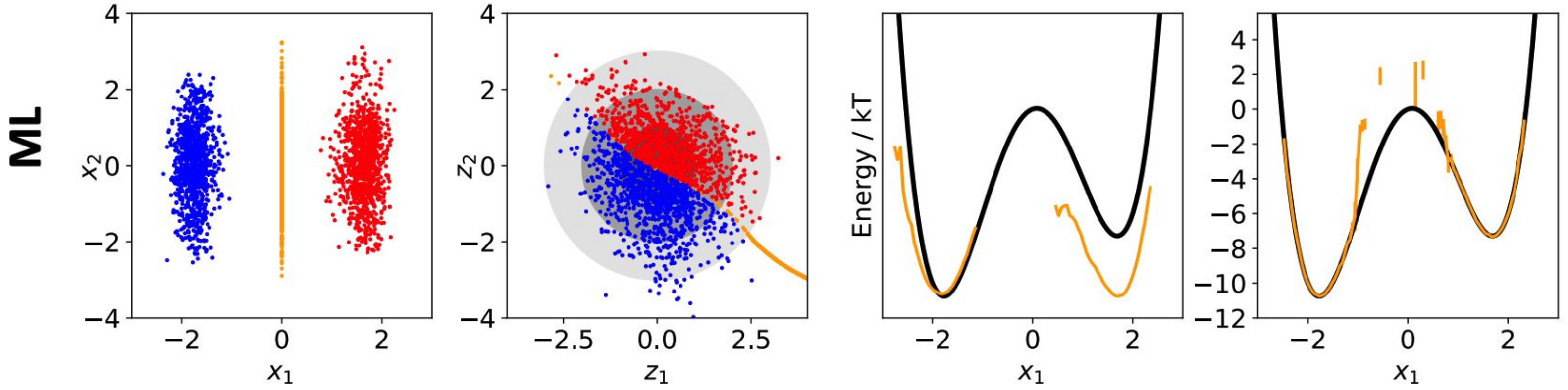
Reference: https://www.science.org/doi/10.1126/science.aaw1147

Training

Train by example

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

Use the standard training method used in other machine learning applications, implemented with the maximum likelihood (ML) principle.

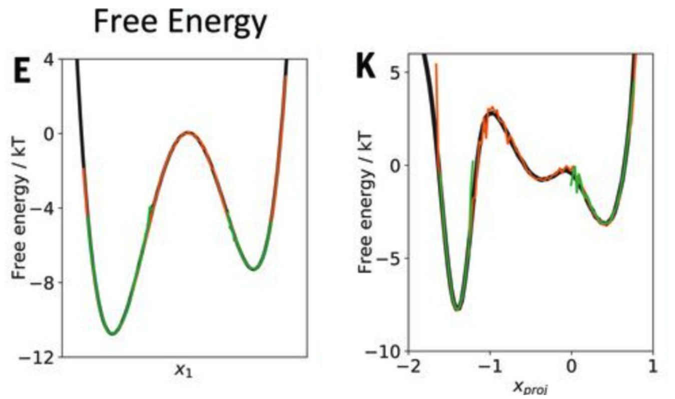


Training

RC loss: $J_{RC} = \int p(r(\mathbf{x})) \log p(r(\mathbf{x})) \mathrm{d}r(\mathbf{x}) = \mathbb{E}_{\mathbf{x}\sim q_X(\mathbf{x})} \log p(r(\mathbf{x}))$

The transformed distribution cover as much as possible the whole of the reaction coordinates defined by us.

We do not want to sample from the Boltzmann distribution but promote the sampling of high-energy states in a specific direction of configuration space, for example, to compute a free-energy profile along a predefined RC.



Training in Boltzmann Generator

$$\text{Prior} \quad \mu_Z(\mathbf{z}) \xrightarrow{F_{ZX}} p_X(\mathbf{x}) \quad \text{generated} \quad \underset{\text{match}}{\longleftrightarrow} \quad \mu_X(\mathbf{x}) = Z_X^{-1} \mathrm{e}^{-u(\mathbf{x})}$$

- Loss function:

$$J = \underbrace{w_{ML} J_{ML}}_{\text{max likelihood}} + \underbrace{w_{KL} J_{KL}}_{\text{Kullback-Leibler}} + \underbrace{w_{RC} J_{RC}}_{\text{reaction coordinate}}.$$

- KL divergence between generated $p_X(\mathbf{x})$ and Boltzmann distribution:

$$\begin{aligned} \mathrm{KL}(p_X \,\|\, \mu_X) &= \mathbb{E}_{\mathbf{x} \sim p_X(\mathbf{x})} [\log p_X(\mathbf{x}) - \log \mu_X(\mathbf{x})] \\ &= \mathbb{E}_{\mathbf{z} \sim \mu_Z(\mathbf{z})} [u_X(F_{ZX}(\mathbf{z})) - \Delta S_{ZX}(\mathbf{z})] + \text{const} \end{aligned}$$

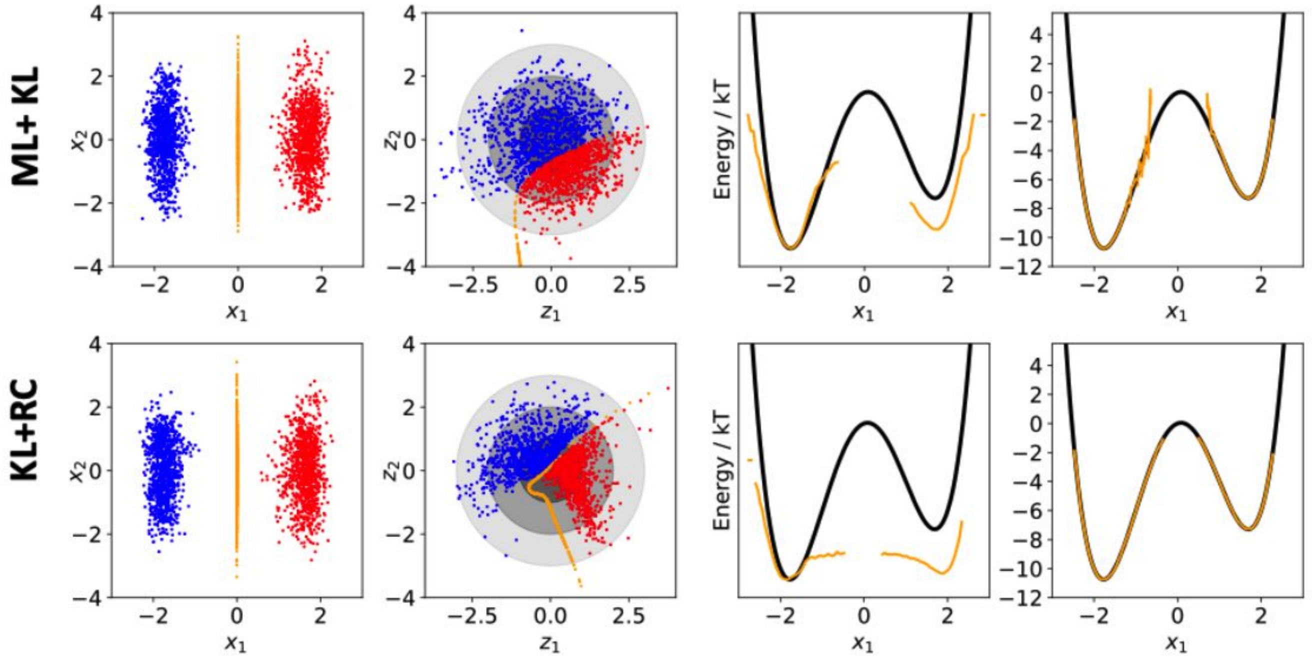
$$\Delta S_t = \log \left|\det \mathbf{J}_t(\mathbf{z})\right|. \qquad \text{log determinant / entropy change}$$

- **Reweighting / Importance sampling**:

$$w_X(\mathbf{x}) = \frac{\mu_X(\mathbf{x})}{p_X(\mathbf{x})} \propto \mathrm{e}^{-u_X(F_{ZX}(\mathbf{z})) + u_Z(\mathbf{z}) + \Delta S_{ZX}(\mathbf{z})}$$

Reference: https://www.youtube.com/watch?v=WuXJRswYIaA

Training in Boltzmann Generator



Outline - Results

- illustration on model systems
- Exploring configuration space
- Complex Molecules

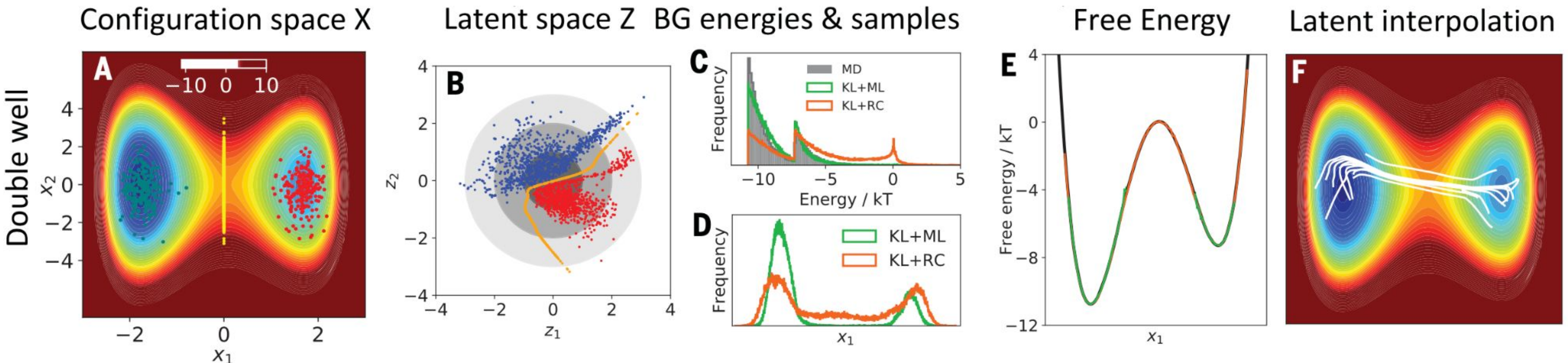
Results: illustration on model systems

A: double-well potential, featuring two minima separated by a high barrier

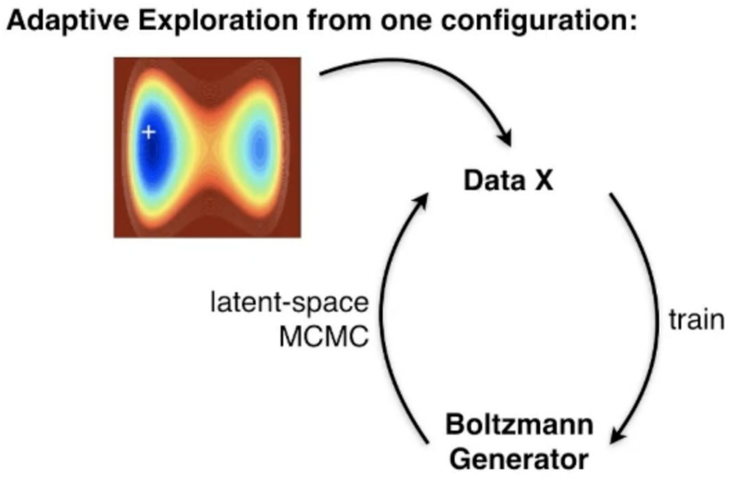
B: Latent- space distribution of trajectories

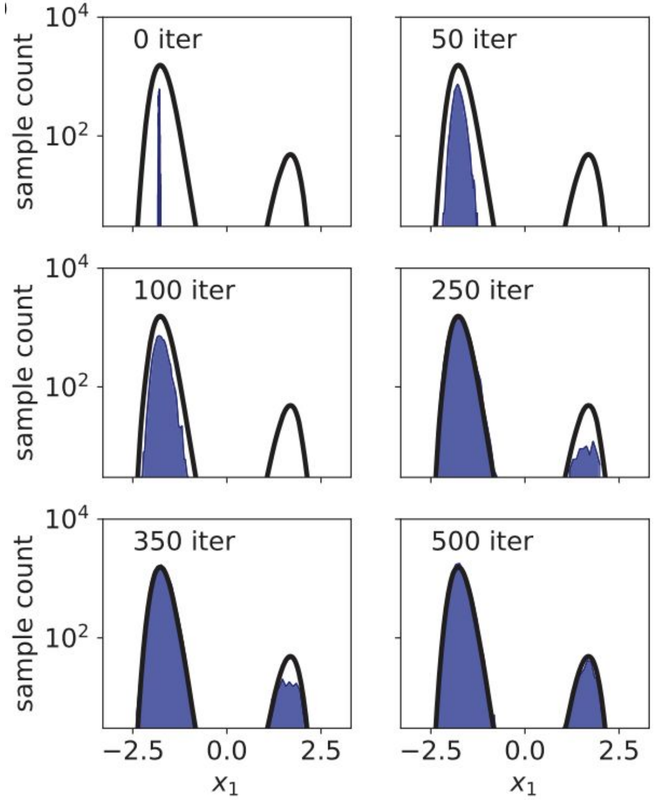
E: Free-energy estimates obtained from Boltzmann generator samples after reweighting

F: Paths generated by linear interpolation in Boltzmann generator latent space (B and H) between random pairs of “blue” and “red” MD samples



Results: Exploring configuration space



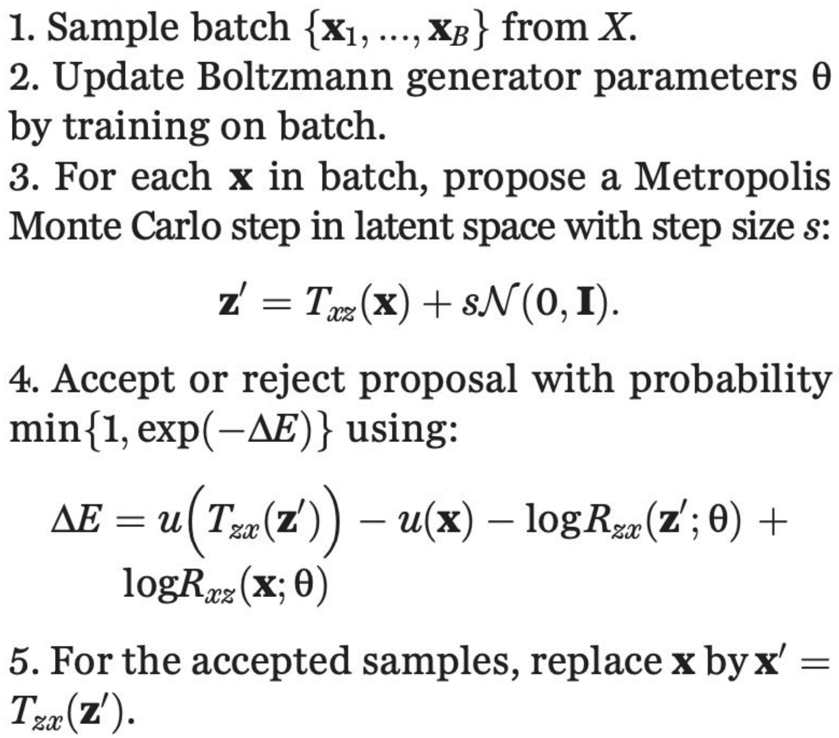


Evolution of sample distribution over MCMC iteration. As soon as sufficient density is available in the states of interest, these distributions can be reweighted to equilibrium

Reference: https://www.youtube.com/watch?v=WuXJRswYIaA&t=1169s

Adaptive sampling and training

Goal: train a Boltzmann generator while simultaneously using it to propose new samples



1. Sample batch $\{\mathbf{x}_1, ..., \mathbf{x}_B\}$ from X.
2. Update Boltzmann generator parameters θ by training on batch.
3. For each $\mathbf{x}$ in batch, propose a Metropolis Monte Carlo step in latent space with step size s:

$$\mathbf{z}' = T_{xz}(\mathbf{x}) + s\mathcal{N}(0, \mathbf{I}).$$

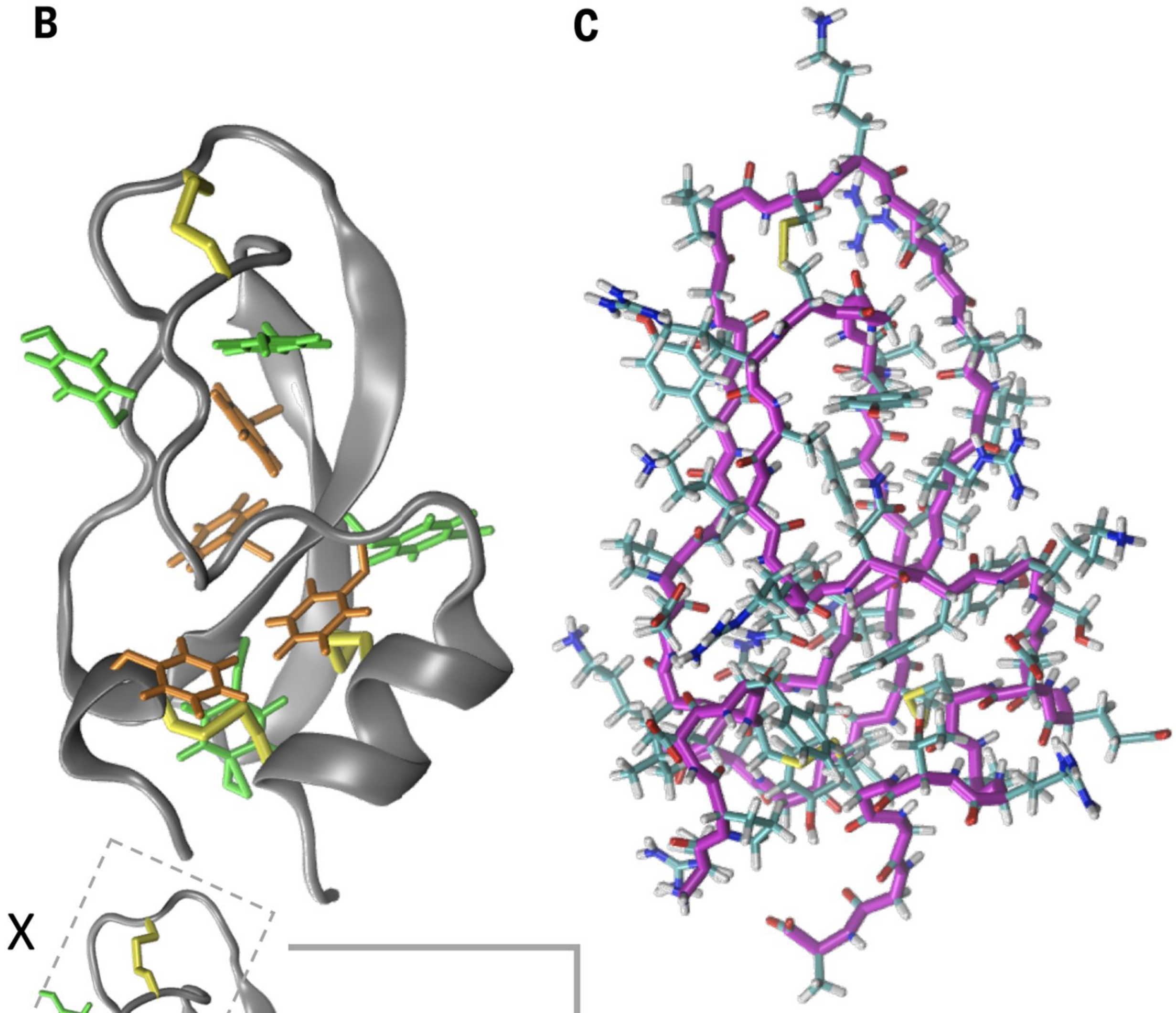
4. Accept or reject proposal with probability $\min\{1, \exp(-\Delta E)\}$ using:

$$\Delta E = u\Big(T_{zx}(\mathbf{z}')\Big) - u(\mathbf{x}) - \log R_{zx}(\mathbf{z}'; \theta) + \log R_{xz}(\mathbf{x}; \theta)$$

5. For the accepted samples, replace $\mathbf{x}$ by $\mathbf{x}' = T_{zx}(\mathbf{z}')$.

Results:

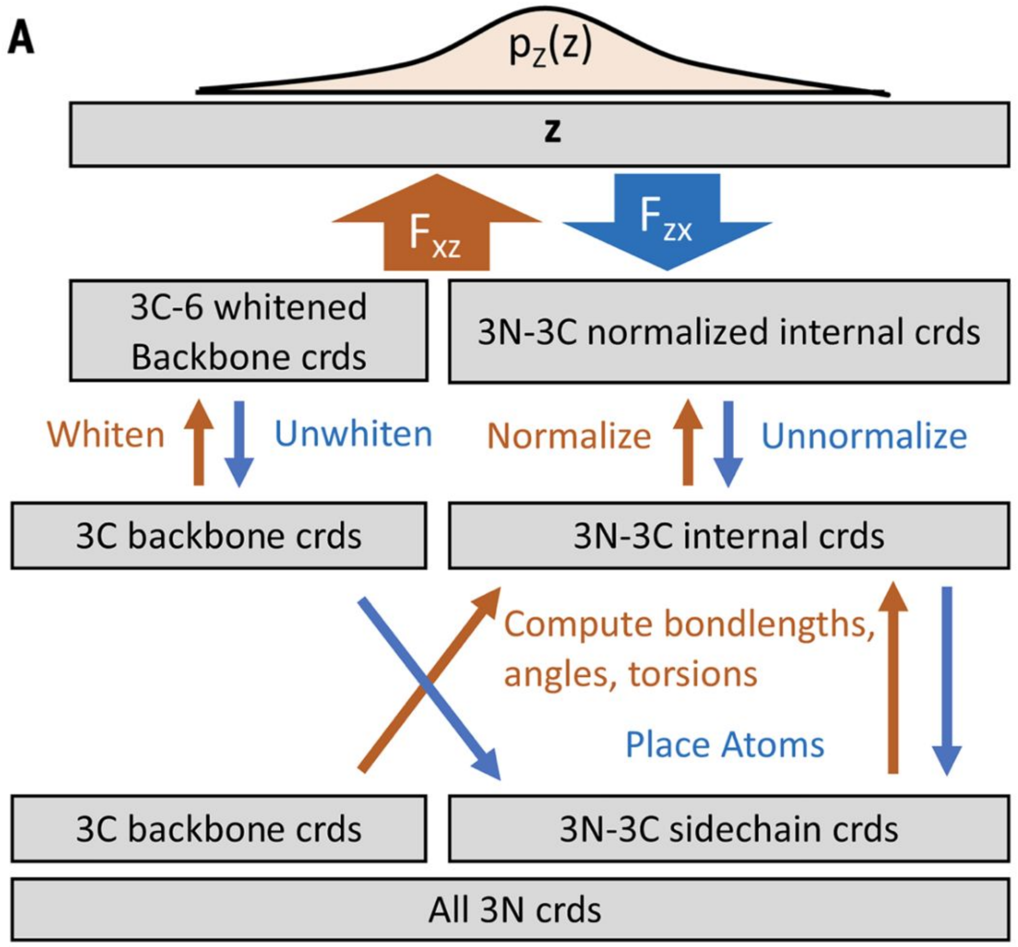
Complex Molecules



Results:

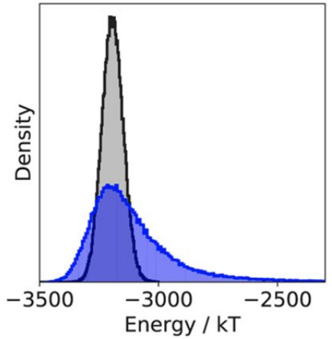
Complex Molecules

A

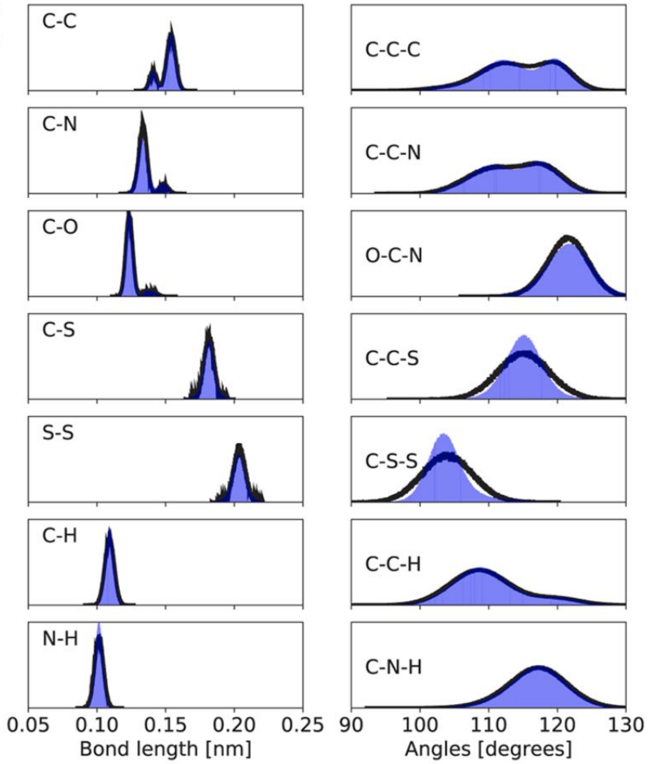


Results:
Complex Molecules

D

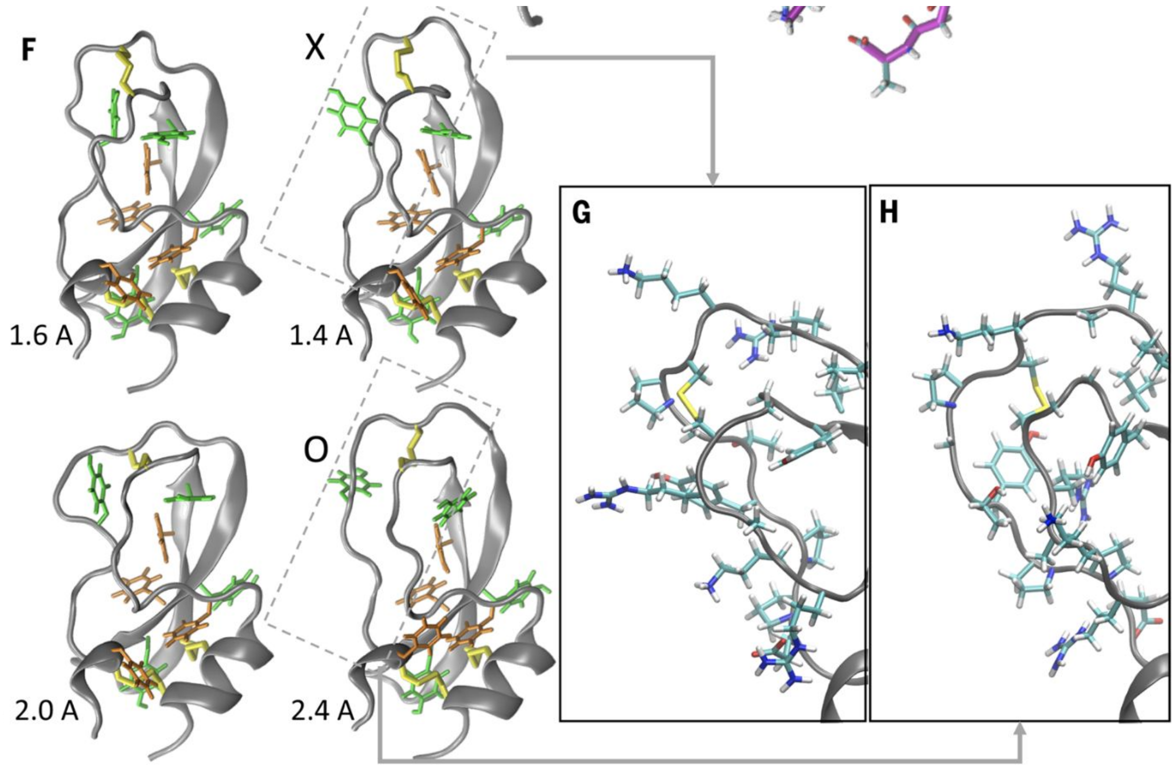


E



Results:

Complex Molecules



Other results

- Thermodynamics of condensed-matter systems
- Thermodynamics between disconnected states

Limitations

- The Boltzmann generator may not sample exactly from the Boltzmann distribution, leading to a slightly different output distribution.
- The larger the system, the more differences there will be between the two distributions, making it harder to match them.
- The acceptance rate of the reweighting step decreases as the system size increases
- The approach can be inefficient for very large systems as the acceptance of samples may eventually stop
- Breaking down a large system into subsystems and applying Boltzmann generators to these subsystems may be necessary, which adds complexity to the process.
- The ideal approach is to exploit the sweet spot between MD and Boltzmann generators, which may not always be feasible or straightforward to identify.

Reference: https://www.youtube.com/watch?v=WuXJRswYIaA&t=1169s

References

- Frank Noe’s talk on MLDS2020
- CS 279 Computational Biology: Structure and Organization of Biomolecules and Cells Fall 2023 Lecture 4
- Glow: Generative Flow with Invertible 1x1 Convolutions
- https://deepgenerativemodels.github.io/notes/
- https://arxiv.org/abs/2101.08176
- https://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2019/Lecture/FLOW%20(v7).pdf
- https://arxiv.org/abs/1808.04730
- https://lilianweng.github.io/posts/2018-10-13-flow-models/
- https://arxiv.org/abs/1707.04585