

Neighborhood-aware training of NeuralODEs to (accurately) predict dynamical invariants of chaotic systems

On going work with Andrus Giraldo & Prof. Deok-sun Lee

2025/05/30

KIAS CAINS Workshop

Joon-Hyuk Ko

Center for AI and Natural Sciences, Korea Institute for Advanced Study

Contents

1. Introduction

Data-driven dynamics discovery

Neural Ordinary Differential Equations

Previous work

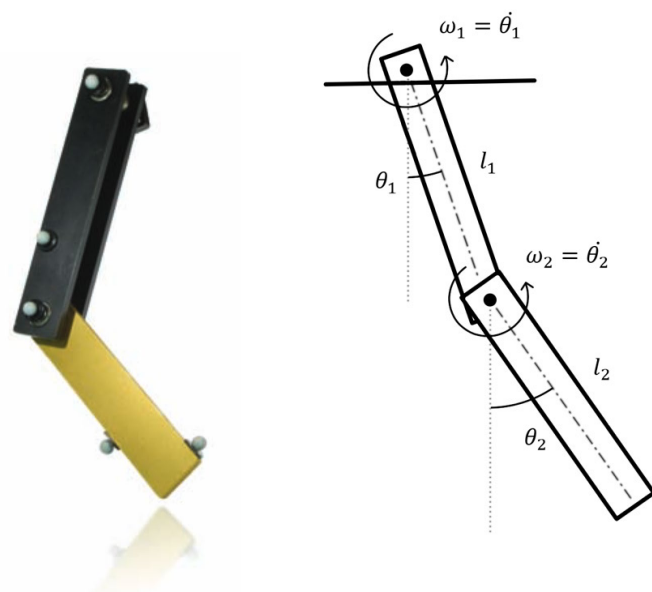
2. Learning chaotic time series data

3. Neighborhood-aware training for Neural ODEs

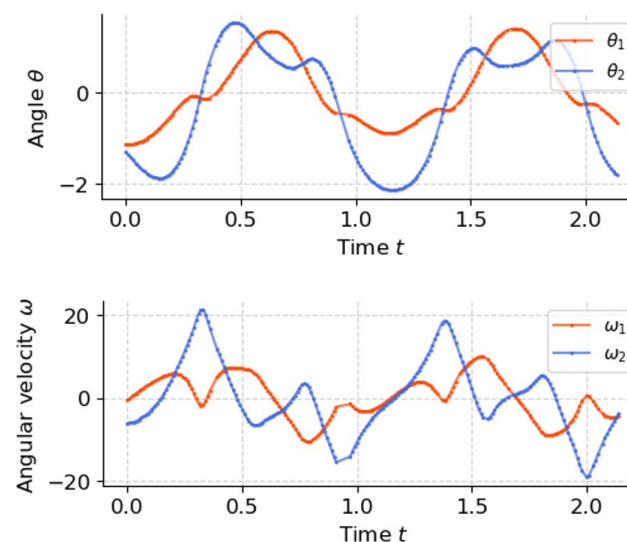
4. Preliminary results and outlook

Data-driven discovery of dynamical systems

Physical system



State observations



Discovered dynamics

$$\begin{aligned} \frac{d\theta_1}{dt} &= \omega_1; \frac{d\theta_2}{dt} = \omega_2; \Delta\theta = \theta_2 - \theta_1 \\ \frac{d\omega_1}{dt} &= \frac{m_2 l_1 \omega_1^2 \sin \Delta\theta \cos \Delta\theta + m_2 l_2 \omega_2^2 \sin \Delta\theta + m_2 g \sin \theta_2 \cos \Delta\theta - (m_1 + m_2) g \sin \theta_1}{(m_1 + m_2) l_1 - m_2 l_1 \cos^2 \Delta\theta} \\ \frac{d\omega_2}{dt} &= -\frac{m_2 l_2 \omega_2^2 \sin \Delta\theta \cos \Delta\theta + (m_1 + m_2) (-l_1 \omega_1^2 \sin \Delta\theta + g \sin \theta_1 \cos \Delta\theta - g \sin \theta_2)}{(m_1 + m_2) l_2 - m_2 l_2 \cos^2 \Delta\theta} \end{aligned}$$

1. Discover new science

2. Forecast future dynamics

3. Use as dynamics surrogates

Autoregressive time-series models

Models of the form:

$$x_{i+1} = \mathcal{F}(t_i, x_i; \theta)$$

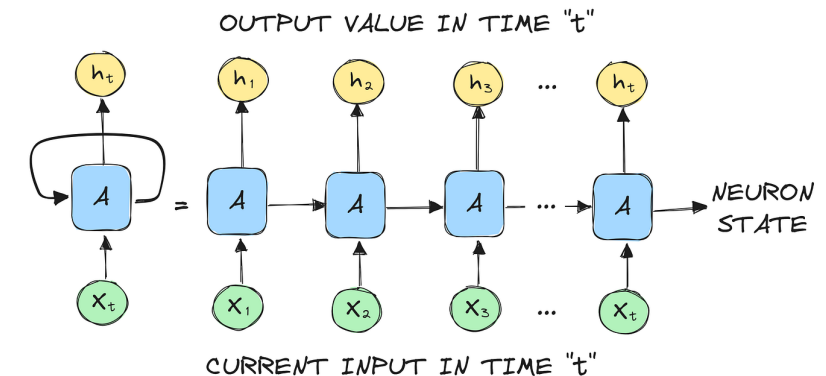
x_i : State prediction at time t_i
 θ : Trainable model parameters

Training : Generate predictions $\{x_i\}_{i=0}^N$ from initial condition x_0 , tune θ to minimize trajectory error

Ex 1) Parametrized ODE models with to-be-determined coefficients

Ex 2) Recurrent Neural Networks (RNNs)

Ex 3) Neural Ordinary Differential Equations (Neural ODEs)



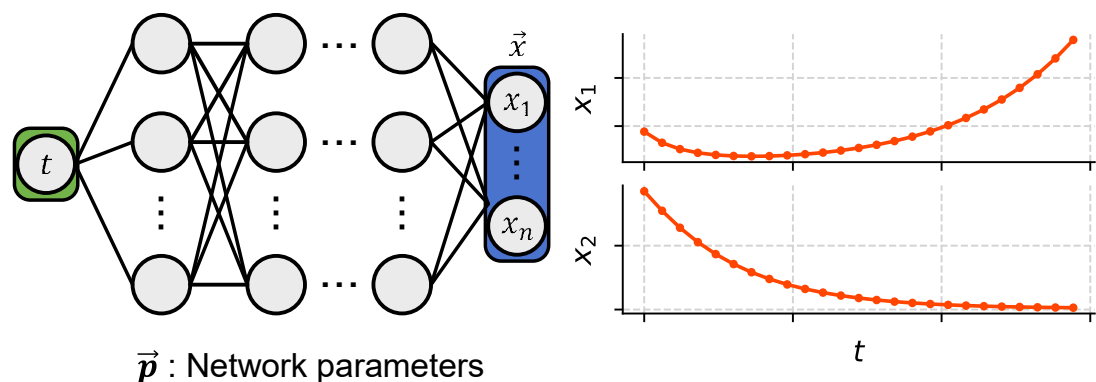
$$h_i = RNN(h_{i-1}, x_i; \theta)$$

Neural Ordinary Differential Equation

A Neural ODE is given by,

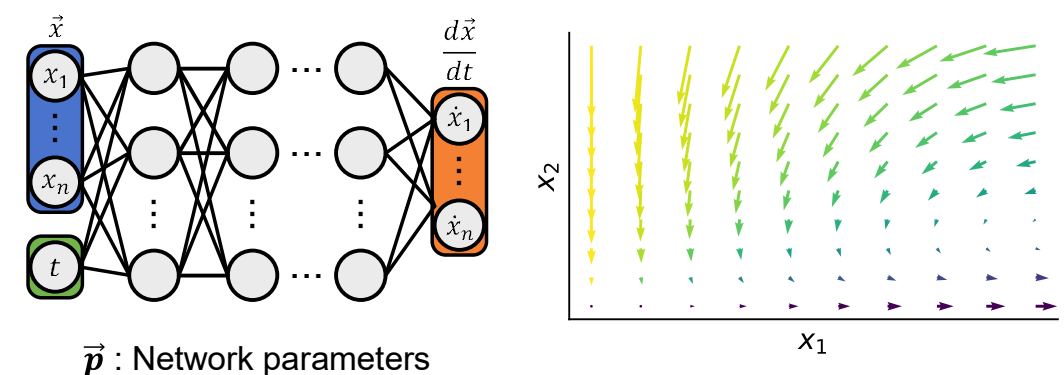
$$\mathbf{x}(t = 0) = \mathbf{x}_0, \quad \frac{d\mathbf{x}}{dt}(t) = f_{NN}(t, \mathbf{x}(t); \boldsymbol{\theta})$$

Neural network : $\vec{x}(t) = NN(t; \vec{p})$



→ models the state trajectory

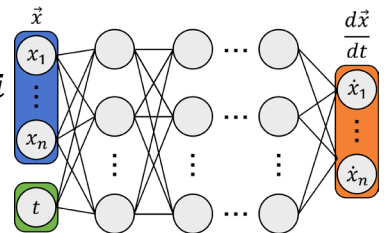
Neural ODE : $\frac{d\vec{x}(t)}{dt} = NN(\vec{x}, t; \vec{p})$



→ models the state dynamics

Obtaining model predictions

Cannot analytically integrate neural networks \rightarrow Use a numerical ODE solver algorithm!

$$\mathbf{u}(t_i) = \mathbf{u}_0 + \int_0^{t_i} \mathbf{NN}(\mathbf{t}, \mathbf{u}; \boldsymbol{\theta}) dt \approx \text{ODESolve}[\mathbf{NN}(\mathbf{t}, \mathbf{u}; \boldsymbol{\theta}), \mathbf{u}_0, t_i]$$


Euler method:

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \mathbf{NN}(n\Delta t, \mathbf{u}_n; \boldsymbol{\theta}) \cdot \Delta t$$

Runge-Kutta 4th order method:

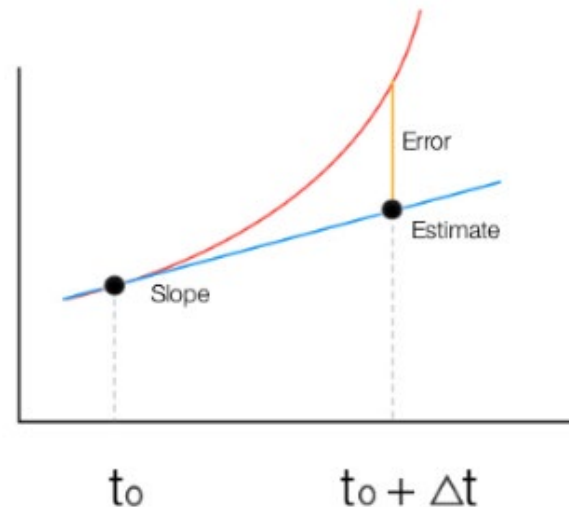
$$\mathbf{u}_{n+1} = \mathbf{u}_n + \frac{1}{6} \cdot (\mathbf{k}_1 + \mathbf{k}_2 + \mathbf{k}_3 + \mathbf{k}_4) \cdot \Delta t$$

$$\mathbf{k}_1 = \mathbf{NN}(n\Delta t, \mathbf{u}_n; \boldsymbol{\theta})$$

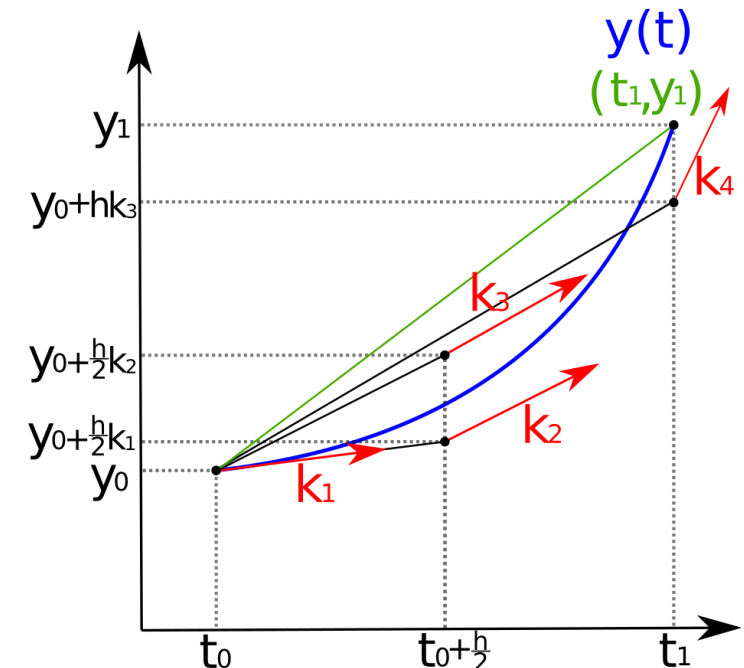
$$\mathbf{k}_2 = \mathbf{NN}\left(\left(n + \frac{1}{2}\right)\Delta t, \mathbf{u}_n + \mathbf{k}_1 \cdot \frac{\Delta t}{2}; \boldsymbol{\theta}\right)$$

$$\mathbf{k}_3 = \mathbf{NN}\left(\left(n + \frac{1}{2}\right)\Delta t, \mathbf{u}_n + \mathbf{k}_2 \cdot \frac{\Delta t}{2}; \boldsymbol{\theta}\right)$$

$$\mathbf{k}_4 = \mathbf{NN}((n + 1)\Delta t, \mathbf{u}_n + \mathbf{k}_3 \cdot \Delta t; \boldsymbol{\theta})$$



<https://www.haroldserrano.com/blog/visualizing-the-runge-kutta-method>



<https://lowebms.readthedocs.io/en/latest/code/rk4.html>

Training Neural ODEs

Trajectory prediction

Loss calculation

$$\frac{d\vec{x}(t)}{dt} = NN(\vec{x}, t; \vec{\theta})$$

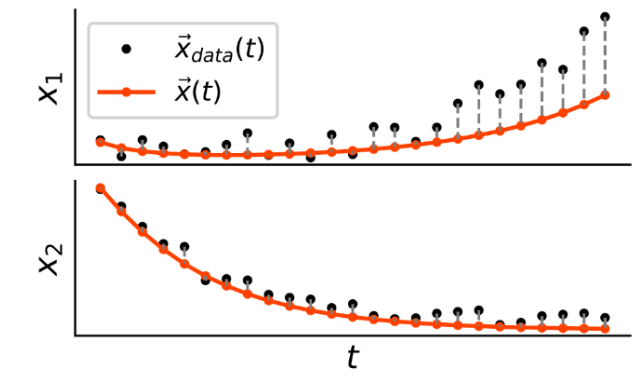
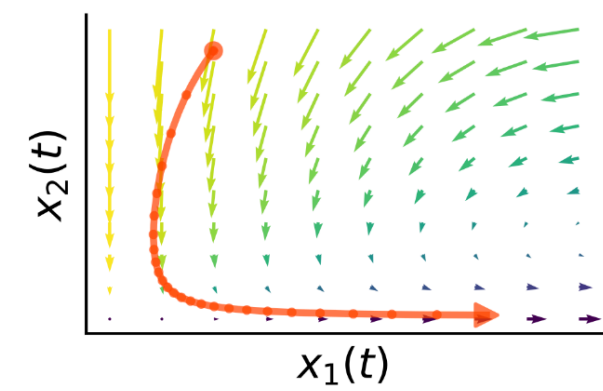
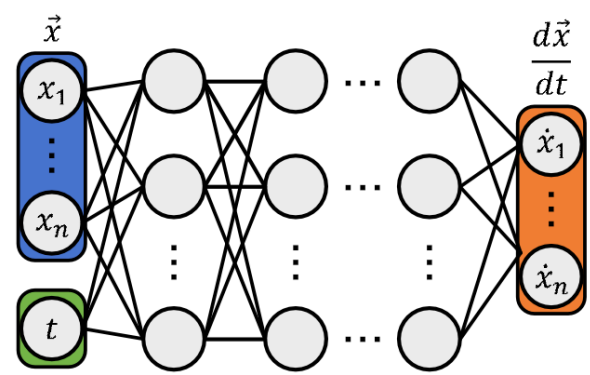


$$\vec{x}(t) = \vec{x}(t_0) + \int_{t_0}^t \text{NN}(\vec{x}, t; \vec{\theta}) dt$$

$$\approx \text{ODESolve}(NN(\vec{x}, t; \vec{\theta}), \vec{x}_0)$$



$$\mathcal{L}(\vec{\theta}) = \sum_{t_i} \{\vec{x}(t_i; \vec{\theta}) - \vec{x}_{data}(t_i)\}^2$$



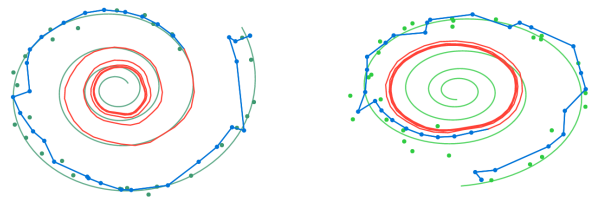
Parameter update (gradient descent)

$$\vec{\theta} \leftarrow \vec{\theta} - \eta \nabla_{\vec{\theta}} \mathcal{L}(\vec{\theta})$$

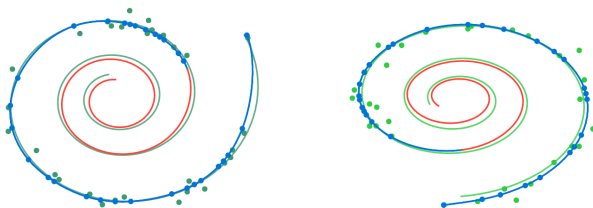


Why Neural ODEs?

- A natural choice to model continuously varying phenomena
- Handling missing / irregular time points is trivial

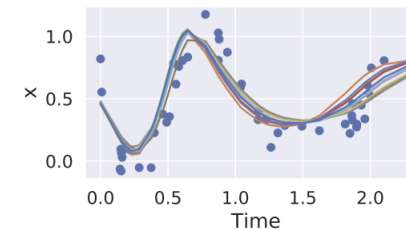


(a) Recurrent Neural Network

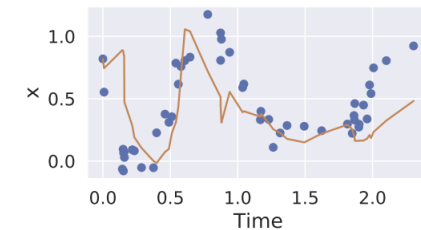


(b) Latent Neural Ordinary Differential Equation

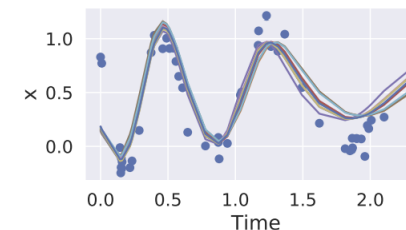
R. T. Q. Chen *et al.* *NeurIPS* **31** (2018).



(a) Latent ODE (ODE enc)



(d) Standard RNN



Y. Rubanova *et al.* *NeurIPS* **32** (2019).

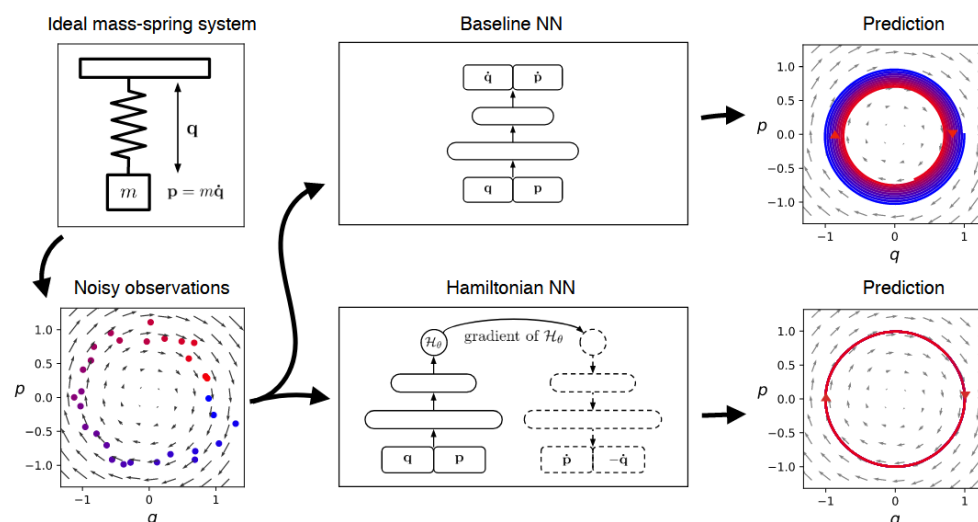
Why Neural ODEs?

- Functional form of the differential equation is always simpler than that of its solution

$$\dot{x} = Ax \quad \text{vs} \quad x(t) = e^{At}$$

- Straightforward to incorporate partial knowledge of the system dynamics

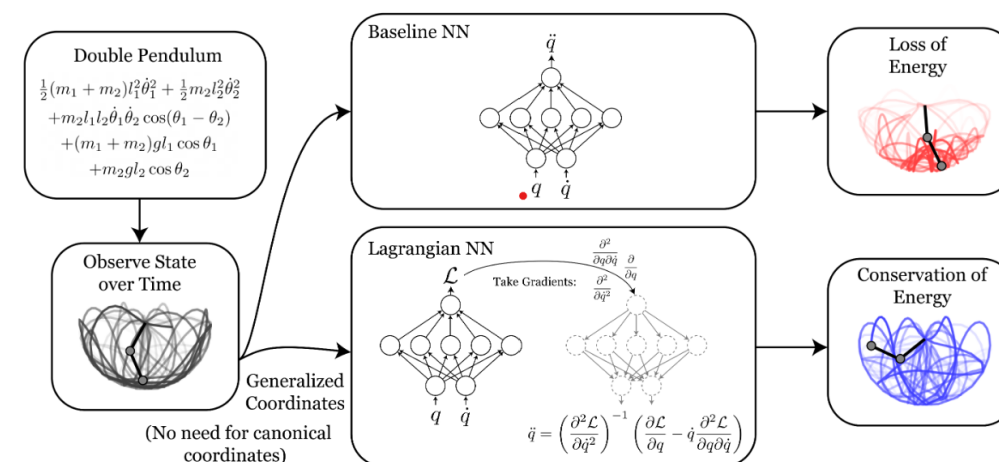
Hamiltonian Neural Network



Hamilton's equations: $\dot{q} = \frac{\partial \mathcal{H}}{\partial p}, \dot{p} = -\frac{\partial \mathcal{H}}{\partial q}$

S. Greydanus et al. *NeurIPS* 32 (2019).

Lagrangian Neural Network



Euler-Lagrange equation: $\frac{\partial \mathcal{L}}{\partial q} - \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{q}} \right) = 0$

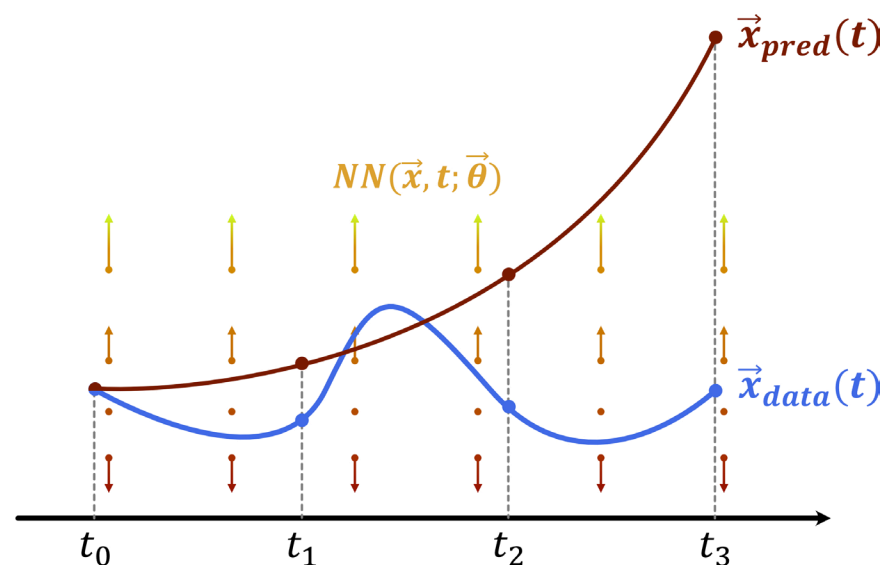
M. Cranmer et al. *ICML Workshop on Integration of Deep Neural Models and Differential Equations* (2020).

Pre-KIAS: Stabilize Neural ODE training on long data

Unknown data dynamics: $\frac{d\vec{x}_{data}}{dt} = \mathbf{F}(t, \vec{x}_{data})$

• Uncoupled

Model: $\frac{d\vec{x}_{pred}}{dt} = NN(t, \vec{x}_{pred}; \vec{\theta})$



• Synchronized

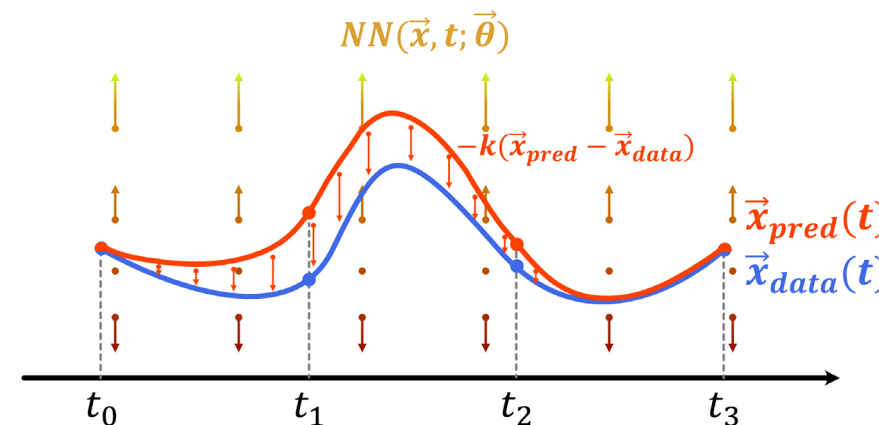
Model: $\frac{d\vec{x}_{pred}}{dt} = NN(t, \vec{x}_{pred}; \vec{\theta}) - k(\vec{x}_{pred} - \vec{x}_{data})$



Coupling term



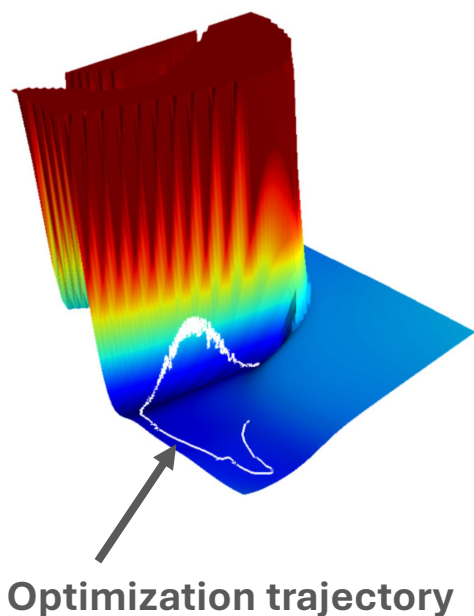
For a large $k > 0$, $\|\vec{x}_{pred} - \vec{x}_{data}\|_2 \rightarrow 0$ as $t \rightarrow \infty$



- A proportional coupling term can synchronize the model dynamics with the data trajectory
- Prevents the untrained model from diverging away from data and produces better loss landscapes

Pre-KIAS: Stabilize Neural ODE training on long data

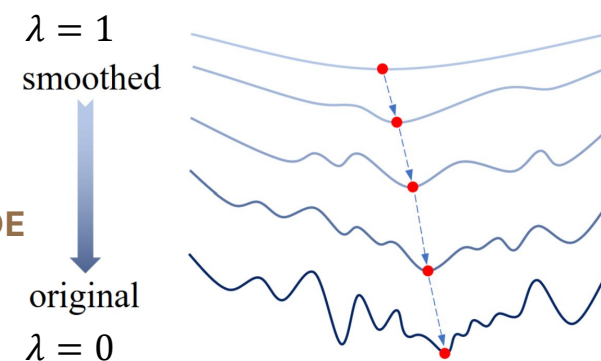
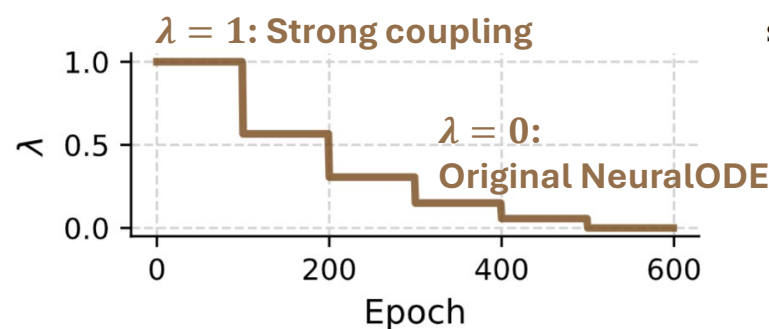
• Vanilla training



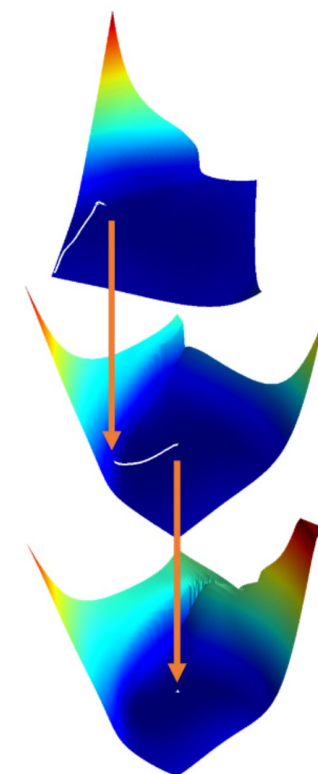
• Homotopy optimization

$$\frac{d\vec{x}_{pred}}{dt} = NN(t, \vec{x}_{pred}; \vec{\theta}) - \lambda k(\vec{x}_{pred} - \vec{x}_{data})$$

Homotopy parameter



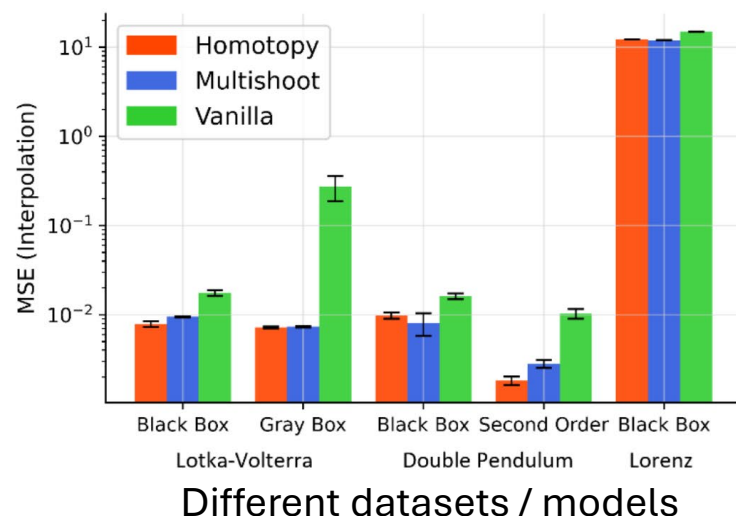
Adapted from: X. Lin et al. *ICML* (2023).



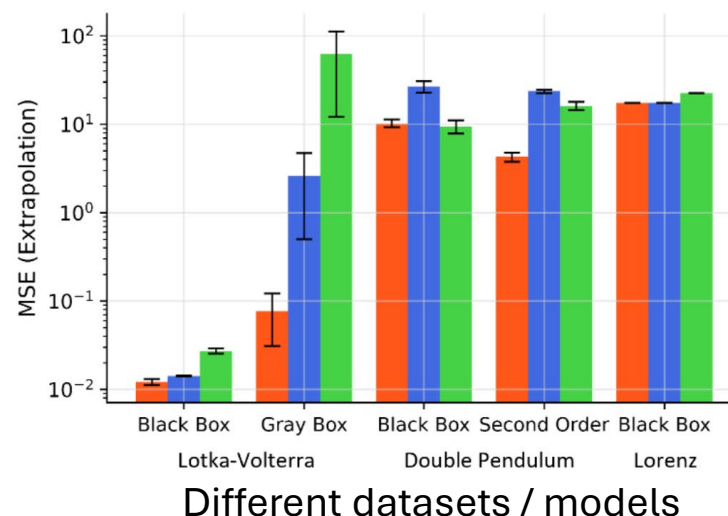
- Start with a simple proxy problem, and continuously transform back to the original difficult problem
- Effectively avoids bad local minima

Pre-KIAS: Stabilize Neural ODE training on long data

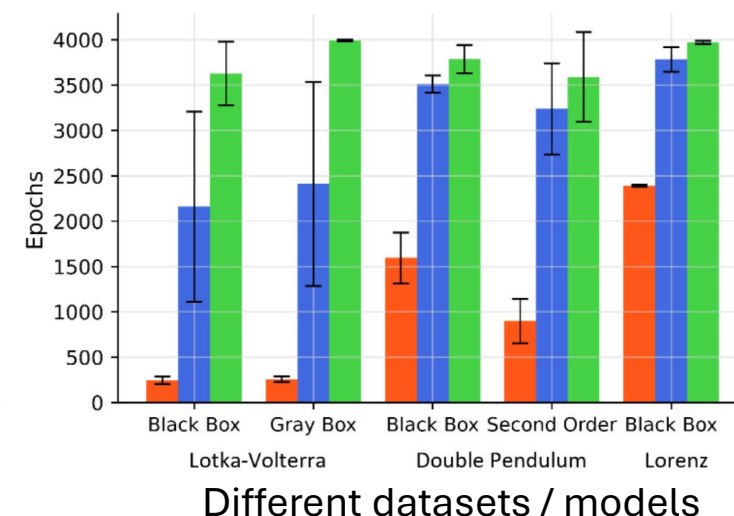
- Interpolation Error



- Extrapolation Error



- Training epochs



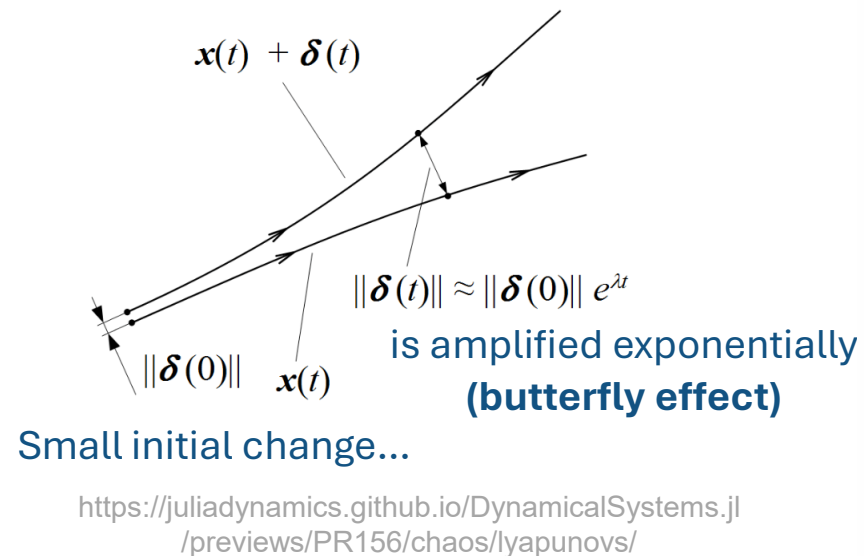
[J. H. Ko et al. NeurIPS \(2023\).](#)

- Models trained with the **homotopy method** have stronger interpolation / extrapolation performance
- Required training epochs are also greatly reduced

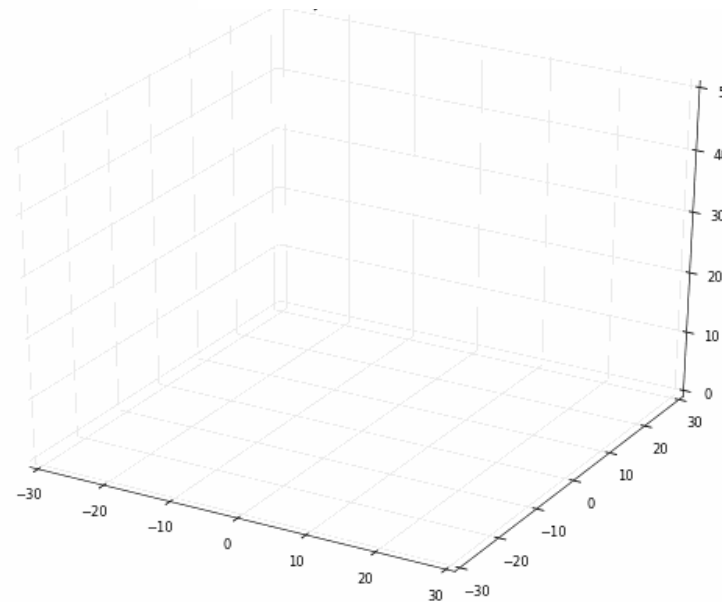
Learning chaotic time series data

Chaotic systems

Chaos: “Aperiodic long-term behavior in a deterministic system that exhibits sensitive dependence on initial conditions” - S. Strogatz, *Nonlinear Dynamics and Chaos*, Third Edition (2024).



Lorenz equations



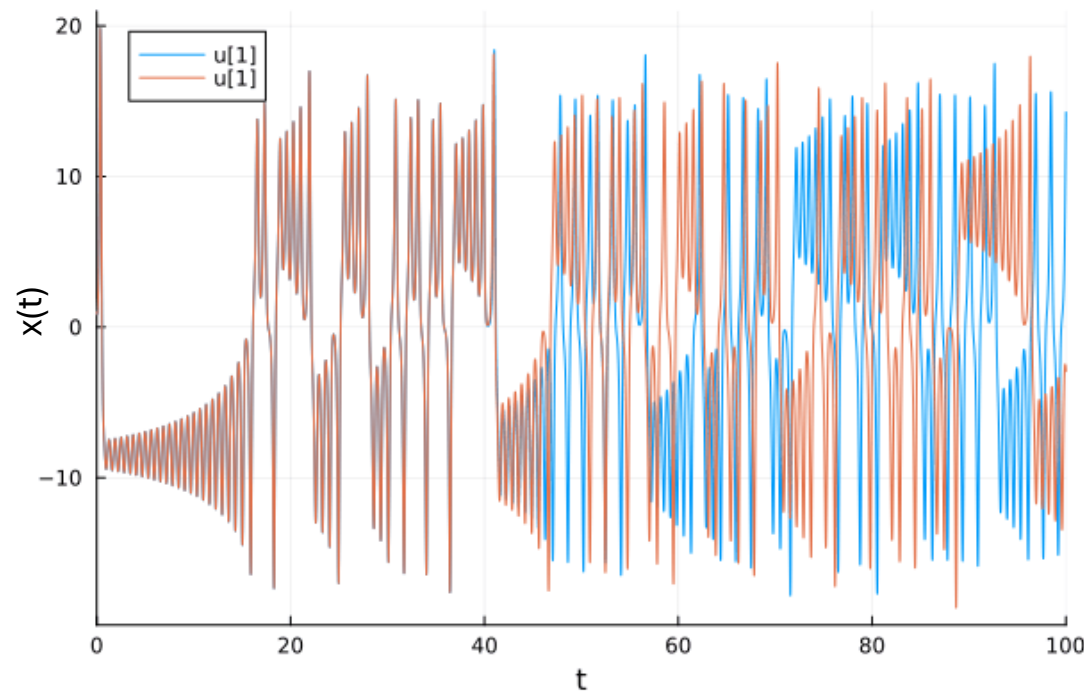
$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z\end{aligned}$$

Ubiquitous in nature: weather data, fluid flow, population dynamics, cardiac signals, etc..

Impossibility of exact long-term predictions

Chaotic dynamics are sensitive to changes in the equation parameters as well

Ex) Lorenz system trajectories with slightly different parameters



$$\dot{x} = \sigma(y - x)$$

$$\dot{y} = x(\rho - z) - y$$

$$\dot{z} = xy - \beta z$$

$$(\sigma, \rho, \beta) = \left(10, 28.0, \frac{8}{3}\right)$$

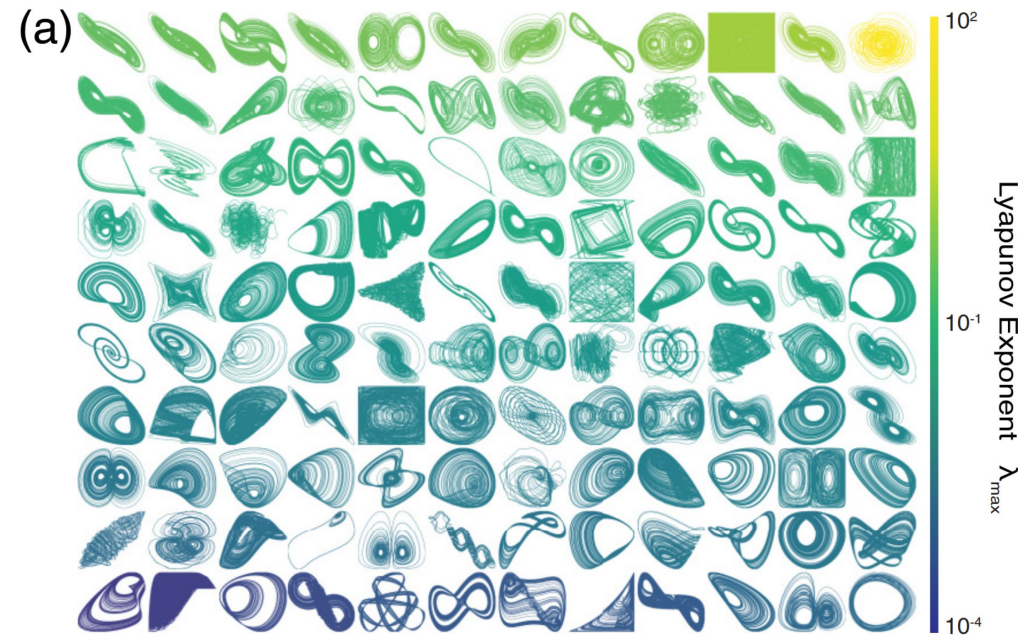
$$(\sigma, \rho, \beta) = \left(10 + 10^{-15}, 28.0, \frac{8}{3}\right)$$

Even if the trained model is very close to the ground truth, exact trajectory predictions fail after some characteristic time

Dynamic invariants of chaotic systems

However, chaotic systems have well-defined long-term, global characteristics

Ex) Strange attractors: Fractal structure in phase space that all trajectories converge onto after some finite time



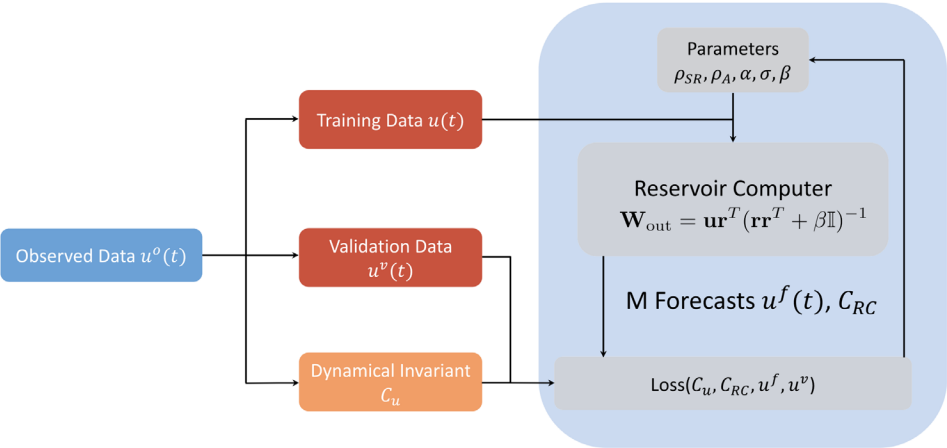
Ex 2) Statistics computed over points on the attractor / along long temporal trajectories: Well-defined, invariant with respect to the dynamics

Goal of chaotic time series learning: Train models that are “statistically accurate”

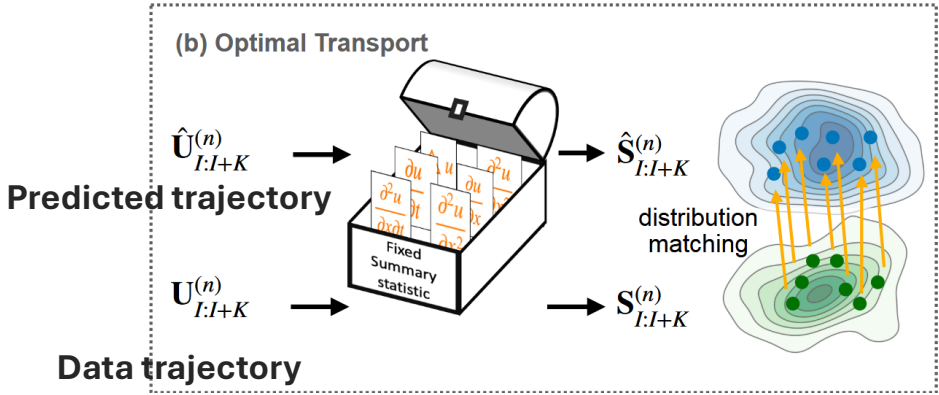
Explicitly using dynamical invariants in training

Dynamical invariants can be explicitly incorporated into model training to enhance model performance

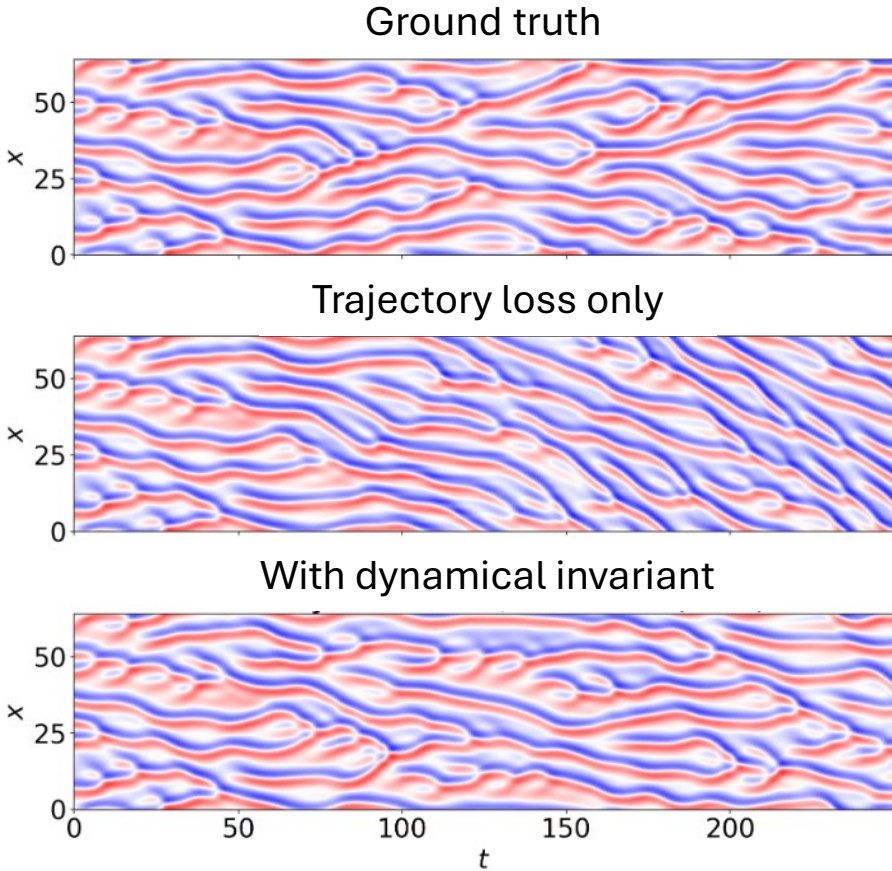
Total loss = Trajectory loss + Dynamical invariant loss



J. Platt et al. *Chaos* **33** 103107 (2023).



R. Jiang et al. *NeurIPS* (2023).



Y. Schiff et al. *ICML* (2024).

Possible alternative: adding Jacobian information

For dynamics $\frac{dx}{dt} = f(x)$, the Jacobian $J(x) = \nabla_{x'} f(x')|_{x'=x(t)}$ governs the dynamics of small perturbations $\delta x(t)$ around the state $x(t)$:

(Variational equation)

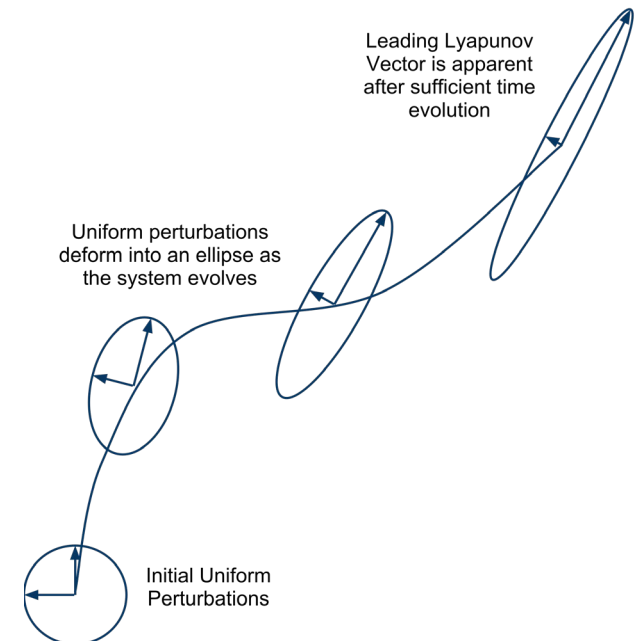
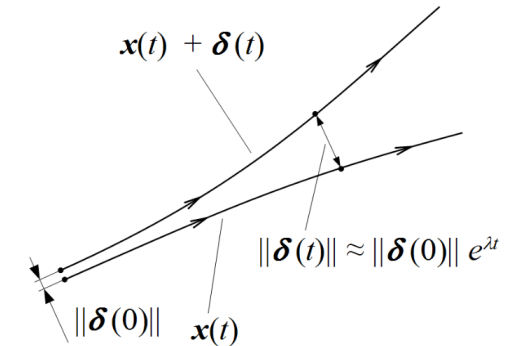
$$\frac{d\delta x}{dt} = \frac{d(x + \delta x)}{dt} - \frac{dx}{dt} = f(x + \delta x) - f(x) = J(x)\delta x + o(\delta x)$$

Maximum Lyapunov exponent $\lambda_1 = \lim_{t \rightarrow \infty} \frac{1}{t} \log \frac{\|\delta x(t)\|_2}{\|\delta x(0)\|_2}$:

→ quantifies sensitivity to initial conditions

For n-D dynamics, there are n Lyapunov exponents $\lambda_1 \geq \dots \geq \lambda_n$

Dynamical invariants!



Possible alternative: adding Jacobian information

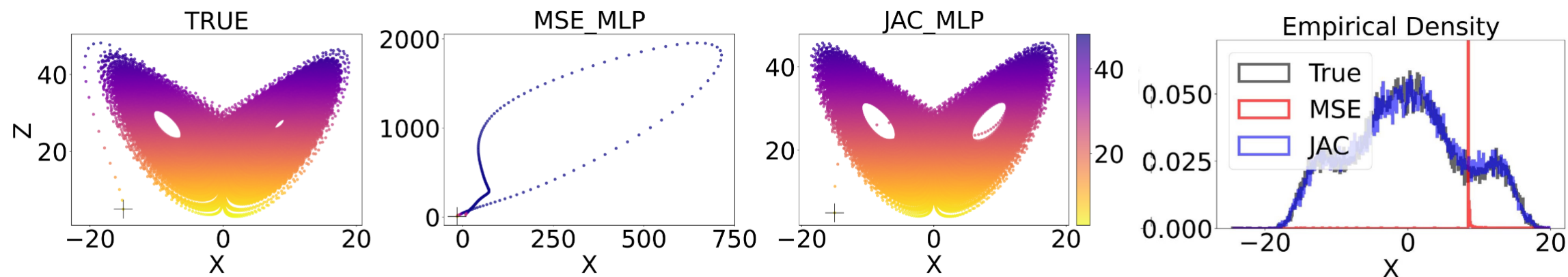
For 1 step training, explicitly providing Jacobian information results in statistically accurate models

$$\mathcal{L}_{MSE}(\theta) = \frac{1}{n} \sum_{i=0}^{n-1} \|x_{i+1} - \tilde{x}_{i+1}\|^2$$

$$\tilde{x}_{i+1} = x_i + \int_{t_i}^{t_{i+1}} NN(x(t); \theta) dt : \text{1-step model predictions}$$

$$\mathcal{L}_{JAC}(\theta) = \mathcal{L}_{MSE}(\theta) + \frac{1}{n} \sum_{i=0}^{n-1} \|J(x_i) - \tilde{J}(x_i)\|_F$$

$$\tilde{J}(x_i) = \nabla_{x'} NN(x'; \theta)|_{x'=x(t)} : \text{Jacobian of the model dynamics}$$



J. Park et al. *NeurIPS* 38 (2024).

No noise, ground truth Jacobian supplied → Can we extend this to a more realistic setting?

Neighborhood-aware training for Neural ODEs

Learning Jacobians from measured data

Given time series data $\{t_i, \mathbf{x}_i\}_{i=0}^n$ generated from $\frac{dx}{dt} = f(\mathbf{x})$, how do we estimate $J(\mathbf{x}_i)$?

Widely researched in the chaos literature from the 80s; two major methods exist

Global method:

Train an autoregressive model $\tilde{\mathbf{x}}_{i+1} = \hat{f}(\tilde{\mathbf{x}}_i; \boldsymbol{\theta})$ to minimize the trajectory error $\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n+1} \sum_i (\mathbf{x}_i - \tilde{\mathbf{x}}_i)^2$

After training, estimate via $\nabla_{\mathbf{x}} \hat{f}(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_i} \approx I + J(\mathbf{x}_i) \Delta t_i$

(+) Only single global model is trained; Basically identical to conventional model training

(-) Learning function values from sampled points do not guarantee accurate derivatives, especially for noisy data

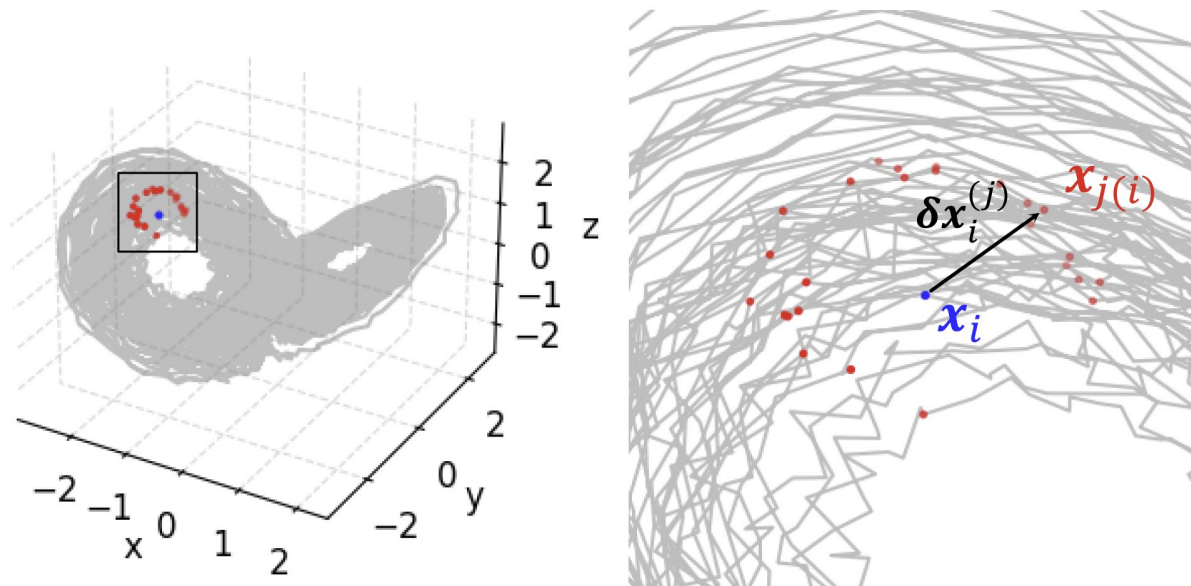
Learning Jacobians from measured data

Given time series data $\{t_i, \mathbf{x}_i\}_{i=0}^n$ generated from $\frac{d\mathbf{x}}{dt} = f(\mathbf{x})$, how do we estimate $J(\mathbf{x}_i)$?

Local method:

Inspired by the variational equation $\frac{d\delta\mathbf{x}}{dt} = J(\mathbf{x})\delta\mathbf{x} + \mathcal{O}((\delta\mathbf{x})^2)$ $\delta\mathbf{x}$: Infinitesimal perturbations around state \mathbf{x}

Perturbations $\delta\mathbf{x}$ are estimated using nearby neighbors in phase space:



Learning Jacobians from measured data

Local method:

For each $i = 0, 1, \dots, n$

Train a local model $\widetilde{\delta x}_{i,1}^{(j)} = \widehat{Df}_i(\delta x_{i,0}^{(j)}; \theta_i)$ by minimizing the 1-step evolution of the neighbor distances:

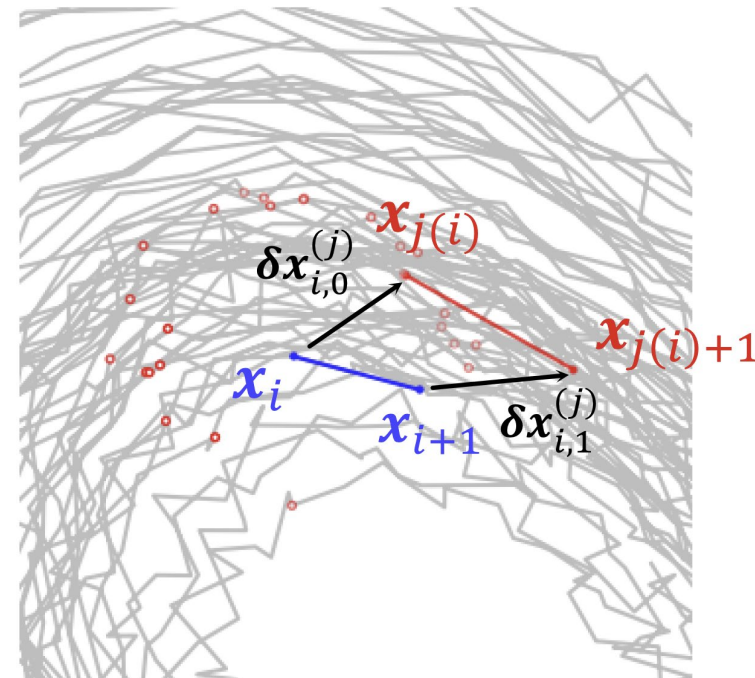
$$\mathcal{L}(\theta_i) = \frac{1}{n_{\text{neighbors}}} \sum_j \left(\delta x_{i,1}^{(j)} - \widetilde{\delta x}_{i,1}^{(j)}(\theta_i) \right)^2$$

For local linear models $\widehat{Df}_i = A_i$,

$$A_i \approx I + J(x_i)\Delta t_i$$

(+) Models are directly conditioned on the dynamics generated by the Jacobian

(-) Requires fitting one model per single point estimation of the Jacobian



Neighborhood-aware training

Goal: For $\dot{\mathbf{x}} = f(\mathbf{x})$, train model to satisfy both $f_{NN}(\mathbf{x}; \boldsymbol{\theta}) \approx f(\mathbf{x})$ and $\nabla_{\mathbf{x}} f_{NN}(\mathbf{x}; \boldsymbol{\theta}) \approx \nabla_{\mathbf{x}} f(\mathbf{x})$

0. Prior to training, identify the indices of the neighborhood points $\{\mathbf{x}_{j(i)}\}_{j=1}^{n_{neighbors}}$ for each data point \mathbf{x}_i

1. Generate predictions from the neural ODE using

$$\frac{d\tilde{\mathbf{x}}}{dt} = f_{NN}(\tilde{\mathbf{x}}; \boldsymbol{\theta})$$

$$\tilde{\mathbf{x}}(0) = \mathbf{x}_i$$

$$\frac{d\tilde{\boldsymbol{\delta}}\mathbf{x}}{dt} = \nabla_{\mathbf{x}} f_{NN}(\tilde{\mathbf{x}}; \boldsymbol{\theta}) \cdot \tilde{\boldsymbol{\delta}}\mathbf{x}$$

$$\tilde{\boldsymbol{\delta}}\mathbf{x}(0) = \boldsymbol{\delta}\mathbf{x}_{i,0}^{(j)}$$

Computed from $f_{NN}(\tilde{\mathbf{x}}; \boldsymbol{\theta})$ using automatic differentiation!

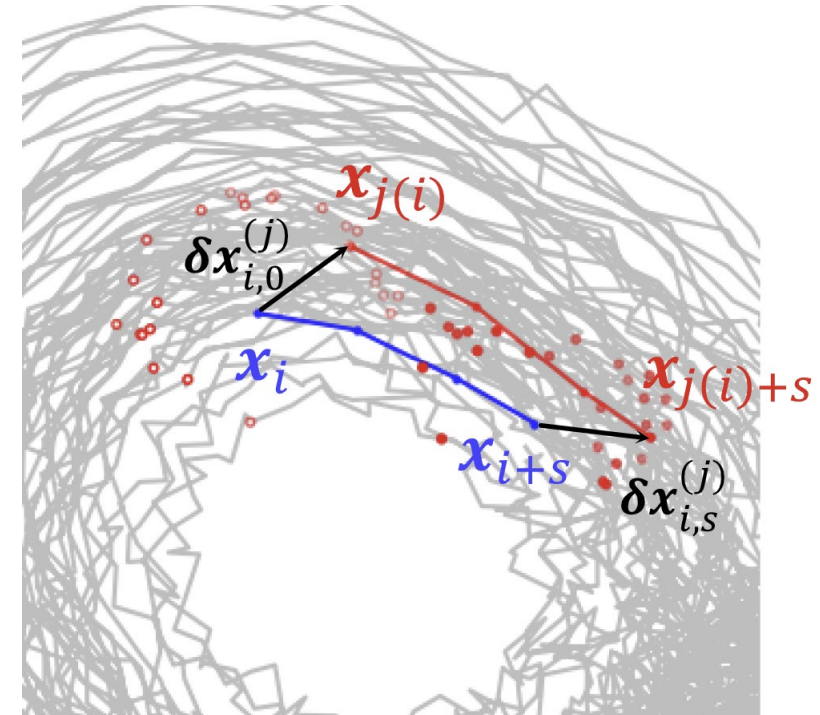
Taylor mode autodiff for higher order expansion also possible

2. Minimize the loss function

$$L(\boldsymbol{\theta}) = \sum_i \sum_{k=1}^s \left(\|\mathbf{x}_{i+k} - \tilde{\mathbf{x}}_{i+k}\|_2^2 + \frac{1}{n_{neighbors}} \sum_{j=1}^{n_{neighbors}} \left\| \mathbf{x}_{j(i)+k} - \tilde{\mathbf{x}}_{i+k} - \tilde{\boldsymbol{\delta}}\mathbf{x}_{i,k}^{(j)} \right\| \right)$$

Trajectory loss

Neighborhood loss

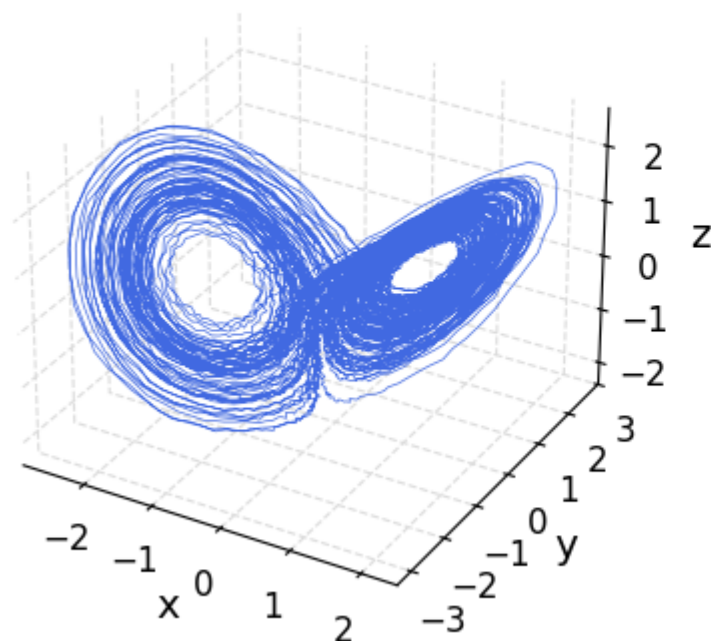


Preliminary results & Outlook

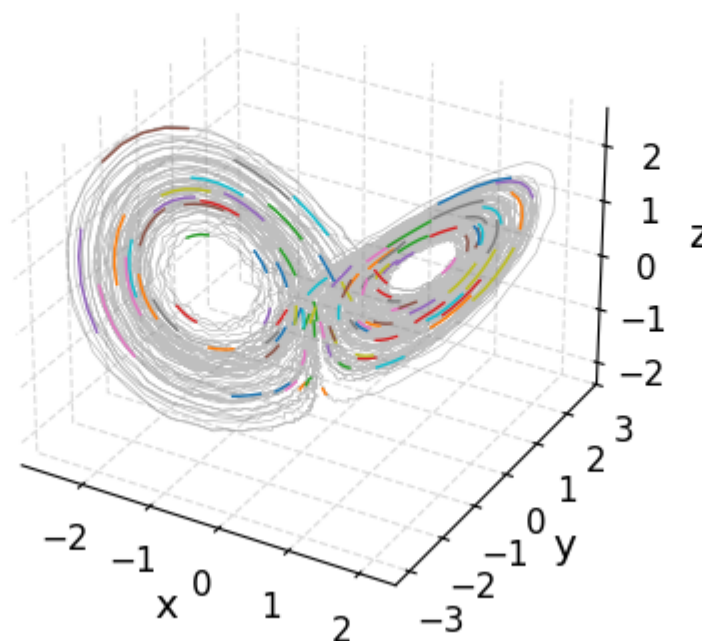
Preliminary results

Models were trained with 5-step segments, and 25 neighbors were used for our method

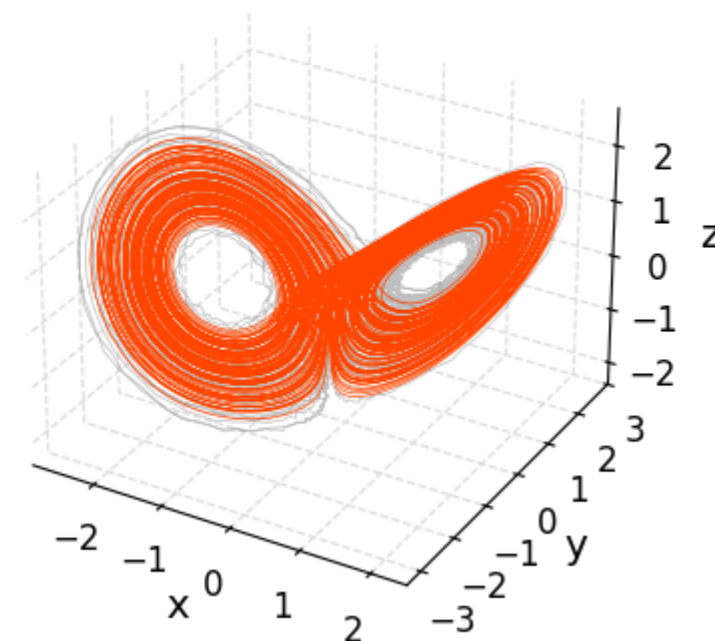
Train data



Train-time prediction

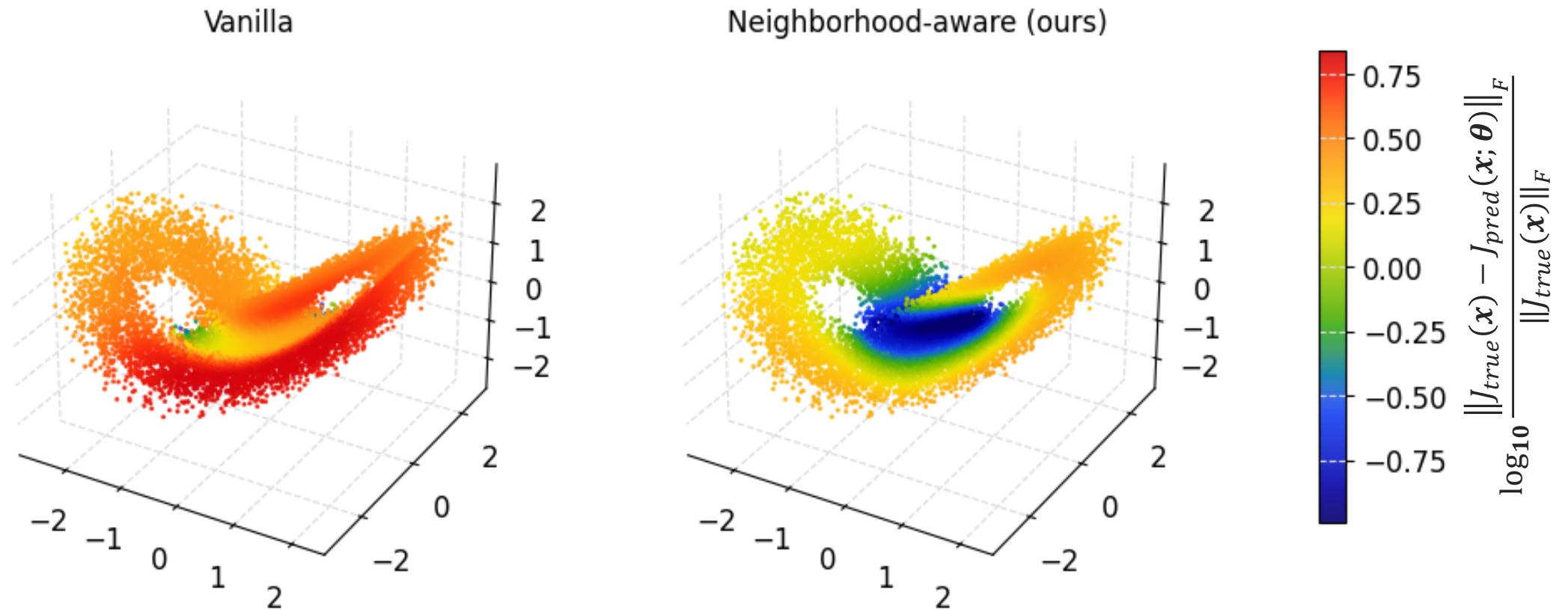


Inference-time prediction



Preliminary results

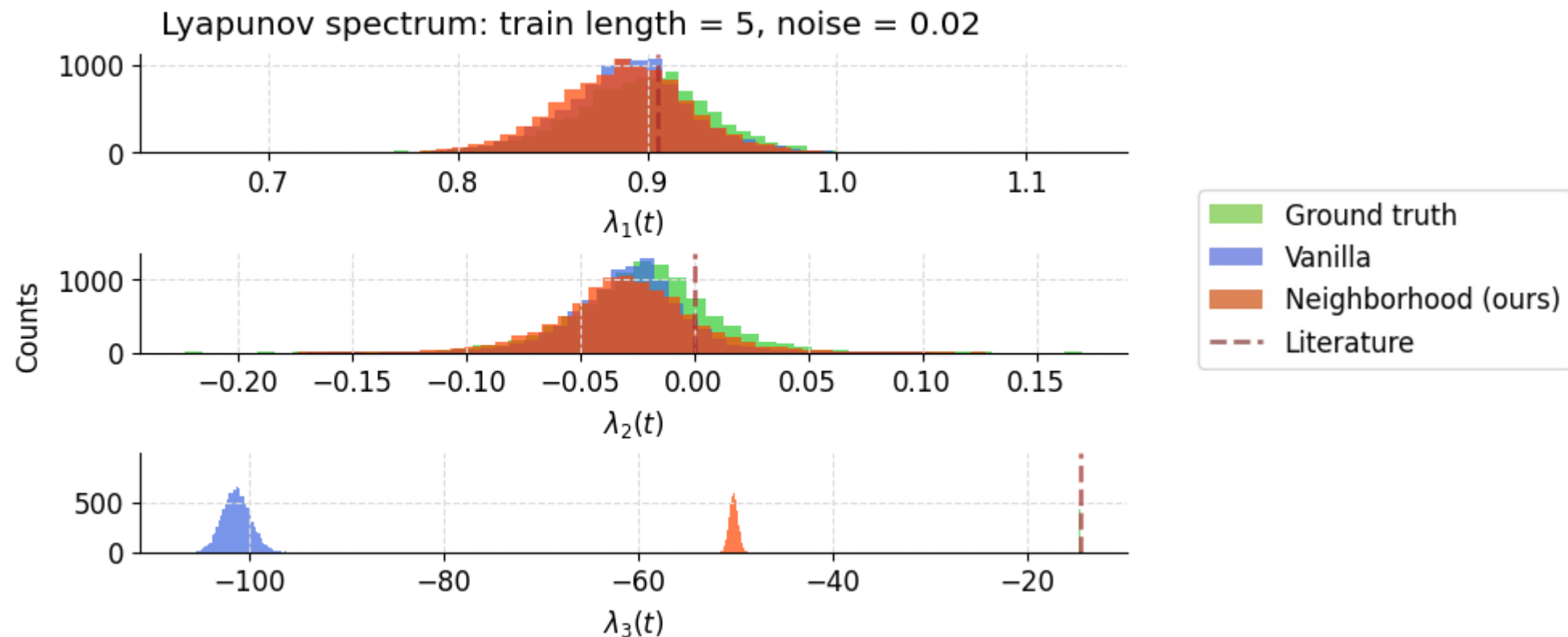
The learned Jacobians were compared against the ground truth analytical Jacobian



Better Jacobians recovered!

Assessing statistical accuracy

To gauge statistical accuracy, the Lyapunov spectrum of the trained models were estimated



Despite better Jacobians, improvement in statistical accuracy is marginal!

Summary & Outlook

- For chaotic systems, training models that are statistically accurate is important
- Training models to properly learn the unknown dynamics and its Jacobian has the potential to produce better models
- Current neighborhood-aware training does produce more accurate Jacobians, but improvement in model performance is marginal
- Further algorithmic improvements & investigations are to be conducted
 - Optimal selection of neighbors
 - Higher order Taylor expansions for neighborhood dynamics
 - Different loss functions

Thank you