Deep Generative Models

Lecture 3

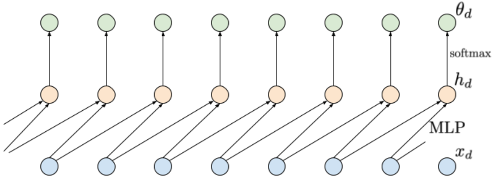
Roman Isachenko

AI Masters

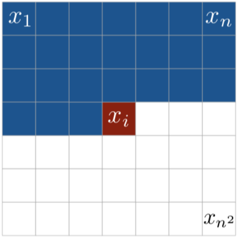
2024, Summer

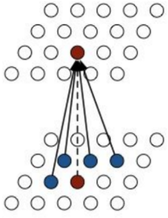
Recap of previous lecture

Autoregressive MLP



Autoregressive CNN





---

image credit: https://jmtomczak.github.io/blog/2/2_ARM.html

Recap of previous lecture

Jacobian matrix

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

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

Change of variable theorem (CoV)

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

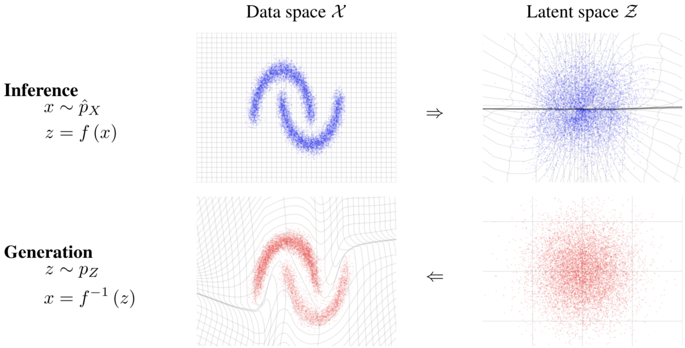
$$p(\mathbf{x}) = p(\mathbf{z})|\det(\mathbf{J_f})| = p(\mathbf{z}) \left|\det\left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}}\right)\right| = p(\mathbf{f}(\mathbf{x})) \left|\det\left(\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|$$

$$p(\mathbf{z}) = p(\mathbf{x})|\det(\mathbf{J_g})| = p(\mathbf{x}) \left|\det\left(\frac{\partial \mathbf{x}}{\partial \mathbf{z}}\right)\right| = p(\mathbf{g}(\mathbf{z})) \left|\det\left(\frac{\partial \mathbf{g}(\mathbf{z})}{\partial \mathbf{z}}\right)\right|.$$

Recap of previous lecture

Definition

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



Log likelihood

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(\mathbf{f}_K \circ \dots \circ \mathbf{f}_1(\mathbf{x})) + \sum_{k=1}^{K} \log |\det(\mathbf{J}_{\mathbf{f}_k})|$$

Dinh L., Sohl-Dickstein J., Bengio S. Density estimation using Real NVP, 2016

Recap of previous lecture

Forward KL for flow model

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

Reverse KL for flow model

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

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;
- $\mathbf{z} \sim p(\mathbf{z})$, $\mathbf{x} = \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{z})$, $\mathbf{x} \sim p(\mathbf{x}|\boldsymbol{\theta})$;
- $\mathbf{x} \sim \pi(\mathbf{x})$, $\mathbf{z} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})$, $\mathbf{z} \sim p(\mathbf{z}|\boldsymbol{\theta})$.

Papamakarios G. et al. Normalizing flows for probabilistic modeling and inference, 2019

Outline

1. NF examples
   Linear normalizing flows
   Gaussian autoregressive NF
   RealNVP: coupling layer

2. Continuous-in-time normalizing flows

Outline

1. NF examples
    - Linear normalizing flows
    - Gaussian autoregressive NF
    - RealNVP: coupling layer

2. Continuous-in-time normalizing flows

Outline

Jacobian structure

Normalizing flows log-likelihood

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})) + \log \left| \det \left(\frac{\partial \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})}{\partial \mathbf{x}} \right) \right|$$

The main challenge is a determinant of the Jacobian matrix.

What is the $det(\mathbf{J})$ in the following cases?

Consider a linear layer $\mathbf{z} = \mathbf{W}\mathbf{x}$, $\mathbf{W} \in \mathbb{R}^{m \times m}$.

1. Let $\mathbf{z}$ be a permutation of $\mathbf{x}$.
2. Let z_j depend only on x_j.

$$\log \left| \det \left(\frac{\partial \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})}{\partial \mathbf{x}} \right) \right| = \log \left| \prod_{j=1}^{m} \frac{\partial f_{j,\boldsymbol{\theta}}(x_j)}{\partial x_j} \right| = \sum_{j=1}^{m} \log \left| \frac{\partial f_{j,\boldsymbol{\theta}}(x_j)}{\partial x_j} \right|.$$

3. Let z_j depend only on $\mathbf{x}_{1:j}$ (autoregressive dependency).

Linear normalizing flows

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

In general, we need $O(m^3)$ to invert matrix.

Invertibility

- Diagonal matrix $O(m)$.
- Triangular matrix $O(m^2)$.
- It is impossible to parametrize all invertible matrices.

Invertible 1x1 conv

$\mathbf{W} \in \mathbb{R}^{c \times c}$ – kernel of 1x1 convolution with c input and c output channels. The computational complexity of computing or differentiating $\det(\mathbf{W})$ is $O(c^3)$. Cost to compute $\det(\mathbf{W})$ is $O(c^3)$. It should be invertible.

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

Linear normalizing flows

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

Matrix decompositions

- **LU-decomposition**

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

  where $\mathbf{P}$ is a permutation matrix, $\mathbf{L}$ is lower triangular with positive diagonal, $\mathbf{U}$ is upper triangular with positive diagonal.

- **QR-decomposition**

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

  where $\mathbf{Q}$ is an orthogonal matrix, $\mathbf{R}$ is an upper triangular matrix with positive diagonal.

Decomposition should be done only once in the beggining. 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

Outline

Gaussian autoregressive model

Consider an autoregressive model

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

Sampling

$$x_j = \sigma_j(\mathbf{x}_{1:j-1}) \cdot z_j + \mu_j(\mathbf{x}_{1:j-1}), \quad z_j \sim \mathcal{N}(0, 1).$$

Inverse transform

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

- We have an **invertible** and **differentiable** transformation from $p(\mathbf{z})$ to $p(\mathbf{x}|\boldsymbol{\theta})$.
- It is an autoregressive (AR) NF with the base distribution $p(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$!
- Jacobian of such transformation is triangular!

Kingma D. P. et al. Improving Variational Inference with Inverse Autoregressive Flow, 2016

Gaussian autoregressive NF

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

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

Generation function $\mathbf{g}_{\boldsymbol{\theta}}(\mathbf{z})$ is **sequential**.
Inference function $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})$ is **not sequential**.

Forward KL for NF

$$KL(\pi || p) = -\mathbb{E}_{\pi(\mathbf{x})} \left[\log p(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})) + \log |\det(\mathbf{J_f})| \right] + \text{const}$$

- We need to be able to compute $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})$ and its Jacobian.
- We need to be able to compute the density $p(\mathbf{z})$.
- We don't need to think about computing the function $\mathbf{g}_{\boldsymbol{\theta}}(\mathbf{z}) = \mathbf{f}_{\boldsymbol{\theta}}^{-1}(\mathbf{z})$ until we want to sample from the model.

Papamakarios G., Pavlakou T., Murray I. Masked Autoregressive Flow for Density Estimation, 2017

Gaussian autoregressive NF

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

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

- Sampling is sequential, density estimation is parallel.
- Forward KL is a natural loss.

Forward transform: $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})$

$$z_j = (x_j - \mu_j(\mathbf{x}_{1:j-1})) \cdot \frac{1}{\sigma_j(\mathbf{x}_{1:j-1})}$$

Inverse transform: $\mathbf{g}_{\boldsymbol{\theta}}(\mathbf{z})$

$$x_j = \sigma_j(\mathbf{x}_{1:j-1}) \cdot z_j + \mu_j(\mathbf{x}_{1:j-1})$$

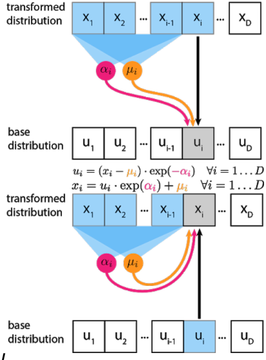


image credit: https://blog.evjang.com/2018/01/nf2.html

Outline

1. NF examples
   - Linear normalizing flows
   - Gaussian autoregressive NF
   - RealNVP: coupling layer

2. Continuous-in-time normalizing flows

RealNVP

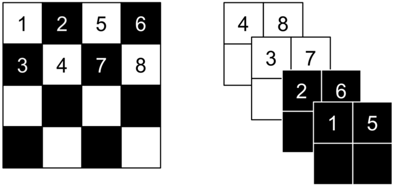
Let split $\mathbf{x}$ and $\mathbf{z}$ in two parts:

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

Coupling layer

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

Image partitioning



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

Dinh L., Sohl-Dickstein J., Bengio S. Density estimation using Real NVP, 2016

RealNVP

Coupling layer

$$\begin{cases} \mathbf{x}_1 = \mathbf{z}_1; \\ \mathbf{x}_2 = \mathbf{z}_2 \odot \boldsymbol{\sigma}_{\boldsymbol{\theta}}(\mathbf{z}_1) + \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}_1). \end{cases} \qquad \begin{cases} \mathbf{z}_1 = \mathbf{x}_1; \\ \mathbf{z}_2 = (\mathbf{x}_2 - \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_1)) \odot \frac{1}{\boldsymbol{\sigma}_{\boldsymbol{\theta}}(\mathbf{x}_1)}. \end{cases}$$

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

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

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

How to get RealNVP coupling layer from gaussian AR NF?

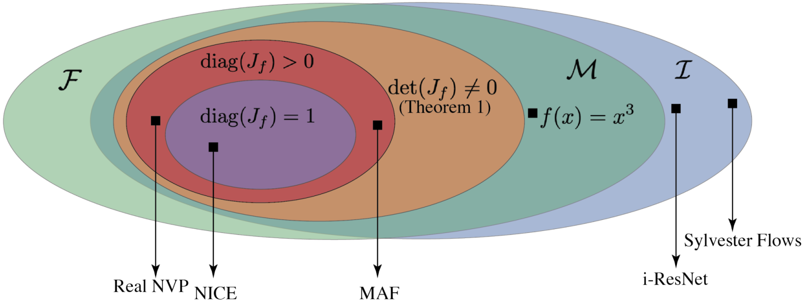
Dinh L., Sohl-Dickstein J., Bengio S. Density estimation using Real NVP, 2016

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 $\mathcal{F}$.

Song Y., Meng C., Ermon S. Mintnet: Building invertible neural networks with masked convolutions, 2019

Outline

Continuous-in-time normalizing flows

Discrete-in-time NF

Previously we assume that the time axis is discrete:

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

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)

$$\frac{d\mathbf{z}(t)}{dt} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{z}(t), t); \quad \text{with initial condition } \mathbf{z}(t_0) = \mathbf{z}_0.$$

$$\mathbf{z}(t_1) = \int_{t_0}^{t_1} \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{z}(t), t) dt + \mathbf{z}_0 \approx \text{ODESolve}(\mathbf{z}(t_0), \mathbf{f}_{\boldsymbol{\theta}}, t_0, t_1).$$

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

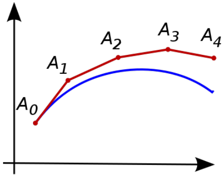
Grathwohl W. et al. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models, 2018

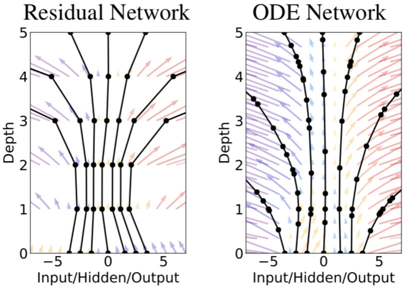
Continuous-in-time normalizing flows

Euler update step

$$\frac{\mathbf{z}(t+\Delta t) - \mathbf{z}(t)}{\Delta t} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{z}(t), t) \;\Rightarrow\; \mathbf{z}(t+\Delta t) = \mathbf{z}(t) + \Delta t \cdot \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{z}(t), t)$$

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).





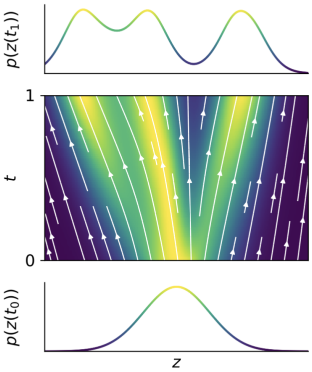
Chen R. T. Q. et al. Neural Ordinary Differential Equations, 2018

Continuous-in-time Normalizing Flows

Neural ODE

$$\frac{d\mathbf{z}(t)}{dt} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{z}(t), t); \quad \text{with initial condition } \mathbf{z}(t_0) = \mathbf{z}_0$$

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



Grathwohl W. et al. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models, 2018

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 $p(\mathbf{x}|\boldsymbol{\theta})$.

Theorem (Picard)

If $\mathbf{f}$ 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

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

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

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

Chen R. T. Q. et al. Neural Ordinary Differential Equations, 2018

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 $\boldsymbol{\theta}$ of the dynamic $\mathbf{f}_{\boldsymbol{\theta}}$.

Theorem (Kolmogorov-Fokker-Planck: special case)

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

$$\frac{d \log p(\mathbf{z}(t), t)}{dt} = -\mathrm{tr}\left(\frac{\partial \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)}\right).$$

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

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.

Chen R. T. Q. et al. Neural Ordinary Differential Equations, 2018

Continuous-in-time Normalizing Flows

Forward transform + log-density

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

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

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

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

Why $O(m^2)$?

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

Chen R. T. Q. et al. Neural Ordinary Differential Equations, 2018

Continuous-in-time Normalizing Flows

Hutchinson's trace estimator

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

$$\mathrm{tr}(\mathbf{A}) = \mathrm{tr}\left(\mathbf{A} \cdot \mathbf{I}\right) = \mathrm{tr}\left(\mathbf{A} \cdot \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T\right]\right) =$$
$$= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\mathrm{tr}\left(\mathbf{A}\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T\right)\right] = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\boldsymbol{\epsilon}^T \mathbf{A} \boldsymbol{\epsilon}\right]$$

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

FFJORD density estimation

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

Grathwohl W. et al. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models, 2018

Summary

- Linear NF try to parametrize set of invertible matrices via matrix decompositions.
- Gaussian autoregressive NF is an autoregressive model with triangular Jacobian. It has fast inference function and slow generation function. Forward KL is a natural loss function.
- The RealNVP coupling layer is an effective type of NF (special case of AR NF) that has fast inference and generation modes.
- Continuous-in-time NF uses neural ODE to define continuous dynamic $\mathbf{z}(t)$. It has less functional restrictions.
- Kolmogorov-Fokker-Planck theorem allows to calculate $\log p(\mathbf{z}, t)$ at arbitrary moment t.
- FFJORD model makes such kind of NF scalable.