

Deep Generative Models

Lecture 3

Roman Isachenko



AI Masters

2024, Spring

Recap of previous lecture

Jacobian matrix

Let $f : \mathbb{R}^m \rightarrow \mathbb{R}^m$ be a differentiable function.

$$\mathbf{z} = f(\mathbf{x}), \quad \mathbf{J} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial z_1}{\partial x_1} & \cdots & \frac{\partial z_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_m}{\partial x_1} & \cdots & \frac{\partial z_m}{\partial x_m} \end{pmatrix} \in \mathbb{R}^{m \times m}$$

Change of variable theorem (CoV)

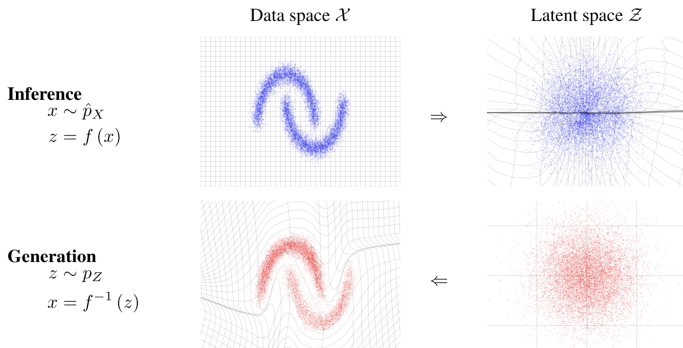
Let \mathbf{x} be a random variable with density function $p(\mathbf{x})$ and $f : \mathbb{R}^m \rightarrow \mathbb{R}^m$ is a differentiable, invertible function. If $\mathbf{z} = f(\mathbf{x})$, $\mathbf{x} = f^{-1}(\mathbf{z}) = g(\mathbf{z})$, then

$$\begin{aligned} p(\mathbf{x}) &= p(\mathbf{z}) |\det(\mathbf{J}_f)| = p(\mathbf{z}) \left| \det \left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right) \right| = p(f(\mathbf{x})) \left| \det \left(\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right) \right| \\ p(\mathbf{z}) &= p(\mathbf{x}) |\det(\mathbf{J}_g)| = p(\mathbf{x}) \left| \det \left(\frac{\partial \mathbf{x}}{\partial \mathbf{z}} \right) \right| = p(g(\mathbf{z})) \left| \det \left(\frac{\partial g(\mathbf{z})}{\partial \mathbf{z}} \right) \right|. \end{aligned}$$

Recap of previous lecture

Definition

Normalizing flow is a *differentiable, invertible* mapping from data \mathbf{x} to the noise \mathbf{z} .



Log likelihood

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(f_K \circ \dots \circ f_1(\mathbf{x})) + \sum_{k=1}^K \log |\det(\mathbf{J}_{f_k})|$$

Recap of previous lecture

Forward KL for flow model

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(f_{\boldsymbol{\theta}}(\mathbf{x})) + \log |\det(\mathbf{J}_f)|$$

Reverse KL for flow model

$$KL(p||\pi) = \mathbb{E}_{p(\mathbf{z})} [\log p(\mathbf{z}) - \log |\det(\mathbf{J}_g)| - \log \pi(g_{\boldsymbol{\theta}}(\mathbf{z}))]$$

Flow KL duality

$$\arg \min_{\boldsymbol{\theta}} KL(\pi(\mathbf{x})||p(\mathbf{x}|\boldsymbol{\theta})) = \arg \min_{\boldsymbol{\theta}} KL(p(\mathbf{z}|\boldsymbol{\theta})||p(\mathbf{z}))$$

- ▶ $p(\mathbf{z})$ is a base distribution; $\pi(\mathbf{x})$ is a data distribution;
- ▶ $\mathbf{z} \sim p(\mathbf{z})$, $\mathbf{x} = g_{\boldsymbol{\theta}}(\mathbf{z})$, $\mathbf{x} \sim p(\mathbf{x}|\boldsymbol{\theta})$;
- ▶ $\mathbf{x} \sim \pi(\mathbf{x})$, $\mathbf{z} = f_{\boldsymbol{\theta}}(\mathbf{x})$, $\mathbf{z} \sim p(\mathbf{z}|\boldsymbol{\theta})$.

Recap of previous lecture

Flow log-likelihood

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(f_{\boldsymbol{\theta}}(\mathbf{x})) + \log |\det(\mathbf{J}_f)|$$

The main challenge is a determinant of the Jacobian.

Linear flows

$$\mathbf{z} = f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{W}\mathbf{x}, \quad \mathbf{W} \in \mathbb{R}^{m \times m}, \quad \boldsymbol{\theta} = \mathbf{W}, \quad \mathbf{J}_f = \mathbf{W}^T$$

- ▶ LU-decomposition

$$\mathbf{W} = \mathbf{P}\mathbf{L}\mathbf{U}.$$

- ▶ QR-decomposition

$$\mathbf{W} = \mathbf{Q}\mathbf{R}.$$

Decomposition should be done only once in the beginning. Next, we fit decomposed matrices ($\mathbf{P}/\mathbf{L}/\mathbf{U}$ or \mathbf{Q}/\mathbf{R}).

Kingma D. P., Dhariwal P. Glow: Generative Flow with Invertible 1x1 Convolutions, 2018

Hoogeboom E., et al. Emerging convolutions for generative normalizing flows, 2019

Recap of previous lecture

Consider an autoregressive model

$$p(\mathbf{x}|\theta) = \prod_{j=1}^m p(x_j|\mathbf{x}_{1:j-1}, \theta), \quad p(x_j|\mathbf{x}_{1:j-1}, \theta) = \mathcal{N}(\mu_j(\mathbf{x}_{1:j-1}), \sigma_j^2(\mathbf{x}_{1:j-1})).$$

Gaussian autoregressive NF

$$\mathbf{x} = g_{\theta}(\mathbf{z}) \quad \Rightarrow \quad x_j = \sigma_j(\mathbf{x}_{1:j-1}) \cdot z_j + \mu_j(\mathbf{x}_{1:j-1}).$$

$$\mathbf{z} = f_{\theta}(\mathbf{x}) \quad \Rightarrow \quad z_j = (x_j - \mu_j(\mathbf{x}_{1:j-1})) \cdot \frac{1}{\sigma_j(\mathbf{x}_{1:j-1})}.$$

- ▶ We have an **invertible** and **differentiable** transformation from $p(\mathbf{z})$ to $p(\mathbf{x}|\theta)$.
- ▶ Jacobian of such transformation is triangular!

Generation function $g_{\theta}(\mathbf{z})$ is **sequential**.

Inference function $f_{\theta}(\mathbf{x})$ is **not sequential**.

Outline

1. RealNVP: coupling layer
2. Continuous-in-time Normalizing Flows
3. Adjoint method

Outline

1. RealNVP: coupling layer
2. Continuous-in-time Normalizing Flows
3. Adjoint method

RealNVP

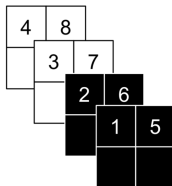
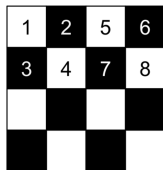
Let split \mathbf{x} and \mathbf{z} in two parts:

$$\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2] = [\mathbf{x}_{1:d}, \mathbf{x}_{d+1:m}]; \quad \mathbf{z} = [\mathbf{z}_1, \mathbf{z}_2] = [\mathbf{z}_{1:d}, \mathbf{z}_{d+1:m}].$$

Coupling layer

$$\begin{cases} \mathbf{x}_1 = \mathbf{z}_1; \\ \mathbf{x}_2 = \mathbf{z}_2 \odot \sigma_{\theta}(\mathbf{z}_1) + \mu_{\theta}(\mathbf{z}_1). \end{cases} \quad \begin{cases} \mathbf{z}_1 = \mathbf{x}_1; \\ \mathbf{z}_2 = (\mathbf{x}_2 - \mu_{\theta}(\mathbf{x}_1)) \odot \frac{1}{\sigma_{\theta}(\mathbf{x}_1)}. \end{cases}$$

Image partitioning



- ▶ Checkerboard ordering uses masking.
- ▶ Channelwise ordering uses splitting.

RealNVP

Coupling layer

$$\begin{cases} \mathbf{x}_1 = \mathbf{z}_1; \\ \mathbf{x}_2 = \mathbf{z}_2 \odot \sigma_{\theta}(\mathbf{z}_1) + \mu_{\theta}(\mathbf{z}_1). \end{cases} \quad \begin{cases} \mathbf{z}_1 = \mathbf{x}_1; \\ \mathbf{z}_2 = (\mathbf{x}_2 - \mu_{\theta}(\mathbf{x}_1)) \odot \frac{1}{\sigma_{\theta}(\mathbf{x}_1)}. \end{cases}$$

Estimating the density takes 1 pass, sampling takes 1 pass!

Jacobian

$$\det \left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right) = \det \begin{pmatrix} \mathbf{I}_d & 0_{d \times m-d} \\ \frac{\partial \mathbf{z}_2}{\partial \mathbf{x}_1} & \frac{\partial \mathbf{z}_2}{\partial \mathbf{x}_2} \end{pmatrix} = \prod_{j=1}^{m-d} \frac{1}{\sigma_j(\mathbf{x}_1)}.$$

Gaussian AR NF

$$\mathbf{x} = g_{\theta}(\mathbf{z}) \quad \Rightarrow \quad \mathbf{x}_j = \sigma_j(\mathbf{x}_{1:j-1}) \cdot \mathbf{z}_j + \mu_j(\mathbf{x}_{1:j-1}).$$

$$\mathbf{z} = f_{\theta}(\mathbf{x}) \quad \Rightarrow \quad \mathbf{z}_j = (\mathbf{x}_j - \mu_j(\mathbf{x}_{1:j-1})) \cdot \frac{1}{\sigma_j(\mathbf{x}_{1:j-1})}.$$

How to get RealNVP coupling layer from gaussian AR NF?

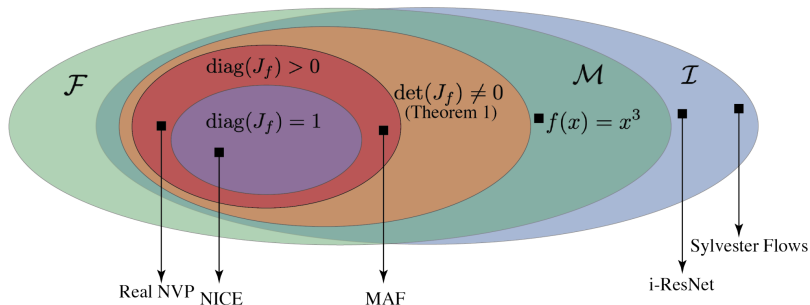
Glow samples

Glow model: coupling layer + linear flows (1x1 convs)



Kingma D. P., Dhariwal P. *Glow: Generative Flow with Invertible 1x1 Convolutions*, 2018

Venn diagram for Normalizing flows



- ▶ \mathcal{I} – invertible functions.
- ▶ \mathcal{F} – continuously differentiable functions whose Jacobian is lower triangular.
- ▶ \mathcal{M} – invertible functions from \mathcal{F} .

Outline

1. RealNVP: coupling layer
2. Continuous-in-time Normalizing Flows
3. Adjoint method

Continuous-in-time Normalizing Flows

Discrete-in-time NF

Previously we assume that the time axis is discrete:

$$\mathbf{z}_{t+1} = f_{\theta}(\mathbf{z}_t); \quad \log p(\mathbf{z}_{t+1}) = \log p(\mathbf{z}_t) - \log \left| \det \frac{\partial f_{\theta}(\mathbf{z}_t)}{\partial \mathbf{z}_t} \right|.$$

Let assume the more general case of continuous time. It means that we will have the dynamic function $\mathbf{z}(t)$.

Continuous-in-time dynamics

Consider Ordinary Differential Equation (ODE)

$$\frac{d\mathbf{z}(t)}{dt} = f_{\theta}(\mathbf{z}(t), t); \quad \text{with initial condition } \mathbf{z}(t_0) = \mathbf{z}_0.$$
$$\mathbf{z}(t_1) = \int_{t_0}^{t_1} f_{\theta}(\mathbf{z}(t), t) dt + \mathbf{z}_0 = \text{ODESolve}(\mathbf{z}(t_0), f_{\theta}, t_0, t_1).$$

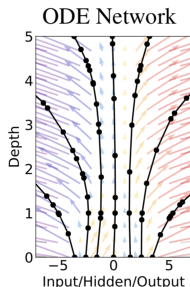
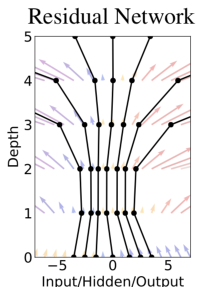
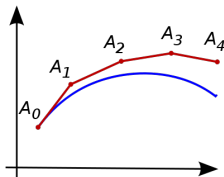
Here we need to define the $\text{ODESolve}(\mathbf{z}(t_0), f_{\theta}, t_0, t_1)$ procedure.

Continuous-in-time Normalizing Flows

Euler update step

$$\frac{\mathbf{z}(t + \Delta t) - \mathbf{z}(t)}{\Delta t} = f_{\theta}(\mathbf{z}(t), t) \Rightarrow \mathbf{z}(t + \Delta t) = \mathbf{z}(t) + \Delta t \cdot f_{\theta}(\mathbf{z}(t), t)$$

Note: Euler method is the simplest version of ODEsolve that is unstable in practice. It is possible to use more sophisticated methods (e.x. Runge-Kutta methods).

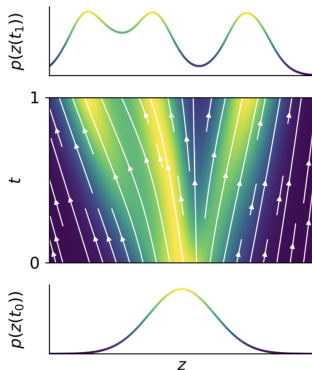


Continuous-in-time Normalizing Flows

Neural ODE

$$\frac{dz(t)}{dt} = f_{\theta}(z(t), t); \quad \text{with initial condition } z(t_0) = z_0$$

- ▶ Let $z(t_0)$ will be a random variable with some density function $p(z(t_0))$.
- ▶ Then $z(t_1)$ will be also a random variable with some other density function $p(z(t_1))$.
- ▶ We could say that we have the joint density function $p(z(t), t)$.
- ▶ What is the difference between $p(z(t), t)$ and $p(z, t)$?



Continuous-in-time Normalizing Flows

Let say that $p(\mathbf{z}, t_0)$ is the base distribution (e.x. standard Normal) and $p(\mathbf{z}, t_1)$ is the desired model distribution $p(\mathbf{x}|\theta)$.

Theorem (Picard)

If f is uniformly Lipschitz continuous in \mathbf{z} and continuous in t , then the ODE has a **unique** solution.

It means that we are able **uniquely revert** our ODE.

Forward and inverse transforms

$$\mathbf{x} = \mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f_{\theta}(\mathbf{z}(t), t) dt$$

$$\mathbf{z} = \mathbf{z}(t_0) = \mathbf{z}(t_1) + \int_{t_1}^{t_0} f_{\theta}(\mathbf{z}(t), t) dt$$

Note: Unlike discrete-in-time NF, f does not need to be bijective (uniqueness guarantees bijectivity).

Continuous-in-time Normalizing Flows

What do we need?

- ▶ We need the way to compute $p(\mathbf{z}, t)$ at any moment t .
- ▶ We need the way to find the optimal parameters θ of the dynamic f_θ .

Theorem (Kolmogorov-Fokker-Planck: special case)

If f is uniformly Lipschitz continuous in \mathbf{z} and continuous in t , then

$$\frac{d \log p(\mathbf{z}(t), t)}{dt} = -\text{tr} \left(\frac{\partial f_\theta(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)} \right).$$

$$\log p(\mathbf{z}(t_1), t_1) = \log p(\mathbf{z}(t_0), t_0) - \int_{t_0}^{t_1} \text{tr} \left(\frac{\partial f_\theta(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)} \right) dt.$$

It means that if we have the value $\mathbf{z}_0 = \mathbf{z}(t_0)$ then the solution of the ODE will give us the density at the moment t_1 .

Continuous-in-time Normalizing Flows

Forward transform + log-density

$$\mathbf{x} = \mathbf{z} + \int_{t_0}^{t_1} f_{\theta}(\mathbf{z}(t), t) dt$$

$$\log p(\mathbf{x}|\theta) = \log p(\mathbf{z}) - \int_{t_0}^{t_1} \text{tr} \left(\frac{\partial f_{\theta}(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)} \right) dt$$

Here $p(\mathbf{x}|\theta) = p(\mathbf{z}(t_1), t_1)$, $p(\mathbf{z}) = p(\mathbf{z}(t_0), t_0)$.

- ▶ **Discrete-in-time NF**: evaluation of determinant of the Jacobian costs $O(m^3)$ (we need invertible f).
- ▶ **Continuous-in-time NF**: getting the trace of the Jacobian costs $O(m^2)$ (we need smooth f).

Why $O(m^2)$?

$\text{tr} \left(\frac{\partial f_{\theta}(\mathbf{z}(t))}{\partial \mathbf{z}(t)} \right)$ costs $O(m^2)$ (m evaluations of f), since we have to compute a derivative for each diagonal element. It is possible to reduce cost from $O(m^2)$ to $O(m)$!

Continuous-in-time Normalizing Flows

Hutchinson's trace estimator

If $\epsilon \in \mathbb{R}^m$ is a random variable with $\mathbb{E}[\epsilon] = 0$ and $\text{cov}(\epsilon) = \mathbf{I}$, then

$$\begin{aligned}\text{tr}(\mathbf{A}) &= \text{tr}(\mathbf{A} \cdot \mathbf{I}) = \text{tr}\left(\mathbf{A} \cdot \mathbb{E}_{p(\epsilon)}\left[\epsilon\epsilon^T\right]\right) = \\ &= \mathbb{E}_{p(\epsilon)}\left[\text{tr}\left(\mathbf{A}\epsilon\epsilon^T\right)\right] = \mathbb{E}_{p(\epsilon)}\left[\epsilon^T \mathbf{A} \epsilon\right]\end{aligned}$$

Jacobian vector products $\mathbf{v}^T \frac{\partial f}{\partial \mathbf{z}}$ can be computed for approximately the same cost as evaluating f (`torch.autograd.functional.jvp`).

FFJORD density estimation

$$\begin{aligned}\log p(\mathbf{z}(t_1)) &= \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \text{tr}\left(\frac{\partial f_{\theta}(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)}\right) dt = \\ &= \log p(\mathbf{z}(t_0)) - \mathbb{E}_{p(\epsilon)} \int_{t_0}^{t_1} \left[\epsilon^T \frac{\partial f}{\partial \mathbf{z}} \epsilon\right] dt.\end{aligned}$$

Outline

1. RealNVP: coupling layer
2. Continuous-in-time Normalizing Flows
3. Adjoint method

Neural ODE

Continuous-in-time NF

$$\begin{aligned}\frac{d\mathbf{z}(t)}{dt} &= f_{\theta}(\mathbf{z}(t), t) & \frac{d \log p(\mathbf{z}(t), t)}{dt} &= -\text{tr} \left(\frac{\partial f_{\theta}(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)} \right) \\ \mathbf{x} &= \mathbf{z} + \int_{t_0}^{t_1} f_{\theta}(\mathbf{z}(t), t) dt & \log p(\mathbf{x}|\theta) &= \log p(\mathbf{z}) - \int_{t_0}^{t_1} \text{tr} \left(\frac{\partial f_{\theta}(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)} \right) dt\end{aligned}$$

How to get optimal parameters of θ ?

For fitting parameters we need gradients. We need the analogue of the backpropagation.

Forward pass (Loss function)

$$\mathbf{z} = \mathbf{x} + \int_{t_1}^{t_0} f_{\theta}(\mathbf{z}(t), t) dt, \quad L(\mathbf{z}) = \log p(\mathbf{z})$$

$$L(\mathbf{z}) = L \left(\mathbf{x} + \int_{t_1}^{t_0} f_{\theta}(\mathbf{z}(t), t) dt \right) = L(\text{ODESolve}(\mathbf{x}, f_{\theta}, t_1, t_0))$$

Neural ODE

Adjoint functions

$$\mathbf{a}_z(t) = \frac{\partial L}{\partial \mathbf{z}(t)}; \quad \mathbf{a}_\theta(t) = \frac{\partial L}{\partial \theta(t)}.$$

These functions show how the gradient of the loss depends on the hidden state $\mathbf{z}(t)$ and parameters θ .

Theorem (Pontryagin)

$$\frac{d\mathbf{a}_z(t)}{dt} = -\mathbf{a}_z(t)^T \cdot \frac{\partial f_\theta(\mathbf{z}(t), t)}{\partial \mathbf{z}}; \quad \frac{d\mathbf{a}_\theta(t)}{dt} = -\mathbf{a}_z(t)^T \cdot \frac{\partial f_\theta(\mathbf{z}(t), t)}{\partial \theta}.$$

Solution for adjoint function

$$\begin{aligned} \frac{\partial L}{\partial \theta(t_1)} &= \mathbf{a}_\theta(t_1) = - \int_{t_0}^{t_1} \mathbf{a}_z(t)^T \frac{\partial f_\theta(\mathbf{z}(t), t)}{\partial \theta(t)} dt + 0 \\ \frac{\partial L}{\partial \mathbf{z}(t_1)} &= \mathbf{a}_z(t_1) = - \int_{t_0}^{t_1} \mathbf{a}_z(t)^T \frac{\partial f_\theta(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)} dt + \frac{\partial L}{\partial \mathbf{z}(t_0)} \end{aligned}$$

Note: These equations are solved in reverse time direction.

Adjoint method

Forward pass

$$\mathbf{z} = \mathbf{z}(t_0) = \int_{t_0}^{t_1} f_{\theta}(\mathbf{z}(t), t) dt + \mathbf{x} \quad \Rightarrow \quad \text{ODE Solver}$$

Backward pass

$$\left. \begin{aligned} \frac{\partial L}{\partial \theta(t_1)} &= \mathbf{a}_{\theta}(t_1) = - \int_{t_0}^{t_1} \mathbf{a}_{\mathbf{z}}(t)^T \frac{\partial f_{\theta}(\mathbf{z}(t), t)}{\partial \theta(t)} dt + 0 \\ \frac{\partial L}{\partial \mathbf{z}(t_1)} &= \mathbf{a}_{\mathbf{z}}(t_1) = - \int_{t_0}^{t_1} \mathbf{a}_{\mathbf{z}}(t)^T \frac{\partial f_{\theta}(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)} dt + \frac{\partial L}{\partial \mathbf{z}(t_0)} \\ \mathbf{z}(t_1) &= - \int_{t_1}^{t_0} f_{\theta}(\mathbf{z}(t), t) dt + \mathbf{z}_0. \end{aligned} \right\} \Rightarrow \text{ODE Solver}$$

Note: These scary formulas are the standard backprop in the discrete case.

Summary

- ▶ The RealNVP coupling layer is an effective type of flow (special case of AR flows) that has fast inference and generation modes.
- ▶ Kolmogorov-Fokker-Planck theorem allows to construct continuous-in-time normalizing flow with less functional restrictions.
- ▶ FFJORD model makes such kind of NF scalable.
- ▶ Adjoint method generalizes backpropagation procedure and allows to train Neural ODE solving ODE for adjoint function back in time.