Dynamical systems and machine learning: combining in a principled way data-driven models and domain-driven models

Michael W. Mahoney
ICSI and Department of Statistics, UC Berkeley

September 2020

(Joint work with Benjamin Erichson, Alejandro Queiruga, Dane Taylor, Michael Muehlebach, and others.)

Outline

Paradigms of Modeling Complex Systems

Statistical Modelling
(Data/Theory-driven)

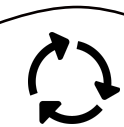
$$y = Ax + \epsilon$$

- Interpretable
- Strong assumptions
- Low expressivity

Machine Learning
(Data-driven)

$$y = F(x) + \epsilon$$

- Black-box
- No assumptions
- High expressivity



Dynamical Systems
(Theory-driven)

$$\dot{y} = f(t, y, x)$$

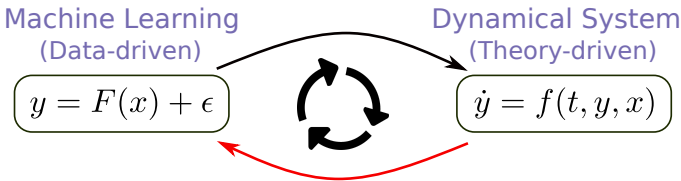
- Gray-box
- Noise ?!
- Robust / Stable

What can we learn from dynamical systems and control theory?

Residual Networks (ResNets)	⟺	Differential Equations
Network Architecture Design	⟺	Numerical Methods
Training	⟺	Optimal Control

Recent Related Mahoney Lab's Research Outcomes

- ML to Dynamical Systems:
  - Shallow neural networks for fluid flow reconstruction with limited sensors (Erichson et al.)



- Ideas from Dynamical Systems to ML:
  - ANODEV2: A coupled neural ODE framework (Gholami et al.)
  - Stochastic normalizing flows (Hodgkinson et al.)
  - Physics-informed autoencoders for lyapunov-stable fluid flow prediction (Erichson et al.)
  - Forecasting sequential data using consistent koopman autoencoders (Azencot et al.)
  - Improving ResNets with a corrected dynamical systems interpretation (Queiruga et al.)
  - Noise-response analysis for rapid detection of backdoors in deep nets (Erichson et al.)

Connection between Deep Learning and Differential Equations

- The essential building blocks of ResNets are so-called residual units.

$$x_{t+1} = \epsilon \cdot x_t + \sigma_t(x_t, \theta_t). \tag{1}$$

- The function $\sigma_t : \mathbb{R}^n \to \mathbb{R}^n$ denotes the t-th residual module (a non-linear map), parameterized by θ_t, which takes a signal $x_t \in \mathbb{R}^n$ as input. ϵ is the step size.
- For simplicity, let's consider a linear unit

$$x_{t+1} = \epsilon \cdot x_t + Ax_t. \tag{2}$$

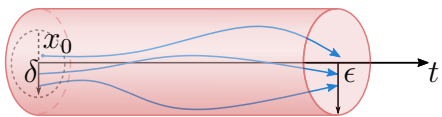
- Through the lens of differential equations, residual units can be seen as a some (!?) discretization scheme for the following ordinary differential equation:

$$\frac{\partial x}{\partial t} = Ax. \tag{3}$$

- This connection between differential equations and residual units can help to study network architecture as well as provide inspiration for the design of new network architectures.

What can we Learn from Dynamical Systems Theory?

- Dynamical systems theory is mainly concerned with describing the **long-term qualitative behavior** of dynamical systems, which typically can be describe as differential equations.
- **Stability theory** plays an essential role in the analysis of differential equation.
- We might be interested to study whether trajectories of a given dynamical systems, under small perturbations of the initial condition x_0, are stable.



- If the dynamics $\frac{\partial x}{\partial t} = Ax$ are linear, stability can be checked with an eigenvalue analysis.
- We can use linearization or input-to-state stability to study nonlinear systems.
- Does stability matter in deep learning? Well, it depends ....
- Feedforward neural networks (FNNs): each residual unit takes only a single step. Thus, stability might not matter?!
- Recurrent neural networks: stability matters! If the recurrent unit is unstable, then we observe exploding gradients. We will discuss this later.

How can we Integrate Prior Physical Knowledge?

- **Option 1: Physics-informed network architectures.** We integrate prior knowledge (e.g., symmetries) via specialized physics-informed layers or convolution kernels.

$$\theta_k = T(W_k) := \beta \cdot (W + W^T) + (1 - \beta) \cdot (W - W^T) \tag{4}$$

- **Option 1: Physics-informed regularizers.** We integrate prior knowledge (e.g., stability) via additional energy terms

$$\min_{\theta} \mathcal{L}(\theta) := \frac{1}{n} \sum_{i=1}^{n} \underbrace{\ell_i(h_\theta(x_i), y_i)}_{\text{Loss}} + \lambda \cdot \underbrace{\mathcal{R}(\theta_k)}_{\text{regularizer}}, \tag{5}$$

- **Option 1: Physics-constrained models.** We integrate prior knowledge (e.g., an ODE model) via additional constraints on the outputs

$$\min_{\theta} \mathcal{L}(\theta) := \frac{1}{n} \sum_{i=1}^{n} \ell_i(h_\theta(x_i), y_i) \quad \text{s.t.} \quad \mathcal{R}(f_\theta(x)) \leq \eta, \tag{6}$$

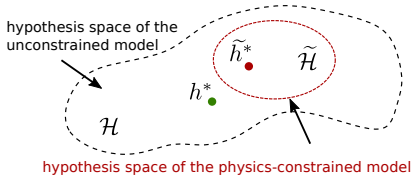
Outline

Physics-constrained learning (PCL)

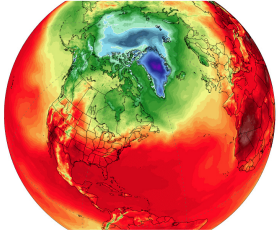
- Supervised ML aims to learn a model $\mathcal{H}$ that best maps a set of inputs $\mathcal{X}$ to a set of outputs $\mathcal{Y}$:

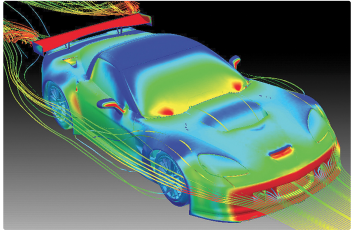
$$\mathcal{H} : \mathcal{X} \to \mathcal{Y}$$

- We hope that this model also works on new inputs.



PCL aims to introduce prior knowledge about the problem into the learning process.





Problem setup: Fluid flow prediction

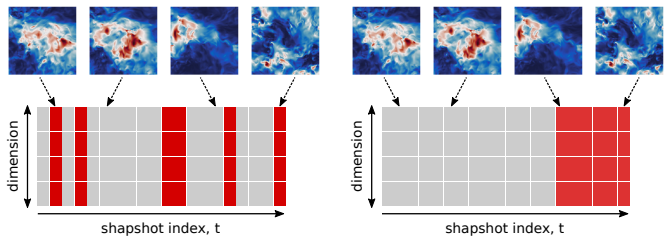
- We assume that the dynamical system of interest can be modeled as

$$\mathbf{x}_{t+1} = \mathcal{A}(\mathbf{x}_t) + \eta_t, \quad t = 0, 1, 2, \ldots, T.$$

- In a data-driven setting we might only have access to (high-dimensional) observations

$$\mathbf{y}_t = \mathcal{G}(\mathbf{x}_t) + \xi_t, \quad t = 0, 1, 2, \ldots, T.$$

- Given a sequence of observations $\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_T \in \mathbb{R}^m$ for training, the objective of this work is to learn a model which maps the snapshot $\mathbf{y}_t$ to $\mathbf{y}_{t+1}$.



Physics-agnostic model

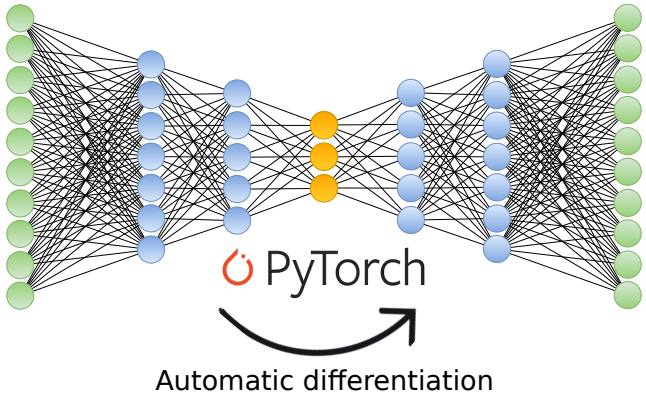
- Given the pairs $\{\mathbf{y}_t, \mathbf{y}_{t+1}\}_{t=1,2,...T}$, we train a model by minimizing the MSE

$$\min \frac{1}{T} \sum_{t=0}^{T} \|\mathbf{y}_{t+1} - \mathcal{F}(\mathbf{y}_t)\|_2^2.$$

- During inference time, we can obtain predictions by composing the learned model k-times

$$\hat{\mathbf{y}}_k = \mathcal{F} \circ \mathcal{F} \circ \mathcal{F} \circ \ ... \ \circ \mathcal{F}(\mathbf{y}_0).$$

A typical black box model



- I will talk more about the specific architecture later....

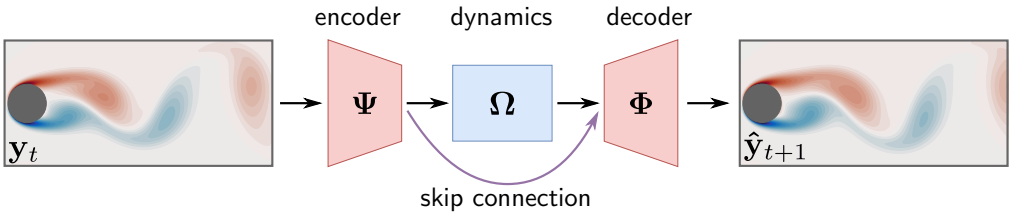
From black box to gray box models

- We to add meaningful constraints to our model:

$$\min \frac{1}{T} \sum_{t=0}^{T} \|\mathbf{y}_{t+1} - \mathbf{\Phi} \circ \mathbf{\Omega} \circ \mathbf{\Psi}(\mathbf{y}_t)\|_2^2 + \lambda \|\mathbf{y}_t - \mathbf{\Phi} \circ \mathbf{\Psi}(\mathbf{y}_t)\|_2^2.$$

- If the model obeys the assumption that $\mathbf{\Psi}$ approximates $\mathcal{G}^{-1}$, then we have that

$$\hat{\mathbf{y}}_k \approx \mathbf{\Phi} \circ \mathbf{\Omega}^k \circ \mathbf{\Psi}(\mathbf{y}_0).$$



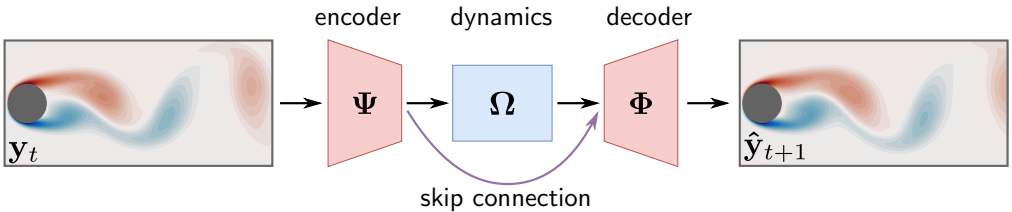
From black box to gray box models

- We start by adding a meaningful constraint to our model:

$$\min \frac{1}{T} \sum_{t=0}^{T} \|\mathbf{y}_{t+1} - \mathbf{\Phi} \circ \mathbf{\Omega} \circ \mathbf{\Psi}(\mathbf{y}_t)\|_2^2 + \lambda \|\mathbf{y}_t - \mathbf{\Phi} \circ \mathbf{\Psi}(\mathbf{y}_t)\|_2^2 + \kappa \ \rho(\mathbf{\Omega}).$$

- If the model obeys the assumption that $\mathbf{\Psi}$ approximates $\mathcal{G}^{-1}$, then we have that

$$\hat{\mathbf{y}}_k \approx \mathbf{\Phi} \circ \mathbf{\Omega}^k \circ \mathbf{\Psi}(\mathbf{y}_0).$$

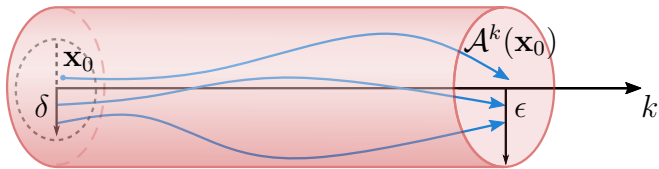


Lyapunov stability in a nutshell

- The origin of a dynamic system

$$\mathbf{x}_{t+1} = \mathcal{A}(\mathbf{x}_t) + \eta_t \quad t = 0, 1, 2, \ldots, T.$$

  is stable if all trajectories starting arbitrarily close to the origin (in a ball of radius δ) remain arbitrarily close (in a ball of radius ϵ).



- If the dynamics $\mathcal{A}$ are linear, stability can be checked with an eigenvalue analysis.

Lyapunov's method... an idea from over 120 years ago[1]

- For linear systems, Lyapunov's second method states that a dynamic system

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \eta_t \quad t = 0, 1, 2, \ldots, T$$

is stable if and only if for any (symmetric) positive definite matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$ there exists a (symmetric) positive definite matrix $\mathbf{P} \in \mathbb{R}^{n \times n}$ satisfying

$$\mathbf{A}^\top \mathbf{P}\mathbf{A} - \mathbf{P} = -\mathbf{Q}.$$

[1] https://stanford.edu/~boyd/papers/pdf/springer_15_colloquium.pdf

Lyapunov's method... an idea from over 120 years ago[1]

- For linear systems, Lyapunov's second method states that a dynamic system

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \eta_t \quad t = 0, 1, 2, \ldots, T$$

  is stable if and only if for any (symmetric) positive definite matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$ there exists a (symmetric) positive definite matrix $\mathbf{P} \in \mathbb{R}^{n \times n}$ satisfying

$$\mathbf{A}^\top \mathbf{P} \mathbf{A} - \mathbf{P} = -\mathbf{Q}.$$

- Using this idea, we impose that the symmetric matrix $\mathbf{P}$, defined by

$$\mathbf{\Omega}^\top \mathbf{P} \mathbf{\Omega} - \mathbf{P} = -\mathbf{I},$$

  is positive definite.

[1] https://stanford.edu/~boyd/papers/pdf/springer_15_colloquium.pdf

To gain some intuition...

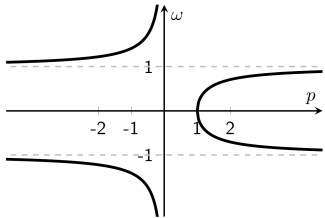
- ... we consider the case where $\mathbf{\Omega}$ is diagonalizable and $\mathbf{Q}$ chosen appropriately.
- Then, for a particular choice of coordinates the following problem

$$\mathbf{\Omega}^\top \mathbf{P} \mathbf{\Omega} - \mathbf{P} = -\mathbf{I}, \tag{1}$$

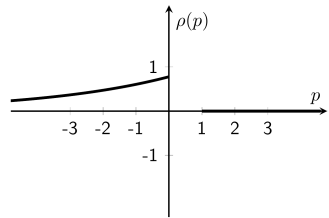
reduces to the system of linear equations

$$\omega_i p_i \omega_i - p_i = -1, \tag{2}$$

where ω_i, p_i, for $i = 1, 2, \ldots, n$, denote the eigenvalues of $\mathbf{\Omega}$ and $\mathbf{P}$, respectively.



(a) Discrete-time Lyapunov function.



(b) Stability promoting prior.

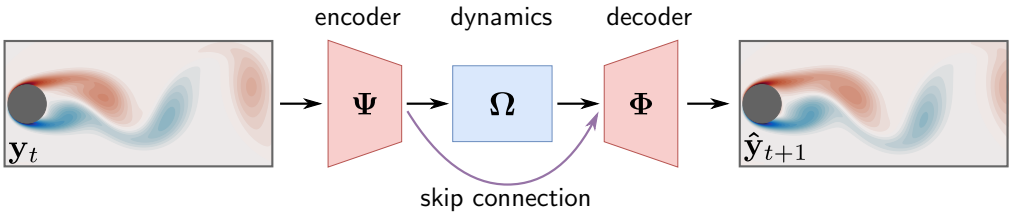
Physics-aware model that preserves stability

- The physics-informed autoencoder is trained by minimizing the following objective

$$\min \frac{1}{T}\sum_{t=0}^{T} \|\mathbf{y}_{t+1} - \mathbf{\Phi}\circ\mathbf{\Omega}\circ\mathbf{\Psi}(\mathbf{y}_t)\|_2^2 \; + \; \lambda\|\mathbf{y}_t - \mathbf{\Phi}\circ\mathbf{\Psi}(\mathbf{y}_t)\|_2^2 + \; \kappa\sum_i \rho(p_i).$$

- The prior p can take various forms. We use the following in our experiments:

$$\rho(p) := \begin{cases} \exp\left(-\frac{|p-1|}{\gamma}\right) & \text{if } p < 0 \\ 0 & \text{otherwise.} \end{cases}$$



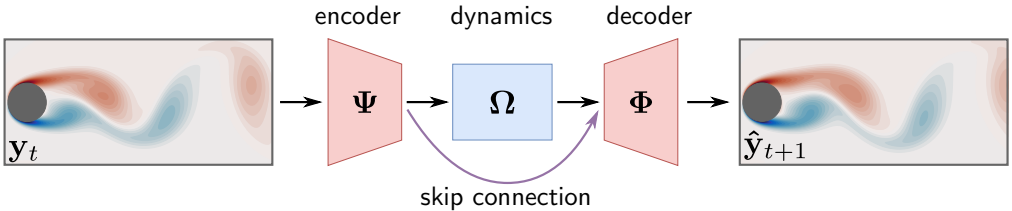
Physics-aware model that preserves stability

- The physics-informed autoencoder is trained by minimizing the following objective

$$\min \frac{1}{T}\sum_{t=0}^{T} \|\mathbf{y}_{t+1}-\mathbf{\Phi}\circ\mathbf{\Omega}\circ\mathbf{\Psi}(\mathbf{y}_t)\|_2^2+\|\mathbf{y}_{t+2}-\mathbf{\Phi}\circ\mathbf{\Omega}\circ\mathbf{\Omega}\circ\mathbf{\Psi}(\mathbf{y}_t)\|_2^2+\lambda\|\mathbf{y}_t-\mathbf{\Phi}\circ\mathbf{\Psi}(\mathbf{y}_t)\|_2^2+\kappa\sum_i \rho(p_i).$$

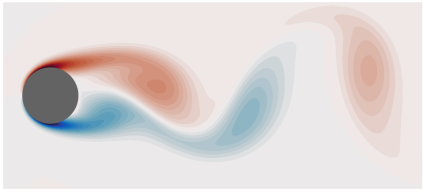
- The prior p can take various forms. We use the following in our experiments:

$$\rho(p) := \begin{cases} \exp\left(-\frac{|p-1|}{\gamma}\right) & \text{if } p < 0 \\ 0 & \text{otherwise.} \end{cases}$$

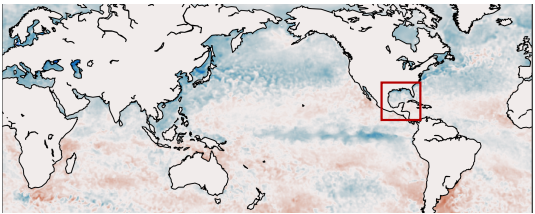


Examples that we consider

- Flow past the cylinder.

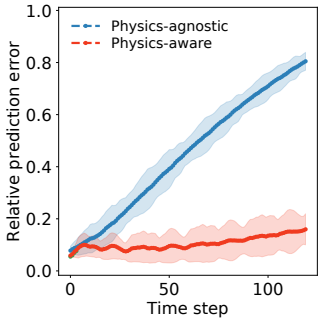


- Daily sea surface temperature data of the gulf of Mexico over a period of 6 years.

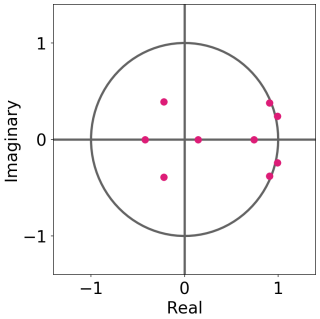


Prediction performance for flow past the cylinder (without weight decay)

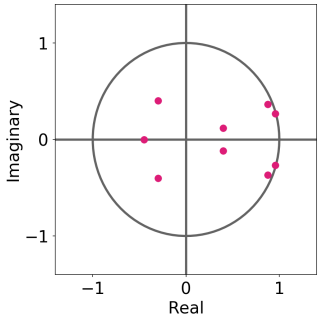
$$\min \frac{1}{T}\sum_{t=0}^{T} \|\mathbf{y}_{t+1}-\mathbf{\Phi}\circ\mathbf{\Omega}\circ\mathbf{\Psi}(\mathbf{y}_t)\|_2^2+\|\mathbf{y}_{t+2}-\mathbf{\Phi}\circ\mathbf{\Omega}\circ\mathbf{\Omega}\circ\mathbf{\Psi}(\mathbf{y}_t)\|_2^2+\lambda\|\mathbf{y}_t-\mathbf{\Phi}\circ\mathbf{\Psi}(\mathbf{y}_t)\|_2^2+\kappa\sum_i \rho(p_i).$$



(a) With LR $1e{-}2$.

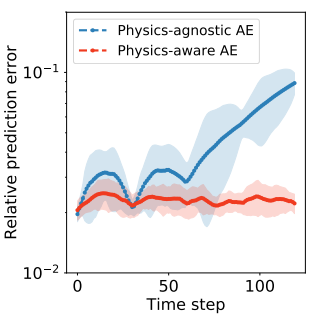


(b) Physics-agnostic (blue).

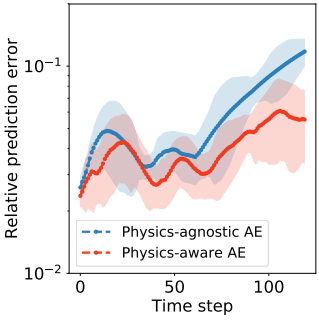


(c) Physics-aware (red).

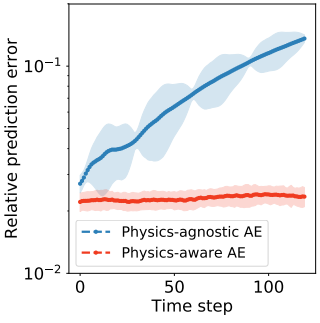
More results for the flow past the cylinder (with weight decay)



(a) With LR $1e{-}2$ and WD $1e{-}6$.



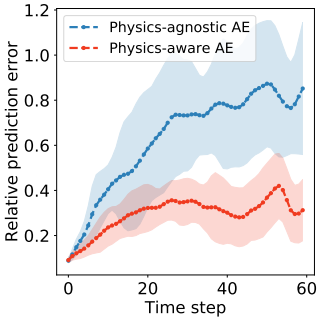
(b) With LR $1e{-}2$ and WD $1e{-}8$.



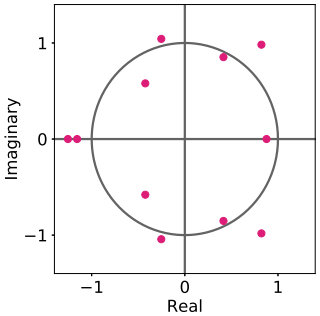
(c) With LR $5e{-}3$ and WD $1e{-}6$.

Results for the sea surface temperature data

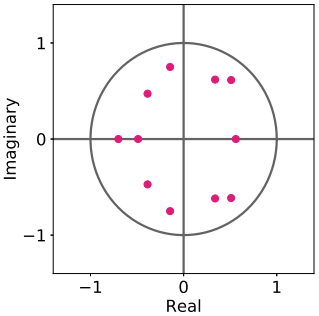
$$\min \frac{1}{T}\sum_{t=0}^{T} \|\mathbf{y}_{t+1} - \mathbf{\Phi}\circ\mathbf{\Omega}\circ\mathbf{\Psi}(\mathbf{y}_t)\|_2^2 + \lambda\|\mathbf{y}_t - \mathbf{\Phi}\circ\mathbf{\Psi}(\mathbf{y}_t)\|_2^2 + \kappa\sum_i \rho(p_i).$$



(a) With LR $1\mathrm{e}{-2}$.

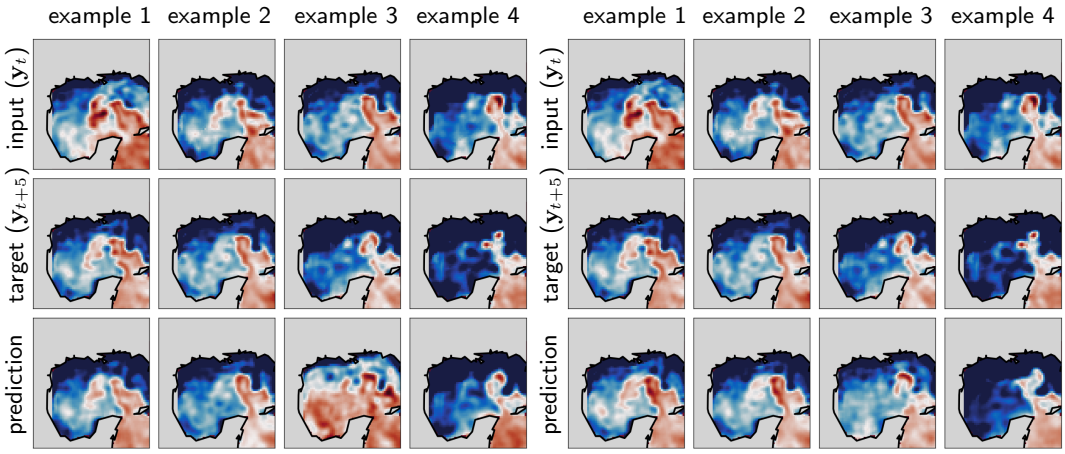


(b) Physics-agnostic (blue).



(c) Physics-aware (red).

Visual results for the sea surface temperature data

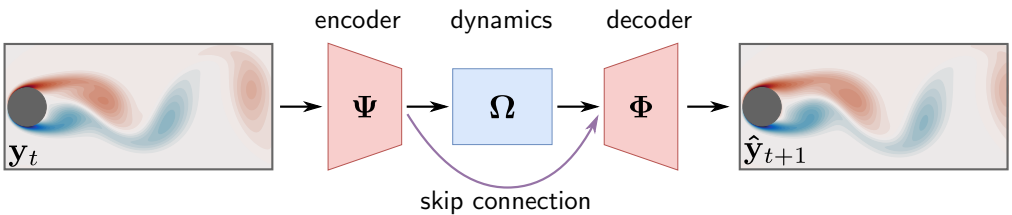


(a) Physics-agnostic model.

(b) Physics-aware model.

Summary

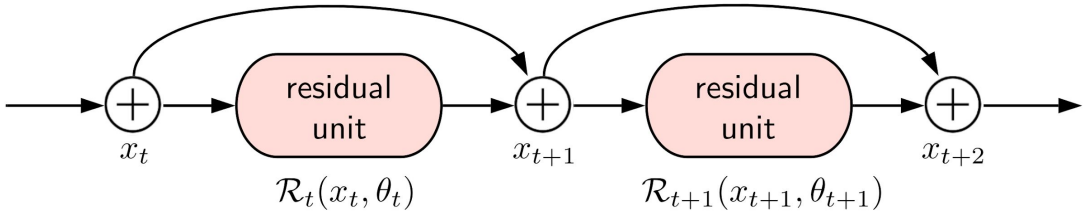
- *Physics-informed* autoencoders can help to improve the generalization performance.
- Caveat of physics-informed learning are complicated loss functions: $\mathcal{L}_1 + \gamma\mathcal{L}_2 + \kappa\mathcal{L}_3 + ...$
- Next steps: non-linear dynamics, recurrent networks and parameterized layers.



Outline

Connection between ResNets and Dynamical Systems

- ResNets are the most popular network architectures on the market.



- **Hypothesis:** Recent literature notes that ResNets learn a forward Euler discretization of a dynamical system:

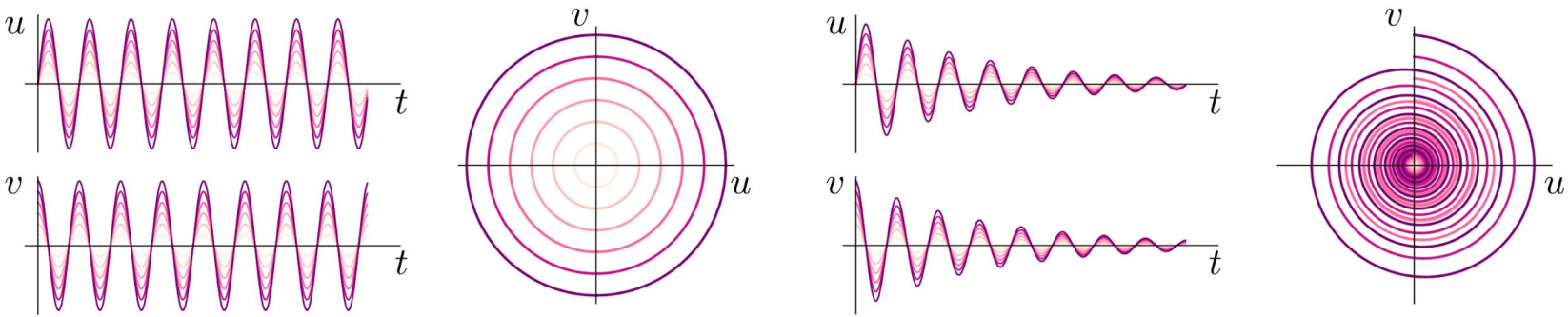
$$x_{k+1} = x_k + \Delta t \mathcal{R}(x_k, \theta_k) \longrightarrow \frac{\partial x(t)}{\partial t} = \mathcal{R}(x(t), t, \theta)$$

=1 sneak it in

- Spoiler: we show that ResNets are not forward Euler discretizations of a dynamical system in a meaningful way due to overfitting.

Experiments in Dynamics

- What does it even mean to say ResNet learns a forward Euler representation of a dynamical system?
- We need context where a dynamical system is meaningful.
- So ... let's try time series prediction of a dynamical system.



Numerical Integration and Machine Learning work in Opposite Directions

Numerical Integration:

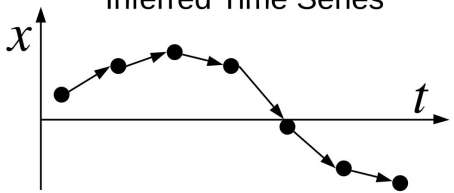
Ground Truth Dynamics

$$f = \frac{\mathrm{d}x}{\mathrm{d}t}$$

Approximation yields a model

$$x_{n+1} = x_n + \Delta t f(x_n)$$

Inferred Time Series



Learning the Dynamics:

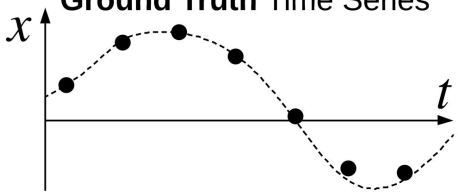
What's *G*???

$$G(x;\theta)$$

Use data to optimize a model

$$\min \|x_{n+1} - (x_n + \Delta t\, G(x_n))\|$$

Ground Truth Time Series



Revisiting a Simple Dynamical System

- Learning a residual makes sense in many contexts:

  Future = Now + Update

- Let's study training such a $F(x)$ based on a neural network $G(x)$:

$$x(t+\Delta t) = \mathbf{F}(x(t)) = x(t)+\mathbf{G}(x(t)) = \text{NumericalMethod}[\mathbf{G}(x)]$$

- Numerical integrators approximate the integral with a discrete series of applications of $f(x,t)=dx/dt$ for a time step Δt:

$$\begin{aligned} x(t+\Delta t) &= x(t) + \int_t^{t+\Delta t} f(x,t)\mathrm{d}t \\ &\approx x(t) + scheme[f, x, t, \Delta t] \end{aligned}$$

Syntactic Similarity is not Sufficient for Correspondence

- Approximations to dynamical systems have richer properties.

  A. For a given integrator, as $\Delta t \rightarrow 0$, error$\rightarrow 0$ (timestep refinement)

  B. Integrators have a rate of convergence: $log(error) \propto r\ log(\Delta t)$

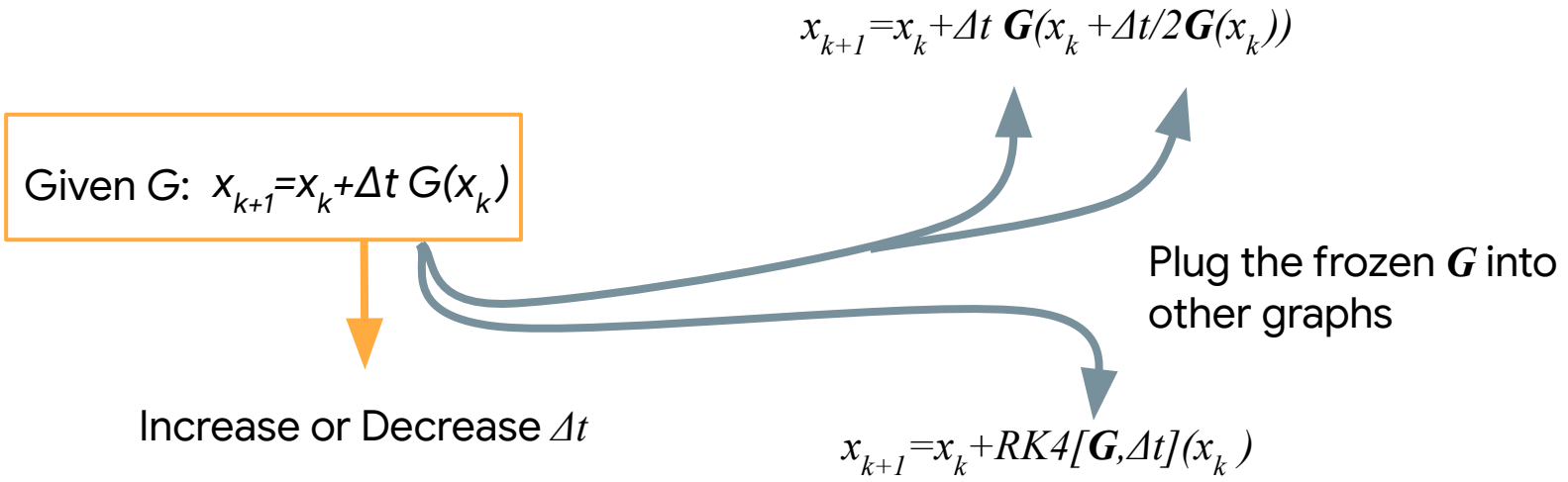
  C. The same dx/dt with different integrators should approach the same $x(t_{max})$ at their respective rates

- We can verify these using a *convergence test*.

- These conditions are critical to deriving integration schemes.
  (They also make great integration tests for numerical software!)

Does ResNet Units Satisfies these Properties?

- If yes, then we should be able to alter the model:



- and predictions should change consistently as expected.
- If no, then the model should behave differently w.r.t. Δt.

Experiment

1. Make a dataset with one Δt: $\{x(0), x(\Delta t), x(2\Delta t) \ldots x(T)\}$
2. Train 3 models using G: a shallow *tanh* NN with 50 hidden units.
3. Then, freeze G, and perform a convergence test.
4. Hypothesis: A, B, & C should hold.

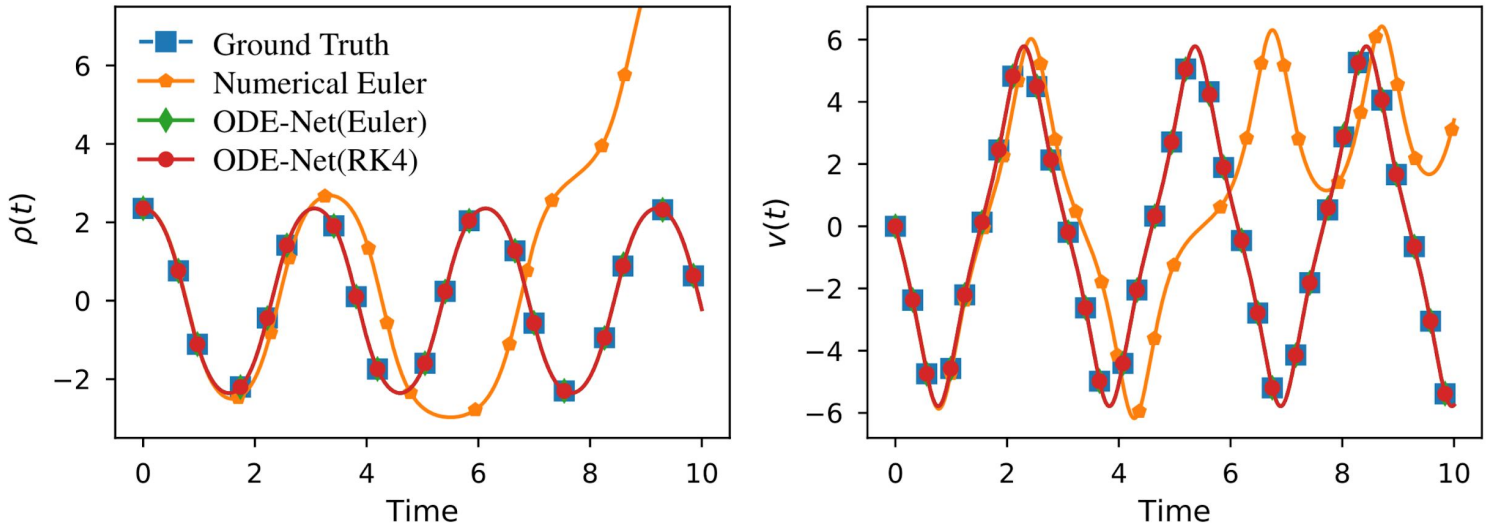
We train three models:

1. Forward Euler: $x_{k+1} = x_k + \Delta t\, G(x_k)$ ← *Looks like a ResNet unit*
2. Midpoint: $x_{k+1} = x_k + \Delta t\, G(x_k + \Delta t/2 G(x_k))$
3. RK4: $x_{k+1} = x_k + \Delta t\, RK4[G, \Delta t](x_k)$

- For ground-truth, use the analytical solution.
- For comparison, plug the known dx/dt into the integrators.

After we train the models, they all perform good

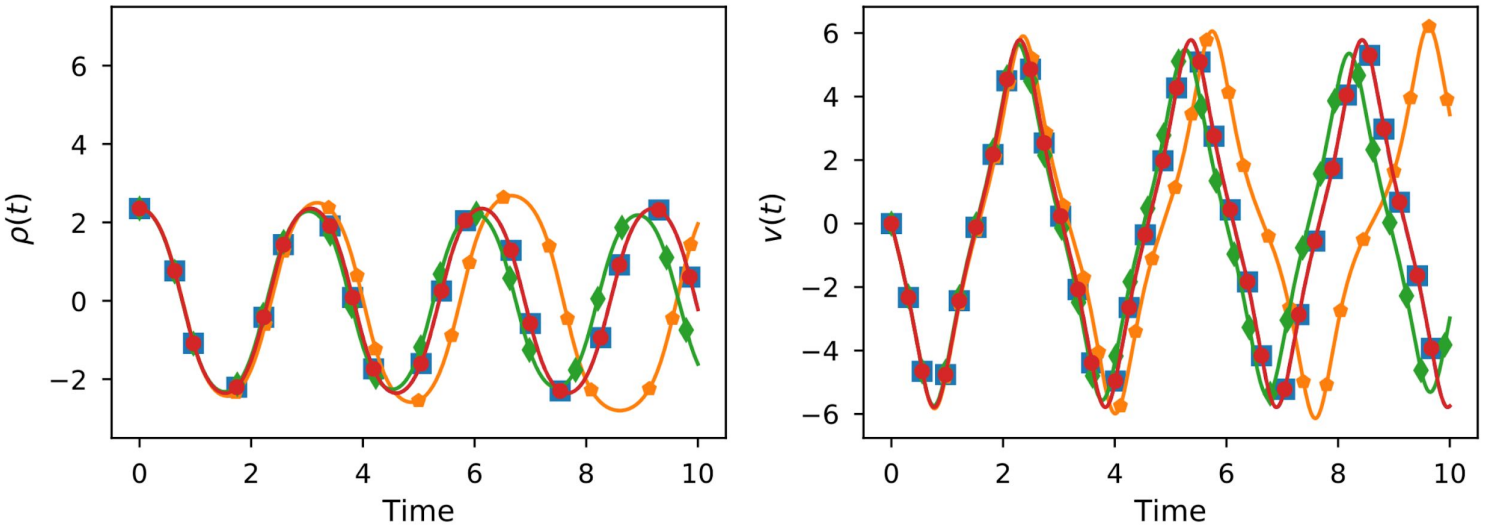
- They are good **discrete models** without changing Δt.
- Note how using Euler as a numerical method is inaccurate.



(a) $\Delta t = \Delta t_{data}$

But if we cut Δt in half, ODE-Net(Euler) gets worse

- Numerical(Euler) improves, as expected.
- Neural(RK4) is still on top of the analytical solution.

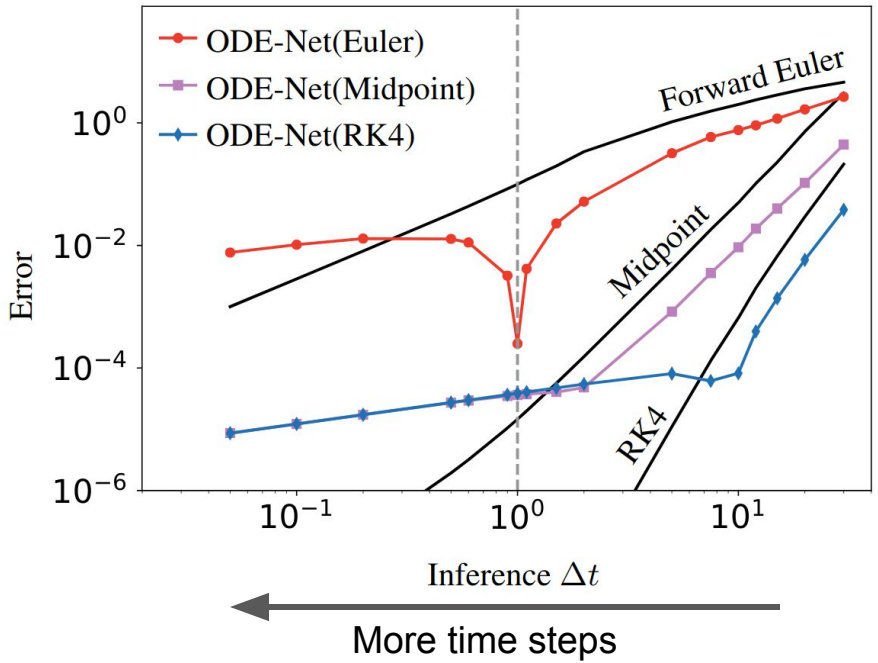


(b) $\Delta t = 0.5\Delta t_{data}$

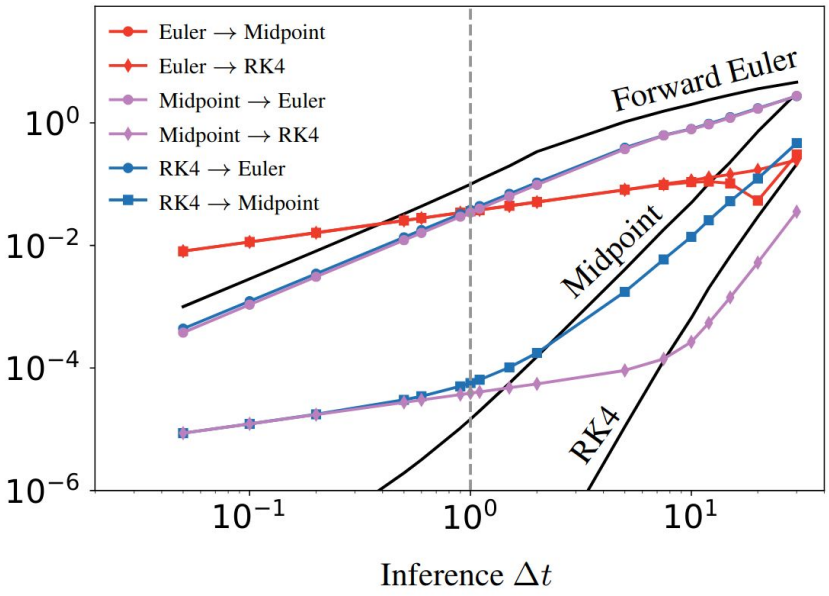
One-off plots aren't sufficient

- Sweep Δt and calculate errors to do the full convergence test.

Part 1: $\Delta t \longrightarrow 0$ with same graph



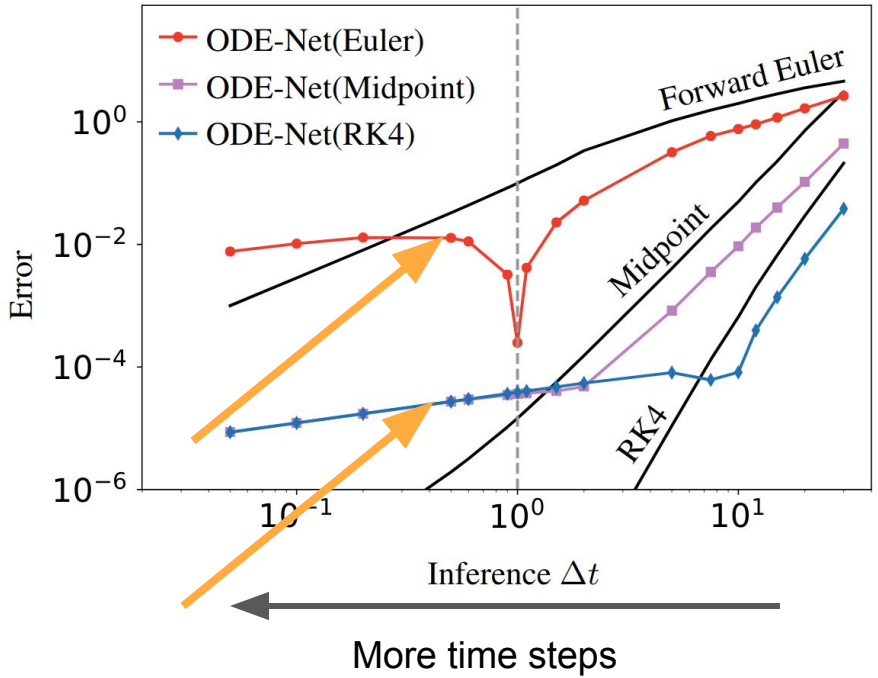
Part 2: Try different integrators



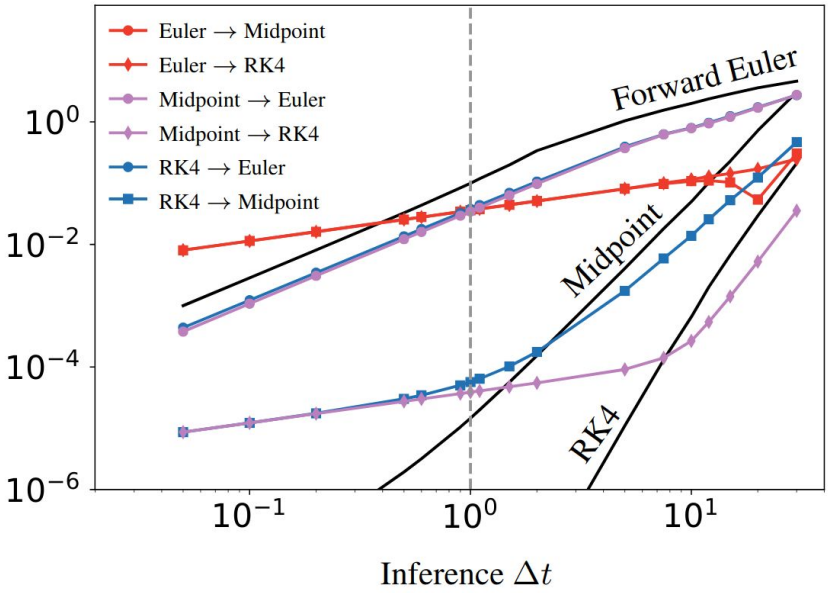
The trajectories on the last slide make one datapoint here:

$$error = ||x_{true}(t_{max}) - F(F(...F(x_0)...))||$$

Part 1: $\Delta t \rightarrow 0$ on the model

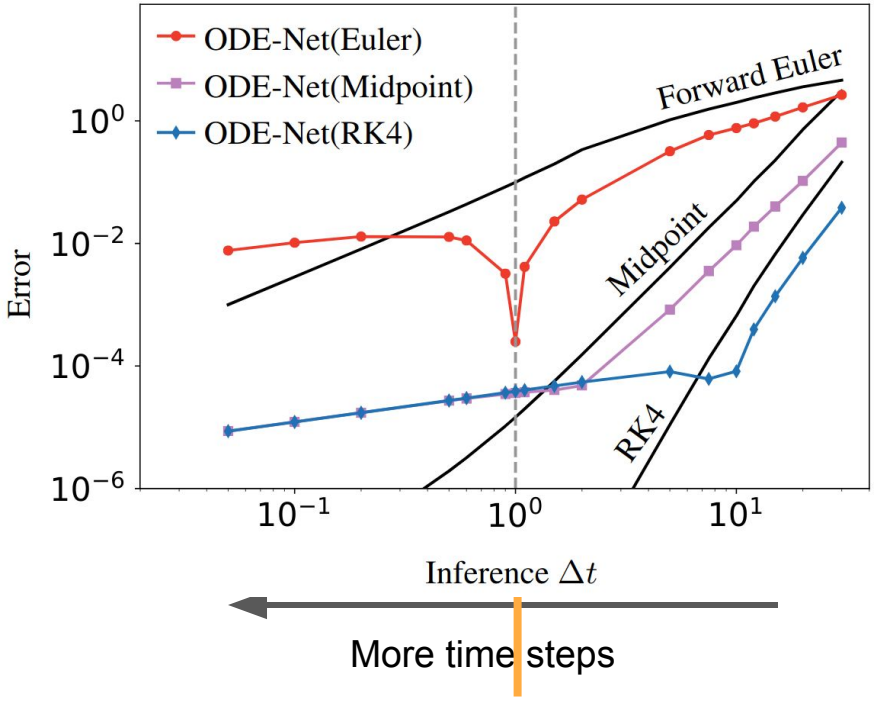


Part 2: Try different integrators

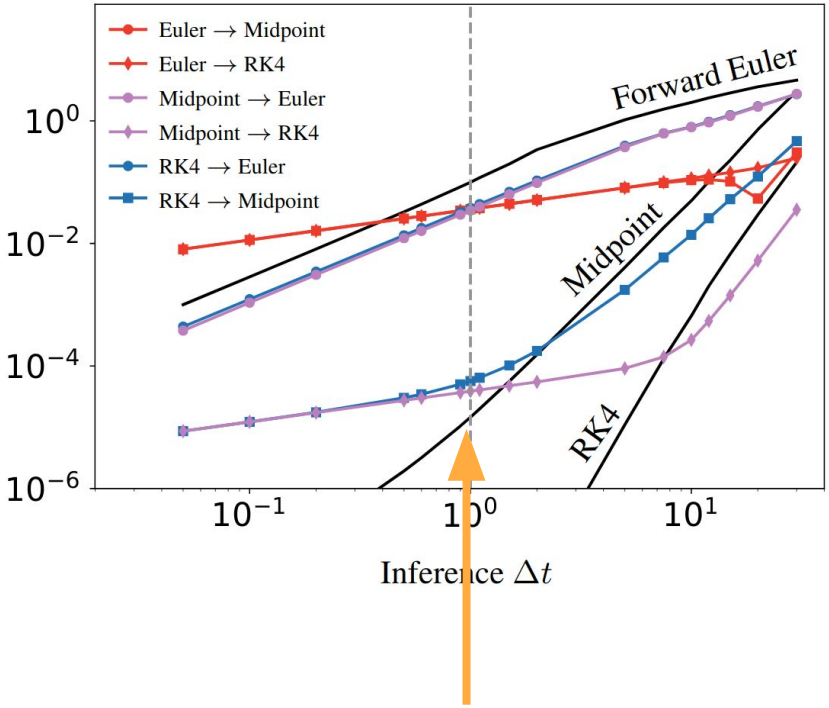


Vertical line is the training dataset sample rate

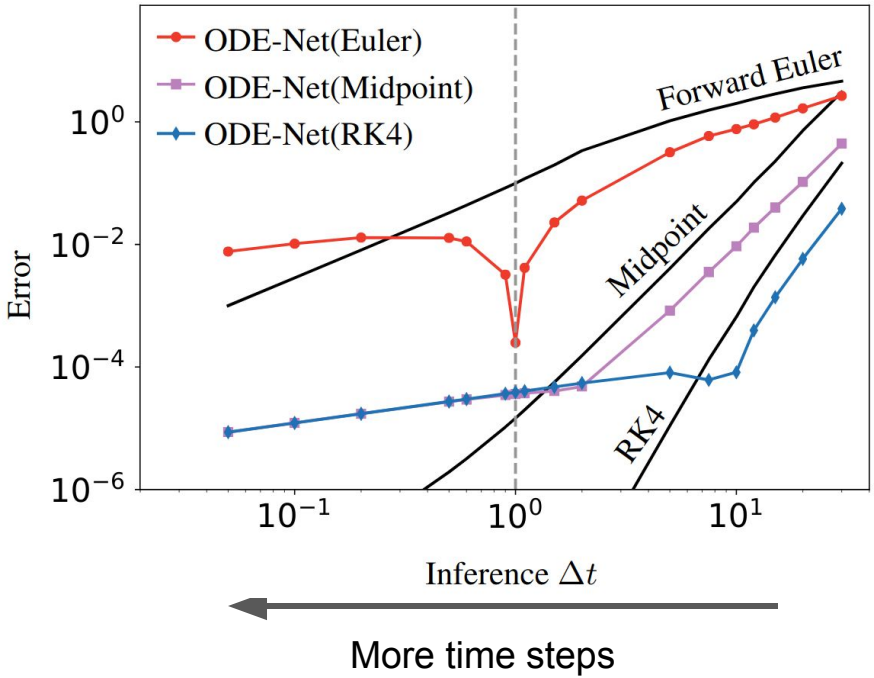
Part 1: $\Delta t \rightarrow 0$ on the model

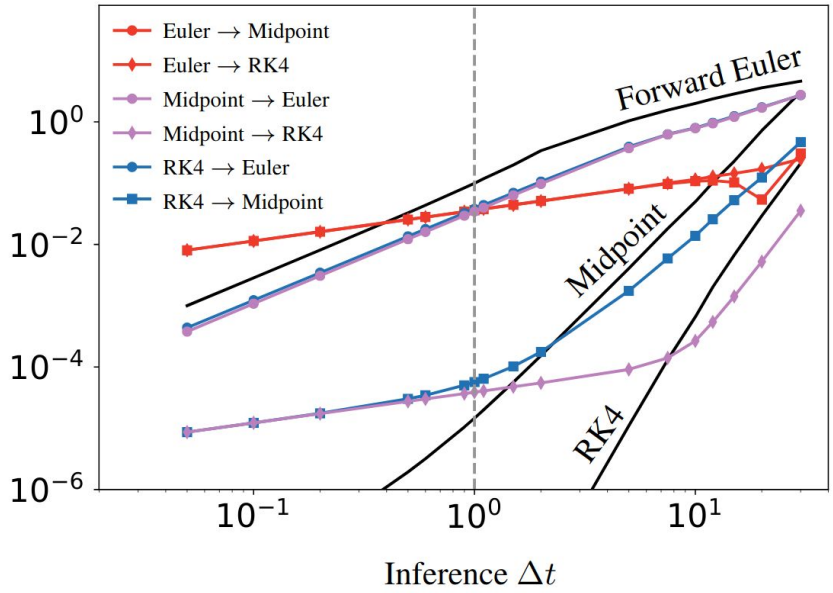


Part 2: Try different integrators

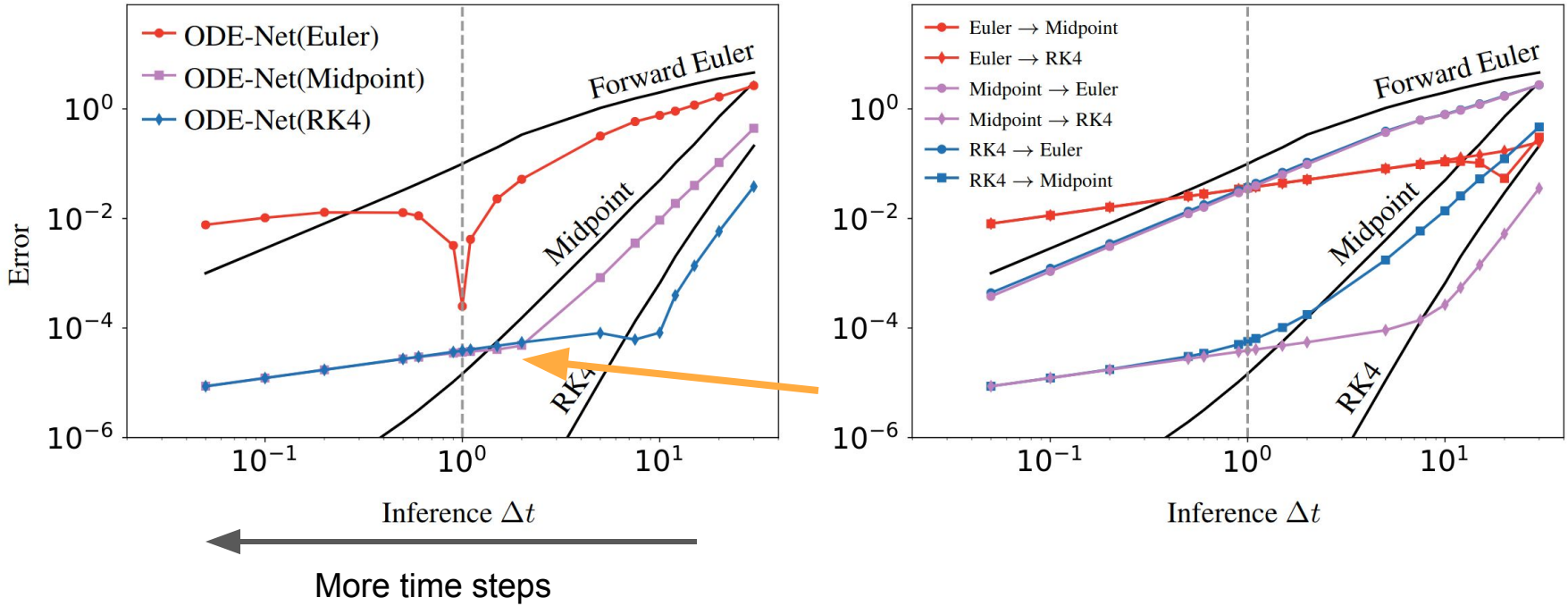


- There's a noticeable dip in error for ODE-Net(Euler).
- ODE-Net(Euler) is extremely sensitive to perturbations in Δt.
- Thus, it is only a discrete model, i.e., it overfits to Δt.

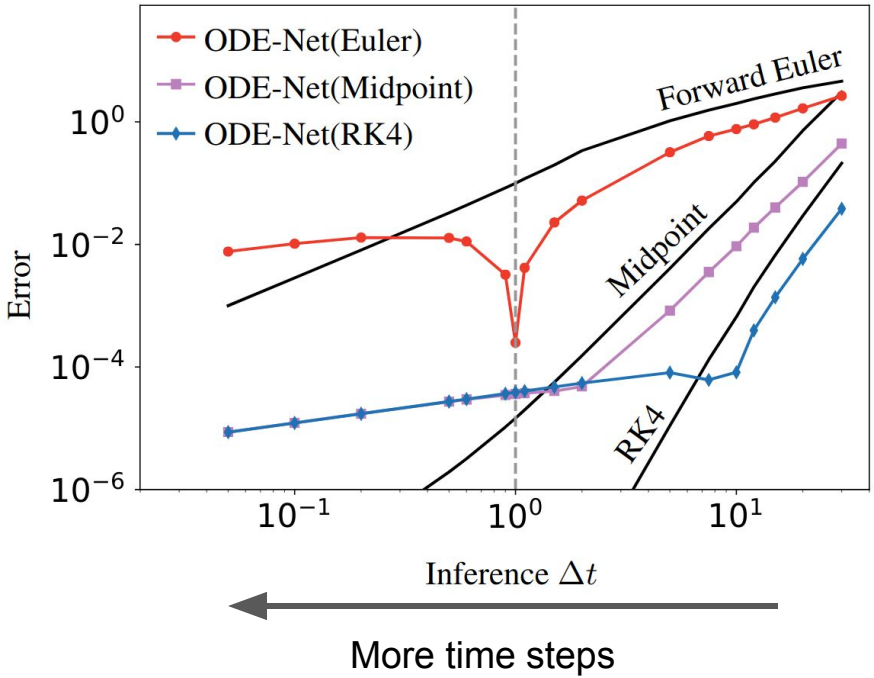


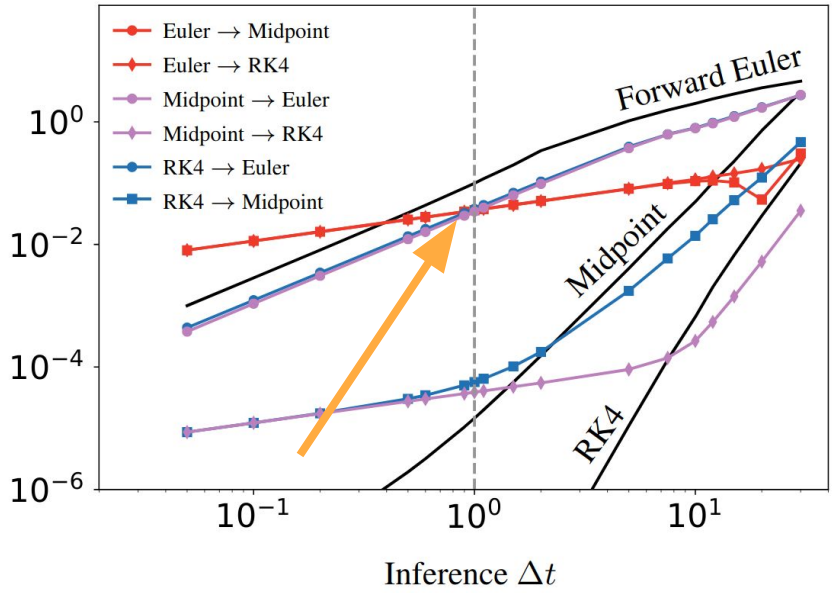


- The models embedded in Midpoint and RK4 have no dip.
- The error changes smoothly for incremental changes in Δt.
- For larger time steps, the slopes match.

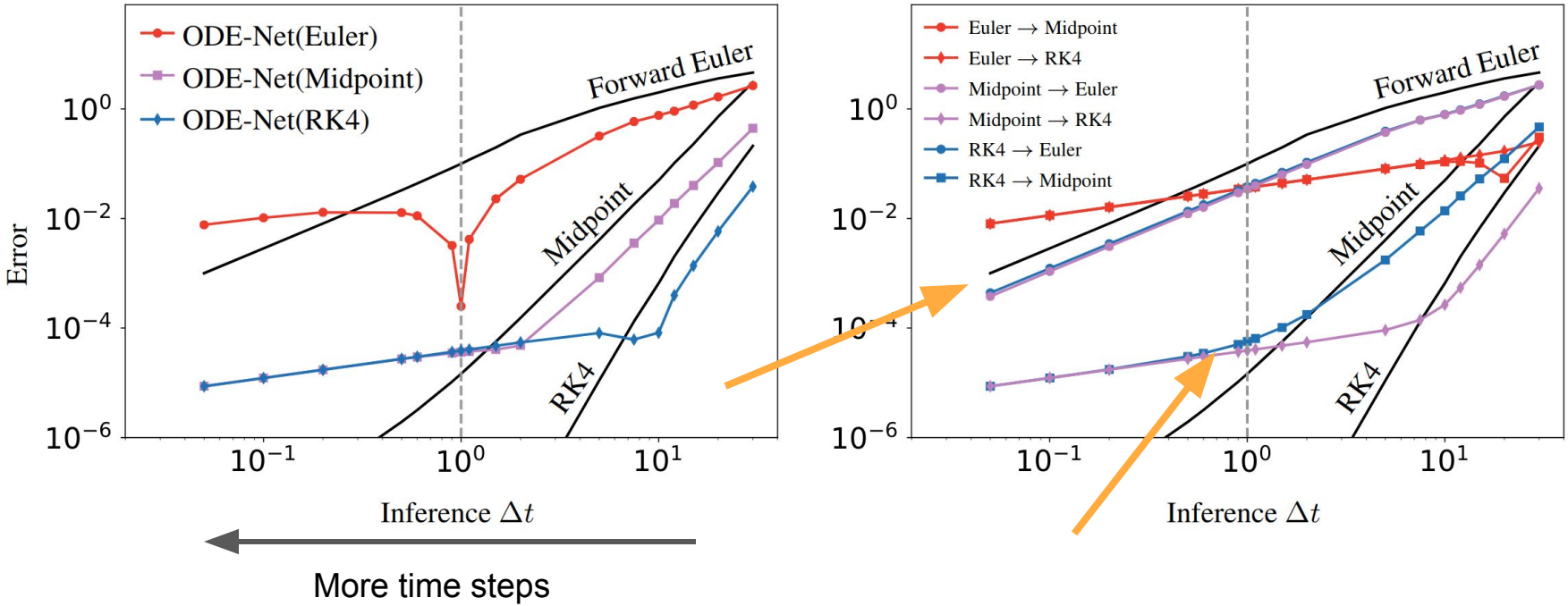


- If we take the NN ***G*** from Euler and put it into another integrator, the error is large.





- But, both of the *G*s trained inside of higher order integrators work as expected when inside of any of the other integrators.



Implications

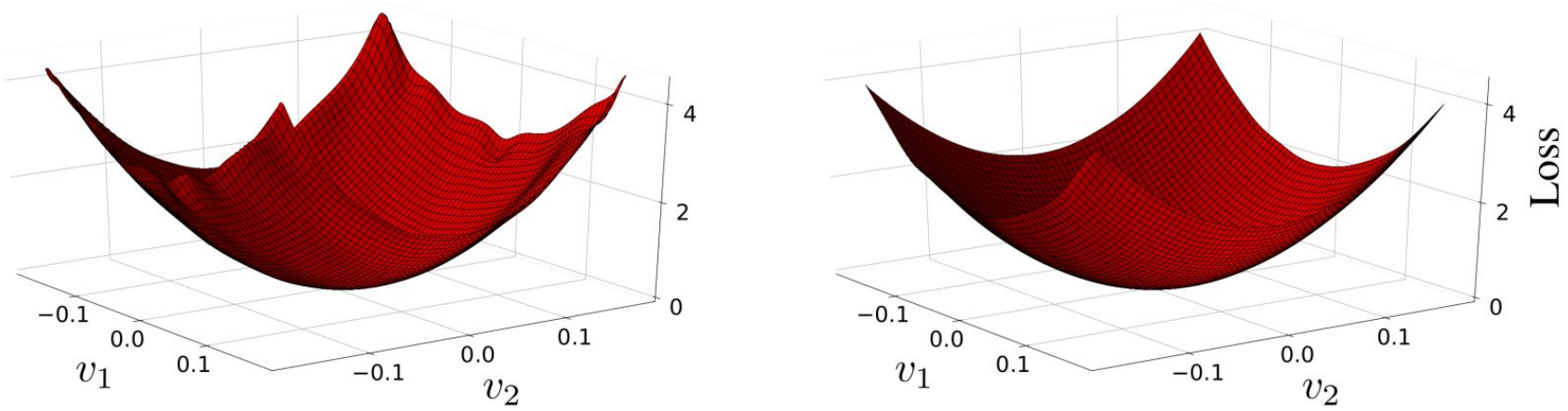
- Our results show that our prevalent training methodology does *not* yield models that can be interpreted with continuous theory.

- Having a peak says that it's fragile to the number of timesteps.
  That means, we can't do ``interpolation".

- The RK4 scheme enables us to interpolate between domain shifted data.
  In turn, this means that we can increase the number of layers (i.e., be continuous-in-depth).

- Our analysis can be seen as a diagnostic tool to assess if the model has overfit: how well does it represent a continuous system, and how well does it exhibit the numerical properties of a continuous operator?

Hessian Loss Landscape



(a) Euler

(b) RK4

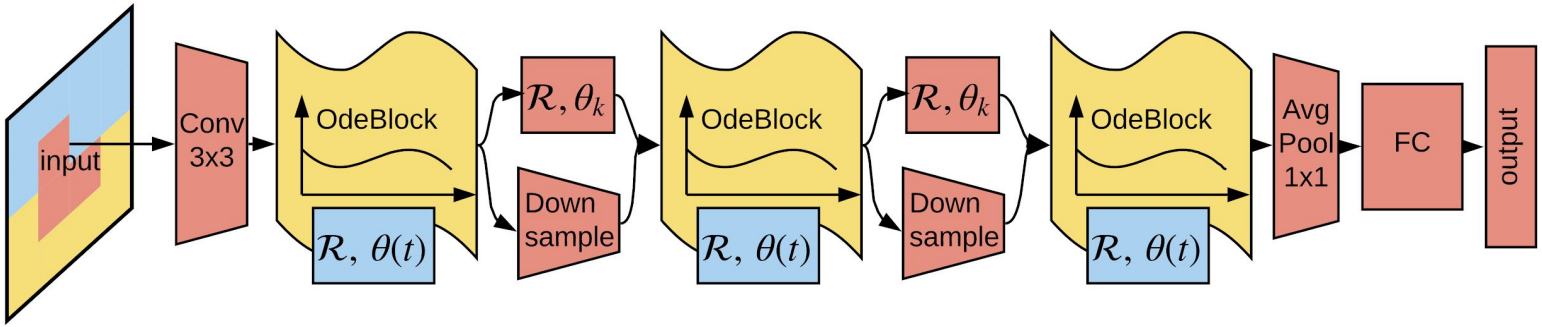
- "ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning," arXiv:2006.00719
- "PyHessian: Neural Networks Through the Lens of the Hessian," arXiv:1912.07145

Continuous-in-Depth Neural Networks

ContinuousNet is a deep model that *is* a dynamical system:

- Basis functions in depth for parameters.
- High-order Runge Kutta-based computation graphs.
- Right timestep refinement and grid refinement.

Goal: recover (then extend) the exact same graph as ResNet, but phrased as a function of time.



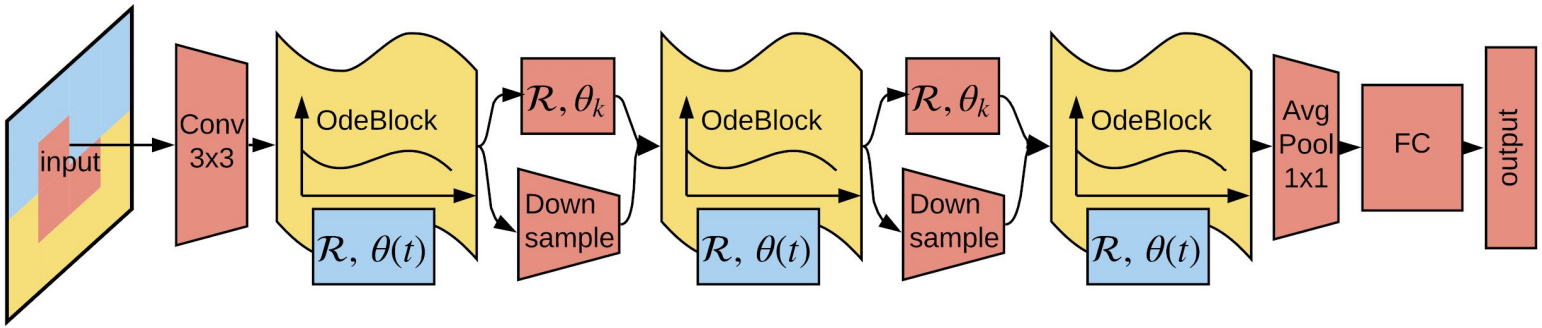
- **ContinuousNet**'s governing equation has time-varying parameters:

$$\dot{x}(t) = \epsilon \mathcal{R}\left(x(t), \theta(t)\right)$$

- Blocks of residual units are replaced by numerical integrators:

$$\begin{aligned} x_{out} =& x_{in} + \int_0^1 \mathcal{R}\left(x(t), \theta(t)\right)\, \mathrm{d}t \\ &= \texttt{OdeBlock}\left[\mathcal{R}, \theta, \Delta t, t \in [0, 1]\right](x_{in}) \end{aligned}$$

- OdeBlocks are assembled into the same ResNet architectures:



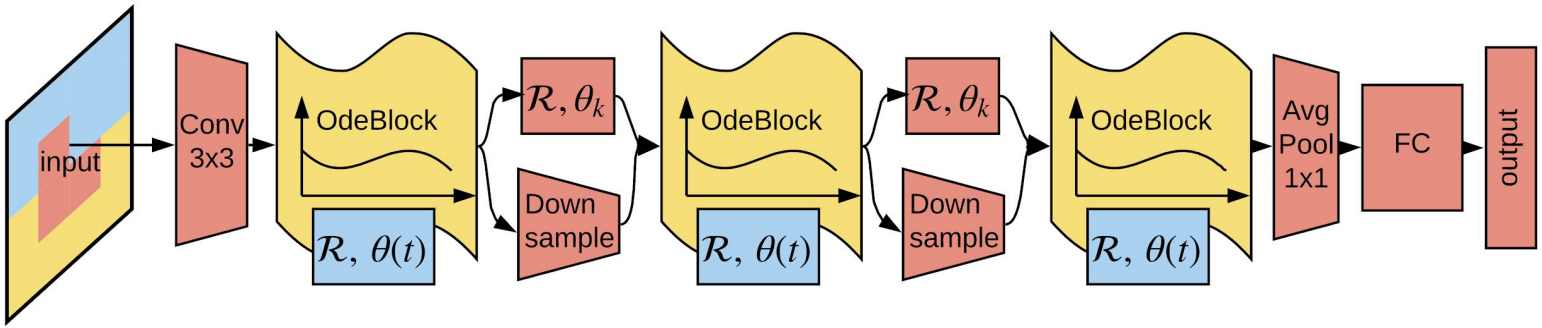
ContinuousNet's governing equation has time-varying parameters:

$$\dot{x}(t) = \epsilon \mathcal{R}\left(x(t), \theta(t)\right)$$

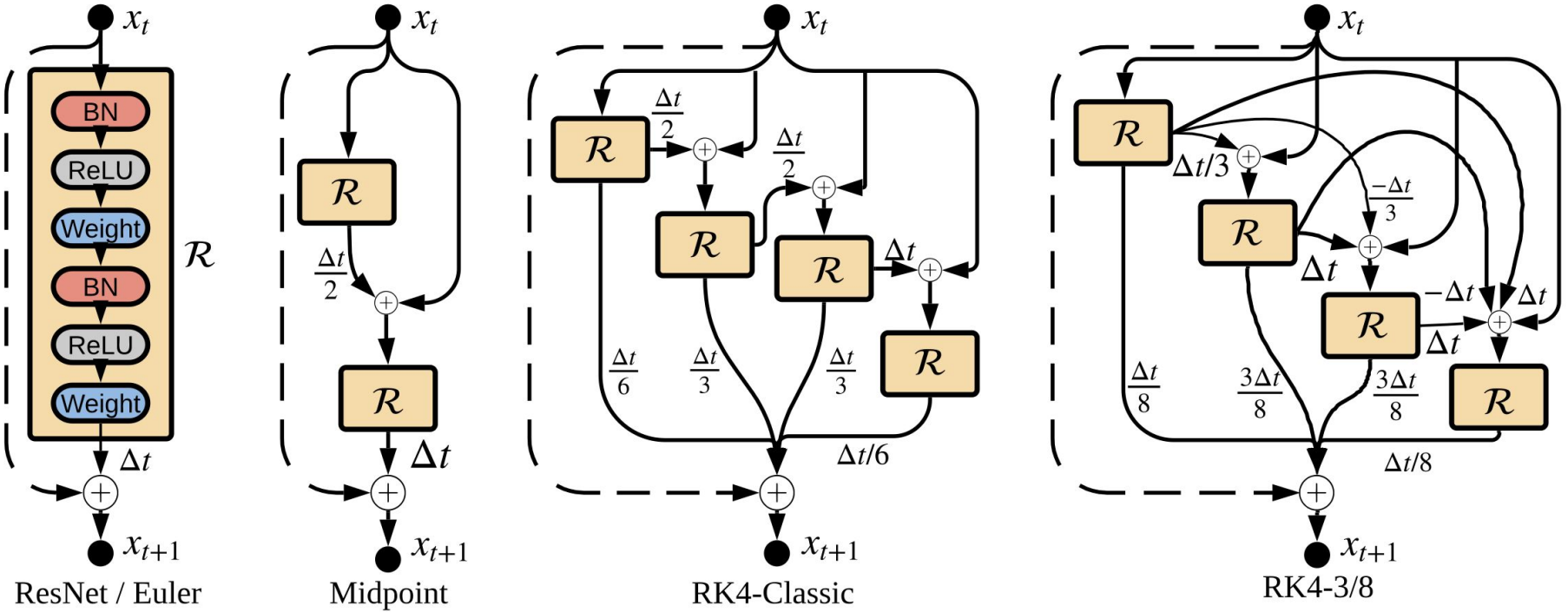
Blocks of residual units are replaced by numerical integrators:

$$\begin{aligned} x_{out} =& x_{in} + \int_0^1 \mathcal{R}\left(x(t), \theta(t)\right)\, \mathrm{d}t \\ &= \texttt{OdeBlock}\left[\mathcal{R}, \theta, \Delta t, t \in [0, 1]\right](x_{in}) \end{aligned}$$

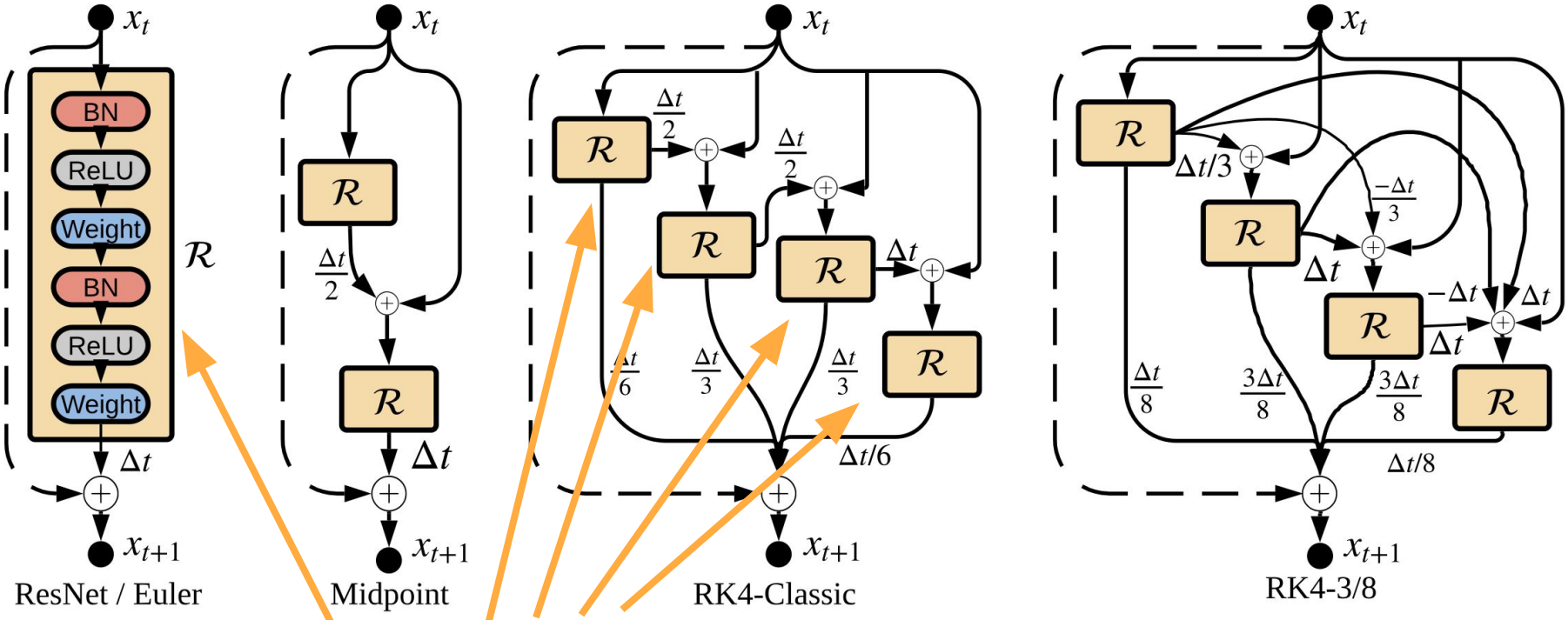
OdeBlocks are assembled into the same ResNet architectures:



- In the NN perspective, integrators prescribe graphs that nest the residual that calculate the same thing:

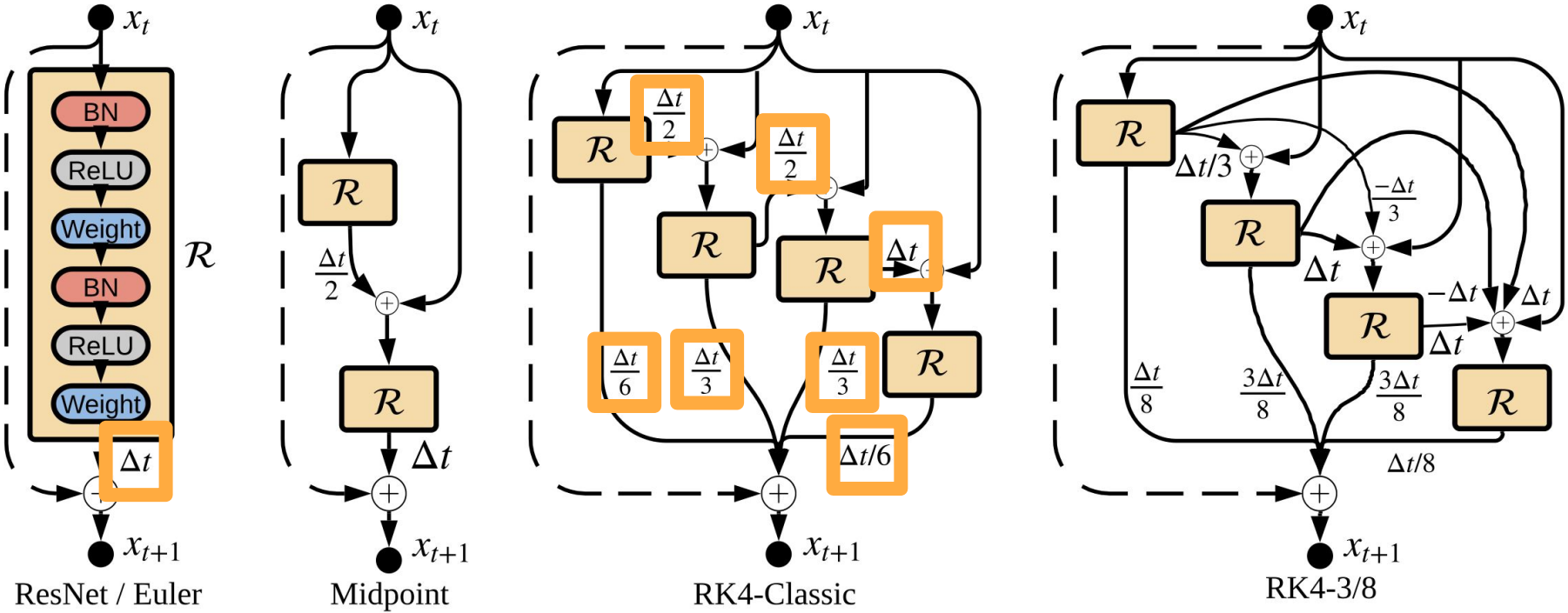


- In the NN perspective, integrators prescribe graphs that nest the residual that calculate the same thing:

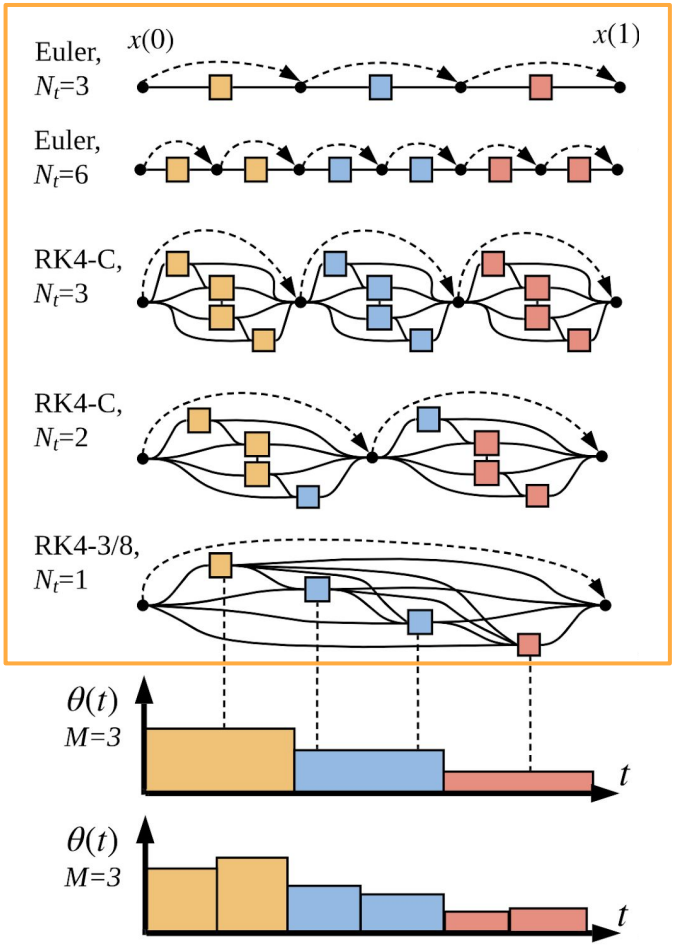


Same R, and the same weights (in the simple case)

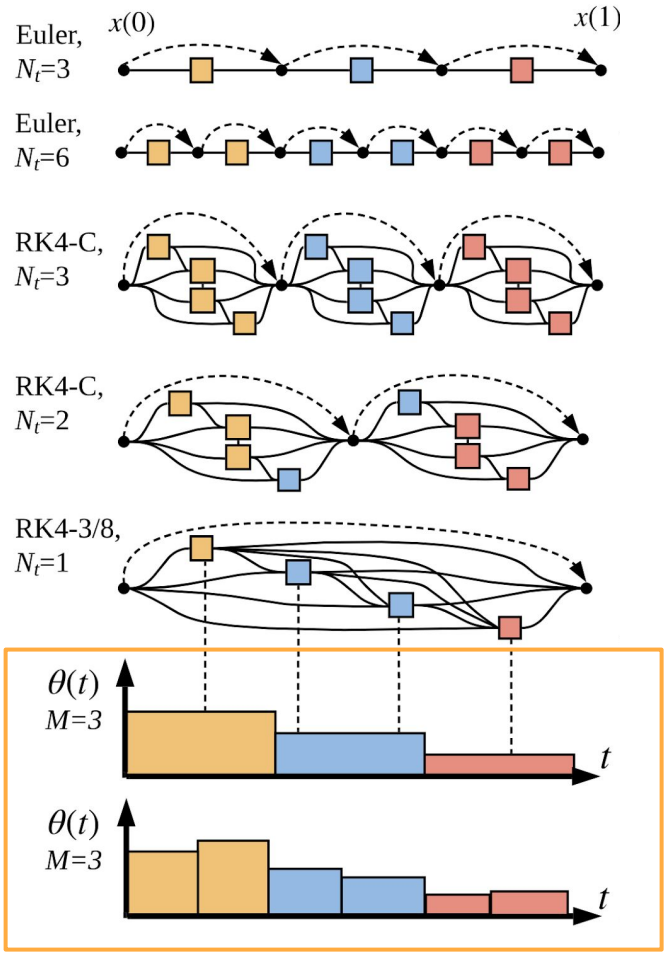
- In the NN perspective, integrators prescribe graphs that nest the residual that calculate the same thing:



- We use edge weights that are well-known in numerical analysis.



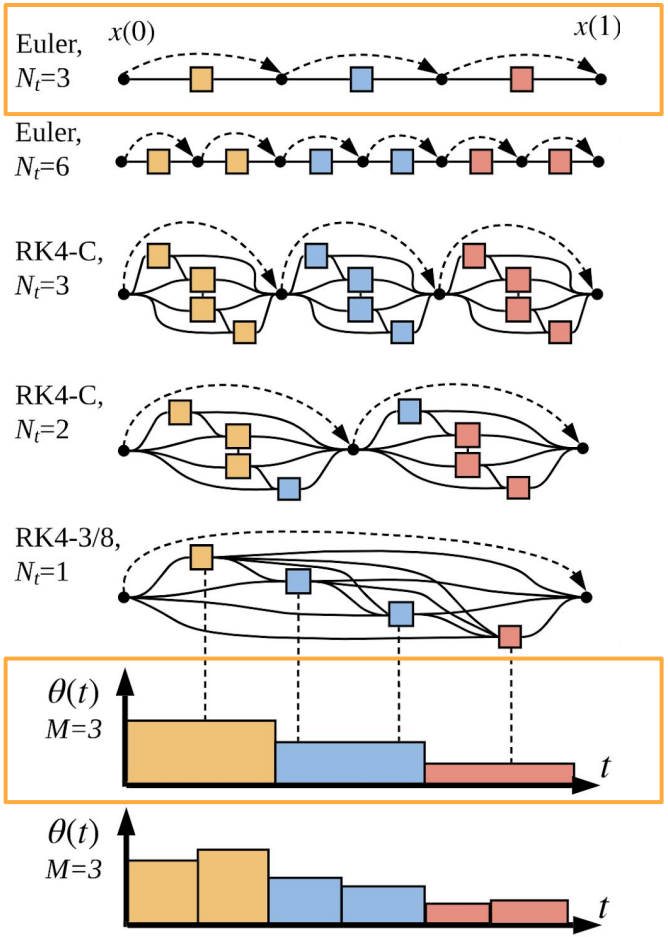
- Numerical integration effectively chains together these units into familiar and unfamiliar graphs.
- Picking a scheme and a N_t specifies how to generate a graph.
- Each one is (approximately) equivalent: each is discrete, but each approximates the same continuous model.



- In ResNet and NNs, parameters are glued to computation nodes in the graph.
- **ContinuousNet**'s computations are assigned weights by evaluating a continuous $\theta(t)$: *What's the value halfway between steps?*
- Basis functions in depth:

$$\theta(t) = \sum_{\beta=1}^{M} \phi^{\beta}(t)\theta^{\beta}$$

- Yields a systematic way to project to different equivalent basis functions.



- **ResNet** is exactly **forward Euler** with the same step size as **piecewise constant** basis functions.

- Construct with a continuous map $[t_k, t_{k+1}) \to \theta_k$ to define values between steps:

$$\theta(t) = \begin{cases} \theta_1 & , \ t \in [0, \Delta t) \\ \theta_2 & , \ t \in [\Delta t, 2\Delta t) \\ \dots & \\ \theta_{N_t} & , \ t \in [T - \Delta t, T) \end{cases}$$

Experiments for Image Classification

The original hypothesis can be tested with a convergence test on a DL problem using ContinuousNet.

Updated hypothesis:

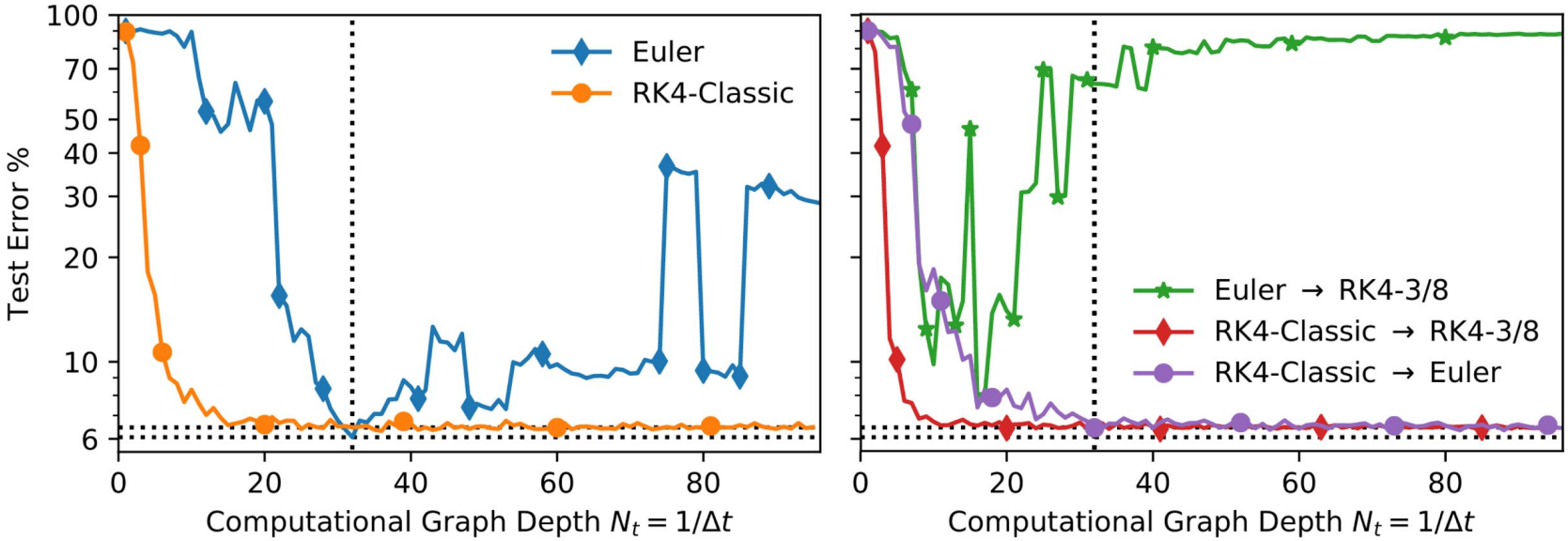
- Expect forward Euler (ResNet) to overfit
- Expect training with Midpoint or RK4 to enable transfer between depths and graph modules

We can perform the same experiment on CIFAR10

- Test set error in place of analytical solution

Train ContinuousNet *M*=32-32-32 with RK4 and Euler (ResNet-198)

- ContinuousNet(Euler): same dip as pendulum
- ContinuousNet(RK4): re-manifests with many N_t and integrators

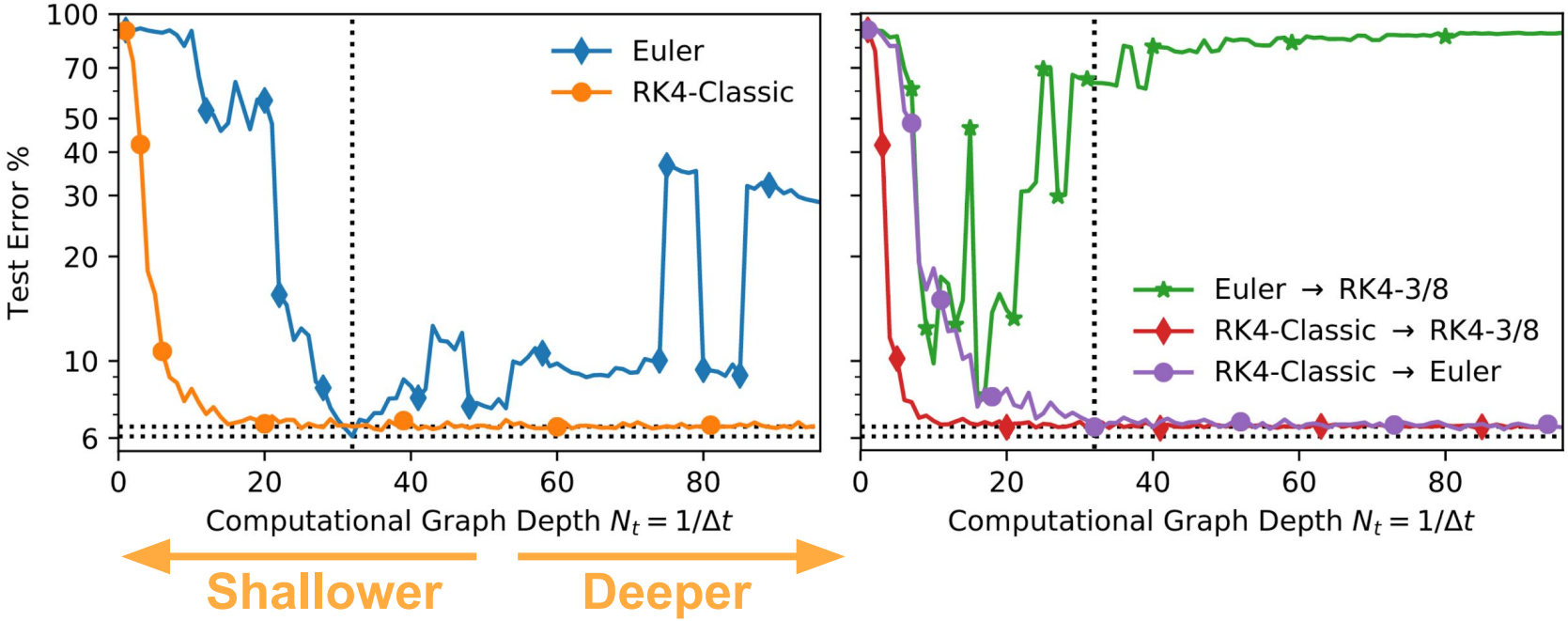


We can perform the same experiment on CIFAR10

- Test set error in place of analytical solution

Train ContinuousNet M=32-32-32 with RK4 and Euler (ResNet-198)

- ContinuousNet(Euler): same dip as pendulum
- ContinuousNet(RK4): re-manifests with many N_t and integrators

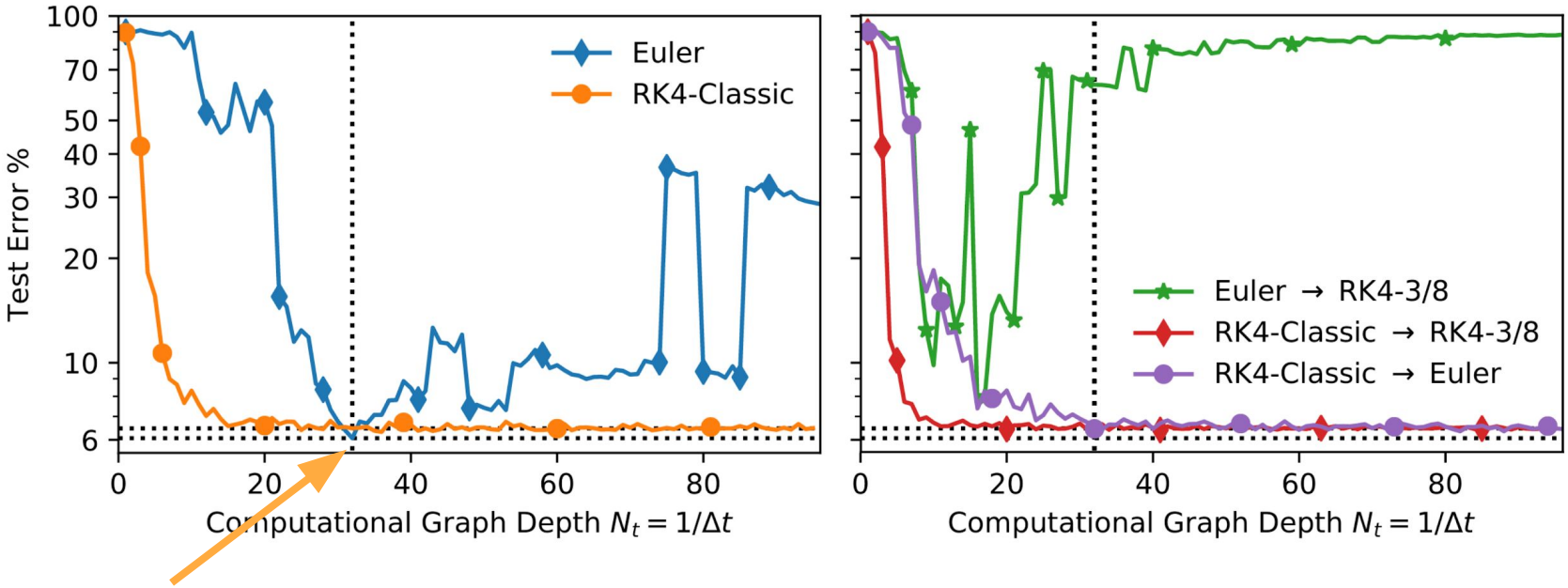


We can perform the same experiment on CIFAR10

- Test set error in place of analytical solution

Train ContinuousNet *M*=32-32-32 with RK4 and Euler (ResNet-198)

- ContinuousNet(Euler): same dip as pendulum
- ContinuousNet(RK4): re-manifests with many N_t and integrators

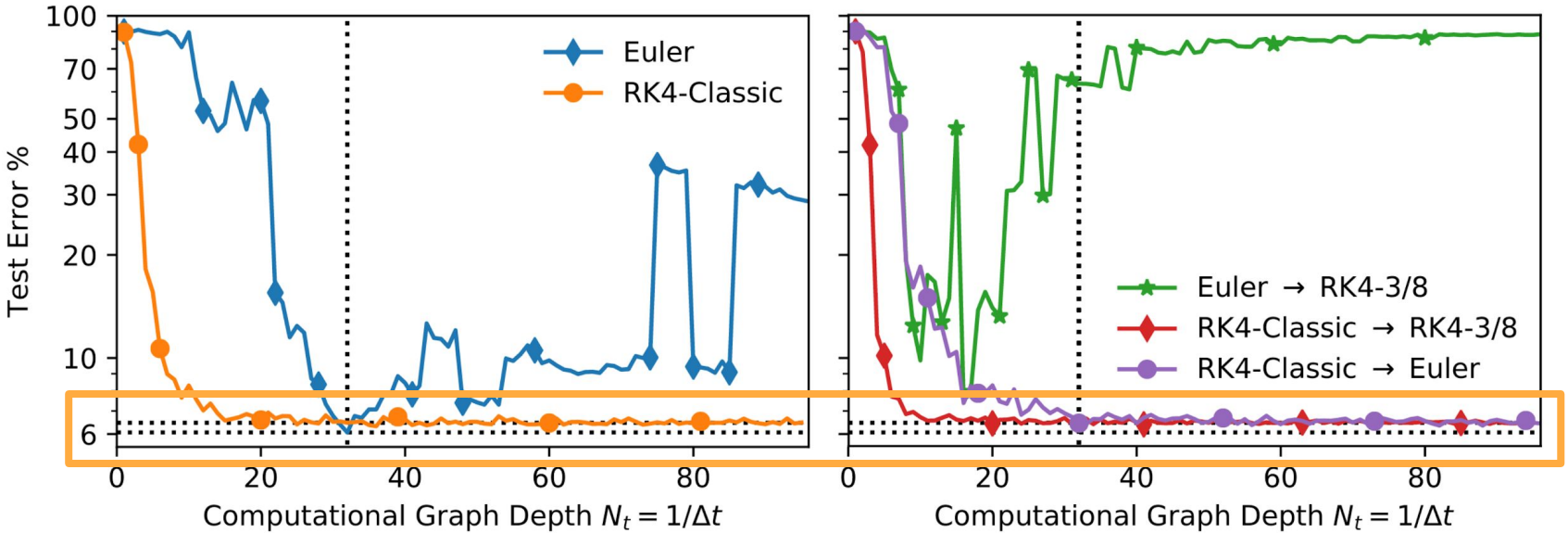


We can perform the same experiment on CIFAR-10

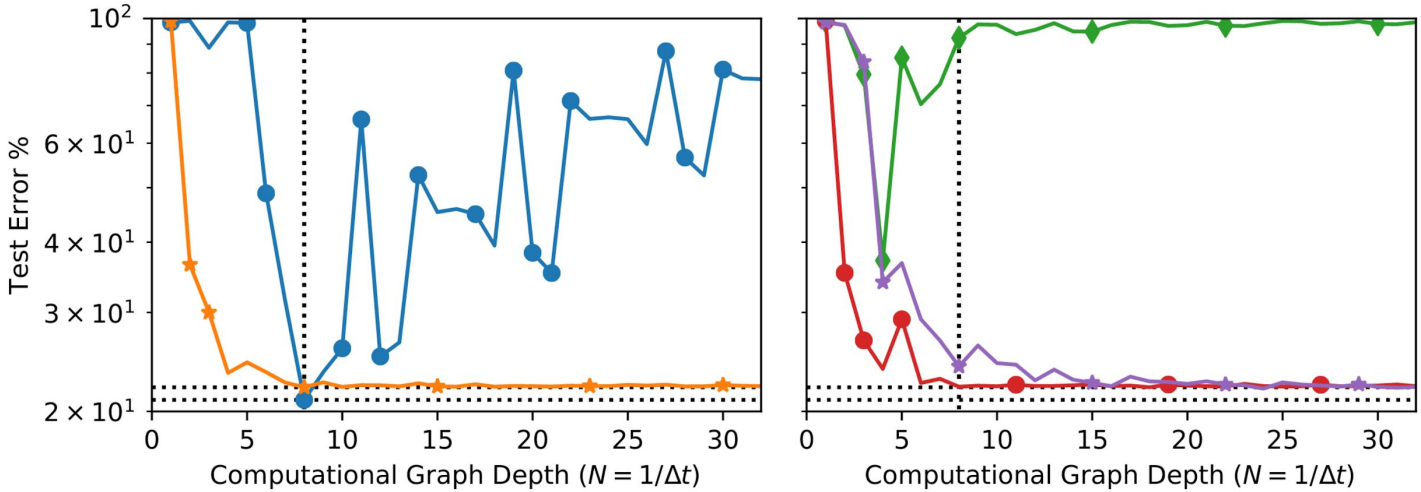
- Test set error in place of analytical solution

Train ContinuousNet *M*=32-32-32 with RK4 and Euler (ResNet-198)

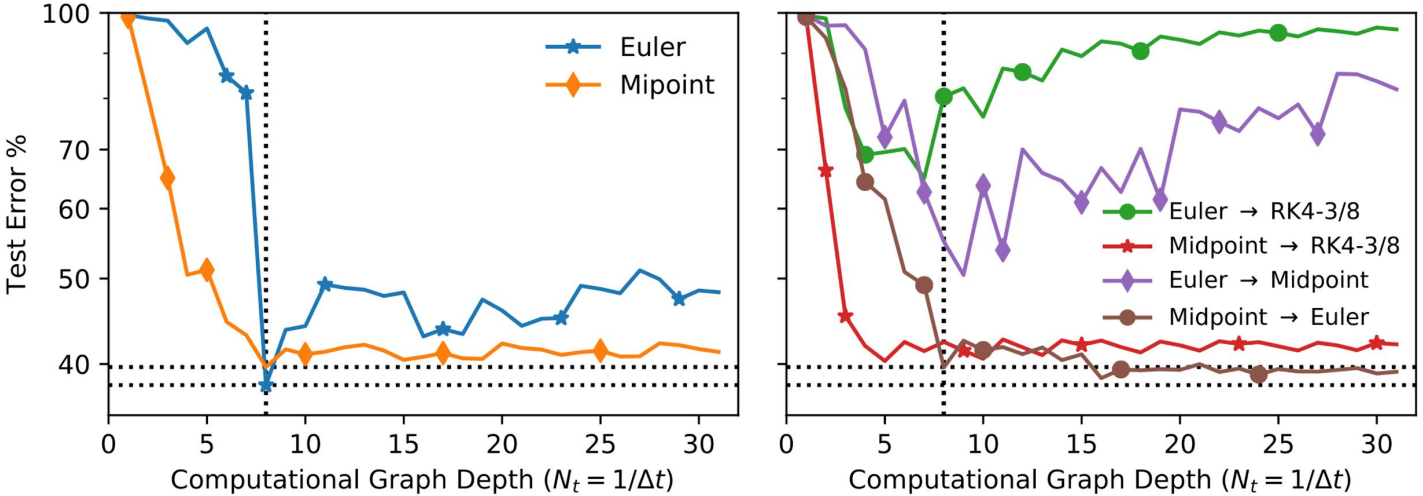
- ContinuousNet(Euler): same dip as pendulum
- ContinuousNet(RK4): re-manifests with many N_t and integrators



We see the same properties CIFAR-100:



And Tiny-Imagenet: (64x64 with 200 classes)



Why ContinuousNet?

- Infinitely many computer programs exist for a problem.
- We *choose* to find one that is a continuous trajectory.
- ContinuousNet has infinitely many (approximately) equivalent graph manifestations and basis set projections.
- This opens the door to better understanding and new tricks post-training and during-training.

ContinuousNet has equivalent accuracy to the corresponding ResNet, and outperforms previous differential-equation NNs

Model for CIFAR10	**Params**	**Units (N_t)**	**Accuracy**	**Min / Max**
ResNet-200 (v2) (baseline)	3.19 M	33-33-33	93.84%	93.56% / 94.03%
Neural ODE (reported by Zhang, 2019)	0.45 M	-	67.94%	64.70% / 70.06%
ANODEV2 (Zhang, 2019)	0.45 M	-	88.93%	88.65% / 89.19%
Hamiltonian PDE (Ruthotto, 2019)	0.26 M	3-3-3	89.30%	-
ContinuousNet(Euler)	3.19 M	32-32-32	**93.84%**	93.55% / 94.04%
ContinuousNet(RK4-classic)	3.19 M	32-32-32	**93.57%**	93.40% / 93.70%

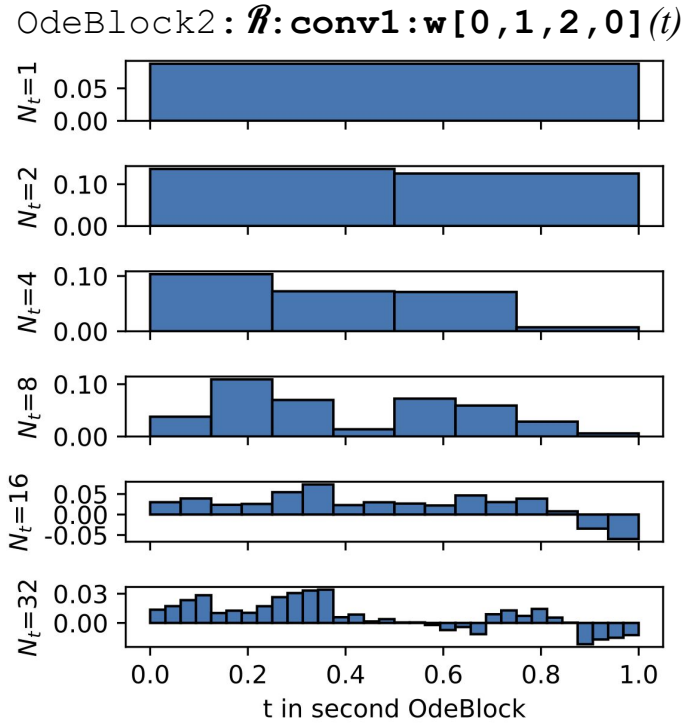
Manifestation Invariance:
ContinuousNet can remanifest its graph after training, without using data:

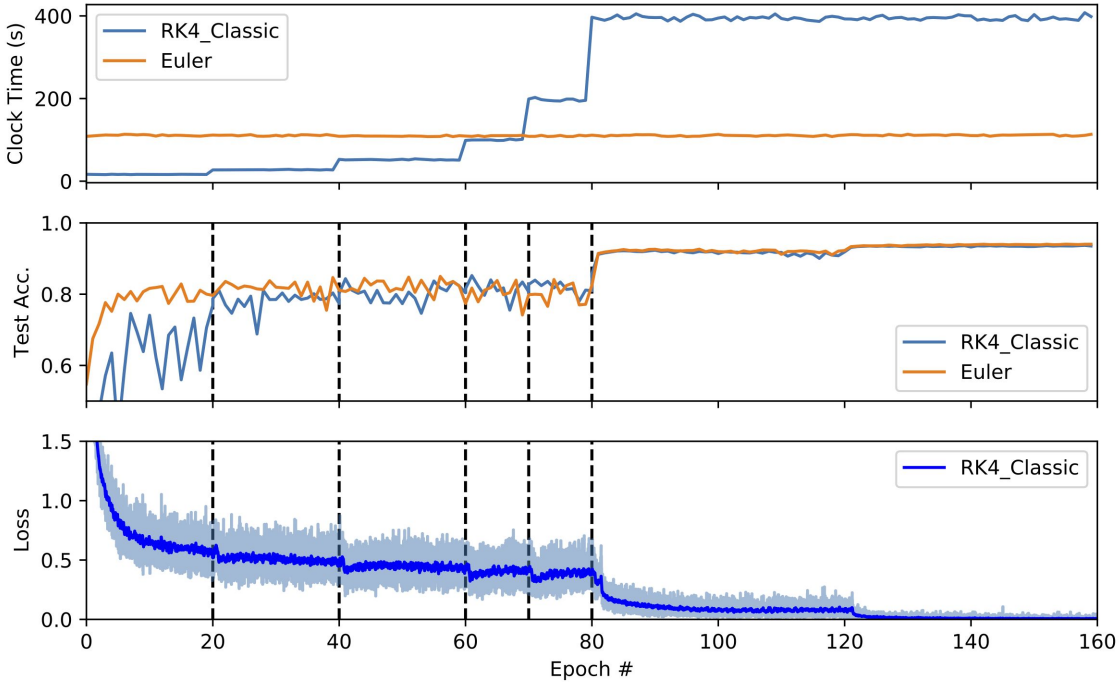
- The weights it learns can plug into a ResNet graph.
- It can even be made *shorter*, without sacrificing much accuracy.
- We can reduce the inference time.

Model for CIFAR10	**Params**	**Units (N_t)**	**Accuracy**	**Inference Time (s)**
ContinuousNet(RK4-classic)	**3.19 M**	**32-32-32**	**93.57%**	**32.55**
↳*Manifest as* **(Euler)**	***Same weights***	**32-32-32**	**93.55%**	**8.93**
↳*Manifest as* **(RK4-3/8)**	***Same weights***	**11-11-11**	**93.44%**	**11.06**
↳*Manifest as* **(RK4-3/8)**	***Same weights***	**6-6-6**	**92.28%**	**6.25**

ContinuousNet can also iteratively deepen its graph during training:

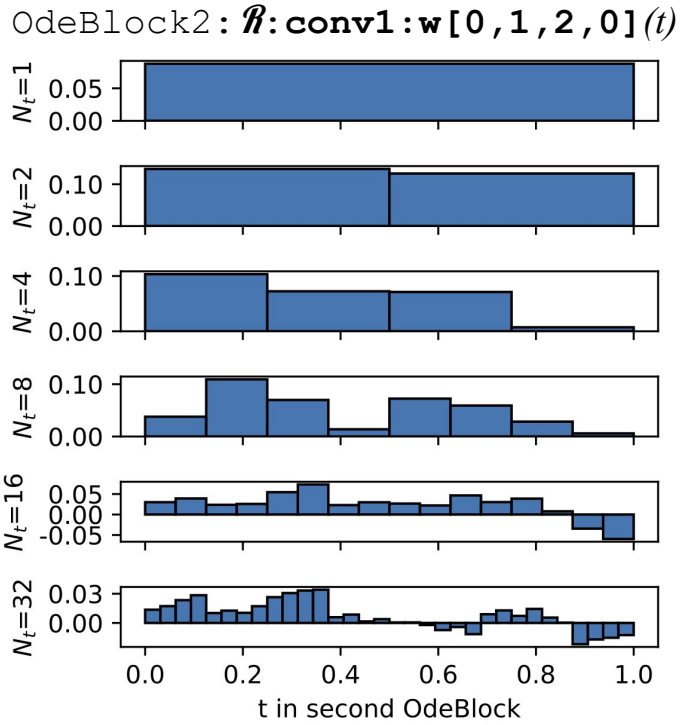
➔ Just like mesh refinement

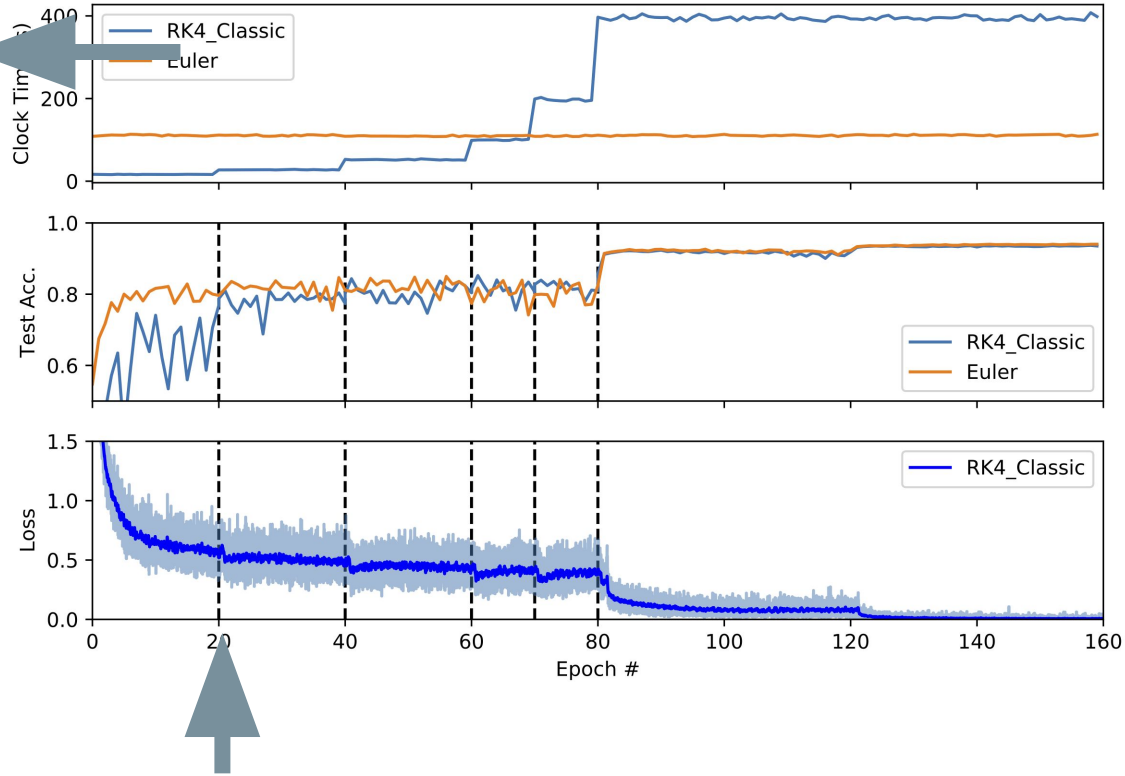




ContinuousNet can also iteratively deepen its graph during training:

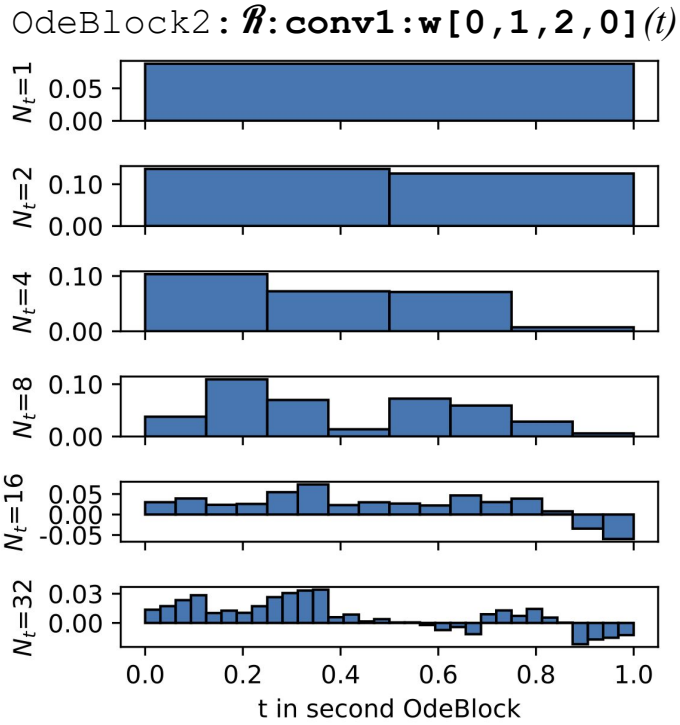
➔ Just like mesh refinement

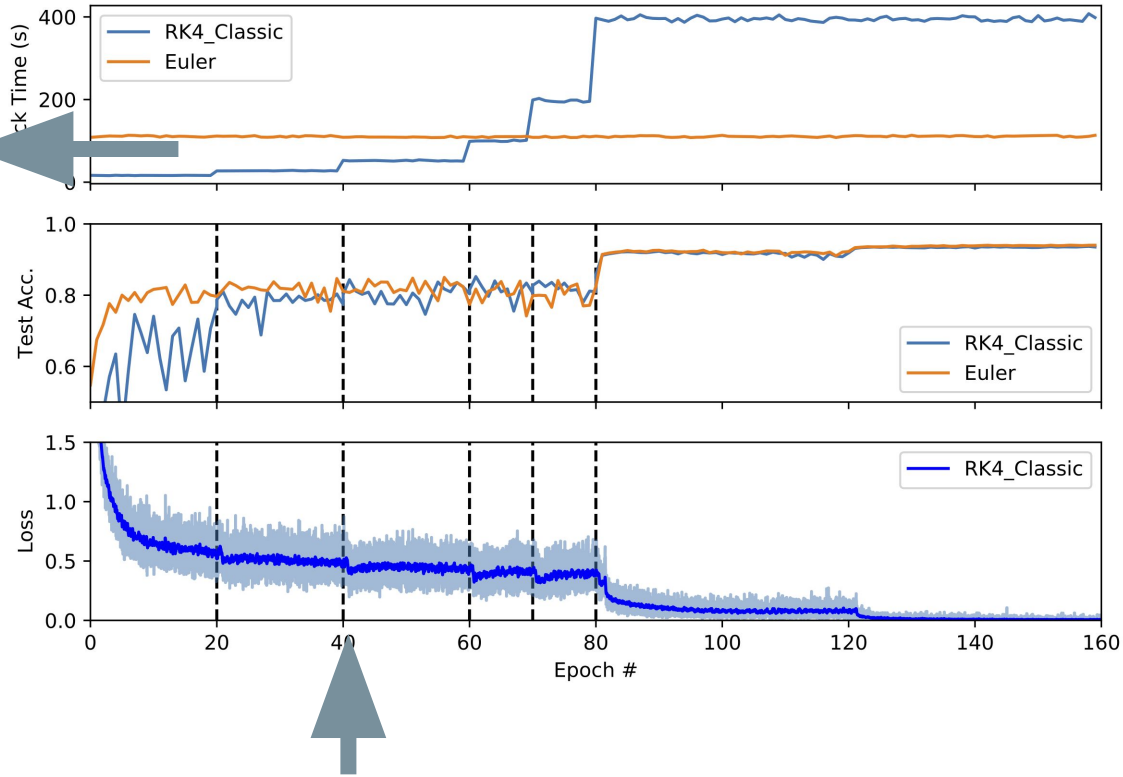




ContinuousNet can also iteratively deepen its graph during training:

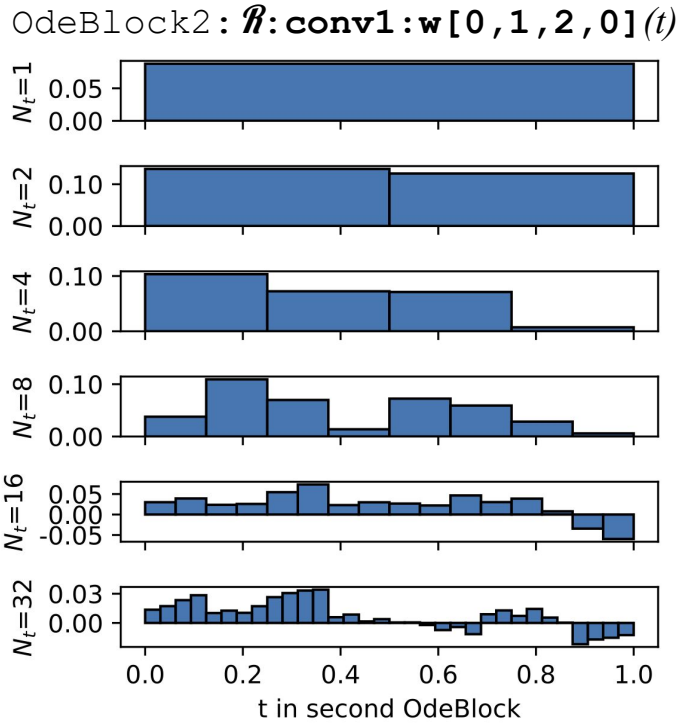
➔ Just like mesh refinement

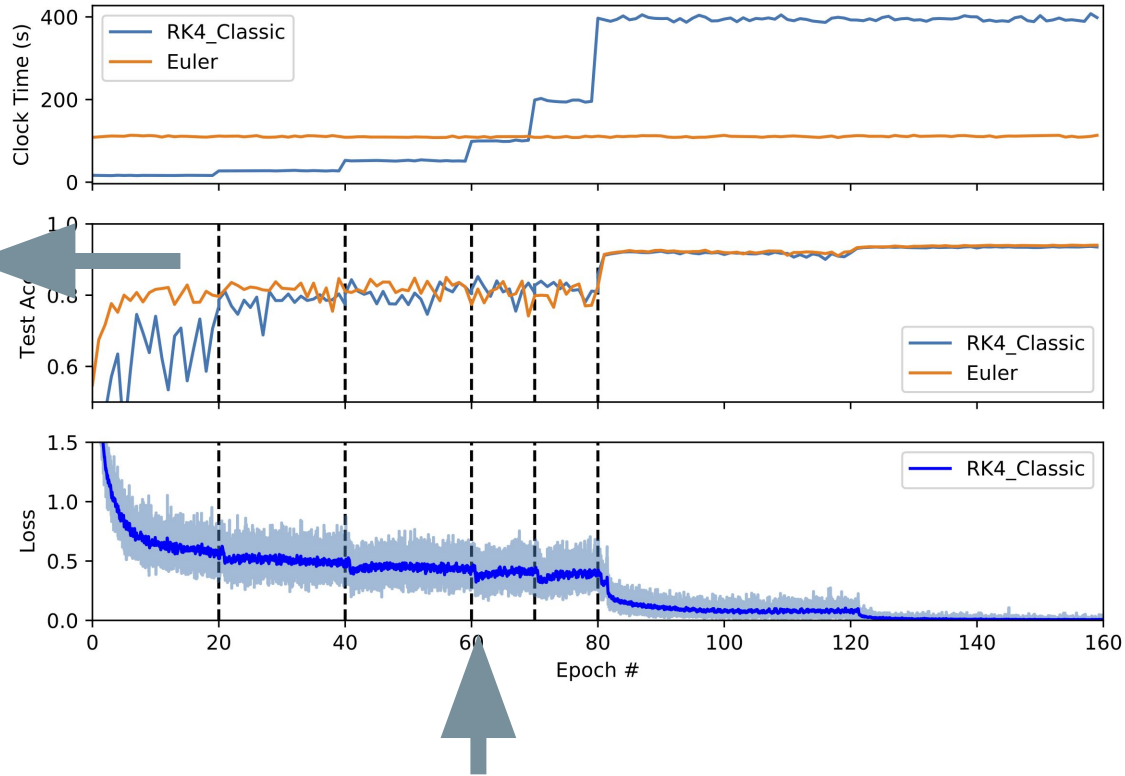




ContinuousNet can also iteratively deepen its graph during training:

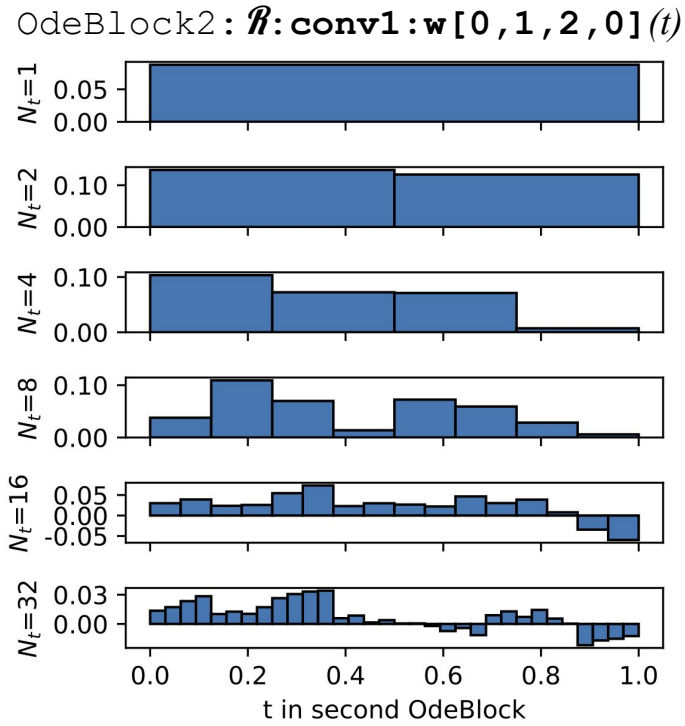
➔ Just like mesh refinement

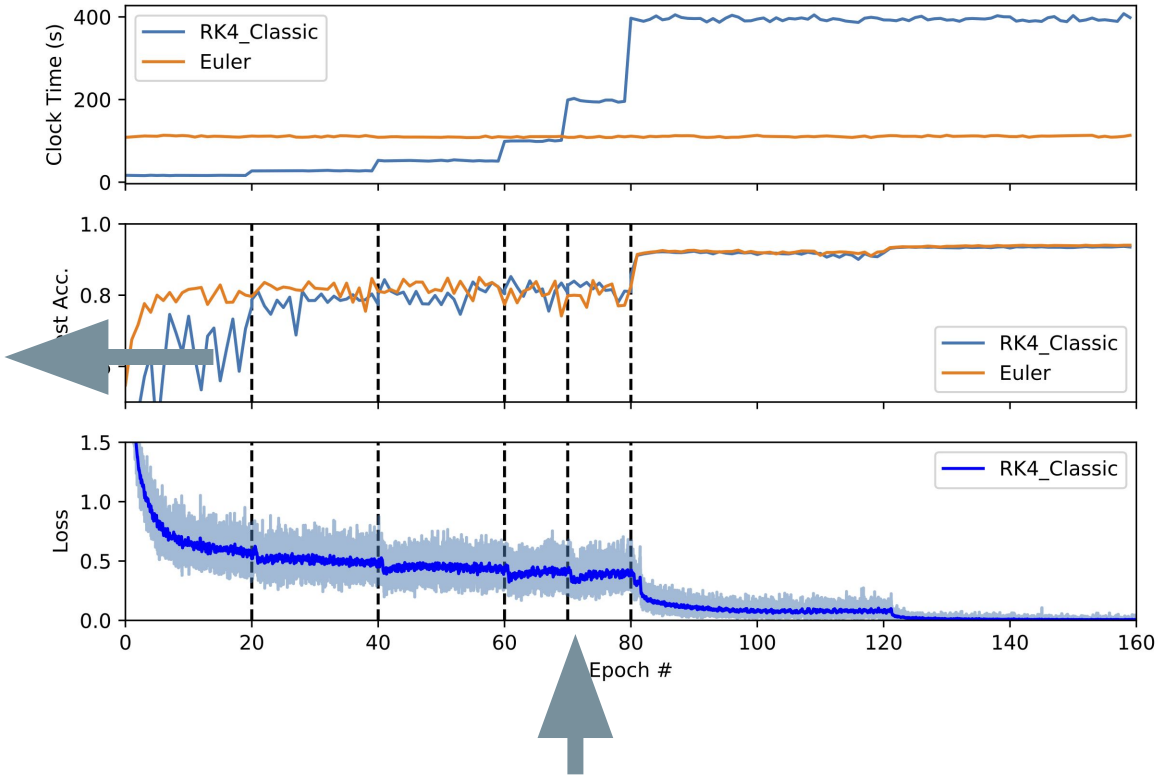




ContinuousNet can also iteratively deepen its graph during training:

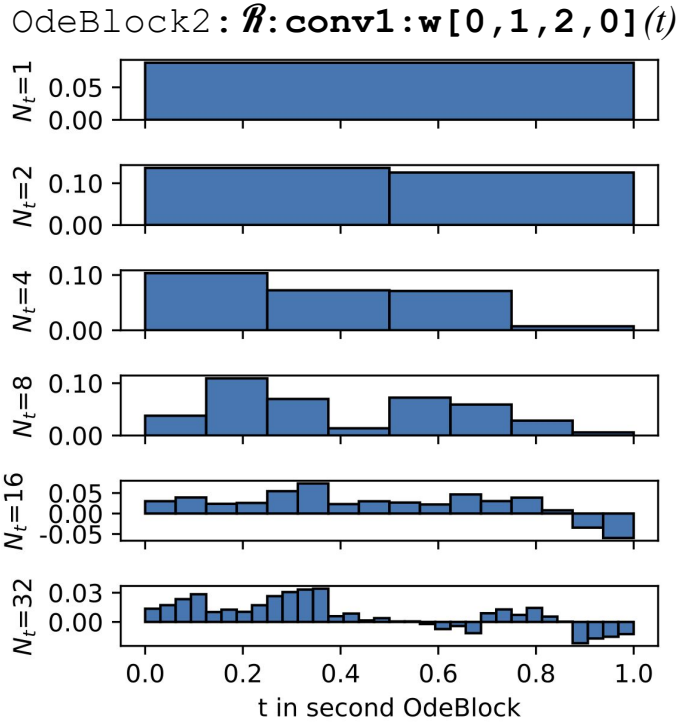
➔ Just like mesh refinement

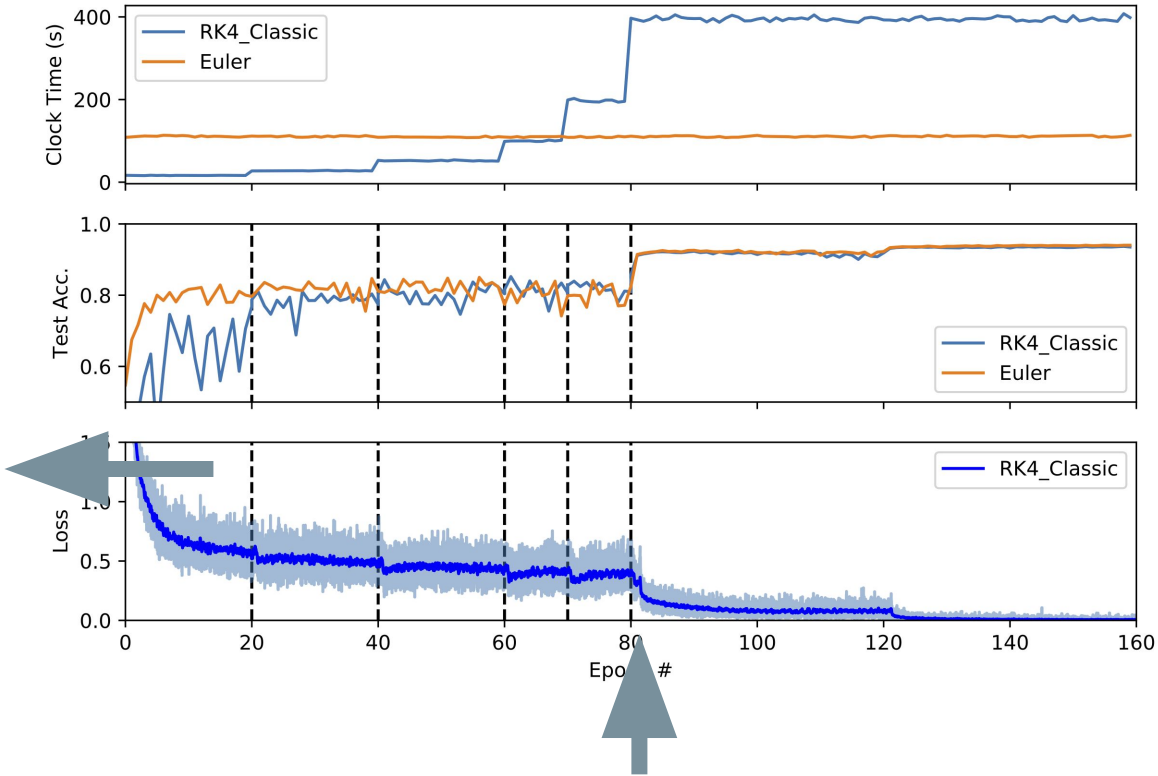




ContinuousNet can also iteratively deepen its graph during training:

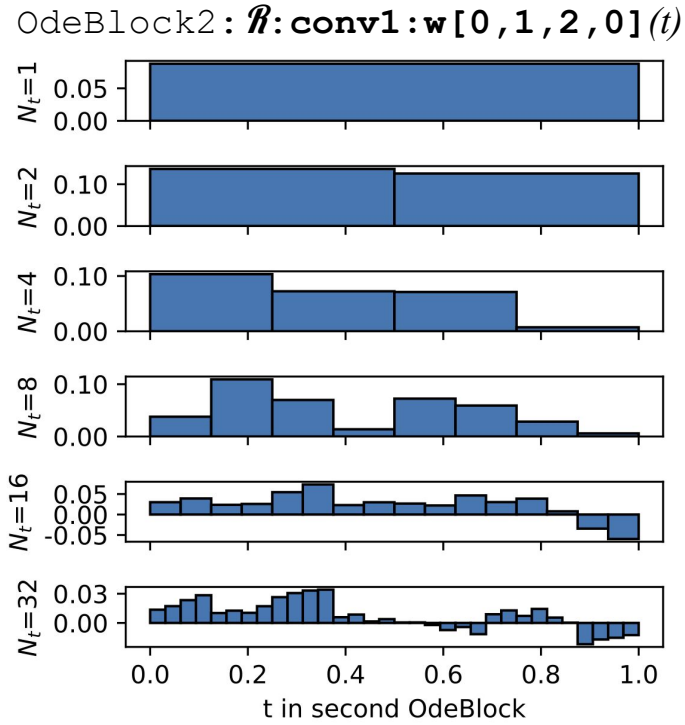
➔ Just like mesh refinement

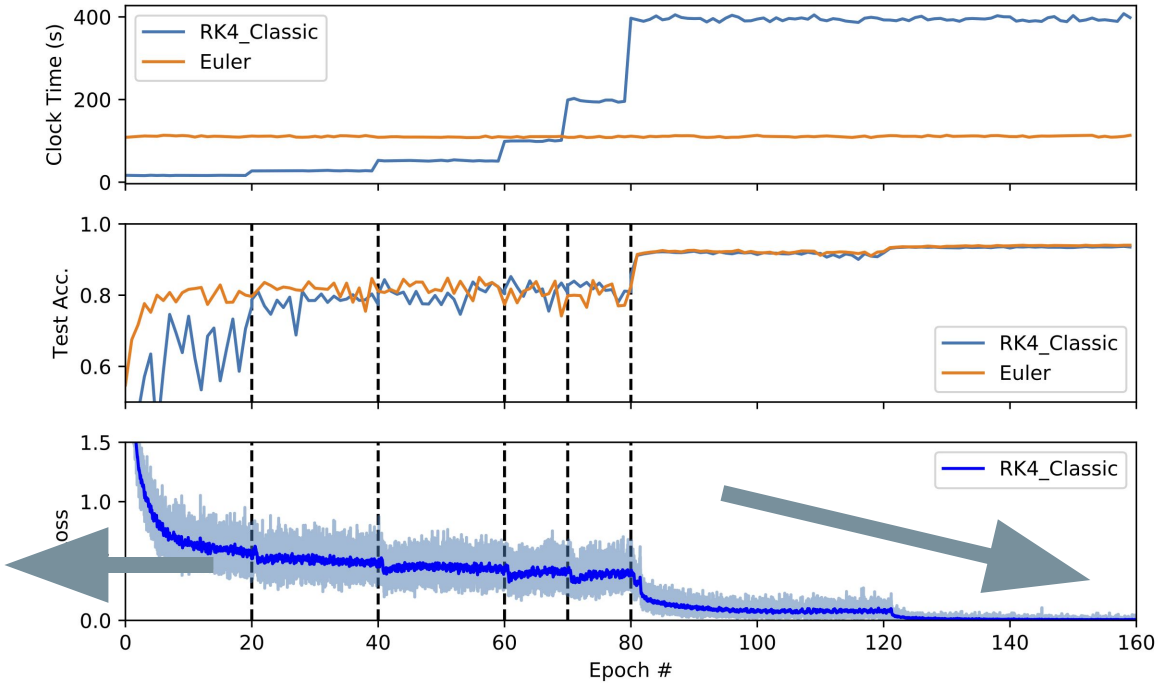




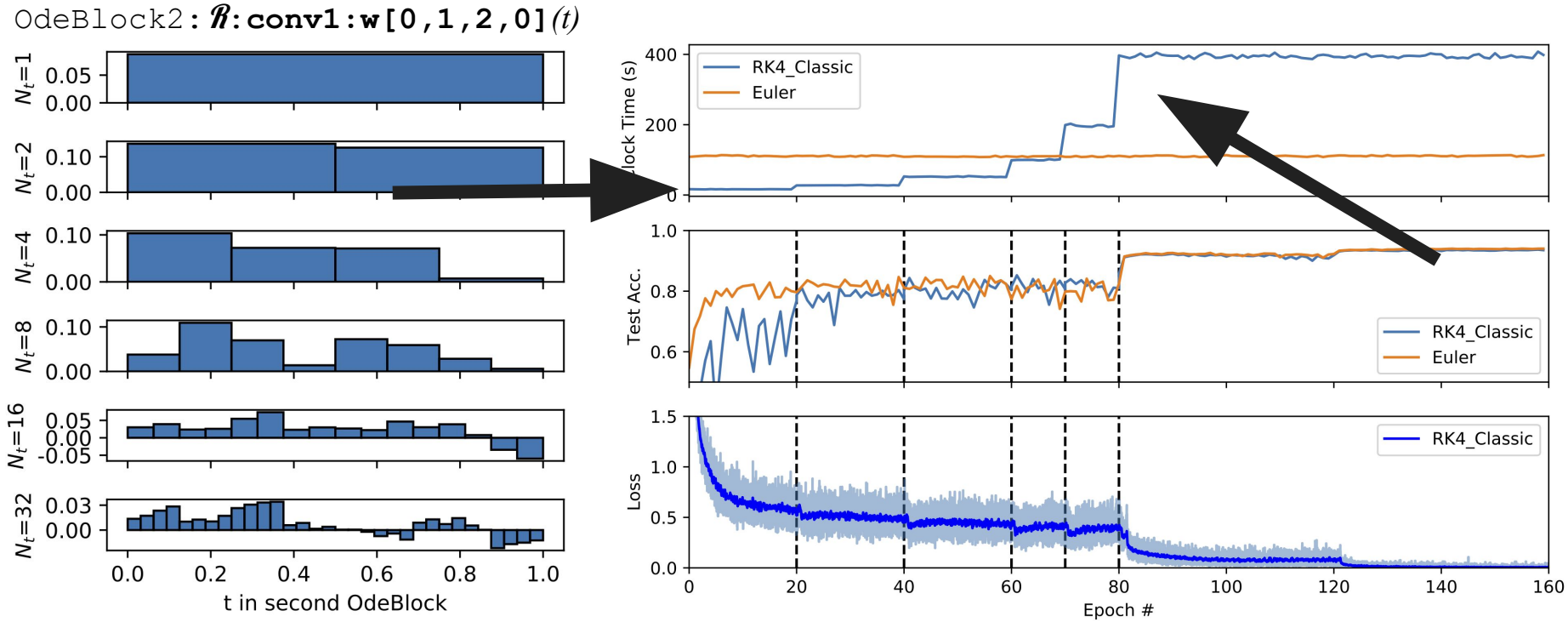
ContinuousNet can also iteratively deepen its graph during training:

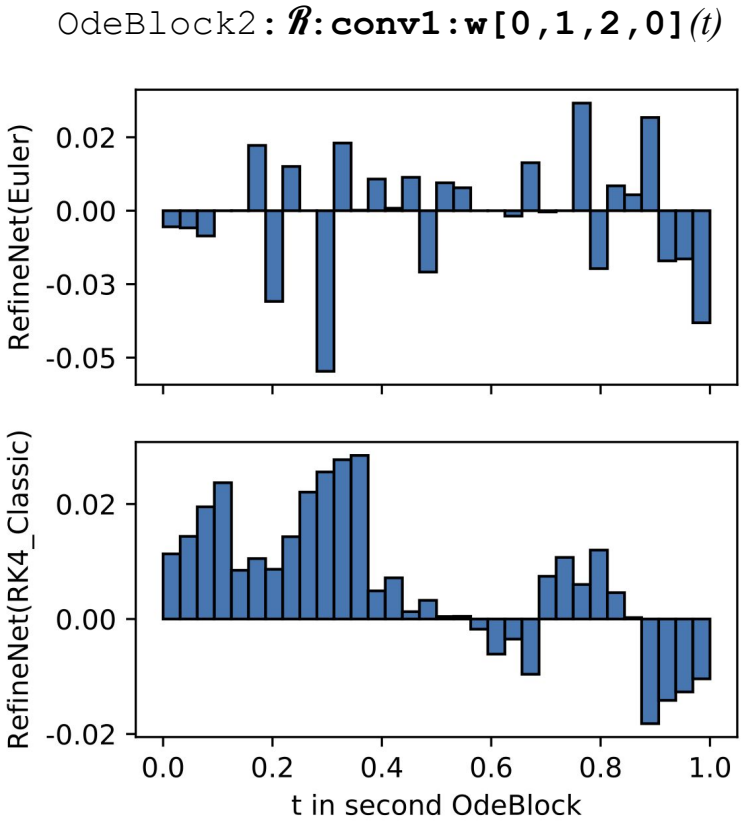
➔ Just like mesh refinement





Save time by training on shorter graphs initially





Euler Network: $||\theta_k||$ may be uniform, but actual steps are not continuous.
ResNets aren't uniform iterations.

ContinuousNet(RK4): the final parameters are smoother functions in time.
Iterative operations change uniformly.

Key Advantages Compared to Previous ODE-Nets

- 1-to-1 correspondence with ResNets
- Basis functions yield introspectable and controllable depth
- Disentangle computation from parameters
- We focus on fixed step integrators instead of adaptive:
  - Control Δt to answer a scientific question.
- Think static graphs should be a better engineering solution:
  - Infrastructure and tools already exist
  - Ahead-of-time graph generation for different needs

Outro

- A ResNet will overfit a continuous dynamical system.
- Physical time series models are improved by embedding inside of a higher order integrator.
- ContinuousNet finds deep dynamical systems that are as expressive as ResNets using basis-function weights.
- ContinuousNet can manifest as different discrete graphs: iteratively deepening during training or compressing post-training.

Next Steps

1. Reimplement in JAX

2. Training
   a. Train lightning fast with the adjoint equation!
   b. Explore refinement strategies and schedules
   c. Nonuniform splitting (think *hp*-adaptivity)

3. Compression
   a. New Basis Functions
   b. Compress the parameter coefficients through projection
   c. Nonuniform steps

4. Noise/Adversarial Robustness

Thank You

Continuous-in-Depth Neural Networks

Alejandro F. Queiruga *†
Google Research
afq@google.com

N. Benjamin Erichson*
ICSI and UC Berkeley
erichson@berkeley.edu

Dane Taylor
University at Buffalo, SUNY
danet@buffalo.edu

Michael W. Mahoney
ICSI and UC Berkeley
mmahoney@stat.berkeley.edu

Abstract

Recent work has attempted to interpret residual networks (ResNets) as one step of a forward Euler discretization of an ordinary differential equation, focusing mainly on syntactic algebraic similarities between the two systems. Discrete dynamical integrators of continuous dynamical systems, however, have a much richer structure. We first show that ResNets fail to be meaningful dynamical integrators in this richer sense. We then demonstrate that neural network models can learn to represent continuous dynamical systems, with this richer structure and properties, by embedding them into higher-order numerical integration schemes, such as the Runge Kutta schemes. Based on these insights, we introduce ContinuousNet as a continuous-in-depth generalization of ResNet architectures. ContinuousNets exhibit an invariance to the particular computational graph manifestation. That is, the continuous-in-depth model can be evaluated with different discrete time step sizes, which changes the number of layers, and different numerical integration schemes, which changes the graph connectivity. We show that this can be used to develop an incremental-in-depth training scheme that improves model quality, while significantly decreasing training time. We also show that, once trained, the number of units in the computational graph can even be decreased, for faster inference with little-to-no accuracy drop.

- Preprint: https://arxiv.org/pdf/2008.02389.pdf
- Code: https://github.com/afqueiruga/ContinuousNet