Deep Generative Models

Lecture 11

Roman Isachenko

Moscow Institute of Physics and Technology
Yandex School of Data Analysis

2024, Autumn

Recap of previous lecture

$$\mathcal{L}_t = \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} \left[\frac{1}{2\tilde{\beta}_t} \left\| \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_{\boldsymbol{\theta},t}(\mathbf{x}_t) \right\|^2 \right]$$

Reparametrization

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{\alpha_t}} \cdot \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{\alpha_t(1 - \bar{\alpha}_t)}} \cdot \boldsymbol{\epsilon}$$

$$\boldsymbol{\mu}_{\boldsymbol{\theta},t}(\mathbf{x}_t) = \frac{1}{\sqrt{\alpha_t}} \cdot \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{\alpha_t(1 - \bar{\alpha}_t)}} \cdot \boldsymbol{\epsilon}_{\boldsymbol{\theta},t}(\mathbf{x}_t)$$

$$\mathcal{L}_t = \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})} \left[\frac{(1 - \alpha_t)^2}{2\tilde{\beta}_t \alpha_t (1 - \bar{\alpha}_t)} \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta},t}\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon} \right) \right\|^2 \right]$$

At each step of reverse diffusion process we try to predict the noise $\boldsymbol{\epsilon}$ that we used in the forward diffusion process!

Simplified objective

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t \sim U\{2, T\}} \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})} \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta},t}\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon} \right) \right\|^2$$

Ho J. Denoising Diffusion Probabilistic Models, 2020

Recap of previous lecture

Training of DDPM

1. Get the sample $\mathbf{x}_0 \sim \pi(\mathbf{x})$.
2. Sample timestamp $t \sim U\{1, T\}$ and the noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$.
3. Get noisy image $\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \cdot \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \boldsymbol{\epsilon}$.
4. Compute loss $\mathcal{L}_{\text{simple}} = \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta},t}(\mathbf{x}_t)\|^2$.

Sampling of DDPM

1. Sample $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$.
2. Compute mean of $p(\mathbf{x}_{t-1}|\mathbf{x}_t, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta},t}(\mathbf{x}_t), \sigma_t^2 \cdot \mathbf{I})$:

$$\boldsymbol{\mu}_{\boldsymbol{\theta},t}(\mathbf{x}_t) = \frac{1}{\sqrt{\alpha_t}} \cdot \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{\alpha_t(1 - \bar{\alpha}_t)}} \cdot \boldsymbol{\epsilon}_{\boldsymbol{\theta},t}(\mathbf{x}_t)$$

3. Get denoised image $\mathbf{x}_{t-1} = \boldsymbol{\mu}_{\boldsymbol{\theta},t}(\mathbf{x}_t) + \sigma_t \cdot \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$.

Ho J. Denoising Diffusion Probabilistic Models, 2020

Recap of previous lecture

DDPM objective

$$\mathbb{E}_{\pi(\mathbf{x}_0)}\mathbb{E}_{t\sim U\{1,T\}}\mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[\frac{(1-\alpha_t)^2}{2\tilde{\beta}_t\alpha_t}\left\|\mathbf{s}_{\boldsymbol{\theta},t}(\mathbf{x}_t)-\nabla_{\mathbf{x}_t}\log q(\mathbf{x}_t|\mathbf{x}_0)\right\|_2^2\right]$$

In practice the coefficient is omitted.

NCSN objective

$$\mathbb{E}_{\pi(\mathbf{x}_0)}\mathbb{E}_{t\sim U\{1,T\}}\mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left\|\mathbf{s}_{\boldsymbol{\theta},\sigma_t}(\mathbf{x}_t)-\nabla_{\mathbf{x}_t}\log q(\mathbf{x}_t|\mathbf{x}_0)\right\|_2^2$$

Note: The objective of DDPM and NCSN is almost identical. But the difference in sampling scheme:

- NCSN uses annealed Langevin dynamics;
- DDPM uses ancestral sampling.

Ho J. Denoising Diffusion Probabilistic Models, 2020

Recap of previous lecture

DDPM vs NCSN: summary

- Different Markov chains:
  - DDPM: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \cdot \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \boldsymbol{\epsilon}$;
  - NCSN: $\mathbf{x}_t = \mathbf{x}_0 + \sigma_t \cdot \boldsymbol{\epsilon}$.
  - It is possible to consider the more general framework $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\alpha_t \cdot \mathbf{x}_0, \sigma_t^2 \cdot \mathbf{I})$
- Identical objectives: ELBO $\equiv$ score-matching.
- Different sampling schemes:
  - ancestral sampling for DDPM;
  - annealed Langevin dynamics for NCSN;
  - there is a combined approach with alternating updates of DDPM and NCSN.

Kingma D. et al. Variational Diffusion Models, 2021
Song Y. et al. Score-Based Generative Modeling through Stochastic Differential Equations, 2020

Recap of previous lecture

Unconditional generation

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{1-\beta_t}} \cdot \mathbf{x}_t + \frac{\beta_t}{\sqrt{1-\beta_t}} \cdot \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\boldsymbol{\theta}) + \sigma_t \cdot \boldsymbol{\epsilon}$$

Conditional generation

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{1-\beta_t}} \cdot \mathbf{x}_t + \frac{\beta_t}{\sqrt{1-\beta_t}} \cdot \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{y}, \boldsymbol{\theta}) + \sigma_t \cdot \boldsymbol{\epsilon}$$

Conditional distribution

$$\begin{aligned}\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{y}, \boldsymbol{\theta}) &= \nabla_{\mathbf{x}_t} \log p(\mathbf{y}|\mathbf{x}_t) + \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\boldsymbol{\theta}) \\ &= \nabla_{\mathbf{x}_t} \log p(\mathbf{y}|\mathbf{x}_t) - \frac{\boldsymbol{\epsilon}_{\boldsymbol{\theta},t}(\mathbf{x}_t)}{\sqrt{1-\bar{\alpha}_t}}\end{aligned}$$

Here $p(\mathbf{y}|\mathbf{x}_t)$ – classifier on noisy samples (we have to learn it separately).

Dhariwal P., Nichol A. Diffusion Models Beat GANs on Image Synthesis, 2021

Recap of previous lecture

Classifier-corrected noise prediction

$$\boldsymbol{\epsilon}_{\boldsymbol{\theta},t}(\mathbf{x}_t, \mathbf{y}) = \boldsymbol{\epsilon}_{\boldsymbol{\theta},t}(\mathbf{x}_t) - \sqrt{1 - \bar{\alpha}_t} \cdot \nabla_{\mathbf{x}_t} \log p(\mathbf{y}|\mathbf{x}_t)$$

Guidance scale

$$\boldsymbol{\epsilon}_{\boldsymbol{\theta},t}(\mathbf{x}_t, \mathbf{y}) = \boldsymbol{\epsilon}_{\boldsymbol{\theta},t}(\mathbf{x}_t) - \gamma \cdot \sqrt{1 - \bar{\alpha}_t} \cdot \nabla_{\mathbf{x}_t} \log p(\mathbf{y}|\mathbf{x}_t)$$

- Train DDPM as usual.
- Train the additional classifier $p(\mathbf{y}|\mathbf{x}_t)$ on the noisy samples $\mathbf{x}_t$.

Guided sampling

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \cdot \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{\alpha_t(1 - \bar{\alpha}_t)}} \cdot \boldsymbol{\epsilon}_{\boldsymbol{\theta},t}(\mathbf{x}_t, \mathbf{y}) + \sigma_t \cdot \boldsymbol{\epsilon}$$

Note: Guidance scale γ tries to sharpen the distribution $p(\mathbf{y}|\mathbf{x}_t)$ (in this case Z should not depend on $\mathbf{x}_t$).

Dhariwal P., Nichol A. Diffusion Models Beat GANs on Image Synthesis, 2021

Recap of previous lecture

- Previous method requires training the additional classifier model $p(\mathbf{y}|\mathbf{x}_t)$ on the noisy data.
- Let try to avoid this requirement.

$$\nabla_{\mathbf{x}_t} \log p(\mathbf{y}|\mathbf{x}_t) = \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{y}, \boldsymbol{\theta}) - \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\boldsymbol{\theta})$$

$$\begin{aligned}\nabla^{\gamma}_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{y}, \boldsymbol{\theta}) &= \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\boldsymbol{\theta}) + \gamma \cdot \nabla_{\mathbf{x}_t} \log p(\mathbf{y}|\mathbf{x}_t) = \\ &= (1-\gamma) \cdot \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\boldsymbol{\theta}) + \gamma \cdot \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{y}, \boldsymbol{\theta})\end{aligned}$$

Classifier-free-corrected noise prediction

$$\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta},t}(\mathbf{x}_t, \mathbf{y}) = \gamma \cdot \boldsymbol{\epsilon}_{\boldsymbol{\theta},t}(\mathbf{x}_t, \mathbf{y}) + (1-\gamma) \cdot \boldsymbol{\epsilon}_{\boldsymbol{\theta},t}(\mathbf{x}_t)$$

- Train the single model $\boldsymbol{\epsilon}_{\boldsymbol{\theta},t}(\mathbf{x}_t, \mathbf{y})$ on **supervised** data alternating with real conditioning $\mathbf{y}$ and empty conditioning $\mathbf{y} = \emptyset$.
- Apply the model twice during inference.

Ho J., Salimans T. Classifier-Free Diffusion Guidance, 2022

Outline

1. Continuous-in-time normalizing flows
2. Continuity equation for NF log-likelihood
3. FFJORD (Hutchinson's trace estimator)
4. Adjoint method for continuous-in-time NF

Outline

1. Continuous-in-time normalizing flows

2. Continuity equation for NF log-likelihood

3. FFJORD (Hutchinson's trace estimator)

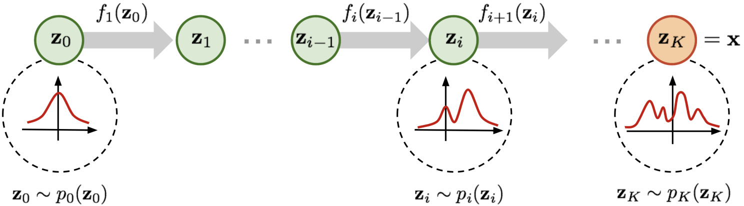
4. Adjoint method for continuous-in-time NF

Discrete-in-time NF

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

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



$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(\mathbf{f}_K \circ \cdots \circ \mathbf{f}_1(\mathbf{x})) + \sum_{k=1}^{K} \log \left|\det\left(\frac{\partial \mathbf{f}_k}{\partial \mathbf{f}_{k-1}}\right)\right|.$$

---

https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

Discrete-in-time NF

- Previously we assumed that the time axis is discrete:
$$\mathbf{x}_{t+1} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_t, t); \quad \log p(\mathbf{x}_{t+1}) = \log p(\mathbf{x}_t) - \log \left| \det \frac{\partial \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_t)}{\partial \mathbf{x}_t} \right| .$$
- Let consider the more general case of continuous time. It means that we will have the function $\mathbf{x}(t) : \mathbb{R} \to \mathbb{R}^m$ of continuous dynamic.

Continuous-in-time dynamics

Consider Ordinary Differential Equation (ODE)

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t); \quad \text{with initial condition } \mathbf{x}(t_0) = \mathbf{x}_0.$$
$$\mathbf{x}(t_1) = \int_{t_0}^{t_1} \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t) dt + \mathbf{x}_0$$

Here $\mathbf{f}_{\boldsymbol{\theta}} : \mathbb{R}^m \times [t_0, t_1] \to \mathbb{R}^m$ is a vector field.

Numerical solution of ODE

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

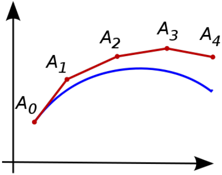
$$\mathbf{x}(t_1) = \int_{t_0}^{t_1} \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)dt + \mathbf{x}_0 \approx \text{ODESolve}_f(\mathbf{x}_0, \boldsymbol{\theta}, t_0, t_1).$$

Here we need to define the computational procedure $\text{ODESolve}_f(\mathbf{x}_0, \boldsymbol{\theta}, t_0, t_1)$.

Euler update step

$$\frac{\mathbf{x}(t + \Delta t) - \mathbf{x}(t)}{\Delta t} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)$$

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t \cdot \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)$$



Note: Euler method is the simplest version of the ODESolve that is unstable in practice.

Continuous-in-time NF

Let consider time interval $[t_0, t_1] = [0, 1]$ without loss of generality.

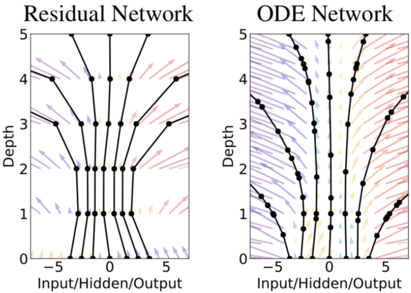
Neural ODE

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

Euler ODESolve

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t \cdot \mathbf{f}_\theta(\mathbf{x}(t), t)$$

Note: It is possible to use more sophisticated numerical methods instead if Euler (e.x. Runge-Kutta methods).

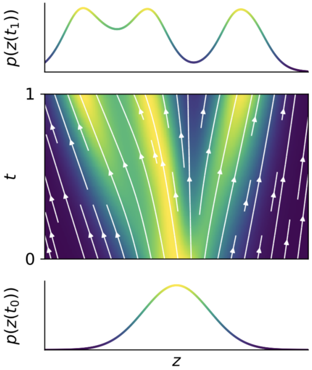


Chen R. T. Q. et al. Neural Ordinary Differential Equations, 2018

Continuous-in-time NF

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

- $\mathbf{x}(0)$ is a random variable with the density function $p(\mathbf{x}(0))$.
- $\mathbf{x}(1)$ is a random variable with the density function $p(\mathbf{x}(1))$.
- $p_t(\mathbf{x}) = p(\mathbf{x}, t)$ is the **probability path** between $p_0(\mathbf{x})$ and $p_1(\mathbf{x})$.
- Let say that $p_0(\mathbf{x}) = \mathcal{N}(0, \mathbf{I})$ is the base distribution and $p_1(\mathbf{x}) = \pi(\mathbf{x})$ is the data distribution (that we try to approximate with the model distribution $p(\mathbf{x}|\boldsymbol{\theta})$.



What is the difference between $p_t(\mathbf{x}(t))$ and $p_t(\mathbf{x})$?

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

Continuous-in-time NF

Theorem (Picard)

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

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

$$\mathbf{x}(1) = \mathbf{x}(0) + \int_0^1 \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)dt$$
$$\mathbf{x}(0) = \mathbf{x}(1) + \int_1^0 \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)dt$$

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

What is left?

- We need the way to compute $p_t(\mathbf{x})$ at any moment t.
- We need the way to find the optimal parameters $\boldsymbol{\theta}$ of the dynamic $\mathbf{f}_{\boldsymbol{\theta}}$.

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

Outline

1. Continuous-in-time normalizing flows

2. Continuity equation for NF log-likelihood

3. FFJORD (Hutchinson's trace estimator)

4. Adjoint method for continuous-in-time NF

Continuous-in-time NF

Theorem (continuity equation)

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

$$\frac{d \log p_t(\mathbf{x}(t))}{dt} = -\text{tr}\left(\frac{\partial \mathbf{f}(\mathbf{x}(t), t)}{\partial \mathbf{x}(t)}\right)$$

It means that if we have the value $\mathbf{x}_0 = \mathbf{x}(0)$ then the solution of the continuity equation will give us the density $p_1(\mathbf{x}(1))$.

Solution of continuity equation

$$\log p_1(\mathbf{x}(1)) = \log p_0(\mathbf{x}(0)) - \int_0^1 \text{tr}\left(\frac{\partial \mathbf{f}(\mathbf{x}(t), t)}{\partial \mathbf{x}(t)}\right) dt.$$

Note: This solution will give us the density along the trajectory (not the total probability path).

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

Continuous-in-time Normalizing Flows

Forward transform + log-density

$$\mathbf{x}(1) = \mathbf{x}(0) + \int_0^1 \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)\,dt$$

$$\log p_1(\mathbf{x}(1)|\boldsymbol{\theta}) = \log p_0(\mathbf{x}(0)) - \int_0^1 \mathrm{tr}\left(\frac{\partial \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)}{\partial \mathbf{x}(t)}\right) dt$$

- **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 $O(m^2)$?

$\mathrm{tr}\left(\frac{\partial \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t))}{\partial \mathbf{x}(t)}\right)$ 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

Outline

1. Continuous-in-time normalizing flows
2. Continuity equation for NF log-likelihood
3. FFJORD (Hutchinson's trace estimator)
4. Adjoint method for continuous-in-time NF

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 $\text{Cov}(\boldsymbol{\epsilon}) = \mathbf{I}$, then

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

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

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

FFJORD density estimation

$$\log p_1(\mathbf{x}(1)) = \log p_0(\mathbf{x}(0)) - \int_0^1 \text{tr}\left(\frac{\partial \mathbf{f}(\mathbf{x}(t), t)}{\partial \mathbf{x}(t)}\right) dt =$$

$$= \log p_0(\mathbf{x}(0)) - \mathbb{E}_{p(\boldsymbol{\epsilon})} \int_0^1 \left[\boldsymbol{\epsilon}^T \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \boldsymbol{\epsilon}\right] dt.$$

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

Outline

1. Continuous-in-time normalizing flows
2. Continuity equation for NF log-likelihood
3. FFJORD (Hutchinson's trace estimator)
4. Adjoint method for continuous-in-time NF

Continuous-in-time NF

Dynamics ODE

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)$$

$$\mathbf{x}(1) = \mathbf{x}(0) + \int_0^1 \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t) dt$$

Continuity ODE

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

$$\log p_1(\mathbf{x}(1)) = \log p_0(\mathbf{x}(0)) - \int_0^1 \mathrm{tr}\left(\frac{\partial \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)}{\partial \mathbf{x}(t)}\right) dt$$

How to get optimal parameters of $\boldsymbol{\theta}$?

- We need the gradients for fitting the parameters
- We need the continuous analogue of the backpropagation.

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

Neural ODE

Forward pass (Loss function)

$$L(\mathbf{x}) = -\log p_1(\mathbf{x}(1)|\boldsymbol{\theta}) = -\log p_0(\mathbf{x}(0)) + \int_0^1 \mathrm{tr}\left(\frac{\partial \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)}{\partial \mathbf{x}(t)}\right) dt$$

Adjoint functions

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

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

Theorem (Pontryagin)

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

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

Adjoint method

Theorem (Pontryagin)

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

Solution for the adjoints function

$$\frac{\partial L}{\partial \boldsymbol{\theta}(0)} = \mathbf{a}_{\boldsymbol{\theta}}(0) = -\int_1^0 \mathbf{a}_{\mathbf{x}}(t)^T \frac{\partial \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)}{\partial \boldsymbol{\theta}(t)} dt + 0$$

$$\frac{\partial L}{\partial \mathbf{x}(0)} = \mathbf{a}_{\mathbf{x}}(0) = -\int_1^0 \mathbf{a}_{\mathbf{x}}(t)^T \frac{\partial \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)}{\partial \mathbf{x}(t)} dt + \frac{\partial L}{\partial \mathbf{x}(1)}$$

- Think about the initial conditions.
- These equations are solved in the reverse time direction.
- Numerical solvers (Euler ODESolve) are used to solve them.

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

Adjoint method

Forward pass

$$\mathbf{x}(1) = \mathbf{x}(0) + \int_0^1 \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t) dt \quad \Rightarrow \quad \text{ODE Solver}$$

Backward pass

$$\left.\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{\theta}(0)} &= \mathbf{a}_{\boldsymbol{\theta}}(0) = -\int_1^0 \mathbf{a}_{\mathbf{x}}(t)^T \frac{\partial \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)}{\partial \boldsymbol{\theta}(t)} dt + 0 \\
\frac{\partial L}{\partial \mathbf{x}(0)} &= \mathbf{a}_{\mathbf{x}}(0) = -\int_1^0 \mathbf{a}_{\mathbf{x}}(t)^T \frac{\partial \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)}{\partial \mathbf{x}(t)} dt + \frac{\partial L}{\partial \mathbf{x}(1)} \\
\mathbf{x}(0) &= -\int_0^1 \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t) dt + \mathbf{x}(1).
\end{aligned}\right\} \Rightarrow \text{ODE Solver}$$

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

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

Summary

- Continuous-in-time NF uses neural ODE to define continuous dynamic $\mathbf{x}(t)$. It has less functional restrictions.
- Continuity equation allows to calculate $\log p(\mathbf{x}, t)$ at arbitrary moment t.
- FFJORD model makes such kind of NF scalable.
- Adjoint method are the continuous analog of backpropagation in the discrete time. Pontryagin theorem gives the way to compute the adjoint functions.
- Using numerical solvers it is possible to make forward and backward passes for the continuous-in-time NF.