Normalizing Flows

Density Estimation and generate model using neural networks learning coordinate transformation from known base distribution to unknown data distribution (only have samples).

Two ways:

- Parametrizing transformation directly using neural network

$$T(\vec{x}) = NN(x;\theta)$$

  - neural network learning transformation directly
  - Planar Flow (x + tanh x)
  - Sylvester flow (NN based)
  - Masked Autoregressive flows
- Parametrizing tangent of transformation trajectory

$$T(\vec{x}) = \int_0^1 dt NN(x;\theta), i.e., \frac{d\vec{x}}{dt} = NN(x;\theta)$$

  - neural networks learning infinitesimal transformations.
  - continuous normalizing flows

Continuous Normalizing Flows

Within the general class of normalizing flows, we have to choose an optimal implementation for smoothly upsampling star particles. Continuous normalizing flows \citep[CNF;][]{NEURIPS2018_69386f6b,grathwohl2019ffjord} are a good candidate with an inductive bias suitable for this problem because the transformation smoothly deforms the base distribution to the target distribution.

More specifically, CNF learns the following infinitesimal transformation,

$$\begin{aligned} g_t &: \vec{y} \to \vec{y} + F(\vec{y}, t) \cdot dt, \\ g_t^{-1} &: \vec{y} \to \vec{y} - F(\vec{y}, t) \cdot dt. \end{aligned}$$

Here, the function F is a neural network representing the derivative $d\vec{y}/dt$ of the trajectory of transformed variables at a latent time t. The full chain of transformations is the integral of this infinitesimal transformation, and it is described by a neural ordinary differential equation \citep[neural ODE;][]{NEURIPS2018_69386f6b},

$$\frac{d}{dt}\vec{y}(t) = F(\vec{y}(t), t).$$

Note that if dt is finite, the transformation g_t is essentially a residual block at a given time, $\vec{y} \to \vec{y} + \mathcal{F}(\vec{y})$, where $\mathcal{F}$ is the difference between the inputs and outputs of the transformation. Therefore, the neural ODE is considered as a generalization of residual networks for normalizing flows \citep{haber2018learning,pmlr-v80-lu18d,Haber_2018,ruthotto2020deep}, and the parameter t takes the role of the flow index in the chain.

The Jacobian determinant of the transformation can be obtained by solving the following form of the Fokker-Planck equation with zero diffusion \citep{NEURIPS2018_69386f6b}, describing the time evolution of log probability $\log p(\vec{y}(t); t)$ along the trajectory $\vec{y}(t)$ at time t:

$$\frac{d}{dt}\log p(\vec{y}(t); t) = -\mathrm{Tr}\left[\frac{\partial F}{\partial \vec{y}(t)}\right].$$

The trace computation of this equation is often a bottleneck during the training, so Hutchinson's trace estimator

\citep{grathwohl2019ffjord} can be used to speed up the training. In our case, the cost of evaluating the trace is manageable since we train CNFs for 3D densities; we explicitly evaluate the trace during the training.

solving equation of motion

Mean square error minimization

$$\mathcal{L} = \sum_{i=1}^{n} |\mathrm{EOM}|^2 = \sum_{i=1}^{n} \left| \vec{v} \cdot \frac{df}{d\vec{x}} + \vec{a} \cdot \frac{df}{d\vec{v}} \right|^2$$

$$\frac{d\mathcal{L}}{da_i} = \sum_{n=1}^{N} \left(\vec{v}^{(n)} \cdot \frac{df}{d\vec{x}^{(n)}} + \vec{a} \cdot \frac{df}{d\vec{v}^{(n)}} \right) \frac{df}{dv_i^{(n)}} = 0$$

Solve linear system

$$M_{ij} a_j + B_i = 0$$

where

$$M_{ij} = \sum_{n=1}^{N} \frac{df}{dv_i^{(n)}} \frac{df}{dv_j^{(n)}}$$

$$B_i = \sum_{n=1}^{N} \vec{v}^{(n)} \cdot \frac{df}{d\vec{x}^{(n)}} \frac{df}{dv_i^{(n)}}$$