Deep Generative Models

Lecture 9

Roman Isachenko

AI Masters

2024, Spring

Recap of previous lecture

f-divergence minimization

$$D_f(\pi||p) = \mathbb{E}_{p(\mathbf{x})} f\left(\frac{\pi(\mathbf{x})}{p(\mathbf{x})}\right) \to \min_p .$$

Here $f : \mathbb{R}_+ \to \mathbb{R}$ is a convex, lower semicontinuous function satisfying $f(1) = 0$.

Variational divergence estimation

$$D_f(\pi||p) \geq \sup_{T \in \mathcal{T}} \left[\mathbb{E}_\pi T(\mathbf{x}) - \mathbb{E}_p f^*(T(\mathbf{x}))\right],$$

Fenchel conjugate

$$f^*(t) = \sup_{u \in \text{dom}_f} (ut - f(u)), \quad f(u) = \sup_{t \in \text{dom}_{f*}} (ut - f^*(t))$$

Note: To evaluate lower bound we only need samples from $\pi(\mathbf{x})$ and $p(\mathbf{x})$. Hence, we could fit implicit generative model.

Nowozin S., Cseke B., Tomioka R. f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization, 2016

Recap of previous lecture

How to evaluate likelihood-free models?

$p(y|\mathbf{x})$ – pretrained image classification model (e.g. ImageNet classifier).

What do we want from samples?

- **Sharpness**

Low sharpness

High sharpness

  $p(y|\mathbf{x})$ has low entropy (each image $\mathbf{x}$ should have distinctly recognizable object).

- **Diversity**

Low diversity

High diversity

  $p(y) = \int p(y|\mathbf{x})p(\mathbf{x})d\mathbf{x}$ has high entropy (there should be as many classes generated as possible).

---

image credit: https://deepgenerativemodels.github.io

Recap of previous lecture

Let take some pretrained image classification model to get the conditional label distribution $p(y|\mathbf{x})$ (e.g. ImageNet classifier).

Evaluation of likelihood-free models

- Sharpness $\Rightarrow$ low $H(y|\mathbf{x}) = -\sum_y \int_{\mathbf{x}} p(y, \mathbf{x}) \log p(y|\mathbf{x}) d\mathbf{x}$.
- Diversity $\Rightarrow$ high $H(y) = -\sum_y p(y) \log p(y)$.

Inception Score

$$IS = \exp(H(y) - H(y|\mathbf{x})) = \exp\left(\mathbb{E}_{\mathbf{x}} KL(p(y|\mathbf{x}) || p(y))\right)$$

Frechet Inception Distance

$$D^2(\pi, p) = \|\mathbf{m}_\pi - \mathbf{m}_p\|_2^2 + \mathrm{Tr}\left(\boldsymbol{\Sigma}_\pi + \boldsymbol{\Sigma}_p - 2\sqrt{\boldsymbol{\Sigma}_\pi \boldsymbol{\Sigma}_p}\right).$$

FID is related to moment matching.

Salimans T. et al. Improved Techniques for Training GANs, 2016
Heusel M. et al. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, 2017

Recap of previous lecture

- $\mathcal{S}_\pi = \{\mathbf{x}_i\}_{i=1}^n \sim \pi(\mathbf{x})$ – real samples;
- $\mathcal{S}_p = \{\mathbf{x}_i\}_{i=1}^n \sim p(\mathbf{x}|\boldsymbol{\theta})$ – generated samples.

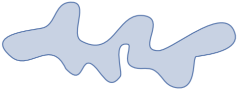
Embed samples using pretrained classifier network (as previously):

$$\mathcal{G}_\pi = \{\mathbf{g}_i\}_{i=1}^n, \quad \mathcal{G}_p = \{\mathbf{g}_i\}_{i=1}^n.$$

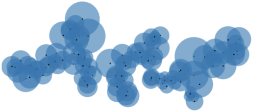
Define binary function:

$$f(\mathbf{g}, \mathcal{G}) = \begin{cases} 1, \text{if exists } \mathbf{g}' \in \mathcal{G} : \|\mathbf{g} - \mathbf{g}'\|_2 \leq \|\mathbf{g}' - \text{NN}_k(\mathbf{g}', \mathcal{G})\|_2; \\ 0, \text{otherwise.} \end{cases}$$

$$\text{Precision}(\mathcal{G}_\pi, \mathcal{G}_p) = \frac{1}{n} \sum_{\mathbf{g} \in \mathcal{G}_p} f(\mathbf{g}, \mathcal{G}_\pi); \quad \text{Recall}(\mathcal{G}_\pi, \mathcal{G}_p) = \frac{1}{n} \sum_{\mathbf{g} \in \mathcal{G}_\pi} f(\mathbf{g}, \mathcal{G}_p).$$



(a) True manifold



(b) Approx. manifold

Kynkäänniemi T. et al. Improved precision and recall metric for assessing generative models, 2019

Outline

1. Neural ODE
2. Adjoint method
3. Continuous-in-time Normalizing Flows

Outline

1. Neural ODE
2. Adjoint method
3. Continuous-in-time Normalizing Flows

Neural ODE

Consider Ordinary Differential Equation (ODE)

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

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

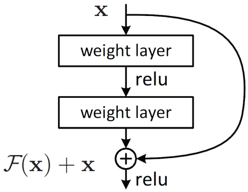
Euler update step

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

Residual block

$$\mathbf{z}_{t+1} = \mathbf{z}_t + f_{\boldsymbol{\theta}}(\mathbf{z}_t)$$

- It is equavalent to Euler update step for solving ODE with $\Delta t = 1$!
- Euler update step is unstable and trivial. There are more sophisticated methods.



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

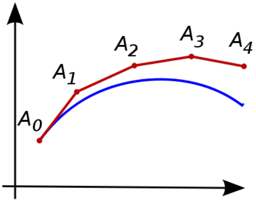
Neural ODE

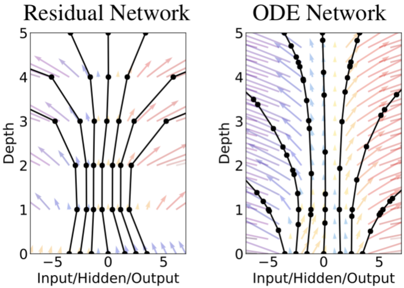
Residual block

$$\mathbf{z}_{t+1} = \mathbf{z}_t + f_\theta(\mathbf{z}_t).$$

In the limit of adding more layers and taking smaller steps, we parameterize the continuous dynamics of hidden units using an ODE specified by a neural network:

$$\frac{d\mathbf{z}(t)}{dt} = f_\theta(\mathbf{z}(t), t); \quad \mathbf{z}(t_0) = \mathbf{x}; \quad \mathbf{z}(t_1) = \mathbf{y}.$$





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

Neural ODE

Forward pass (loss function)

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

Note: ODESolve could be any method (Euler step, Runge-Kutta methods).

Backward pass (gradients computation)

For fitting parameters we need gradients:

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

In theory of optimal control these functions called **adjoint** functions. They show how the gradient of the loss depends on the hidden state $\mathbf{z}(t)$ and parameters $\boldsymbol{\theta}$.

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

Outline

1. Neural ODE

2. Adjoint method

3. Continuous-in-time Normalizing Flows

Adjoint method

Adjoint functions

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

Theorem (Pontryagin)

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

Do we know any initilal condition?

Solution for adjoint function

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

$$\frac{\partial L}{\partial \mathbf{z}(t_0)} = \mathbf{a}_{\mathbf{z}}(t_0) = -\int_{t_1}^{t_0} \mathbf{a}_{\mathbf{z}}(t)^T \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)} dt + \frac{\partial L}{\partial \mathbf{z}(t_1)}$$

Note: These equations are solved back in time.

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

Adjoint method

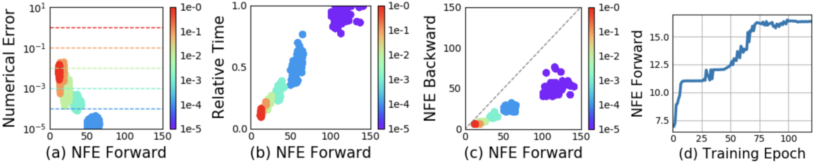
Forward pass

$$\mathbf{z}(t_1) = \int_{t_0}^{t_1} f_{\boldsymbol{\theta}}(\mathbf{z}(t), t)dt + \mathbf{z}_0 \quad \Rightarrow \quad \text{ODE Solver}$$

Backward pass

$$\left.\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{\theta}(t_0)} &= \mathbf{a}_{\boldsymbol{\theta}}(t_0) = -\int_{t_1}^{t_0} \mathbf{a}_{\mathbf{z}}(t)^T \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{z}(t), t)}{\partial \boldsymbol{\theta}(t)} dt + 0 \\
\frac{\partial L}{\partial \mathbf{z}(t_0)} &= \mathbf{a}_{\mathbf{z}}(t_0) = -\int_{t_1}^{t_0} \mathbf{a}_{\mathbf{z}}(t)^T \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)} dt + \frac{\partial L}{\partial \mathbf{z}(t_1)} \\
\mathbf{z}(t_0) &= -\int_{t_1}^{t_0} f_{\boldsymbol{\theta}}(\mathbf{z}(t), t)dt + \mathbf{z}_1.
\end{aligned}\right\} \Rightarrow \text{ODE Solver}$$

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



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

Outline

1. Neural ODE
2. Adjoint method
3. Continuous-in-time Normalizing Flows

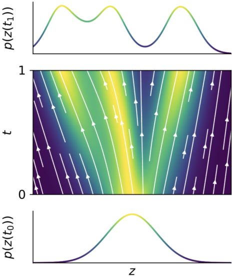
Continuous-in-time Normalizing Flows

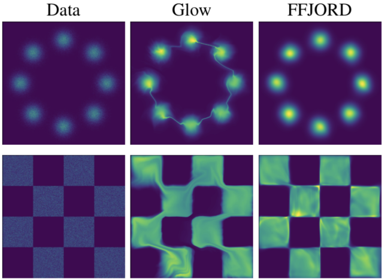
Discrete-in-time NF

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

Continuous-in-time dynamics

$$\frac{d\mathbf{z}(t)}{dt} = f_{\boldsymbol{\theta}}(\mathbf{z}(t), t).$$





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

Continuous-in-time Normalizing Flows

Theorem (Picard)

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

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

- Discrete-in-time NF need invertible f. Here we have sequence of $\log p(\mathbf{z}_t)$.
- Continuous-in-time NF require only smoothness of f. Here we need to get $\log(p(\mathbf{z}(t), t))$

Forward and inverse transforms

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

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

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

Continuous-in-time Normalizing Flows

Theorem (Kolmogorov-Fokker-Planck: special case)

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

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

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

Here $p(\mathbf{x}|\boldsymbol{\theta}) = p(\mathbf{z}(t_1), t_1)$, $p(\mathbf{z}) = p(\mathbf{z}(t_0), t_0)$. **Adjoint** method is used for getting the derivatives.

Forward transform + log-density

$$\begin{bmatrix} \mathbf{x} \\ \log p(\mathbf{x}|\boldsymbol{\theta}) \end{bmatrix} = \begin{bmatrix} \mathbf{z} \\ \log p(\mathbf{z}) \end{bmatrix} + \int_{t_0}^{t_1} \begin{bmatrix} f_{\boldsymbol{\theta}}(\mathbf{z}(t), t) \\ -\mathrm{tr}\left(\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)}\right) \end{bmatrix} dt.$$

It costs $O(m^2)$ to get the trace of the Jacobian (evaluation of determinant of the Jacobian costs $O(m^3)$!).

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

Continuous-in-time Normalizing Flows

- $\text{tr}\left(\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{z}(t))}{\partial \mathbf{z}(t)}\right)$ costs $O(m^2)$ (m evaluations of f), since we have to compute a derivative for each diagonal element.
- Jacobian vector products $\mathbf{v}^T \frac{\partial f}{\partial \mathbf{z}}$ can be computed for approximately the same cost as evaluating f.

It is possible to reduce cost from $O(m^2)$ to $O(m)$!

Hutchinson's trace estimator

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

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

FFJORD density estimation

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

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

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

Summary

- Residual networks could be interpreted as solution of ODE with Euler method.
- Adjoint method generalizes backpropagation procedure and allows to train Neural ODE solving ODE for adjoint function back in time.
- Kolmogorov-Fokker-Planck theorem allows to construct continuous-in-time normalizing flow with less functional restrictions.
- FFJORD model makes such kind of NF scalable.