MACHINE LEARNING IN PHYSICS
FLOW AND DIFFUSION MODELS

HARRISON B. PROSPER

PHY6938

Introduction

1. Convolutional neural networks (CNN)
2. Autoencoder (AE)
3. Physics-informed neural networks (PINN)
4. Flow and diffusion models
5. Graph neural networks (GNN)
6. Transformer neural networks (TNN)

Introduction

- Probability density estimation (pdf) and sampling from a pdf are important tasks in many fields.

- Earlier we saw how a binary classifier, $C(x)$, can be used to approximate an unknown density $p(x|1)$ given a known density $p(x|0)$ using

$$p(x|1) = p(x|0)\left[\frac{C(x)}{1 - C(x)}\right]$$

- Today, we introduce two other methods normalizing flows and diffusion models for approximating and/or sampling from a D-dimensional density, $p(x)$.

NORMALIZING FLOWS

Change of Variables

Consider two probability densities $p(x)$ and $q(y)$ that are related through the function $y = f(x)$.

Suppose that $q(y)$ is known and our goal is to model $p(x)$.

Since $p(x)$ and $q(y)$ are probability densities, then necessarily,

$$p(x)dx = q(y)dy$$

and, therefore,

$$p(x) = q(f(x)) \left| \frac{df}{dx} \right|$$

Change of Variables

Generally, $y = f(x)$ is *vector*-valued with $x, y \in \mathbb{R}^D$,

In this case, the change of variables formula generalizes to

$$p(x) = q(y)\,|\det J|$$

or

$$\log p(x) = \log q(f(x)) + \log|\det J|$$

where $J = \nabla_x f(x)$ is the Jacobian matrix of the transformation.

Change of Variables: Example

Given

$$y = f(x) = \begin{pmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{pmatrix}$$

and

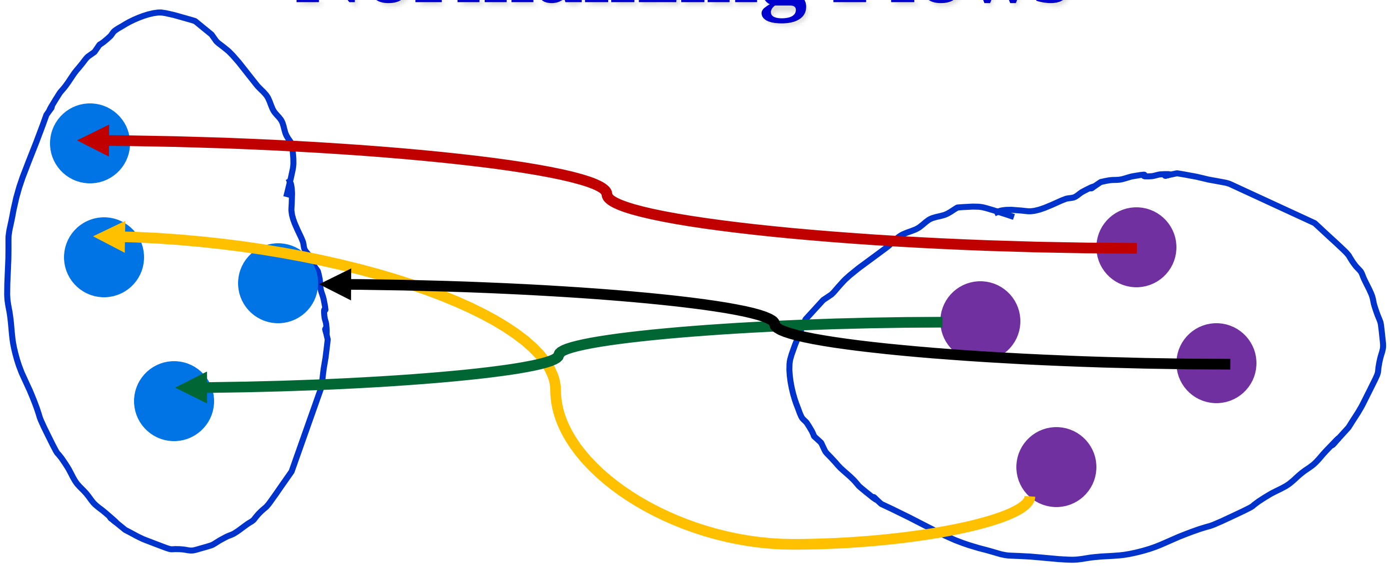
$$\nabla_x = \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2} \right)$$

the Jacobian matrix is

$$\nabla_x f = \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2} \right) \begin{pmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{pmatrix}$$

Normalizing Flows



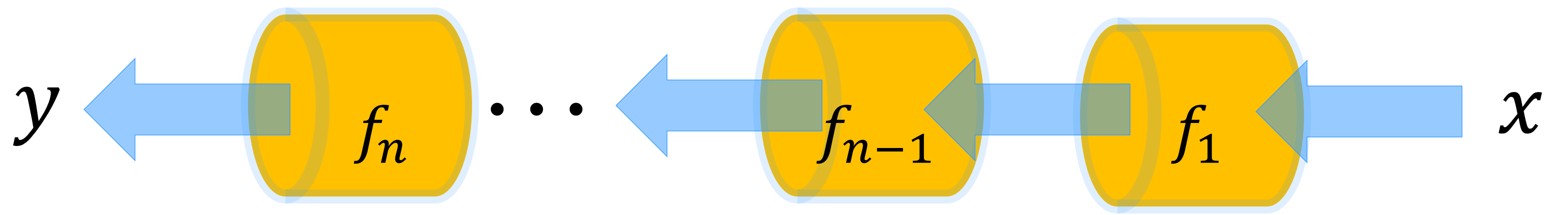
codomain $y = f(x)$ domain

A normalizing flow is a way to approximate the function $f(x)$ by modeling the latter as a conjunction of bijections f_i.

$$f(x) = f_n \circ f_{n-1} \circ \cdots f_1$$
$$= f_n(f_{n-1}(\ldots f_1(x)) \ldots)$$

(A bijection is a one-to-one invertible function.)

Normalizing Flows



Since the functions f_i are bijections, the inverse function $x = f^{-1}(y)$ exists.

Therefore, we can both model $p(x) = q(y)\,|\det J|$ (provided that the Jacobian is tractable) as well as sample $x \sim p(x)$ by first sampling $y \sim q(y)$ and then using $x = f^{-1}(y)$.

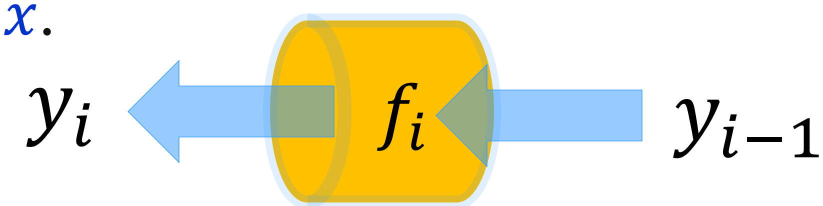
Normalizing Flows

The Jacobian $|\det J|$ of $f(x) = f_n \circ f_{n-1} \cdots f_1$ is simply the product of the Jacobians of each of the functions f_i.

Therefore,

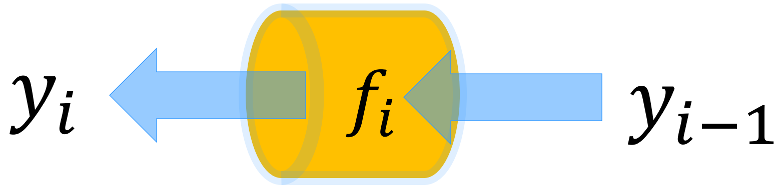
$$\log|\det \nabla_x f| = \sum_i \log|\det \nabla_x f_i|$$

The key to approximating $f(x)$ is computing the individual log Jacobians $\log|\det \nabla_x f_i|$ of the bijection $y_i = f_i(y_{i-1})$ where $y_0 \equiv x$.



Normalizing Flows: Bijection

To simplify the calulation of the Jacobian for $y_i = f_i(y_{i-1})$,



Dinh et al.[1] suggest splitting the function values y_i as follows

$$y_i = (y_A, y_B),$$

where y_A is of dimension d and y_B is of dimension $D - d$ with the inputs y_{i-1} split in a similar manner

$$y_{i-1} = (x_A, x_B).$$

1. Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio, *Density Estimation using Real NVP*, https://arxiv.org/abs/1605.08803

Normalizing Flows: Bijection

Dinh et al.[1] suggest the following form for the bijections f_i

$$\begin{pmatrix} y_A \\ y_B \end{pmatrix} = f_i(x_A, x_B) = \begin{pmatrix} x_A \\ x_B \exp(s(x_A)) + t(x_A) \end{pmatrix},$$

where $s(x_A)$ and $t(x_A)$ are $(D-d)$-dimensional functions with $\exp$ applied *elementwise* to its vector argument.

This form yields a Jacobian that is easy to compute. The function $f_i(x_A, x_B)$ is called non-volume preserving (NVP).

1. Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio, *Density Estimation using Real NVP*, https://arxiv.org/abs/1605.08803

Normalizing Flows: Jacobian

Writing ∇_x as $(\nabla_{x_A}, \nabla_{x_B})$ and applying it to

$$\begin{pmatrix} y_A \\ y_B \end{pmatrix} = f_i(x_A, x_B) = \begin{pmatrix} x_A \\ x_B \exp(s(x_A)) + t(x_A) \end{pmatrix},$$

yields the Jacobian matrix

$$\begin{pmatrix} \nabla_{x_A} y_A & \nabla_{x_B} y_A \\ \nabla_{x_A} y_B & \nabla_{x_B} y_B \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \nabla_{x_A} y_B & \exp(s(x_A)) \end{pmatrix},$$

whose determinant is simply

$$\exp\left(\sum_{j=1}^{D-d} s_j(x_A) \right)$$

and hence $\log|\det J_i| = \sum_{j=1}^{D-d} s_j(x_A)$.

Normalizing Flows: Pseudocode

The pseudocode that follows is inspired by François Fleuret's excellent course on deep learning https://fleuret.org/dlc/, where the quantities s and t, which are intrinsically $(D - d)$-dimensional functions, are modeled with D-dimensional tensors by using a mask, b.

Normalizing Flows: $\boldsymbol{y} = \boldsymbol{f}_{\boldsymbol{i}}(\boldsymbol{x})$

Given $b = [1, 0]$ # dim(1) = d, dim(0) = $D - d$

and $M = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$:

$x = xM$ # $x = [x_B, x_A] \to x = [x_A, x_B]$

$s = S(b \odot x)$ # dim(s) = D

$t = T(b \odot x)$ # dim(t) = D

$s' = (1 - b) \odot s$ # $s' = [0, s_B]$

$\log J = \mathrm{sum}(s')$

$y = b \odot x + (1 - b) \odot [x \odot \exp(s) + t]$

return $y, \log J$

François Fleuret, https://fleuret.org/dlc/

Normalizing Flows: $\boldsymbol{x} = \boldsymbol{f_i^{-1}}(\boldsymbol{y})$

Given $b = [1, 0]$ # dim(1) = d, dim(0) = $D - d$

and $M = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

$\boldsymbol{x} = \boldsymbol{f}^{-1}(\boldsymbol{y})$:

$y = [y_A, y_B]$

$s = S(b \odot y)$ # dim(s) = D

$t = T(b \odot y)$ # dim(t) = D

$x = b \odot y + (1 - b) \odot (y - t) \odot \exp(-s)$

$x = xM$ # $x = [x_A, x_B] \rightarrow x = [x_B, x_A]$

return x

François Fleuret, https://fleuret.org/dlc/

Normalizing Flows: Training

- Data

$$x_i \sim p(x)$$

- Loss function

$$\begin{aligned} L(\theta) &= -\log p(x) \\ &= -\left[\log q\big(f(x)\big) + \log|\det J|\right] \end{aligned}$$

- $q(y = f(x))$ is usually chosen to be a zero mean, unit variance, diagonal D-dimensional normal.

- The unknown functions S and T are modeled with neural networks.

DIFFUSION MODELS

Diffusion Model

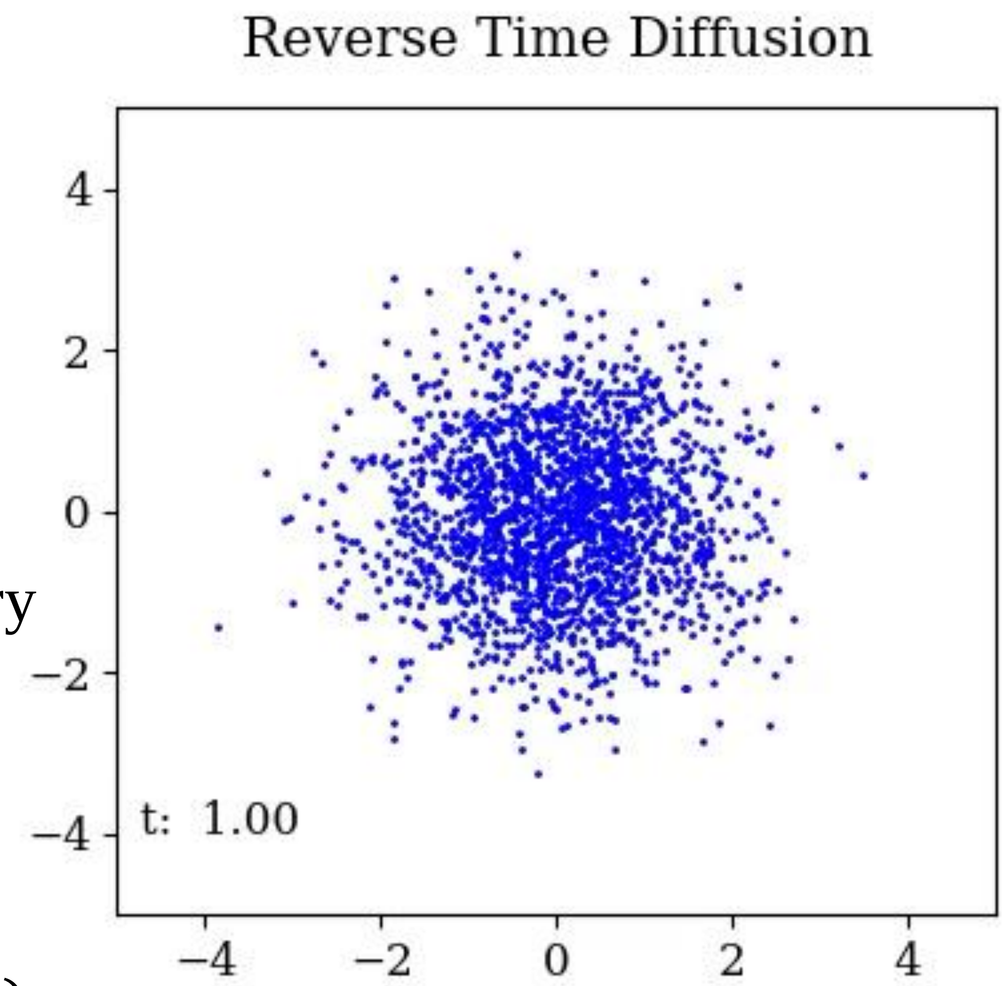
Like a normalizing flow, a diffusion model maps a point y to a point x, where $x, y \in \mathbb{R}^D$.

One way to do this is by solving the 1st order ordinary differential equation:

$$\frac{dx_t}{dt} = G(t, x_t)$$

where

$$G(t, x_t) = \lambda_t x_t + \mu_t q(t, x_t)$$



Diffusion Model

The main difficulty in solving the ODE[1,2]

$$\frac{dx_t}{dt} = \lambda_t x_t + \mu_t q(t, x_t)$$

from $t = 1$ to $t = 0$, where $\lambda_t \equiv \lambda(t)$ and $\mu_t \equiv \mu(t)$ are known functions of "time", is approximating

$$q(t, x_t) = \int_{\mathbb{R}^D} x_0 \, \frac{p(x_t \mid x_0)}{p(x_t)} p(x_0) dx_0$$

where $p(x_t \mid x_0)$ is a *D*-dimensional Gaussian and $p(x_0)$ is the target density.

1. Cheng Lu†, Yuhao Zhou†, Fan Bao†, Jianfei Chen†, Chongxuan Li‡, Jun Zhu, DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps*, arXiv:2206.00927v3, 2022.
2. Yanfang Lui, Minglei Yang, Zezhong Zhang, Feng Bao, Yanzhao Cao, and Guannan Zhang, Diffusion-Model-Assisted Supervised Learning of Generative Models for Density Estimation, arXiv:2310.14458v1, 2023

Diffusion Model

The functions $\lambda_t \equiv \lambda(t)$ and $\mu_t \equiv \mu(t)$ are given by

$$\lambda(t) = \frac{d \log \sigma_t}{dt}, \qquad \mu(t) = \alpha_t \frac{d \log \alpha_t / \sigma_t}{dt}$$

where a typical choice for $\alpha_t \equiv \alpha(t)$ is $\alpha_t = 1 - t$ and $\sigma_t = \sigma(t) = t$ or $\sigma_t = \sqrt{t}$.

The conditional density $p(x_t \,|x_0)$ is given by

$$p(x_t \,|x_0) = A \exp\left(-\frac{1}{2} z_t^2\right), \qquad z_t \equiv z(t) = \frac{x_t - \alpha_t x_0}{\sigma_t}$$

where $z(t)$ is a D-dimensional vector.

Diffusion Model

Given a sample $\left\{x_0^{(i)}\right\}_{i=1}^{K}$ from the target density $p(x_0)$, typically from a simulation, the integral

$$q(t, x_t) = \frac{1}{p(x_t)} \int_{\mathbb{R}^D} x_0 \, p(x_t \,|x_0) \, p(x_0) dx_0$$

can be approximated with Monte Carlo integration[1]. Define

$$\hat{p}(x_t) \equiv \frac{1}{K} \sum_{i=1}^{K} p(x_t \,|x_0^{(i)}) , \qquad \hat{p}_1(x_t) \equiv \frac{1}{K} \sum_{i=1}^{K} x_0^{(i)} \, p(x_t \,|x_0^{(i)})$$

Then

$$q(t, x_t) \approx \hat{p}_1(x_t)/\hat{p}(x_t)$$

1. Yanfang Lui, Minglei Yang, Zezhong Zhang, Feng Bao, Yanzhao Cao, and Guannan Zhang, Diffusion-Model-Assisted Supervised Learning of Generative Models for Density Estimation, arXiv:2310.14458v1, 2023

Diffusion Model

Advantages

- One significant advantage of this diffusion model is that the accuracy of the mapping from $y = x_1$ to $x = x_0$ depends solely on the accuracy of the numerical ODE solver and the sample size K used in approximating the two integrals.
- The solution for multiple points can be trivially parallelized.

Disadvantages

- Solving an integro-differential equation may become computational demanding for high-dimensional problems.
- A very fast mapping would require training a neural network on the data generated from the ODE solutions, which may impair the accuracy of the mapping.

Summary

- Finding good approximations to multidimensional probability densities is an important task in many fields.

- Normalizing flows have been shown to yield satisfactory results, though they may not scale well to very high-dimensional problems.

- Diffusion models can be used to map a point sampled from a Gaussian to another sampled from a desired target density. We briefly described a method that "simply" requires solving an ODE.