Generative Models

ML models that generates data similar to the training dataset.

Neural network transforming random noise (usually Gaussian) to samples from the unknown data distribution $p(x)$.

- Generative Adversarial Neural Networks (GAN)
- Variational Autoencoders (VAE)
- Normalizing Flows
- Diffusion Models

In particular, noramlzing flows and diffusion models are interesting since they're also giving us estimated probablity function $p(x)$. GAN and VAE is not estimating the density but only generates samples.

Physics Problem: measuring stellar phase space density and DM density of Milky Way

by measuring star's trajectory, we could measure gravitational acceleration field in the Milky way,

But we can measure MW in only limited time, so the trajectory information is limited.

Instead, we may consider the stars as hydrodynamic system, and solve equation of motion to infer the gravitational field.

each stars is I.I.D, so we may write the phase space density as 6D probability function of position and velocity $f(\vec{x}, \vec{v})$

$$\left[\frac{\partial}{\partial t} + \vec{v} \cdot \frac{\partial}{\partial \vec{x}} + \vec{a} \cdot \frac{\partial}{\partial \vec{v}}\right] f(\vec{x}, \vec{v}) = 0$$

If Milky way is in approximate kinematic equilibrium, $\partial f / \partial t = 0$, and the above equation can be solved for $\vec{a}$ if we estimate $f(x, v)$ from the data.

Density estimation

Statement of problem: given data $\{\vec{x}^{(1)}, \cdots, \vec{x}^{(N)}\}$ from unknown probablity denstiy $p(x)$, estimate $p(\vec{x})$ only using the data.

Histograms

- cons
  - binned. (result is boxy)
  - doesn't scale well with data dimension

Kernel Density Estimation

kernel density estimation [https://en.wikipedia.org/wiki/kernel density estimation]

- cons

- expensive
- estimating smeared density

Alternatives?

Parametric Density Estimation

Predefine parametric family of densities interested (more likely)

- Gaussian $N(\mu, \sigma^2)$

If this family of densities cover the true density, we could estimate the parameter of family using MLE and use the selected model as a smooth density estimation result.

Loss function: again, negative log likelihood (Maximum Likelihood estimation of parameters)

$$\mathcal{L} = -\int dx p(x) \log q(x;\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log q(x^{(i)};\theta)$$

Proof:

Lagrange multiplier method with constraint $g(x) = \int dx\, q(x;\theta) - 1 = 0$: integral of probablity density = 1

$$\mathrm{LM}(q,\lambda) = -\int dx p(x) \log q(x;\theta) + \lambda \left[\int dx\, q(x;\theta) - 1\right]$$

$$\frac{d\mathrm{LM}(q,\lambda)}{dq} = -\left[\frac{p(x)}{q(x)} - \lambda\right], \quad \frac{d\mathrm{LM}(q,\lambda)}{d\lambda} = \int dx\, q(x;\theta) - 1$$

First equation yields the constraint between λ and $q(x)$

$$\lambda = \frac{p(x)}{q(x)}$$

So plugging this to second equation yields

$$\frac{1}{\lambda}\int dx p(x) - 1 = \frac{1}{\lambda} - 1 \rightarrow \lambda = 1$$

This implies the stationary point of $\mathcal{L}$ is at

$$\frac{p(x)}{q(x;\theta)} - 1 = 0$$

i.e.,

$$q(x;\theta) = p(x)$$

Example: 1D Gaussian

let $\{x^{(i)}\}$ sample drawn from Gaussian $N(\mu_{true}, \sigma^2_{true})$

$$-\log q(x;\mu,\sigma^2) = \frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i-\mu)^2 + \frac{1}{2}\log 2\pi\sigma^2$$

$$\mathcal{L}(\mu,\sigma^2) = \frac{N}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i-\mu)^2$$

The maximum likelihood estimator

$$\frac{d}{d\mu}\mathcal{L} = 0 \quad \to \quad \hat{\mu} = \frac{1}{N}\sum_{i=1}^{N}x_i$$

$$\frac{d}{d\sigma^2}\mathcal{L} = 0 \quad \to \quad \hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i-\hat{\mu})^2$$

Gives us mean and variance of the data, estimating the Gaussian distribution correctly. (up to bias factor)

Normalizing Flows

Reviews:

- 1908.09257 [https://arxiv.org/abs/1908.09257]
- 1912.02762 [https://arxiv.org/abs/1912.02762]

Neural network learning coordinate transformations between random variables

$$T: Z \to Y$$

$$\vec{y} = T(\vec{x}), \quad p(y) = p(x)\left|\frac{\partial\vec{x}}{\partial\vec{y}}\right|$$

This is coordinate transform, so it requires

- T must be bijective

But if T is very complicated, the Jacobian determinant is complicated to calculate

- we use relatively simple transformation T_i whose Jacobian is easy to evaluate.
- after then we stack up transformations in order to model highly expressive coordinate transformation.
    - $T = T_n \circ \cdots \circ T_1$

Examples:

- affine flow: $T: x \to Wx + B$
- planar flow: $T: x \to x + U\,\tanh(Wx+B)$
- sylvester flow: $T: x \to x + NN(x;\theta)$ (resnet)

Masked Autoregressive Flows

ref: 1705.07057 [https://arxiv.org/abs/1705.07057]

In the case of modeling multivariate distributions, constructing simple multivariate bijections is not itself straightforward. To overcome this, MAFs use autoregressive modeling of probability density, which utilizes the chain rule of probability in order to model a simple multivariate bijection as a product of simple conditioned univariate bijections. The chain rule says that a joint probability $p(\vec{u})$ can be written in terms of conditionals $p(u_k|u_{1:k-1})$,

$$p(\vec{u}) = p(u_1) \times p(u_2|u_1) \times \cdots \times p(u_n|u_{1:n-1}),$$

where $u_{1:k}$ denotes a tuple $(u_1, \cdots, u_k)$ which is a subset of the elements of $\vec{u}$. We only need to model conditioned univariate bijections for each $p(u_k|u_{1:k-1})$ in order to construct a simple multivariate bijection.

In particular, we use affine MAFs whose conditional probability $p(u_k|u_{1:k-1})$ is Gaussian if the base distribution is the standard normal distribution. The corresponding transformation from $\vec{h}$ to $\vec{u}$ is defined as follows.

$$h_k \to u_k = h_k\,\sigma(u_{1:k-1}) + \mu(u_{1:k-1}),$$

where $\mu(u_{1:k-1})$ and $\sigma(u_{1:k-1})$ are the mean and standard deviation parameters of $p(u_k|u_{1:k-1})$, respectively.

- Advantage: Jacobian matrix is triangular, so we can simply estimate the Jacobian determinant as the product of diagonals.

$$|\frac{d\vec{u}}{d\vec{h}}| = \prod_{i=1}^{N} \sigma(u_{1:k-1})$$

$$|\frac{d\vec{h}}{d\vec{u}}| = \prod_{i=1}^{N} 1/\sigma(u_{1:k-1})$$

- inverse transformation is also easy to evaluate.: $h_k = \frac{u_k - \mu(u_{1:k-1})}{\sigma(u_{1:k-1})}$
- so the density can be evaluated very fast!

But this algorithm is slow for sample generation evaluating $\vec{u} = T(\vec{h})$ in the following way.

- $u_1 = h_1\,\sigma + \mu$
- $u_2 = h_2\,\sigma(u_1) + \mu(u_1)$
- $u_3 = h_3\,\sigma(u_{1:2}) + \mu(u_{1:2})$
- $\cdots$
- $u_k = h_k\,\sigma(u_{1:k-1}) + \mu(u_{1:k-1})$

The parameter functions $\mu(u_{1:k-1})$ and $\sigma(u_{1:k-1})$ are modeled by a multilayer perceptron with masking, following the Masked Autoencoder for Density Estimation (MADE) 1502.03509 [https://arxiv.org/abs/1502.03509] approach. The masking turns off connections to k-th output from i-th inputs with $i > k$, so that we may use this MADE block to build a model for the parameter functions of accumulated tuples $u_{1:k-1}$ as follows.

$$\mathrm{MADE}(\vec{u}) = \begin{pmatrix} \mu & \log\sigma \\ \mu(u_1) & \log\sigma(u_1) \\ \mu(u_{1:2}) & \log\sigma(u_{1:2}) \\ \vdots & \vdots \\ \mu(u_{1:n-1}) & \log\sigma(u_{1:n-1}) \end{pmatrix}$$

We use $\log\sigma(u_{1:n-1})$ here to make $\sigma(u_{1:k-1})$ positive definite.

Show video

Continuous Normalizing Flows

next lecture…

- 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