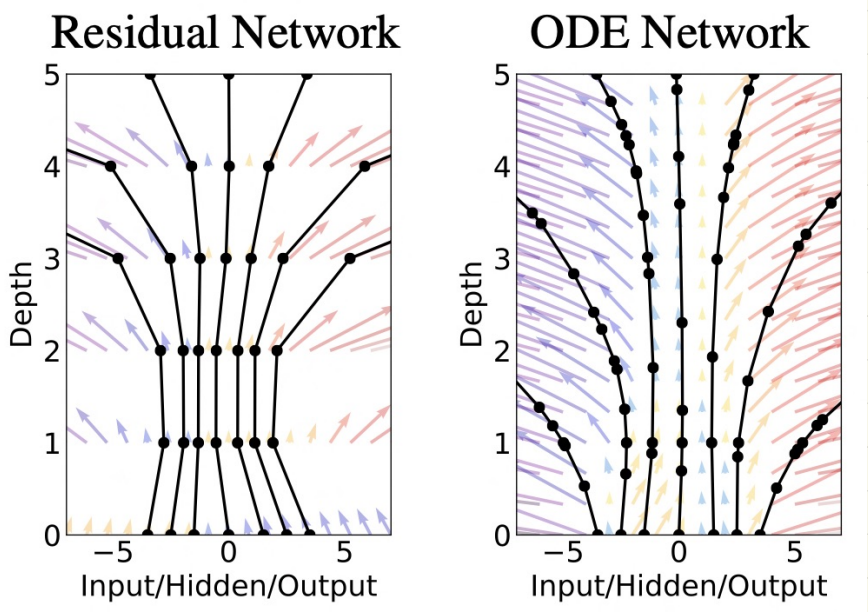
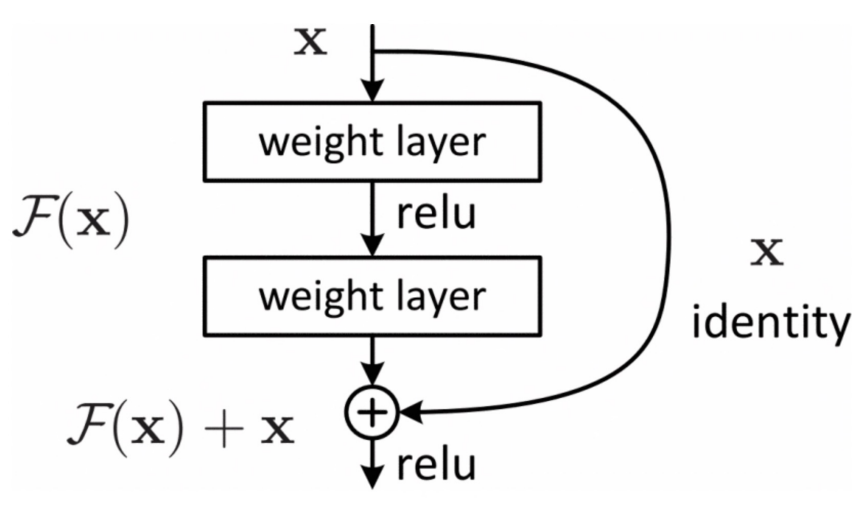
[Submitted on 19 Jun 2018 (v1), last revised 14 Dec 2019 (this version, v5)]

Neural Ordinary Differential Equations

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, David Duvenaud



Resnets



Plain NN: $x_{n+1} = F(x_n)$

ResNet: $x_{n+1} = x_n + F(x_n)$

residual

K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

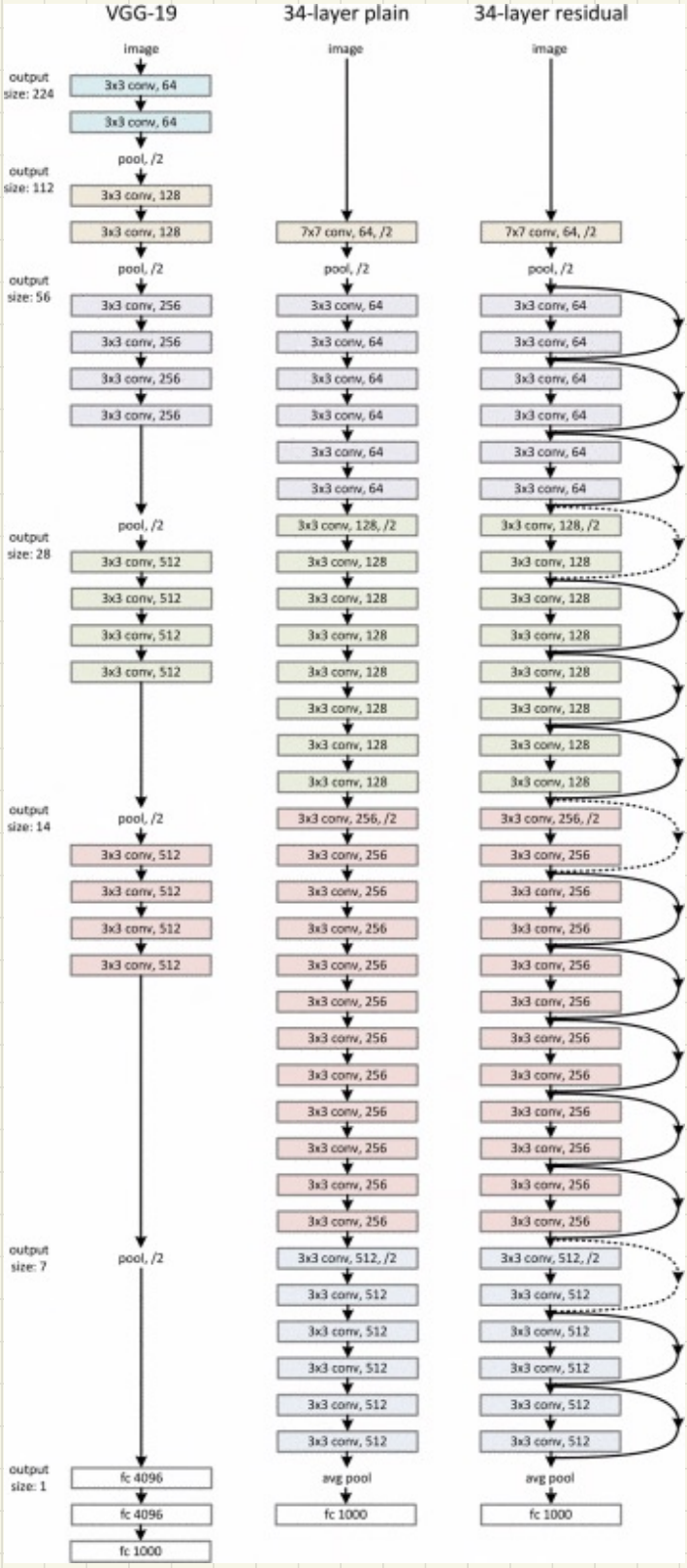
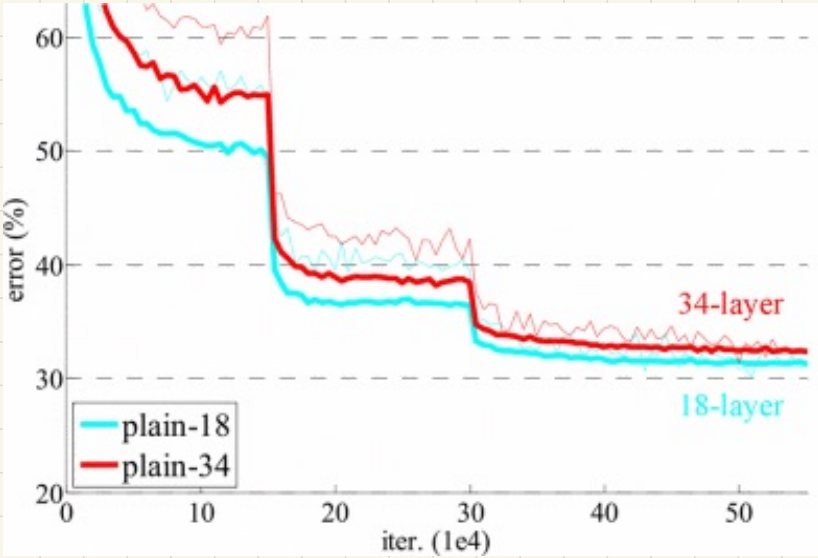
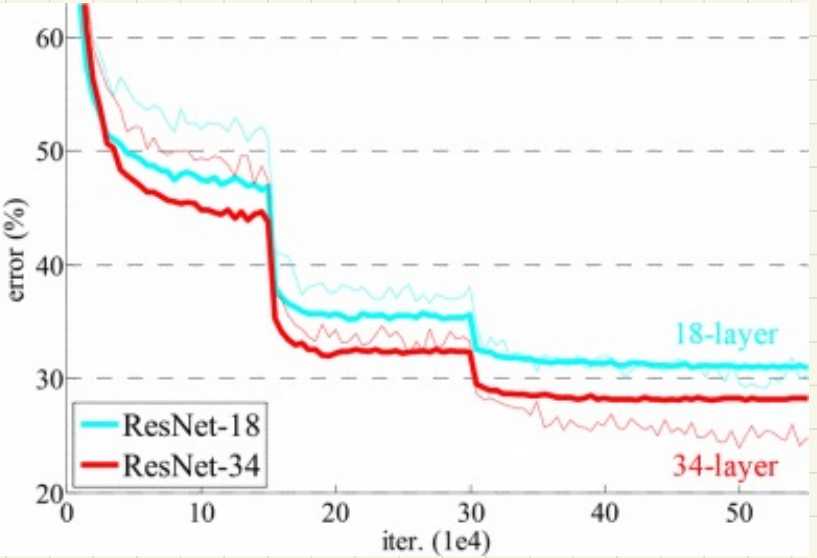

VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
output size: 28
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 14
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 7
fc 4096
fc 4096
fc 1000
output size: 1
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 256, /2
3x3 conv, 512, /2
avg pool
fc 1000

Image Net results:





$x_1 = f_1(x_0)$

$x_2 = f_2(x_1) = f_2(f_1(x_0))$

$\nabla_\theta x_2 = \nabla_\theta f_2(f_1(x_0)) + \nabla_x f_2 \cdot \nabla_\theta f_1(x_0)$

$x_2 = x_1 + f_2(x_0 + f_1(x_0))$

$\nabla_\theta x_2 = \nabla_\theta x_1 + \nabla_x f_2 \cdot (\nabla_\theta x_0 + \nabla_\theta f_1)$

Resnets: $z_{n+1} = z_n + F_\theta(z_n)$

Neural ODE: $z_{n+1} = z_n + \Delta t\, f_\theta(z_n)$ ← Euler discretization of $\dot{z} = f_\theta(z)$

$$\boxed{\dot{z} = f_\theta(z)}$$

↑ NN with params θ

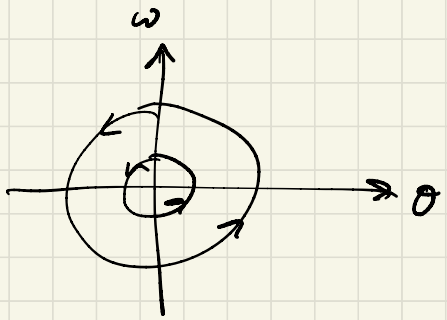
Solve: $z(t_1) = z(t_0) + \int_{t_0}^{t_1} f(z(t))\, dt$

$$L = L(z(t_0), z(t_1), z(t_2), \ldots)$$

Ex pendulum (linearized)

$\dot{\theta} = \omega$

$\dot{\omega} = -\sin\theta$



$\dot{\theta} = \omega$

$\dot{\omega} = -\theta$

$$z = \begin{bmatrix} \theta \\ \omega \end{bmatrix}$$

$$\dot{z} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} z$$

NODE: $\dot{z} = \begin{bmatrix} \dot{\theta} \\ \dot{\omega} \end{bmatrix} = f_\theta(z)$

$= f_\theta\left(\begin{bmatrix} \theta \\ \omega \end{bmatrix}\right)$

Adjoints

- x state $\in \mathbb{R}^d$
- $Ax = b$ $\quad A = A(\theta)$ $\quad b = b(\theta)$ $\quad \theta \in \mathbb{R}^p$
- loss $L(x)$
- find θ to minimize $L(x)$

$$\frac{d}{d\theta} L = \nabla_x L \cdot \nabla_\theta x$$

$$= L_x x_\theta \quad (\in \mathbb{R})$$

$$A x = b$$

$$A_\theta x + A x_\theta = b_\theta$$

$$x_\theta = A^{-1}(b_\theta - A_\theta x)$$

$$\frac{d}{d\theta} L = L_x A^{-1}(b_\theta - A_\theta x)$$

dimensions: $1 \times p$ $\quad 1 \times d$ $\quad d \times d$ $\quad d \times p$

$O(d^2 p)$

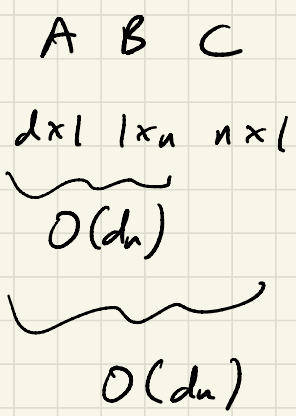
λ^T $1 \times d$

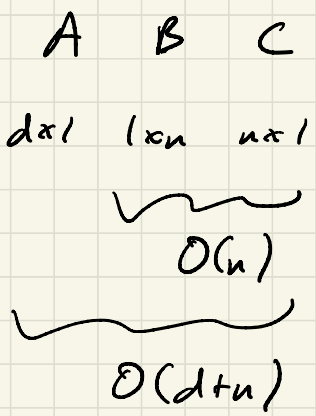
$$\lambda^T = L_x A^{-1}$$

$$\lambda = A^{-T} L_x^T$$

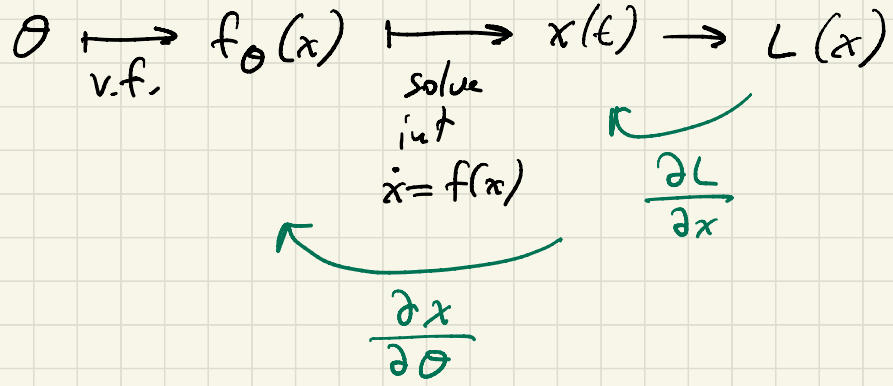
$$\boxed{A^T \lambda = L_x^T}$$

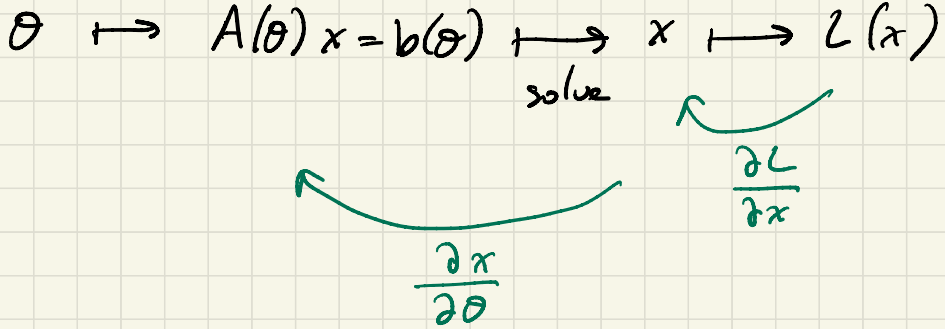
← adjoint system


A B C
d×1 1×n n×1
O(dn)
O(dn)


A B C
d×1 1×n n×1
O(n)
O(d+n)

Neural ODE:


θ
v.f.
$f_\theta(x)$
solve
int
$\dot{x} = f(x)$
$x(t)$
$L(x)$
$\frac{\partial L}{\partial x}$
$\frac{\partial x}{\partial \theta}$


θ
$A(\theta)\, x = b(\theta)$
solve
x
$L(x)$
$\frac{\partial L}{\partial x}$
$\frac{\partial x}{\partial \theta}$

Linear

$Ax = b \qquad A = A(\theta) \quad b = b(\theta)$

$A_\theta x + A x_\theta = b_\theta$

$A x_\theta = (b_\theta - A_\theta x)$

$x_\theta = A^{-1}(b_\theta - A_\theta x)$

want $L_\theta = L_x x_\theta$

$= L_x \underbrace{A^{-1}}_{\lambda^T} (b_\theta - A_\theta x)$

$$\boxed{A^T \lambda = L_x^T}$$

adjoint equ

Nonlinear

$f(x, \theta) = 0$

$f_x x_\theta + f_\theta = 0$

$f_x \leftarrow A$

$x_\theta = -f_x^{-1} f_\theta$

want $L_\theta = L_x x_\theta$

$= -L_x \underbrace{f_x^{-1}}_{\lambda^T} f_\theta$

$$\boxed{f_x^T \lambda = L_x^T}$$

Linear ODEs

$$\dot{x} = Bx \quad x(0) = b \quad t \in [0, T]$$

$$\bar{x} = x(T) = e^{BT} b$$

$$e^{-BT} \bar{x} = b$$

e^{-BT} ↖ A

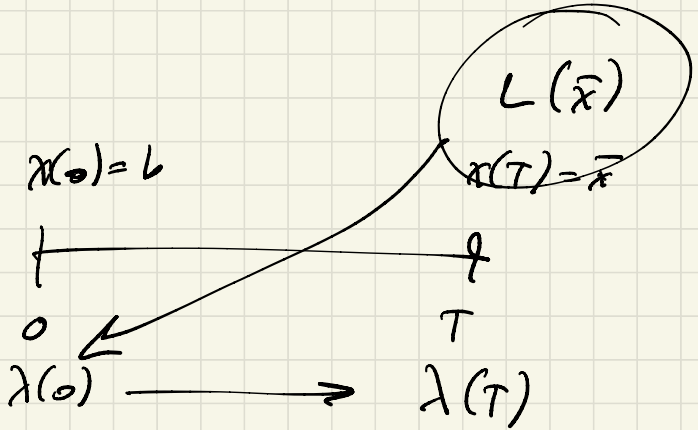
Assume $L = L(\bar{x})$

$$A^T \lambda = L_{\bar{x}}^T$$

$$e^{-B^T T} \lambda = L_{\bar{x}}^T$$

$$\lambda(T) = \lambda = e^{B^T T} L_{\bar{x}}^T$$

$$\dot{\lambda} = B^T \lambda \quad \lambda(0) = L_{\bar{x}}^T$$



---

define $\tau = -t$ $\quad \frac{\partial}{\partial t} = \frac{\partial \tau}{\partial t} \frac{\partial}{\partial \tau} = -\frac{\partial}{\partial \tau}$

$$\frac{d}{d\tau} \lambda = -B^T \lambda$$

$$\lambda(T) = L_{\bar{x}}^T$$

want $\lambda(0)$

Nonlinear ODEs

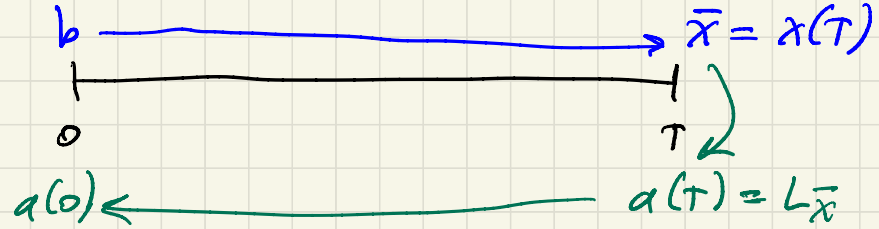
$$\dot{x} = f(x) \qquad x(0) = b$$

linearize: $\delta\dot{x} = \underbrace{\frac{\partial f}{\partial x}(x(t))}_{B}\,\delta x$

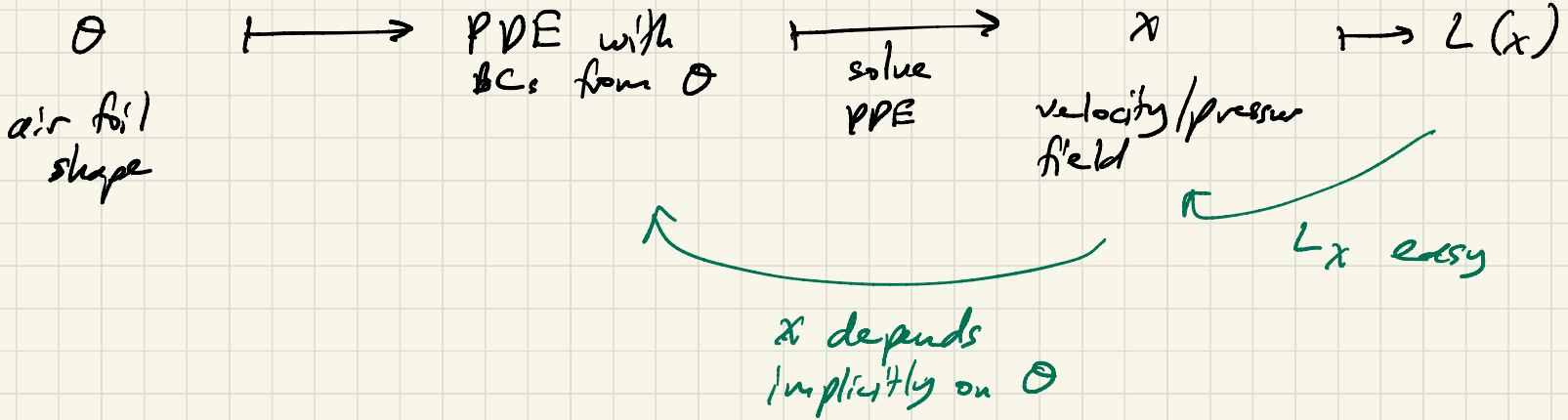
$\Rightarrow$ adjoint eqn: $\frac{d}{dt}\lambda = -\left(\frac{\partial f}{\partial x}\right)^T \lambda$

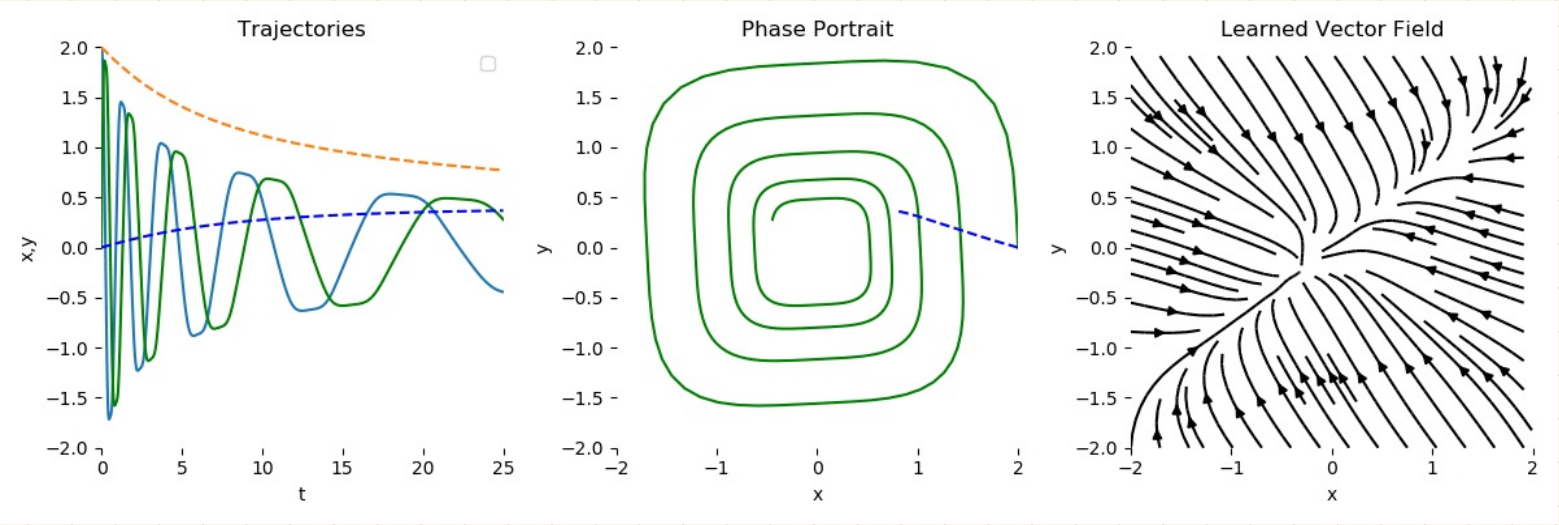
or $a = \lambda^T \Rightarrow$ $\frac{d}{dt}a = -a\,\frac{\partial f}{\partial x}(x(t))$

$$a(T) = L_{\bar{x}}$$



PDE optimization (e.g. airfoils)




Trajectories
x,y
t
Phase Portrait
y
x
Learned Vector Field
y
x

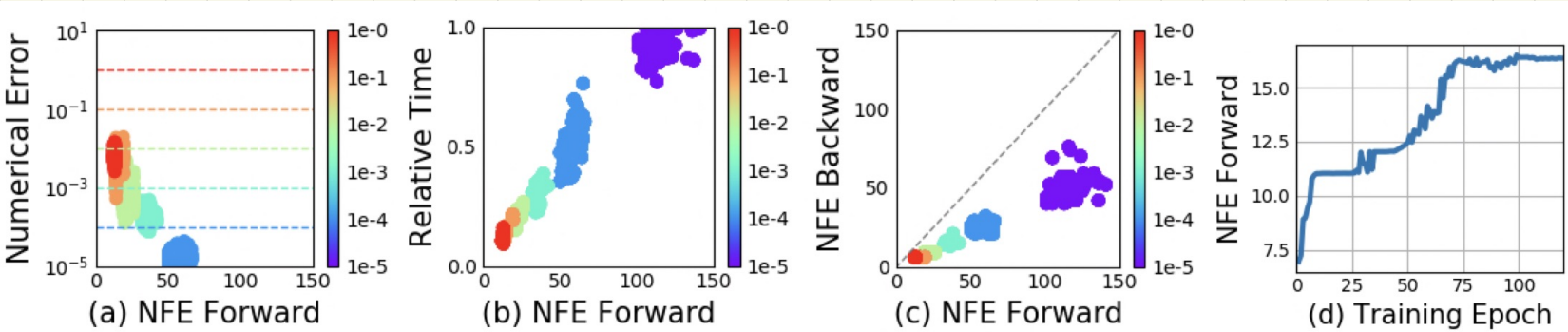
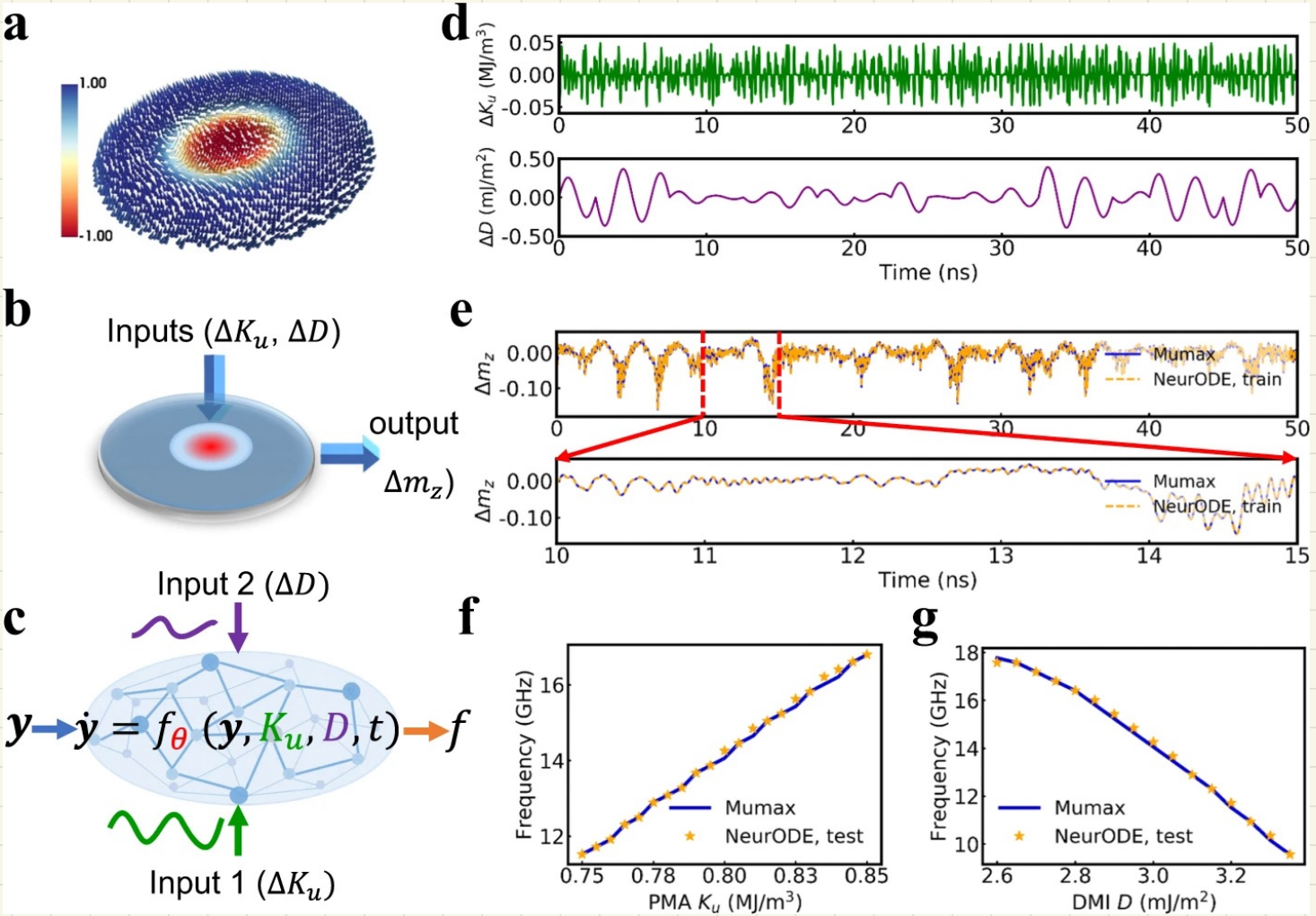


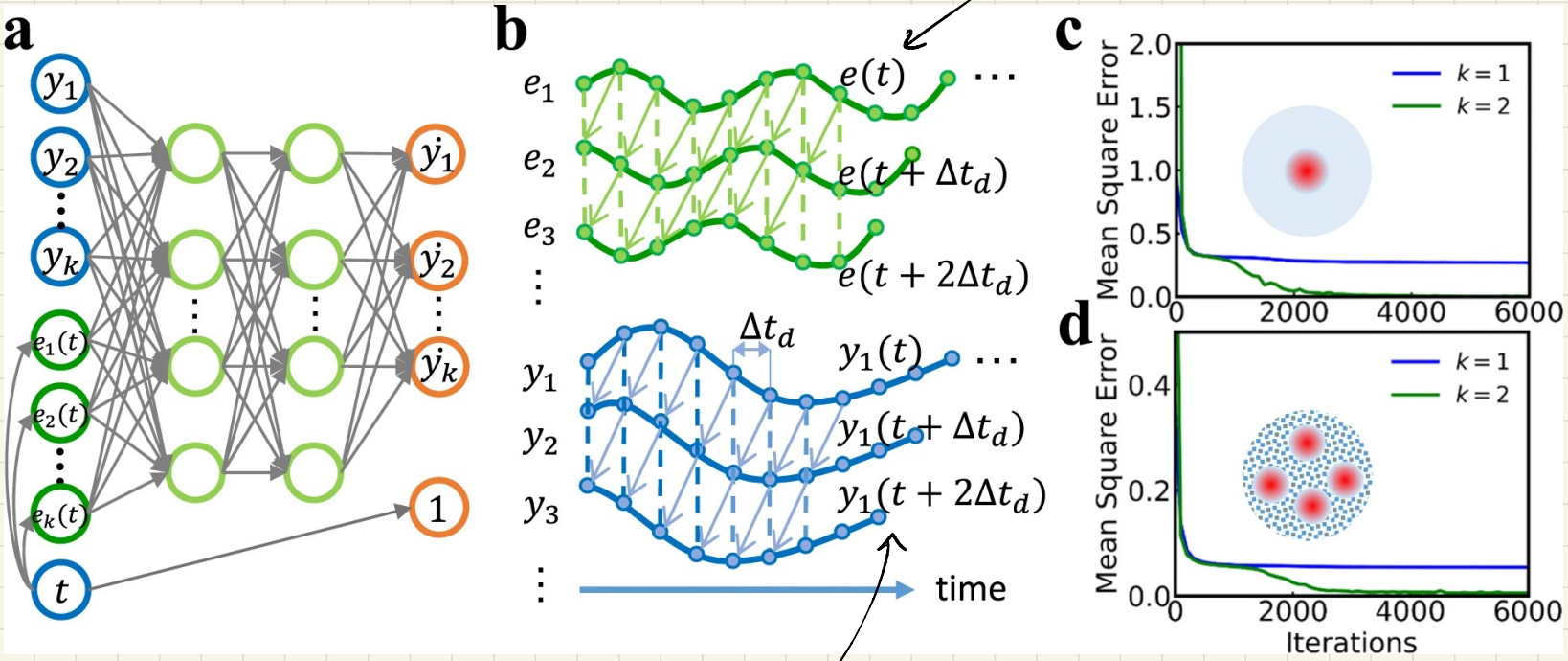
Figure 3: Statistics of a trained ODE-Net. (NFE = number of function evaluations.)

Forecasting the outcome of spintronic experiments with Neural Ordinary Differential Equations

Xing Chen, Flavio Abreu Araujo, Mathieu Riou, Jacob Torrejon, Dafiné Ravelosona, Wang Kang, Weisheng Zhao, Julie Grollier & Damien Querlioz

Nature Communications **13**, Article number: 1016 (2022) | Cite this article


a
1.00
-1.00
b
Inputs (ΔK_u, ΔD)
output
Δm_z)
c
Input 2 (ΔD)
$y \rightarrow \dot{y} = f_\theta(y, K_u, D, t) \rightarrow f$
Input 1 (ΔK_u)
d
ΔK_u (MJ/m^3)
ΔD (mJ/m^2)
Time (ns)
e
Δm_z
Mumax
NeurODE, train
Time (ns)
f
Frequency (GHz)
Mumax
NeurODE, test
PMA K_u (MJ/m^3)
g
Frequency (GHz)
Mumax
NeurODE, test
DMI D (mJ/m^2)



k = # delayed states

$k=2 \Rightarrow (y_1(t), y_1(t+\Delta t))$

Time-delay embeddings

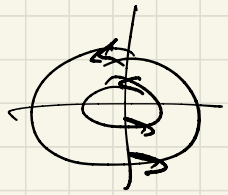
2D ODE $\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = f\begin{pmatrix} x \\ y \end{pmatrix}$

observe only $x_0, x_1, x_2, \ldots$

do NOT observe $y_0, y_1, y_2, \ldots$

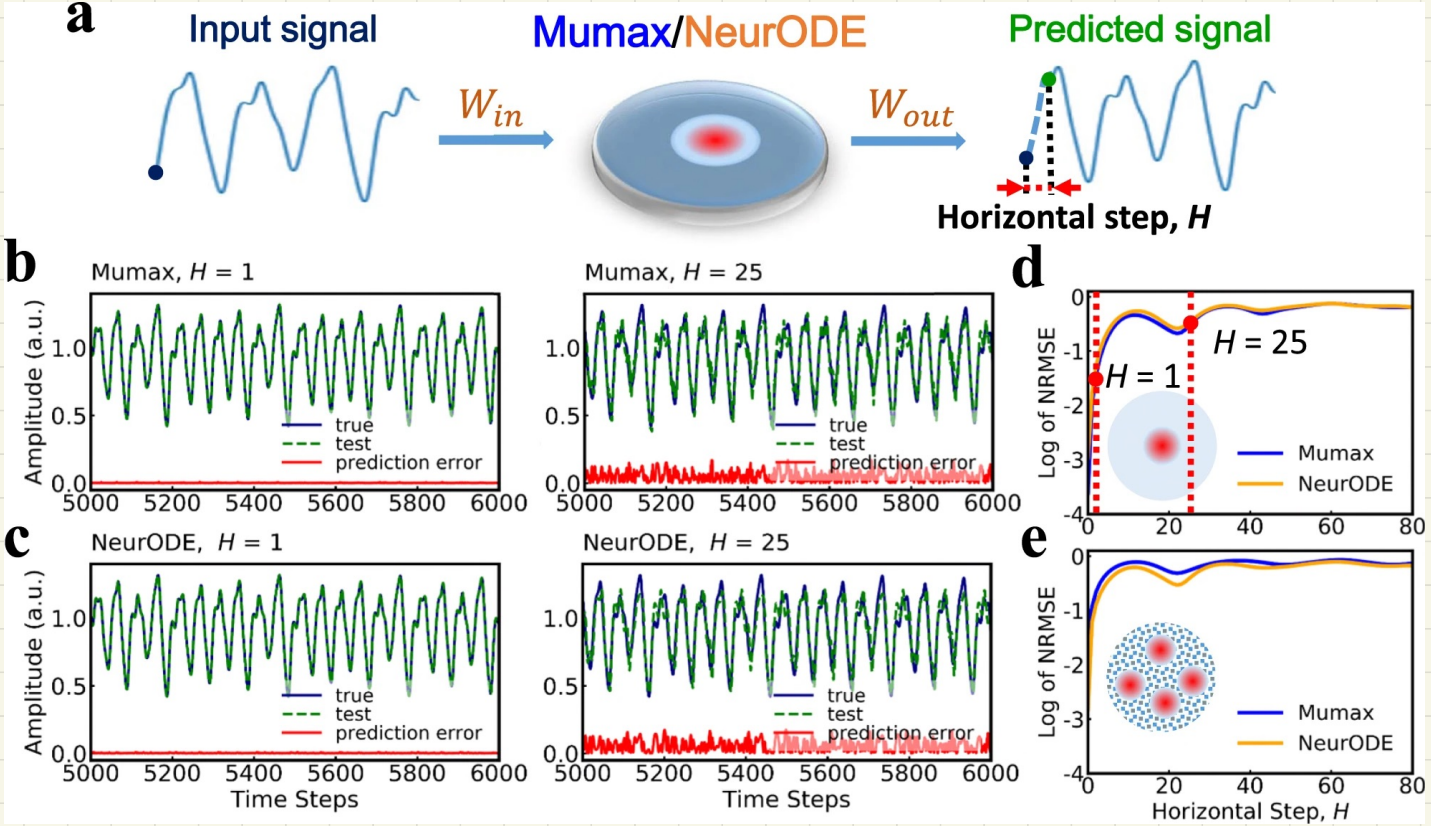
Thm (Takens embedding) there exists an equivalent (homeomorphic) system $\begin{pmatrix} x_t \\ x_{t+1} \end{pmatrix} = \tilde{f}\begin{pmatrix} x_{t-1} \\ x_t \end{pmatrix}$

Ex $\begin{pmatrix} \dot{\theta} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \omega \\ -\sin\theta \end{pmatrix}$



$\begin{pmatrix} \theta_t \\ \theta_{t+1} \end{pmatrix} = \tilde{f}\begin{pmatrix} \theta_{t-1} \\ \theta_t \end{pmatrix}$

$\omega_t \approx \frac{\theta_t - \theta_{t-1}}{\Delta t}$


a
Input signal
Mumax/NeurODE
Predicted signal
W_{in}
W_{out}
Horizontal step, H
b
Mumax, H = 1
Mumax, H = 25
c
NeurODE, H = 1
NeurODE, H = 25
Amplitude (a.u.)
true
test
prediction error
Time Steps
d
e
Log of NRMSE
H = 1
H = 25
Mumax
NeurODE
Horizontal Step, H

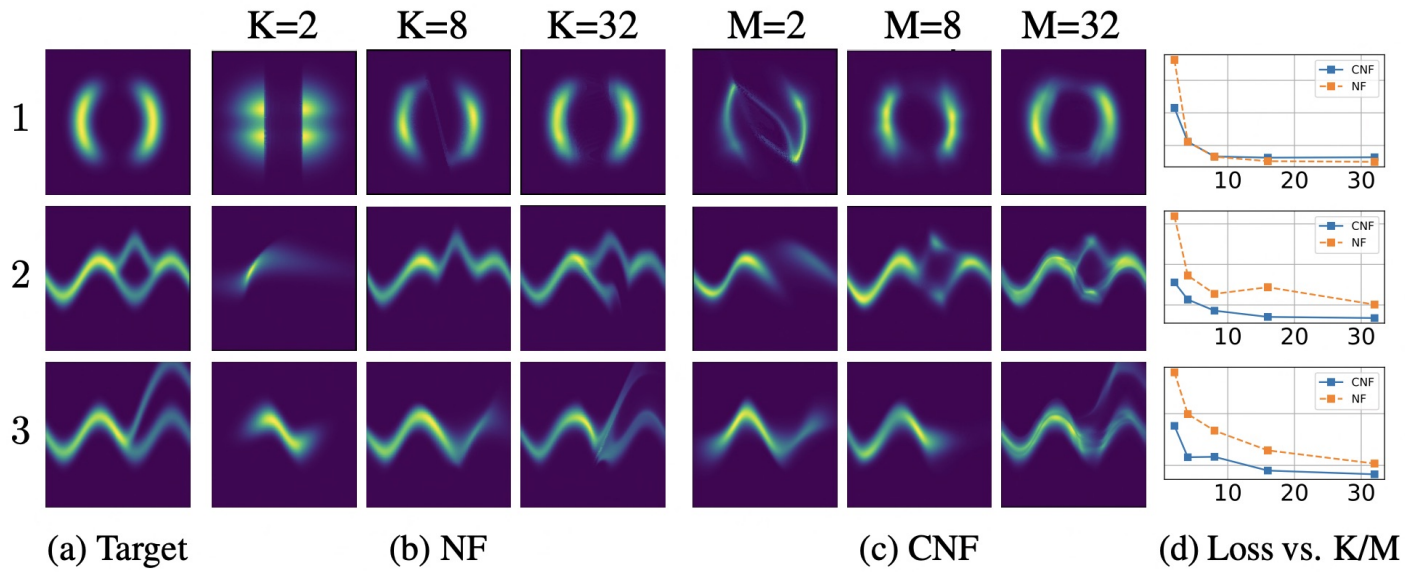
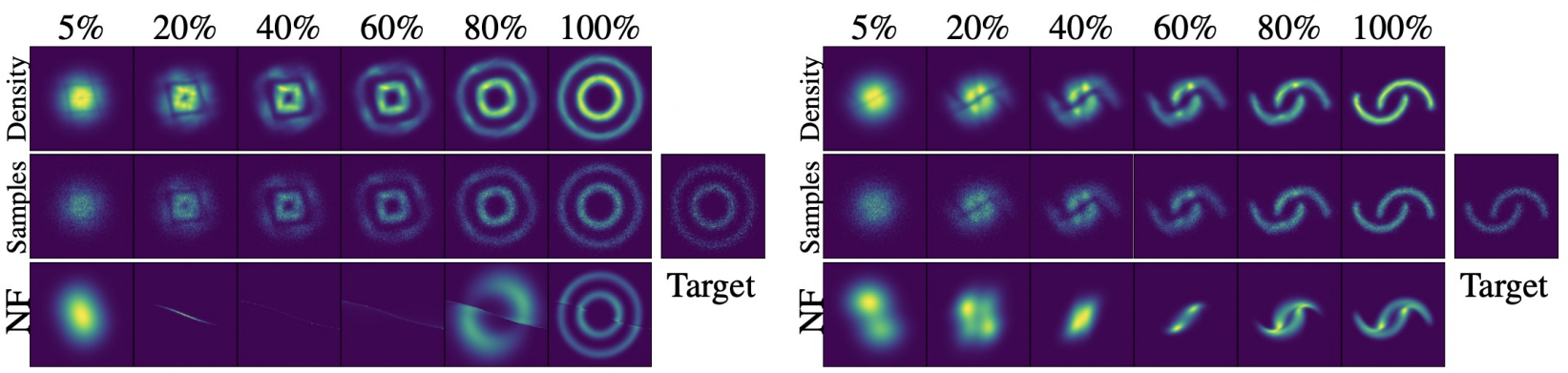


Figure 4: Comparison of normalizing flows versus continuous normalizing flows. The model capacity of normalizing flows is determined by their depth (K), while continuous normalizing flows can also increase capacity by increasing width (M), making them easier to train.



(a) Two Circles

(b) Two Moons

Figure 5: **Visualizing the transformation from noise to data.** Continuous-time normalizing flows are reversible, so we can train on a density estimation task and still be able to sample from the learned density efficiently.

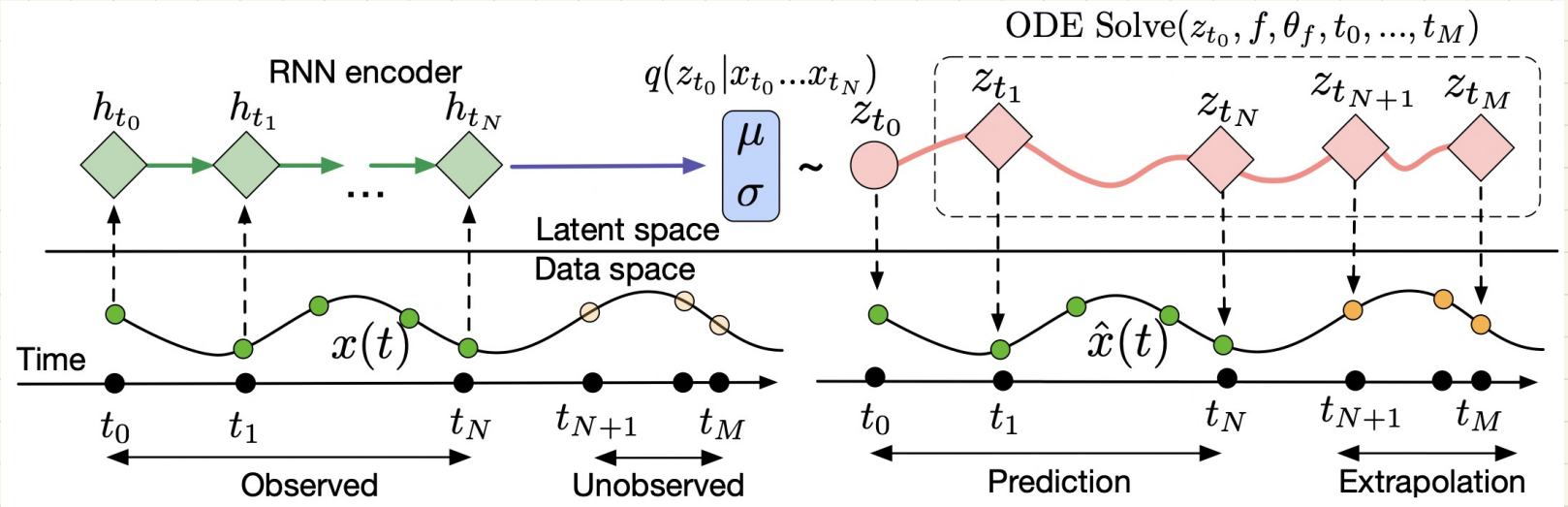
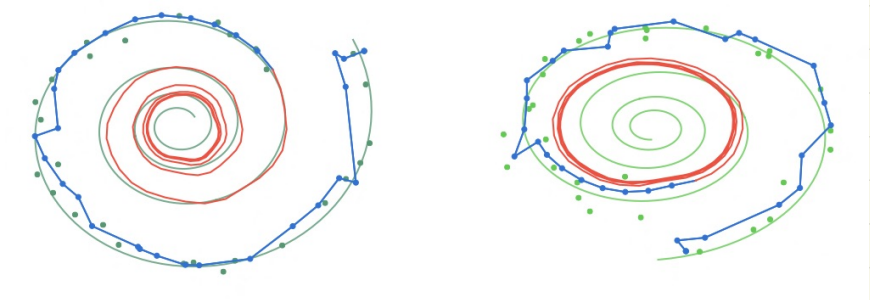
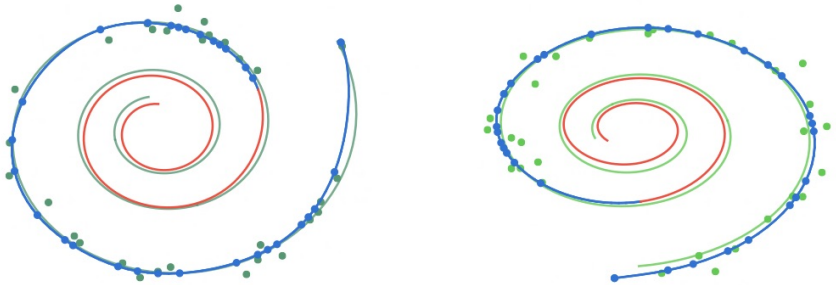


Figure 6: Computation graph of the latent ODE model.



(a) Recurrent Neural Network



(b) Latent Neural Ordinary Differential Equation

Algorithm 2 Complete reverse-mode derivative of an ODE initial value problem

Input: dynamics parameters θ, start time t_0, stop time t_1, final state $\mathbf{z}(t_1)$, loss gradient $\partial L / \partial \mathbf{z}(t_1)$

$\frac{\partial L}{\partial t_1} = \frac{\partial L}{\partial \mathbf{z}(t_1)}^\mathsf{T} f(\mathbf{z}(t_1), t_1, \theta)$ ▷ Compute gradient w.r.t. t_1

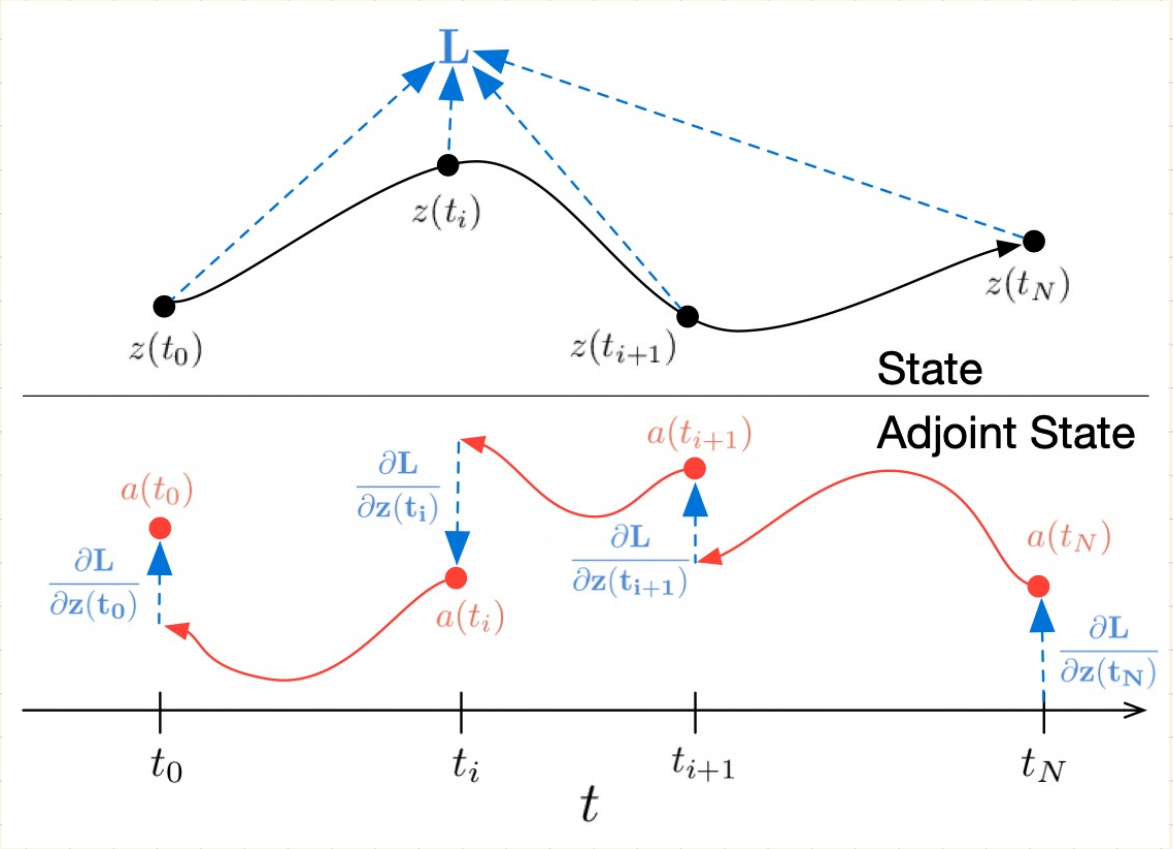
$s_0 = [\mathbf{z}(t_1), \frac{\partial L}{\partial \mathbf{z}(t_1)}, \mathbf{0}_{|\theta|}, -\frac{\partial L}{\partial t_1}]$ ▷ Define initial augmented state

def aug_dynamics($[\mathbf{z}(t), \mathbf{a}(t), \cdot, \cdot], t, \theta$): ▷ Define dynamics on augmented state

return $[f(\mathbf{z}(t), t, \theta), -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial \mathbf{z}}, -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial \theta}, -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial t}]$ ▷ Compute vector-Jacobian products

$[\mathbf{z}(t_0), \frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial t_0}] = \text{ODESolve}(s_0, \text{aug\_dynamics}, t_1, t_0, \theta)$ ▷ Solve reverse-time ODE

return $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial t_0}, \frac{\partial L}{\partial t_1}$ ▷ Return all gradients


L
$z(t_i)$
$z(t_N)$
$z(t_0)$
$z(t_{i+1})$
State
Adjoint State
$a(t_0)$
$\frac{\partial \mathbf{L}}{\partial \mathbf{z}(\mathbf{t_0})}$
$\frac{\partial \mathbf{L}}{\partial \mathbf{z}(\mathbf{t_i})}$
$a(t_i)$
$a(t_{i+1})$
$\frac{\partial \mathbf{L}}{\partial \mathbf{z}(\mathbf{t_{i+1}})}$
$a(t_N)$
$\frac{\partial \mathbf{L}}{\partial \mathbf{z}(\mathbf{t_N})}$
t_0
t_i
t_{i+1}
t_N
t

Resources

https://github.com/rtqichen/torchdiffeq

https://github.com/diffeqml/torchdyn