September 1, 2020

Continuous Normalizing Flows

Vikram Voleti

PhD student, Mila, University of Montreal

Supervisor: **Prof. Christopher Pal**

voletiv.github.io

@ (virtual) Mila, Montreal, Canada

1. Ordinary Differential Equations

2. Neural ODEs
3. Continuous Normalizing Flows
4. Later research

Ordinary Differential Equations (ODEs)

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = \ ?$$

Solution:

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

Example:

$$\frac{dx}{dt} = 2t; \ x(0) = 2; \ x(1) = \ ?$$

$$\begin{aligned} \Rightarrow x(1) &= x(0) + \int_0^1 2t\, dt \\ &= x(0) + (t^2|_{t=1} - t^2|_{t=0}) \\ &= 2 + 1^2 - 0^2 \\ &= 3 \end{aligned}$$

Ordinary Differential Equations (ODEs)

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \;\; x(t_0) \text{ is given}; \;\; x(t_1) = \; ?$$

Solution:

$$x(t_1) = x(t_0) + \boxed{\int_{t_0}^{t_1} f(x(t), t, \theta)\, dt}$$

What if this cannot be analytically integrated?

Example:

$$\frac{dx}{dt} = 2xt \; ; \; x(0) = 3$$

$$\Rightarrow \int \frac{1}{2x}\, dx = \int t\, dt$$

$$\Rightarrow \frac{1}{2}\log x = \frac{1}{2}t^2 + c_0$$

$$\Rightarrow x(t) = ce^{t^2}$$

$$x(0) = 3 \Rightarrow c = 2$$

$$\therefore x(t) = 2e^{t^2}$$

$$\Rightarrow x(1) = 5.436$$

Ordinary Differential Equations (ODEs)

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given;} \quad x(t_1) = \ ?$$

Solution:

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

Approximations to $\int_{t_0}^{t_1} f(x(t), t, \theta)\, dt$

i.e. **Numerical Integration :**

- Euler method
- Runge-Kutta methods
- ...

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = \ ?$$

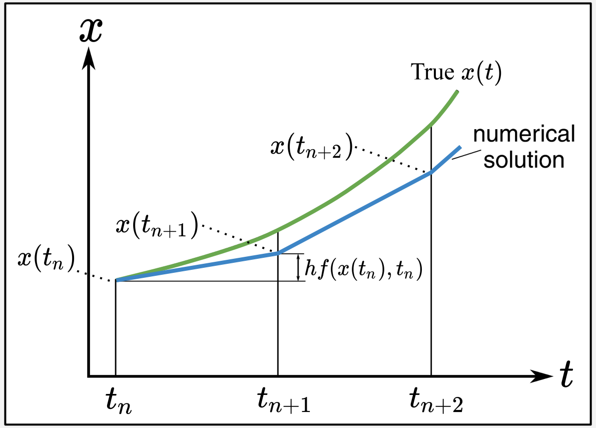
Solution:

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

1st-order Runge-Kutta / Euler's method:

$t_{n+1} = t_n + h$ → Step size h

$x(t_{n+1}) = x(t_n) + hf(x(t_n), t_n)$ → Update using derivative f



https://guide.freecodecamp.org/mathematics/differential-equations/eulers-method/

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \;\; x(t_0) \text{ is given; } \;\; x(t_1) = \;?$$

Solution:

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

1st-order Runge-Kutta / Euler's method:

$$t_{n+1} = t_n + h$$
$$x(t_{n+1}) = x(t_n) + hf(x(t_n), t_n)$$

Example:

$$\frac{dx}{dt} = f(x,t) = 2xt \;; \; x(0) = 3; \; x(1) = \;?$$

$$(\text{Solution: } \; x(t) = 2e^{t^2}; \; x(1) = 5.436)$$

$$\boxed{h = 0.25}$$

$$\begin{aligned}
x(0.25) &= x(0) + 0.25 * f(x(0), 0) \\
&= 3 + 0.25 * (2 * 3 * 0) \\
&= 3 \\
x(0.5) &= x(0.25) + 0.25 * f(x(0.25), 0.25) \\
&= 3 + 0.25 * (2 * 3 * 0.25) \\
&= 3.375 \\
x(0.75) &= x(0.5) + 0.25 * f(x(0.5), 0.5) \\
&= 3.375 + 0.25 * (2 * 3.375 * 0.5) \\
&= 4.21875 \\
x(1) &= x(0.75) + 0.25 * f(x(0.75), 0.75) \\
&= 4.21875 + 0.25 * (2 * 4.21875 * 0.75) \\
&= 5.8008
\end{aligned}$$

Ordinary Differential Equations (ODEs)

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = \ ?$$

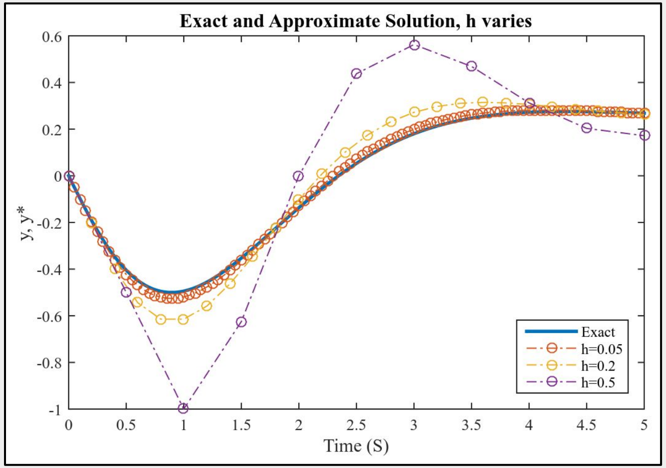
Solution:

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

1st-order Runge-Kutta / Euler's method:

$$t_{n+1} = t_n + h$$
$$x(t_{n+1}) = x(t_n) + hf(x(t_n), t_n)$$

Step size matters!



https://lpsa.swarthmore.edu/NumInt/NumIntFirst.html

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = \ ?$$

Solution:

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

★ **4th-order Runge-Kutta method:**

$$t_{n+1} = t_n + h$$
$$s_1 = f(x(t_n),\ t_n)$$
$$s_2 = f(x(t_n) + \tfrac{h}{2} s_1,\ t_n + \tfrac{h}{2})$$
$$s_3 = f(x(t_n) + \tfrac{h}{2} s_2,\ t_n + \tfrac{h}{2})$$
$$s_4 = f(x(t_n) + h s_3,\ t_n + h)$$
$$x(t_{n+1}) = x(t_n) + \tfrac{h}{6}(s_1 + 2s_2 + 2s_3 + s_4)$$

Default ODE solver used in MATLAB:
https://blogs.mathworks.com/loren/2015/09/23/ode-solver-selection-in-matlab/

Ordinary Differential Equations (ODEs)

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given;} \quad x(t_1) = \ ?$$

Solution:

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

$$\boxed{x(t_1) = \text{ODESolve}(\ f(x(t), t, \theta),\ x(t_0),\ t_0,\ t_1\ )}$$

Final time

Initial time

Initial value

Differential

Any ODE solver of our choice!

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = \ ?$$

Solution:

$$x(t_1) = \text{ODESolve}(\ f(x(t), t, \theta),\ x(t_0),\ t_0,\ t_1\ )$$

Fundamental Theorem of ODEs

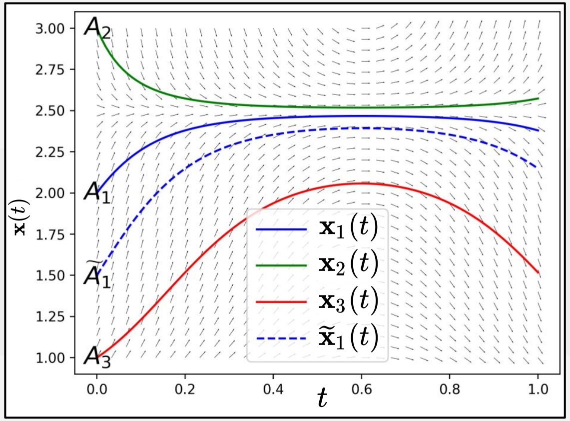
Suppose f is continuously differentiable.

1. The ... differential equation ... tely fil

2. Solution curves for different solutions **do not intersect**.

Geometrically, $x(t)$ is a flow!

http://faculty.bard.edu/belk/math213/InitialValueProblems.pdf



https://openreview.net/pdf?id=B1e9Y2NYvS

1. Ordinary Differential Equations

2. **Neural ODEs**

3. Continuous Normalizing Flows

Neural ODEs (Chen et al., NeurIPS 2018)

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = \ ?$$

Solution:

$$x(t_1) = \text{ODESolve}(\ f(x(t), t, \theta),\ x(t_0),\ t_0,\ t_1\ )$$

f **is a neural network!**

Paradigm shift: whereas earlier *f* was pre-defined/hand-designed according to the domain, here we would like to estimate an *f* that suits our objective.

https://arxiv.org/pdf/1806.07366.pdf

Neural ODEs (Chen et al., 2018)

ODEs

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

Euler discretization

Vector notation

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\, f(\mathbf{x}_n, t_n, \theta)$$

Residual networks

$$\mathbf{x}_{l+1} = \mathrm{ResBlock}(\mathbf{x}_l, \theta)$$

$$\mathbf{x}_{l+1} = \mathbf{x}_l + g(\mathbf{x}_l, \theta)$$

Skip connection

https://arxiv.org/pdf/1806.07366.pdf

https://arxiv.org/pdf/1512.03385.pdf

ODEs

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

Euler discretization

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\, f(\mathbf{x}_n, t_n, \theta)$$

Forward propagation:

$$\mathbf{x}(t_1) = \mathrm{ODESolve}(\ f(\mathbf{x}(t), t, \theta),\ \mathbf{x}(t_0),\ t_0,\ t_1\ )$$

$$L(\mathbf{x}(t_1)) \rightarrow \frac{\partial L}{\partial \theta}$$

How to compute this?

Update θ to reduce L

https://arxiv.org/pdf/1806.07366.pdf

Residual networks

$$\mathbf{x}_{l+1} = \mathrm{ResBlock}(\mathbf{x}_l, \theta)$$

$$\mathbf{x}_{l+1} = \mathbf{x}_l + g(\mathbf{x}_l, \theta)$$

Skip connection

$$\mathbf{y}_{pred} = \mathrm{ResNet}(\mathbf{x})$$

Stacked ResBlocks

$$L(\mathbf{y}_{pred}) \rightarrow \frac{\partial L}{\partial \theta}$$

Update θ to reduce L

https://arxiv.org/pdf/1512.03385.pdf

Neural ODEs (Chen et al., 2018)

ODEs

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

Euler discretization

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\, f(\mathbf{x}_n, t_n, \theta)$$

Forward propagation:

$$\mathbf{x}(t_1) = \text{ODESolve}(\ f(\mathbf{x}(t), t, \theta),\ \mathbf{x}(t_0),\ t_0,\ t_1\ )$$

$$L(\mathbf{x}(t_1)) \rightarrow \boxed{\frac{\partial L}{\partial \theta}}$$

Update θ to reduce L

Back-propagate through the ODE Solver!

High memory cost -

need to save all activations of all iterations of ODESolve.

Can we do better?

Yes.

https://arxiv.org/pdf/1806.07366.pdf

$$\boxed{L(\text{ODESolve}(\ f(\mathbf{x}(t), t, \theta),\ \mathbf{x}(t_0),\ t_0,\ t_1\ )) \rightarrow \frac{\partial L}{\partial \theta}}$$

Adjoint method (Pontryagin et al., 1962)

$$\text{adjoint } \mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{x}}\ ;\ \frac{d\mathbf{a}}{dt} = -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \mathbf{x}}$$

Forward propagation: $\mathbf{x}(t_1) = \text{ODESolve}(\ f(\mathbf{x}(t), t, \theta),\ \mathbf{x}(t_0),\ t_0,\ t_1\ ) \ \Rightarrow \mathbf{a}(t_1) = \frac{\partial L}{\partial \mathbf{x}(t_1)}$

? ?

$$\boxed{\frac{\partial L}{\partial \theta}} = \int_{t_1}^{t_0} -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \theta}\, dt$$

https://arxiv.org/pdf/1806.07366.pdf

Neural ODEs (Chen et al., 2018)

$$\boxed{L(\text{ODESolve}(\ f(\mathbf{x}(t), t, \theta),\ \mathbf{x}(t_0),\ t_0,\ t_1\ )) \rightarrow \frac{\partial L}{\partial \theta}}$$

Adjoint method (Pontryagin et al., 1962)

$$\text{adjoint } \mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{x}}\ ;\ \frac{d\mathbf{a}}{dt} = -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \mathbf{x}}$$

Forward propagation: $\mathbf{x}(t_1) = \text{ODESolve}(\ f(\mathbf{x}(t), t, \theta),\ \mathbf{x}(t_0),\ t_0,\ t_1\ ) \quad \Rightarrow \mathbf{a}(t_1) = \frac{\partial L}{\partial \mathbf{x}(t_1)}$

Back-propagation:

$$x(t_0) = \text{ODESolve}(\quad f(\mathbf{x}(t), t, \theta) \quad,\ \mathbf{x}(t_1),\ t_1,\ t_0)$$

$$\Rightarrow \mathbf{a}(t_0) = \frac{\partial L}{\partial \mathbf{x}(t_0)} = \text{ODESolve}(-\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \mathbf{x}},\ \frac{\partial L}{\partial \mathbf{x}(t_1)},\ t_1,\ t_0)$$

$$\therefore \boxed{\frac{\partial L}{\partial \theta}} = \int_{t_1}^{t_0} -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \theta}\, dt = \text{ODESolve}(-\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \theta},\quad \mathbf{0}_{|\theta|}\quad,\ t_1,\ t_0)$$

Initial value is 0

https://arxiv.org/pdf/1806.07366.pdf

Neural ODEs (Chen et al., 2018)

Mila

Forward propagation:

$$\mathbf{x}(t_1) = \text{ODESolve}(\ f(\mathbf{x}(t), t, \theta),\ \mathbf{x}(t_0),\ t_0,\ t_1\ )$$

Compute $L(\mathbf{x}(t_1))$.

$$\mathbf{a}(t_1) = \frac{\partial L}{\partial \mathbf{x}(t_1)}$$

Back-propagation:

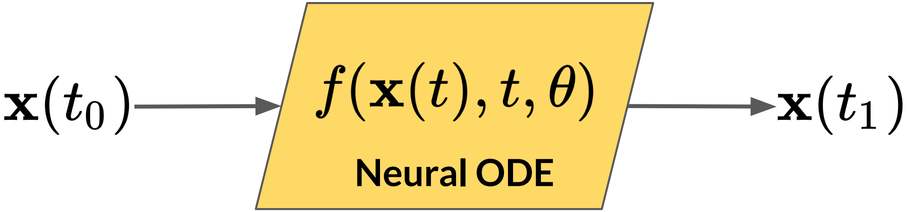
$$\begin{bmatrix} \mathbf{x}(t_0) \\ \dfrac{\partial L}{\partial \mathbf{x}(t_0)} \\ \boxed{\dfrac{\partial L}{\partial \theta}} \end{bmatrix} = \text{ODESolve}\left(\begin{bmatrix} f(\mathbf{x}(t), t, \theta) \\ -\mathbf{a}(t)^\top \dfrac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \mathbf{x}} \\ -\mathbf{a}(t)^\top \dfrac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \theta} \end{bmatrix}, \begin{bmatrix} \mathbf{x}(t_1) \\ \dfrac{\partial L}{\partial \mathbf{x}(t_1)} \\ \mathbf{0}_{|\theta|} \end{bmatrix}, t_1, t_0 \right)$$

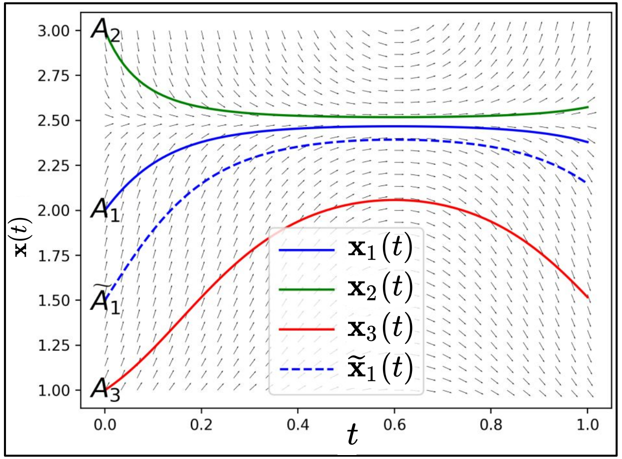
Update θ to reduce L

https://arxiv.org/pdf/1806.07366.pdf

Neural ODEs (Chen et al., 2018)

https://arxiv.org/pdf/1806.07366.pdf





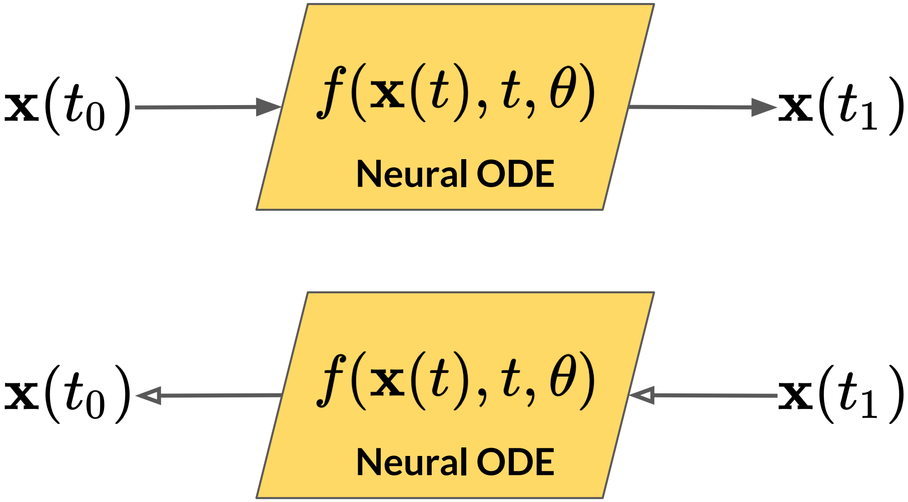
https://openreview.net/pdf?id=B1e9Y2NYvS

Neural ODEs describe a homeomorphism (flow).

- They preserve dimensionality.
- They form non-intersecting trajectories.

Neural ODEs (Chen et al., 2018)

https://arxiv.org/pdf/1806.07366.pdf

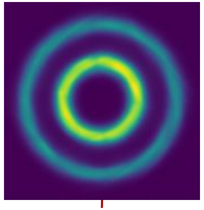


Neural ODEs are **reversible** models!
Just integrate forward/backward in time.

1. Ordinary Differential Equations
2. Neural ODEs
3. **Continuous Normalizing Flows**

Continuous Normalizing Flows

Target distribution



(such as real image manifold)

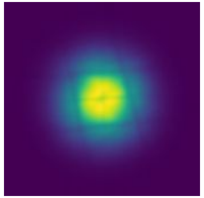
$\mathbf{x}_{im}$

$f_\theta(\mathbf{x}(t), t)$

Neural ODE

$\mathbf{z}$

Noise distribution



Sample from target distribution (such as an image)

Likelihood estimation

using Change of Variables formula

$$\mathbf{z} = g(\mathbf{x}) \Rightarrow \log p(\mathbf{x}) = \log p(\mathbf{z}) + \log \left| \det \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right|$$

Sample from noise distribution (such as Gaussian)

Train *f* to maximize the likelihood of the samples from target distribution $\log p(\mathbf{x})$

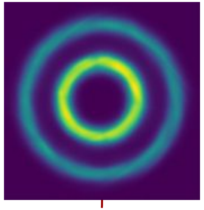
https://arxiv.org/abs/1810.01367

Mila

"FFJORD"

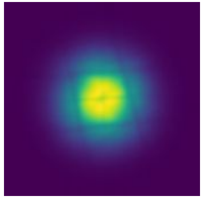
(Free-Form Jacobian Of Reversible Dynamics)

Target distribution



(such as real image manifold)

Noise distribution



$\mathbf{x}_{im}$ ⇄ $f_\theta(\mathbf{x}(t), t)$ **Neural ODE** ⇄ $\mathbf{z}$

Sample from target distribution (such as an image)

Sample from noise distribution (such as Gaussian)

Likelihood estimation →

using Change of Variables formula

← Generate samples

Sample from the noise distribution, transform it into a sample from the target distribution using the trained Neural ODE.

https://arxiv.org/abs/1810.01367

FFJORD (ICLR 2019)

Change of variables:

$$\log p(\mathbf{x}_{im}) - \log p(\mathbf{z}) = \log\det\left|\frac{\mathrm{d}f_\theta}{\mathrm{d}\mathbf{x}(t)}\right|$$

Instantaneous change of variables:

$$\frac{\partial \log p(\mathbf{x}(t))}{\partial t} = -\mathrm{Tr}\left(\frac{\partial f_\theta}{\partial \mathbf{x}(t)}\right)$$

Initial value: $\begin{bmatrix} \mathbf{x}_{im} \\ 0 \end{bmatrix}$

$$\begin{bmatrix} \mathbf{z} \\ \log p(\mathbf{x}_{im}) - \log p(\mathbf{z}) \end{bmatrix} = \int_{t_0}^{t_1} \begin{bmatrix} f_\theta(\mathbf{x}(t), t) \\ -\mathrm{Tr}\left(\frac{\partial f_\theta}{\partial \mathbf{x}(t)}\right) \end{bmatrix} \mathrm{d}t$$

Hutchinson's trace estimator:

$$\begin{aligned} \log p(\mathbf{z}(t_1)) &= \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) dt \\ &= \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}(t)} \boldsymbol{\epsilon}\right] dt \\ &= \log p(\mathbf{z}(t_0)) - \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\int_{t_0}^{t_1} \boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}(t)} \boldsymbol{\epsilon} dt\right] \end{aligned}$$

https://arxiv.org/abs/1810.01367

Continuous Normalizing Flows

FFJORD (ICLR 2019)

Change of variables:

$$\log p(\mathbf{x}_{im}) - \log p(\mathbf{z}) = \log\det\left|\frac{\mathrm{d}f_\theta}{\mathrm{d}\mathbf{x}(t)}\right|$$

Instantaneous change of variables:

$$\frac{\partial \log p(\mathbf{x}(t))}{\partial t} = -\mathrm{Tr}\left(\frac{\partial f_\theta}{\partial \mathbf{x}(t)}\right)$$

Initial value: $\begin{bmatrix} \mathbf{x}_{im} \\ 0 \end{bmatrix}$

$$\begin{bmatrix} \mathbf{z} \\ \log p(\mathbf{x}_{im}) - \log p(\mathbf{z}) \end{bmatrix} = \int_{t_0}^{t_1} \begin{bmatrix} f_\theta(\mathbf{x}(t), t) \\ -\mathrm{Tr}\left(\frac{\partial f_\theta}{\partial \mathbf{x}(t)}\right) \end{bmatrix} \mathrm{d}t$$

CIFAR10		ImageNet64	
BPD	**Time**	**BPD**	**Time**
3.40	≥5 days	-	-

https://arxiv.org/abs/1810.01367

How to Train your Neural ODE (ICML 2020)

Introduces 2 regularization terms:
1) Kinetic energy of flow
2) Jacobian norm of flow

$$\mathcal{K}(\theta) = \int_{t_0}^{t_1} \|f(\mathbf{x}(t), t, \theta)\|_2^2 \,\mathrm{d}t$$

$$\mathcal{B}(\theta) = \int_{t_0}^{t_1} \|\epsilon^\top \nabla_z f(\mathbf{x}(t), t, \theta)\|_2^2 \,\mathrm{d}t$$

CIFAR10		ImageNet64	
BPD	**Time**	**BPD**	**Time**
3.38	31.84	3.83	64.1

https://arxiv.org/abs/2002.02798

STEER

Introduces temporal regularization:

$$\mathbf{x}(t_1) = \mathbf{x}(t_0) + \int_{t_0}^{T} f_\theta(\mathbf{x}(t), t)\mathrm{d}t$$
$$= \mathrm{ODESolve}(\mathbf{x}(t_0), f_\theta, t_0, T)$$

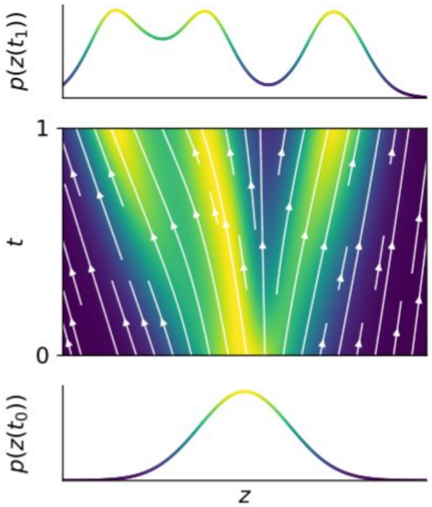
$$T \sim \mathrm{Uniform}(t_1 - b, t_1 + b)$$
$$b < t_1 - t_0$$

CIFAR10		ImageNet64	
BPD	**Time**	**BPD**	**Time**
3.39	22.24	-	-

https://arxiv.org/abs/2006.10711

3. Image Generation

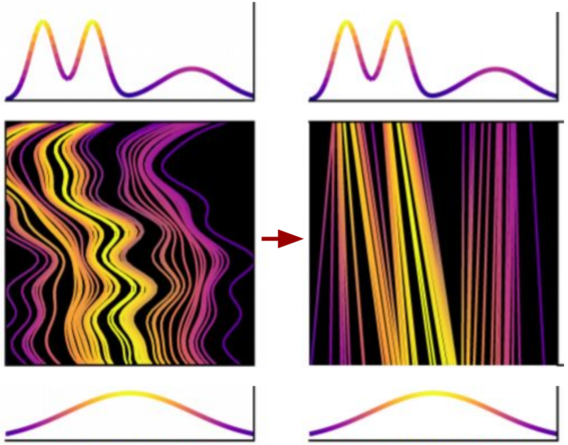
FFJORD (ICLR 2019)



CIFAR10		ImageNet64	
BPD	**Time**	**BPD**	**Time**
3.40	≥5 days	-	-

https://arxiv.org/abs/1810.01367

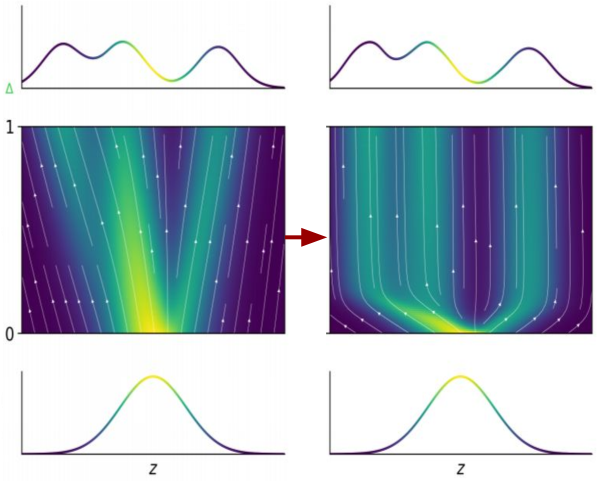
How to Train your Neural ODE (ICML 2020)



CIFAR10		ImageNet64	
BPD	**Time**	**BPD**	**Time**
3.38	31.84	3.83	64.1

https://arxiv.org/abs/2002.02798

STEER



CIFAR10		ImageNet64	
BPD	**Time**	**BPD**	**Time**
3.39	22.24	-	-

https://arxiv.org/abs/2006.10711

Thank you!

voletiv.github.io