Diffusion Models Chapter 0: Neural ODE and Continuous-Depth Flow Models

Generative AI and Foundation Models

Spring 2024

Department of Mathematical Sciences

Ernest K. Ryu

Seoul National University

Depth-L residual network

Consider the depth-L residual network

$$\begin{aligned} h_\theta(X) &= z_L \\ z_L &= z_{L-1} + f(z_{L-1}, \theta, L-1) \\ &\vdots \\ z_2 &= z_1 + f(z_1, \theta, 1) \\ z_1 &= z_0 + f(z_0, \theta, 0) \\ z_0 &= X \end{aligned}$$

where $z_0, \dots, z_L \in \mathbb{R}^D$, $\theta \in \mathbb{R}^P$, and $f : \mathbb{R}^D \times \mathbb{R}^P \times \mathbb{N} \to \mathbb{R}^D$.

Note that the trainable parameter θ is shared across all layers.

Loss function

Consider the loss function

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} l(h_\theta(X_i), Y_i)$$

For simplicity, assume $N = 1$ and write

$$\mathcal{L}(\theta) = l(h_\theta(X), Y)$$

So $\mathcal{L}$ is the output scalar loss value.

Neural ODE

The Neural ODE is a continuous-depth (or infinite-depth) analog:

$$\begin{aligned} h_\theta(X) &= z(1) \\ \dot{z}(s) &= f\left(z(s), \theta, s\right) \qquad \text{for } s \in [0,1] \\ z(0) &= X, \end{aligned}$$

where $z(s) \in \mathbb{R}^D$ for $s \in [0,1]$, $\theta \in \mathbb{R}^D$, and $f : \mathbb{R}^D \times \mathbb{R}^P \times [0,1] \to \mathbb{R}^D$. We refer to s as *pseudo-time*. Assume f is continuous in (z, θ, s) and continuously differentiable in (z, θ). We will represent $f(z, \theta, s)$ as a neural network.

We say $\{z(s)\}_{s \in [0,1]}$ is a solution to this ODE if

$$z(s) = X + \int_0^s f\left(z(s'), s', \theta\right)\, ds', \qquad s \in [0,1].$$

R. T. Q. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud, Neural ordinary differential equations, *NeurIPS*, 2018.

Forward-solve of Neural ODE

Option 1. Implement a simpler Euler discretization

set Δs

for $k = 0, \dots, \left\lfloor \frac{1}{\Delta s} \right\rfloor - 1$

$z_{k+1} = z_k + \Delta s \, f(z_k, \theta, k\Delta s)$

endfor

return $z_{\left\lfloor \frac{1}{\Delta s} \right\rfloor}$

Option 2. Call an ODE solver. (Often the better option, since ODE solvers are quite sohpisticated and can solve the ODE to high accuracy.)

Loss function for Neural ODE

Generally, Consider the loss function

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} l(h_\theta(X_i), Y_i)$$

where $h_\theta(X_i)$ is the solution to the ODE at pseudo-time $s = 1$ with initial condition $z(0) = X_i$ at pseudo-time $s = 0$.

For simplicity, assume $N = 1$ and write

$$\mathcal{L} = l(h_\theta(X), Y) = l(z(1), Y)$$

So $\mathcal{L}$ is the output scalar loss value.

Backprop warmup

$$
\begin{aligned}
\mathcal{L} &= l(h_\theta(X), Y) \\
h_\theta(X) &= z_L \\
z_L &= z_{L-1} + f(z_{L-1}, \theta, L-1) \\
&\vdots \\
z_2 &= z_1 + f(z_1, \theta, 1) \\
z_1 &= z_0 + f(z_0, \theta, 0) \\
z_0 &= X
\end{aligned}
$$

As a warmup exercise, let's work out backpropagation of the discrete-depth ResNet.

Assume the forward pass has been performed, i.e., $z_1, \dots, z_L$ have been sequentially computed and their values been stored in memory. For notational simplicity,

$$
a_l = \frac{\partial \mathcal{L}}{\partial z_l} = \frac{\partial \mathcal{L}}{\partial z_L} \frac{\partial z_L}{\partial z_{L-1}} \cdots \frac{\partial z_{l+2}}{\partial z_{l+1}} \frac{\partial z_{l+1}}{\partial z_l}, \qquad l = 0, \dots, L.
$$

Then,

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial z_L} \frac{\partial z_L}{\partial \theta} &= a_L \left(\frac{\partial f}{\partial \theta}(z_{L-1}, \theta, L-1) + \frac{\partial f}{\partial z_{L-1}}(z_{L-1}, \theta, L-1) \frac{\partial z_{L-1}}{\partial \theta} + \frac{\partial z_{L-1}}{\partial \theta} \right) \\
&= a_L \frac{\partial f}{\partial \theta}(z_{L-1}, \theta, L-1) + a_L \left(\frac{\partial f}{\partial z_{L-1}}(z_{L-1}, \theta, L-1) + I \right) \frac{\partial z_{L-1}}{\partial \theta} \\
&= a_L \frac{\partial f}{\partial \theta}(z_{L-1}, \theta, L-1) + a_L \frac{\partial z_L}{\partial z_{L-1}} \frac{\partial z_{L-1}}{\partial \theta} \\
&= a_L \frac{\partial f}{\partial \theta}(z_{L-1}, \theta, L-1) + a_{L-1} \frac{\partial z_{L-1}}{\partial \theta} \\
&= a_L \frac{\partial f}{\partial \theta}(z_{L-1}, \theta, L-1) + a_{L-1} \frac{\partial f}{\partial \theta}(z_{L-2}, \theta, L-2) + a_{L-2} \frac{\partial z_{L-2}}{\partial \theta} \\
&= \sum_{l=1}^{L} a_l \frac{\partial f}{\partial \theta}(z_{l-1}, \theta, l-1).
\end{aligned}
$$

Backprop warmup

The formula can be implemented in a backward for loop:

$a_L = \frac{\partial \mathcal{L}}{\partial z_L}$

$g = 0$

for $\ell = L, L-2, \dots, 1$

  $g \mathrel{+}= a_\ell \frac{\partial f(z_{\ell-1}, \theta, \ell-1)}{\partial \theta}$

  $a_{\ell-1} = a_\ell \frac{\partial z_\ell}{\partial z_{\ell-1}}$

endfor

return g

Backpropagation for neural ODE

We start with a warmup exercise. The full derivation of the continuous-depth backprop will be carried out soon. For $s, t \in [0,1]$, define the *flow operator* (also called the time evolution operator) $\mathcal{F}^{s,t} : \mathbb{R}^D \to \mathbb{R}^D$ as

$$\begin{aligned} \mathcal{F}^{s,t}(z) &= z(t) \\ \dot{z}(s') &= f\left(z(s'), \theta, s'\right) \qquad \text{for } s' \in [s,t] \\ z(s) &= z. \end{aligned}$$

Then

$$z(1) = \mathcal{F}^{0,1}(X) = \mathcal{F}^{s,1}(\mathcal{F}^{0,s}(X))$$

for any $s \in [0,1]$.

Forward and backward flow operator

The flow operator can evolve the initial condition forward in pseudo-time ($t > s$) and also backwards in pseudo-time ($t < s$) since the ODE can be solved both forwards and backwards in pseudo-time.

In fact, if $z(1)$ is known, then the initial condition $z(0) = \mathcal{F}^{1,0}(z(1))$ can be recovered through solving the ODE

$$\dot{z}(s) = f\left(z(s), \theta, s\right) \qquad \text{for } s \in [0,1]$$
$$z(1) \qquad \text{“initial” condition.}$$

Obtaining $z(0)$ from $z(1)$ is no more difficult than Obtaining $z(1)$ from $z(0)$. This wasn't the case for discrete-depth ResNet; knowing z_L does not allow one to recover z_0 or z_{L-1}.

Integral form of the flow operator

Both when $s < t$ and $s > t$, we have the integral form

$$z(t) = \mathcal{F}^{s,t}(z(s))$$

$$z(t) = z(s) + \int_s^t \dot{z}(s')\, ds'$$

Backprop for Neural ODE: Warmup

Define

$$\frac{\partial \mathcal{L}}{\partial z(s)} = \big(D(\mathcal{L} \circ \mathcal{F}^{s,1})\big)(z(s)) = \frac{\partial \mathcal{L}(\mathcal{F}^{s,1}(z))}{\partial z}\bigg|_{z=z(s)} \quad \text{and} \quad \frac{\partial z(t)}{\partial z(s)} = \big(D(\mathcal{F}^{s,t})\big)(z(s)) = \frac{\partial \mathcal{F}^{s,t}(z)}{\partial z}\bigg|_{z=z(s)}$$

for $s, t \in [0,1]$. Then, we have the chain rule

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial z(s)} = \big(D(\mathcal{L} \circ \mathcal{F}^{s,1})\big)(z(s)) &= \big(D(\mathcal{L} \circ \mathcal{F}^{t,1} \circ \mathcal{F}^{s,t})\big)(z(s)) \\
&= \big(D(\mathcal{L} \circ \mathcal{F}^{t,1})(z(t))\big)\big(D(\mathcal{F}^{s,t})\big)(z(s)) \quad \text{(matrix-matrix product)} \\
&= \frac{\partial \mathcal{L}}{\partial z(t)} \frac{\partial z(t)}{\partial z(s)} \quad \text{(matrix-matrix product)}
\end{aligned}$$

for $s, t \in [0,1]$. So $\frac{\partial \mathcal{L}}{\partial z(s)}$ represents the infinitesimal change in $\mathcal{L}$ if the neural ODE started at pseudo-time s with initial value $z(s) + \delta$, where δ is an infinitesimal perturbation.

Backprop for Neural ODE: Warmup

Let

$$a(s) = \frac{\partial \mathcal{L}}{\partial z(s)} \in \mathbb{R}^{1\times D}, \qquad s \in [0,1].$$

Then

$$\dot{a}(s) = -a(s)\frac{\partial f}{\partial z}(z(s),\theta,s), \qquad s\in[0,1]$$
$$a(1) = \frac{\partial \mathcal{L}}{\partial z(1)}$$

and $\{a(s)\}_{s\in[0,1]}$ can be solved by solving the ODE backwards in pseudo-time. We provide an ODE solver with “initial condition” $a(1) = \frac{\partial \mathcal{L}}{\partial z(1)}$ and solves for $\{a(s)\}_{s\in[0,1]}$. We then return

$$\frac{\partial \mathcal{L}}{\partial X} = \frac{\partial \mathcal{L}}{\partial z(0)} = a(0)$$

We now show

$$\dot a(s) = -a(s)\frac{\partial f}{\partial z}(z(s),\theta,s), \qquad s\in[0,1]$$
$$a(1) = \frac{\partial \mathcal{L}}{\partial z(1)}$$

Proof) This follows from

$$h_\theta(X) = z(1)$$
$$\dot z(s) = f\left(z(s),\theta,s\right) \qquad \text{for } s\in[0,1]$$
$$z(0) = X,$$

$$\dot a(s) = \lim_{\varepsilon\to 0}\frac{a(s+\varepsilon)-a(s)}{\varepsilon} \overset{(i)}{=} \lim_{\varepsilon\to 0}\frac{a(s+\varepsilon)}{\varepsilon}\left(I-\frac{\partial z(s+\varepsilon)}{\partial z(s)}\right)$$
$$= \lim_{\varepsilon\to 0}\frac{a(s+\varepsilon)}{\varepsilon}\left(I-\frac{\partial}{\partial z(s)}\left(z(s)+\int_s^{s+\varepsilon} f(z(s'),\theta,s')\,ds'\right)\right) \overset{(ii)}{=} -\lim_{\varepsilon\to 0}\left(a(s+\varepsilon)\frac{\partial f(z(s),\theta,s)}{\partial z(s)}+\mathcal{O}(\varepsilon)\right)$$
$$= -\underbrace{a(s)}_{1\times D}\underbrace{\frac{\partial f}{\partial z}(z(s),\theta,s)}_{D\times D}.$$

$$(i)\colon a(s) = \frac{\partial \mathcal{L}}{\partial z(s)} = \frac{\partial \mathcal{L}}{\partial z(s+\varepsilon)}\frac{\partial z(s+\varepsilon)}{\partial z(s)} = a(s+\varepsilon)\frac{\partial z(s+\varepsilon)}{\partial z(s)}$$
$$(ii)\colon \int_s^{s+\epsilon} f(z(s'),\theta,s')ds' = \varepsilon f(z(s),\theta,s) + \text{h.o.t.}$$

■

Backprop for Neural ODE

Ultimately, we want $\frac{\partial \mathcal{L}}{\partial \theta}$. (Previous derivation was for $\frac{\partial \mathcal{L}}{\partial X}$.) However, infinitesimal changes of θ to $\theta + \delta$ affects the update via

$$z(s+\epsilon) \approx z(s) + \epsilon f(z(s), \theta + \delta, s)$$

and making sense of this precisely and correctly is tricky.

Therefore, we employ a technique of converting θ into an initial condition (rather than a parameter) of an augmented ODE.

Backprop for Neural ODE

Theorem. (Adjoint state method) Consider the neural ODE

$$\dot{z}(s) = f(z(s), \theta, s), \qquad \text{for } s \in [0,1]$$

with initial condition $z(0)$. Assume $\{z(s)\}_{s\in[0,1]}$ has been solved in a "forward pass". Let $\mathcal{L} : \mathbb{R}^D \to \mathbb{R}$ be loss function depending on $z(1)$. The solution to the ODE

$$\begin{aligned}
\dot{a}(s) &= -a(s)\frac{\partial f}{\partial z}(z(s), \theta, s), && \text{for } s \in [0,1] \\
\dot{b}(s) &= -a(s)\frac{\partial f}{\partial \theta}(z(s), \theta, s), && \text{for } s \in [0,1] \\
a(1) &= \frac{\partial \mathcal{L}}{\partial z(1)} \in \mathbb{R}^{1\times D} \\
b(1) &= 0 \in \mathbb{R}^{1\times P}
\end{aligned}$$

yields $\frac{\partial \mathcal{L}}{\partial \theta} = b(0)$.

Proof) Augment the ODE as follows:

$$
\begin{aligned}
\dot{z}(s) &= f(z(s), \varphi(s), s), && \text{for } s \in [0,1] \\
\dot{\varphi}(s) &= 0, && \text{for } s \in [0,1] \\
z(0) &= X \\
\varphi(0) &= \theta.
\end{aligned}
$$

Define the augmented notation

$$
z_{\text{aug}}(s) = \begin{bmatrix} z(s) \\ \varphi(s) \end{bmatrix} \in \mathbb{R}^{D+P}
$$

$$
f_{\text{aug}}(z_{\text{aug}}(s), s) = \begin{bmatrix} f(z(s), \varphi(s), s) \\ 0 \end{bmatrix} \in \mathbb{R}^{D+P}.
$$

Then

$$
\begin{aligned}
\dot{z}_{\text{aug}}(s) &= f_{\text{aug}}(z_{\text{aug}}(s), s), && \text{for } s \in [0,1] \\
z_{\text{aug}}(0) &= \begin{bmatrix} X \\ \theta \end{bmatrix}.
\end{aligned}
$$

For $s, t \in [0,1]$, define the augmented flow operator $\mathcal{F}_{aug}^{s,t} : \mathbb{R}^{D+P} \to \mathbb{R}^{D+P}$ as

$$\begin{aligned}
\mathcal{F}_{\mathrm{aug}}^{s,t}(z,\varphi) &= (z(t), \varphi(t)) \\
\dot{z}_{\mathrm{aug}}(s') &= f_{\mathrm{aug}}(z_{\mathrm{aug}}(s'), s'), \qquad \text{for } s' \in [s,t] \\
z_{\mathrm{aug}}(s) &= \begin{bmatrix} z \\ \varphi \end{bmatrix}.
\end{aligned}$$

Then define

$$\begin{aligned}
a_{\mathrm{aug}}(s) = \frac{\partial \mathcal{L}}{\partial z_{\mathrm{aug}}(s)} &= \left.\frac{\partial \mathcal{L}(\mathcal{F}_{\mathrm{aug}}^{s,1}(z_{\mathrm{aug}}))}{\partial z_{\mathrm{aug}}}\right|_{z_{\mathrm{aug}}=z_{\mathrm{aug}}(s)} \in \mathbb{R}^{1\times(D+P)} \\
&\overset{\text{(i)}}{=} \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z(s)} & \frac{\partial \mathcal{L}}{\partial \varphi(s)} \end{bmatrix} \\
&\overset{\text{(ii)}}{=} \begin{bmatrix} a(s) & b(s) \end{bmatrix},
\end{aligned}$$

where (i) defines $\frac{\partial \mathcal{L}}{\partial z(s)}$ and $\frac{\partial \mathcal{L}}{\partial \varphi(s)}$ and (ii) defines $a(s)$ and $b(s)$.

In other words,

$$a(s) = \frac{\partial \mathcal{L}}{\partial z(s)} = \left. \frac{\partial \mathcal{L}(\mathcal{F}^{s,1}_{\text{aug}}(z,\varphi))}{\partial z} \right|_{\substack{z=z(s)\\ \varphi=\varphi(s)}} \quad \text{and} \quad b(s) = \frac{\partial \mathcal{L}}{\partial \varphi(s)} = \left. \frac{\partial \mathcal{L}(\mathcal{F}^{s,1}_{\text{aug}}(z,\varphi))}{\partial \varphi} \right|_{\substack{z=z(s)\\ \varphi=\varphi(s)}}.$$

The meaning of $a(s) = \frac{\partial \mathcal{L}}{\partial z(s)}$ is the same as what we saw in the warmup derivation.

The meaning of $b(s) = \frac{\partial \mathcal{L}}{\partial \varphi(s)}$ is the infinitesimal change in $\mathcal{L}$ if the neural ODE started at pseudo-time s with initial value $z(s)$ and parameter $\theta + \delta$, where δ is an infinitesimal perturbation. Since $\mathcal{L}$ ultimately only depends on $z(1)$, we have $\frac{\partial \mathcal{L}}{\partial \varphi(1)} = 0$. The gradient we wish to obtain is $\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial \varphi(0)}$.

By the same reasoning as before, we have

$$\dot{a}_{\text{aug}}(s) = -a_{\text{aug}}(s) \frac{\partial f_{\text{aug}}}{\partial z_{\text{aug}}}(z_{\text{aug}}(s), s) = -\begin{bmatrix} a(s) & b(s) \end{bmatrix} \begin{bmatrix} \frac{\partial f}{\partial z}(z(s), \varphi(s), s) & \frac{\partial f}{\partial \theta}(z(s), \varphi(s), s) \\ 0 & 0 \end{bmatrix}.$$

Multiplying out this leads to the stated result. ■

Backprop for Neural ODE v.1

Finally, we are ready to describe the algorithm to perform backpropagation with the neural ODE.

1. With initial condition $z(0)$, call an ODE solver to compute and store $\{z(s)\}_{s=0}^{1}$.

2. With initial condition $a(1) = \frac{\partial L}{\partial z(1)}$, and $b(1) = 0$, call an ODE solver (backwards in pseudo-time) to compute $\big(a(0), b(0)\big)$. Return $b(0) = \frac{\partial \mathcal{L}}{\partial \theta}$.

$$\dot{a}(s) = -a(s)\frac{\partial f}{\partial z}(z(s), \theta, s)$$

$$\dot{b}(s) = -a(s)\frac{\partial f}{\partial \theta}(z(s), \theta, s)$$

Note that step 2 uses with $\{z(s)\}_{s\in[0,1]}$ computed from step 1. However, storing the entire trajectory $\{z(s)\}_{s\in[0,1]}$ can be inefficient in terms of memory usage.

Backprop for Neural ODE v.2

A more efficient backprop method for

1. With initial condition $z(0)$, call an ODE solver to compute and store $z(1)$.

2. With initial condition $z(1)$, $a(1) = \frac{\partial L}{\partial z(1)}$, and $b(1) = 0$, call an ODE solver (backwards in pseudo-time) to compute $\big(z(0), a(0), b(0)\big)$. Return $b(0) = \frac{\partial L}{\partial \theta}$.

$$\dot{z}(s) = f\left(z(s), \theta, s\right)$$

$$\dot{a}(s) = -a(s)\frac{\partial f}{\partial z}(z(s), \theta, s)$$

$$\dot{b}(s) = -a(s)\frac{\partial f}{\partial \theta}(z(s), \theta, s)$$

Recomputing $\{z(s)\}_{s\in[0,1]}$ anew from $z(1)$ together with the computation of $\{a(s)\}_{s\in[0,1]}$ and $\{b(s)\}_{s\in[0,1]}$ is much more memory efficient, although it does require slightly more computation.

ODE solver uses autograd(backprop)

To solve,

$$\dot{z}(s) = f\left(z(s), \theta, s\right)$$

$$\dot{a}(s) = -a(s)\frac{\partial f}{\partial z}(z(s), \theta, s)$$

$$\dot{b}(s) = -a(s)\frac{\partial f}{\partial \theta}(z(s), \theta, s)$$

The ODE solver is provided with functions that can evaluate $f\left(z(s), \theta, s\right)$, $-a(s)\frac{\partial f}{\partial z}(z(s), \theta, s)$ and $-a(s)\frac{\partial f}{\partial \theta}(z(s), \theta, s)$.

The evaluation of $-a(s)\frac{\partial f}{\partial z}(z(s), \theta, s)$ and $-a(s)\frac{\partial f}{\partial \theta}(z(s), \theta, s)$ themselves requires the use of autograd or backprop, since they are derivatives. Since backprop requires a scalar output, we use

$$-a(s)\frac{\partial f}{\partial z}(z(s), \theta, s) = \frac{\partial}{\partial z}\big(-a(s)f(z(s), \theta, s)\big)$$

$$-a(s)\frac{\partial f}{\partial \theta}(z(s), \theta, s) = \frac{\partial}{\partial \theta}\big(-a(s)f(z(s), \theta, s)\big)$$

Flow models

A flow model is a generative model with samples $X = h_\theta(Z)$ with $Z \sim p_Z$, where the "prior distribution", often a simple IID Gaussian vector. Crucially, h_θ is invertible.

Sampling requires evaluation of $\boldsymbol{h_\theta}$.

Training is done via maximum likelihood on X. Therefore, we must be able to compute $\log p_\theta^{(\mathrm{gen})}(X)$ and its stochastic gradients efficiently.

Training requires evaluation of $\boldsymbol{h_\theta^{-1}}$.

FFJORD: Flow model with Neural ODE

Free-form Jacobian of Reversible Dynamics (FFJORD) samples X with

$$\begin{aligned} X &= z(1) \\ \dot{z}(s) &= f\left(z(s), \theta, s\right) \qquad \text{for } s \in [0,1] \\ z(0) &= Z \sim p_Z, \end{aligned}$$

Once trained, i.e., once θ is fixed, sample $X \sim p_\theta^{(\mathrm{gen})}$ by:

1. Sample $Z \sim p_Z$.
2. Call an ODE solver with initial condition Z. (So $X = h_\theta(Z) = \mathcal{F}^{0,1}(Z)$.)

W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, FFJORD: Free-form continuous dynamics for scalable reversible generative models, *ICLR*, 2019.

FFJORD: Flow model with Neural ODE

Given data $X_1, \dots, X_N$, FFJORD is trained by solving the maximum likelihood estimation problem (equivalently, minimizing the sum of negative log likelihoods):

$$\underset{\theta}{\text{minimize}} \quad \mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log p_\theta^{(\text{gen})}(X_i)$$

$$\begin{aligned} X &= z(1) \\ \dot{z}(s) &= f\left(z(s), \theta, s\right) \\ z(0) &= Z \sim p_Z, \end{aligned}$$

Training requires stochastic gradients, unbiased estimates of

$$\nabla_\theta \log p_\theta^{(\text{gen})}(X) = \nabla_\theta \log p_1(z(1))$$

FFJORD: Flow model with Neural ODE

Theorem. Let p_s be the density function of $z(s)$. Then,

$$\frac{d}{ds}\log p_s(z(s)) = -(\nabla_z \cdot f)(z(s), \theta, s), \qquad s \in [0, 1]$$

The solution also has an integral form

$$\log p_1(z(1)) = \log p_0(z(0)) - \int_0^1 (\nabla_z \cdot f)(z(s), \theta, s)\, ds$$

(Note, the integrand does not involve $p_s\big(z(s)\big)$.)

The other statements follow the fundamental theorem of calculus.

Proof of Theorem) We now show

$$\frac{d}{ds}\log p_s(z(s)) = -\mathrm{Tr}\left(\frac{\partial f}{\partial z}(z(s),\theta,s)\right), \qquad s \in [0,1]$$

The proof will utilize Jacobi's formula and the change of variables formula for random variables.

Jacobi's formula) Let A be an $n \times n$ matrix with eigenvalues $\lambda_1, \dots, \lambda_n$. Consider the limit $\varepsilon \to 0$. Then

$$|I + \varepsilon A| = \prod_{i=1}^{n}(1 + \varepsilon\lambda_i) = 1 + \varepsilon\sum_{i=1}^{n}\lambda_i + O(\varepsilon^2)$$

Therefore,

$$\lim_{\varepsilon\to 0}\frac{\partial}{\partial\varepsilon}|I + \varepsilon A| = \mathrm{Tr}(A)$$

Actually, our Jacobi's formula is a special case of the more general Jacobi's formula.

Change of variables for RV

Let $h : \mathbb{R}^n \to \mathbb{R}^n$ be an invertible function such that both h and h^{-1} are differentiable. Let Z be a continuous random variable with probability density function p_Z and let $X = h(Z)$ have density p_X. Then

$$p_X(x) = p_Z(z) \left| \frac{\partial h^{-1}}{\partial x}(x) \right| = \frac{p_Z(z)}{\left| \frac{\partial h}{\partial z}(z) \right|}$$

where $x = h(z)$.

Invertibility of h is essential; it is not a minor technical issue.

Proof of $\frac{d}{ds}\log p_s(z(s)) = -\mathrm{Tr}\left(\frac{\partial f}{\partial z}(z(s),\theta,s)\right)$

(i) Change of variables formula with $z(t+\varepsilon) = \mathcal{F}^{t,t+\varepsilon}\big(z(t)\big)$

(ii) L'Hôpital's rule

(iii) Jacobi’s formula

$$
\begin{aligned}
\frac{d}{ds}\log p_s(z(s)) &= \lim_{\varepsilon\to 0}\frac{\log p_{s+\varepsilon}(z(s+\varepsilon)) - \log p_s(z(s))}{\varepsilon} \\
&\overset{\text{(i)}}{=} \lim_{\varepsilon\to 0}\frac{\log p_s(z(s)) - \log\left|\frac{\partial\mathcal{F}^{s,s+\varepsilon}}{\partial z}(z(s))\right| - \log p_s(z(s))}{\varepsilon} \\
&\overset{\text{(ii)}}{=} \lim_{\varepsilon\to 0} -\frac{\partial}{\partial\varepsilon}\log\left|\frac{\partial\mathcal{F}^{s,s+\varepsilon}}{\partial z}(z(s))\right| \\
&= \lim_{\varepsilon\to 0} -\frac{\frac{\partial}{\partial\varepsilon}\left|\frac{\partial\mathcal{F}^{s,s+\varepsilon}}{\partial z}(z(s))\right|}{\left|\frac{\partial\mathcal{F}^{s,s+\varepsilon}}{\partial z}(z(s))\right|} \\
&= -\lim_{\varepsilon\to 0}\frac{\partial}{\partial\varepsilon}\left|\frac{\partial\mathcal{F}^{s,s+\varepsilon}}{\partial z}(z(s))\right| \\
&= -\lim_{\varepsilon\to 0}\frac{\partial}{\partial\varepsilon}\left|\frac{\partial}{\partial z}\left((z(s)) + \int_s^{s+\varepsilon} f(z(s'),\theta,s')\,ds'\right)\right| \\
&= -\lim_{\varepsilon\to 0}\frac{\partial}{\partial\varepsilon}\left|\frac{\partial}{\partial z}z(s) + \varepsilon\frac{\partial f}{\partial z}(z(s),\theta,s) + \mathcal{O}(\varepsilon^2)\right| \\
&\overset{\text{(iii)}}{=} -\mathrm{Tr}\left(\frac{\partial f}{\partial z}\right) = -\nabla_z\cdot f(z(s),\theta,s)
\end{aligned}
$$

■

FFJORD: Flow model with Neural ODE

Corollary. We get $\log p_X(X) = \log p_1\big(z(1)\big)$ by solving the forward pseudo-time ODE

$$\begin{aligned}\log p_\theta^{(\text{gen})}(X) &= \ell(1)\\ \dot{\ell}(s) &= -(\nabla_z \cdot f)(z(s), \theta, s), \qquad s \in [0,1]\\ \ell(0) &= \log p_0(z(0)) = \log p_Z(z(0))\end{aligned}$$

or the backward pseudo-time ODE

$$\begin{aligned}\log p_\theta^{(\text{gen})}(X) &= \log p_0(z(0)) - \ell(0) = \log p_Z(z(0)) - \ell(0)\\ \dot{\ell}(s) &= -(\nabla_z \cdot f)(z(s), \theta, s), \qquad s \in [0,1]\\ \ell(1) &= 0\end{aligned}$$

where $\nabla \cdot$ denotes the divergence. (Recall, $p_Z\big(z(0)\big) = p_0\big(z(0)\big)$.)

Exact log-likelihood computation v.1

The following is an exact log-likelihood computation for FFJORD:

1. Given a data X, solve the ODE $\dot{z}(s) = f(z(s), \theta, s)$ in <u>reverse</u> pseudo-time with initial condition $z(1) = X$ to obtain $\{z(s)\}_{s\in[0,1]}$.
2. Given $\{z(s)\}_{s\in[0,1]}$, solve the ODE $\dot{\ell}(s) = -(\nabla_z \cdot f)(z(s), \theta, s)$ in <u>reverse</u> pseudo-time with initial condition $\ell(1) = 0$ to obtain $\log p_\theta^{(\mathrm{gen})}(X) = \log p_Z\big(z(0)\big) - \ell(0)$.

Problem: We must store $\{z(s)\}_{s\in[0,1]}$, which is memory inefficient.

Exact log-likelihood computation v.2

Improvement: Just solve for $z(s)$ and $\ell(s)$ together by using an ODE solver in reverse pseudo-time.

$$\begin{aligned} \log p_\theta^{(\mathrm{gen})}(X) &= \log p_Z(z(0)) - \ell(0) \\ \frac{d}{ds}\begin{bmatrix} z(s) \\ \ell(s) \end{bmatrix} &= \begin{bmatrix} f\left(z(s), \theta, s\right) \\ -(\nabla_z \cdot f)(z(s), \theta, s) \end{bmatrix} \qquad \text{for } s \in [0,1] \\ \begin{bmatrix} z(1) \\ \ell(1) \end{bmatrix} &= \begin{bmatrix} X \\ 0 \end{bmatrix} \end{aligned}$$

Problem: $\mathrm{Tr}\left(\frac{\partial f}{\partial z}\right) = \nabla \cdot f$ requires D backprop calls to evaluate, when $f(z(s), \theta, s) \in \mathbb{R}^D$ and $z \in \mathbb{R}^D$. We want to avoid computing divergences.

Background: Hutchinson's trace estimator

Let $\nu \in \mathbb{R}^D$ be a random vector such that

$$\mathbb{E}_\nu[\nu_i \nu_j] = \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

i.e., $\mathbb{E}_\nu[\nu\nu^\intercal] = I \in \mathbb{R}^{D\times D}$.

One example is $\nu_1, \ldots, \nu_D \sim \mathcal{N}(0,1)$ IID Gaussian.

Another example is $\nu_1, \ldots, \nu_D$ drawn as IID Rademacher (1/2 Bernoulli) random variables. In what follows.

M. F. Hutchinson, A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines, *Communications in Statistics - Simulation and Computation*, 1990.

Background: Hutchinson's trace estimator

Let $A \in \mathbb{R}^{D\times D}$. Then

$$\begin{aligned}
\mathbb{E}_\nu[\nu^\intercal A\nu] &= \mathbb{E}_\nu[\mathrm{Tr}(\nu^\intercal A\nu)] \\
&= \mathbb{E}_\nu[\mathrm{Tr}(A\nu\nu^\intercal)] \\
&= \mathrm{Tr}\left(\mathbb{E}_\nu[A\nu\nu^\intercal]\right) \\
&= \mathrm{Tr}\left(A\mathbb{E}_\nu[\nu\nu^\intercal]\right) \\
&= \mathrm{Tr}\left(AI\right) \\
&= \mathrm{Tr}(A)
\end{aligned}$$

So

$$\mathrm{Tr}(A) = \mathbb{E}_\nu[\nu^\intercal A\nu]$$

and $\nu^\top A\nu$ serves as an unbiased estimator of $\mathrm{Tr}(A)$.

M. F. Hutchinson, A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines, *Communications in Statistics - Simulation and Computation*, 1990.

Stochastic estimate of log-likelihood

We can express the likelihood as

$$\begin{aligned}\log p_\theta^{(\mathrm{gen})}(X) &= \log p_Z(z(0)) + \int_0^1 -(\nabla_z \cdot f)(z(s),\theta,s)\, ds \\ &= \log p_Z(z(0)) + \int_0^1 -\mathrm{Tr}\left(\frac{\partial f}{\partial z}(z(s),\theta,s)\right)\, ds \\ &= \log p_Z(z(0)) + \int_0^1 -\mathbb{E}_\nu\left[\nu^\intercal \frac{\partial f}{\partial z}\nu\right]\, ds \\ &= \log p_Z(z(0)) + \mathbb{E}_\nu\left[\int_0^1 -\nu^\intercal \frac{\partial f}{\partial z}\nu\, ds\right]\end{aligned}$$

and we can an unbiased estimate of the likelihood.

$$\log \widehat{p_\theta^{(\mathrm{gen})}}(X) = \log p_Z(z(0)) + \underbrace{\int_0^1 -\nu^\intercal \frac{\partial f}{\partial z}\nu\, ds}_{=-\hat{\ell}(0)}$$

Directional derivative

$\mathrm{Tr}\left(\frac{\partial f}{\partial z}\right) = \nabla \cdot f$ requires D backprop calls to evaluate. The Hutchinson estimator reduces the backprop cost. The directional derivative

$$v^\top \frac{\partial f}{\partial z} v = \frac{\partial g}{\partial v} = \frac{\partial g}{\partial z} v$$

where $g = v^\top f$, can be valuated with a single call to backprop:

```
z = torch.randn((D,), requires_grad=True)
theta = torch.randn((D,), requires_grad=False)
v = torch.randn((D,), requires_grad=False)

f = . . .
g = torch.dot(v, f)
grad = torch.autograd.grad(outputs=g, inputs=z)[0]
grad_v = torch.dot(grad, v)
```

Stochastic log-likelihood computation

Instead of the trace of the Jacobian (the divergence), use the Hutchinson trace estimator and solve for $z(s)$ and $\hat{\ell}(s)$ together by using an ODE solver in reverse pseudo-time.

$$\log \widehat{p_\theta^{(\text{gen})}}(X) = \log p_Z(z(0)) - \hat{\ell}(0)$$

$$\frac{d}{ds}\begin{bmatrix} z(s) \\ \hat{\ell}(s) \end{bmatrix} = \begin{bmatrix} f\left(z(s), \theta, s\right) \\ -\nu^\intercal \frac{\partial f}{\partial z} \nu(z(s), \theta, s) \end{bmatrix} \qquad \text{for } s \in [0, 1]$$

$$\begin{bmatrix} z(1) \\ \hat{\ell}(1) \end{bmatrix} = \begin{bmatrix} X \\ 0 \end{bmatrix}$$

(For an X, we sample a random ν and keep the ν fixed throughout the ODE solve.)

Problem: We have computed a stochastic estimate of log-likelihood, but we need the gradient of the log likelihood.

Stochastic gradient of log-likelihood v.1

Since $\log \widehat{p_\theta^{(\text{gen})}}(X) = \log p_Z(z(0)) - \hat{\ell}(0)$ is computed by solving an ODE in reverse pseudo-time, we compute its gradient $\nabla_\theta \log \widehat{p_\theta^{(\text{gen})}}(X)$ using the adjoint state method.

Step 1. In reverse pseudo-time, solve the ODE

$$\begin{aligned} \log \widehat{p_\theta^{(\text{gen})}}(X) &= \log p_Z(z(0)) - \hat{\ell}(0) \\ \frac{d}{ds}\begin{bmatrix} z(s) \\ \hat{\ell}(s) \end{bmatrix} &= \begin{bmatrix} f\left(z(s), \theta, s\right) \\ -\nu^\intercal \frac{\partial f}{\partial z} \nu(z(s), \theta, s) \end{bmatrix} \qquad \text{for } s \in [0,1] \\ \begin{bmatrix} z(1) \\ \hat{\ell}(1) \end{bmatrix} &= \begin{bmatrix} X \\ 0 \end{bmatrix} \end{aligned}$$

to compute and store $\{z(s)\}_{s\in[0,1]}$.

Stochastic gradient of log-likelihood v.1

Step 2. In forward pseudo-time, using $\{z(s)\}_{s\in[0,1]}$, solve the ODE

$$\frac{\partial \log \widehat{p_\theta^{(\mathrm{gen})}}(X)}{\partial\theta} = b(1)$$

$$\dot a(s) = -a\frac{\partial f}{\partial z}(z(s),\theta,s) - \frac{\partial}{\partial z}\nu^\intercal\frac{\partial f}{\partial z}(z(s),\theta,s)\nu, \qquad s\in[0,1]$$

$$\dot b(s) = -a\frac{\partial f}{\partial \theta}(z(s),\theta,s) - \frac{\partial}{\partial \theta}\nu^\intercal\frac{\partial f}{\partial z}(z(s),\theta,s)\nu, \qquad s\in[0,1]$$

$$a(0) = \frac{\log p_Z(z)}{\partial z}\Big|_{z=z(0)} \in \mathbb{R}^{1\times D}, \qquad b(0) = 0 \in \mathbb{R}^{1\times P}$$

to compute $\frac{\partial \log \widehat{p_\theta^{(\mathrm{gen})}}(X)}{\partial\theta}$. Proof is left to homework.

Stochastic gradient of log-likelihood v.2

Storing $\{z(s)\}_{s\in[0,1]}$ is inefficient. Also, the value of $\widehat{\log p_\theta^{(\text{gen})}}(X) = p_Z(z(0)) - \hat{\ell}(0)$ is not actually used in Step 2.

Step 1. In reverse pseudo-time, solve the ODE

$$\begin{aligned} \dot{z}(s) &= f\left(z(s), \theta, s\right), \qquad \text{for } s \in [0,1] \\ z(1) &= X \end{aligned}$$

to compute $z(0)$.

Stochastic gradient of log-likelihood v.2

Step 2. In forward pseudo-time, using $z(0)$, solve the ODE

$$\frac{\partial \log \widehat{p_\theta^{(\text{gen})}}(X)}{\partial \theta} = b(1)$$

$$\begin{aligned}
\dot{z}(s) &= f\left(z(s), \theta, s\right), && s \in [0,1] \\
\dot{a}(s) &= -a \frac{\partial f}{\partial z}(z(s), \theta, s) - \frac{\partial}{\partial z} \nu^\intercal \frac{\partial f}{\partial z}(z(s), \theta, s)\nu, && s \in [0,1] \\
\dot{b}(s) &= -a \frac{\partial f}{\partial \theta}(z(s), \theta, s) - \frac{\partial}{\partial \theta} \nu^\intercal \frac{\partial f}{\partial z}(z(s), \theta, s)\nu, && s \in [0,1] \\
z(0) &= z(0), \quad a(0) = \frac{\log p_Z(z)}{\partial z}\Big|_{z=z(0)} \in \mathbb{R}^{1\times D}, \quad b(0) = 0 \in \mathbb{R}^{1\times P}
\end{aligned}$$

to compute $\dfrac{\partial \log \widehat{p_\theta^{(\text{gen})}}(X)}{\partial \theta}$.

Mixed partial derivatives

Computation of $\frac{\partial}{\partial z}\nu^\intercal\frac{\partial f}{\partial z}(z(s),\theta,s)\nu$ and $\frac{\partial}{\partial \theta}\nu^\intercal\frac{\partial f}{\partial z}(z(s),\theta,s)\nu$ require computing mixed partial derivatives. Modern deep learning libraries such as PyTorch support the computation of higher order derivatives.

```
z = torch.randn((D,), requires_grad=True)
theta = torch.randn((D,), requires_grad=True)
v = torch.randn((D,), requires_grad=False)

f = . . .
g = torch.dot(v, f)
grad = torch.autograd.grad(outputs=g, inputs=z, create_graph=True)[0]
directional_derivative_v = torch.dot(grad, v)

grad_z, grad_theta = torch.autograd.grad(directional_derivative_v, [z, theta])
```

Training FFJORD

while not converged:

X from dataset

$z(0)$ by solving

$$\dot{z}(s) = f\left(z(s), \theta, s\right), \qquad \text{for } s \in [0,1]$$
$$z(1) = X$$

$g = 0$

for _ = 0, … , K ($K \geq 1$ is a hyper parameter, batch size for ν)

ν from IID Gaussian or Rademacher

solve

$$\frac{\partial \log \widehat{p_\theta^{(\mathrm{gen})}}(X)}{\partial \theta} = b(1)$$
$$\dot{z}(s) = f\left(z(s), \theta, s\right), \qquad s \in [0,1]$$
$$\dot{a}(s) = -a\frac{\partial f}{\partial z}(z(s), \theta, s) - \frac{\partial}{\partial z}\nu^\intercal \frac{\partial f}{\partial z}(z(s), \theta, s)\nu, \qquad s \in [0,1]$$
$$\dot{b}(s) = -a\frac{\partial f}{\partial \theta}(z(s), \theta, s) - \frac{\partial}{\partial \theta}\nu^\intercal \frac{\partial f}{\partial z}(z(s), \theta, s)\nu, \qquad s \in [0,1]$$
$$z(0) = z(0), \quad a(0) = \frac{\log p_Z(z)}{\partial z}\Big|_{z=z(0)} \in \mathbb{R}^{1\times D}, \quad b(0) = 0 \in \mathbb{R}^{1\times P}$$

$g \mathrel{+}= b(1)$

endfor

call optimizer with g

endwhile

This version has batch size 1.
We can also use a larger batch size (use more X's) before calling the optimizer with stochastic gradient g.