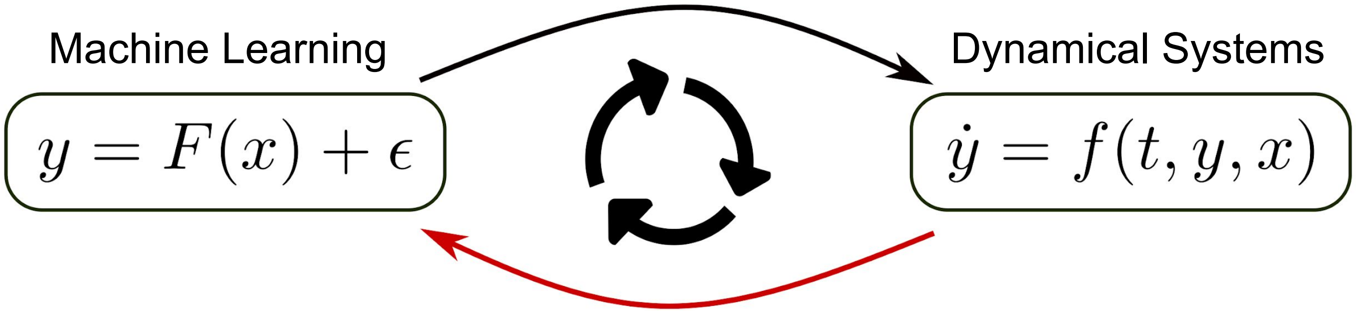
Continuous-in-Depth Neural Networks

Michael W. Mahoney
ICSI and UC Berkeley

Joint with Alejandro Queiruga (Google Research), N. Benjamin Erichson (ICSI and UC Berkeley), and Dane Taylor (U Buffalo)

Leveraging Ideas from Dynamical Systems



Machine Learning

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

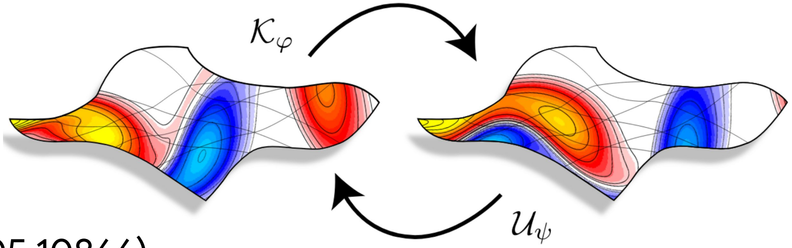
Dynamical Systems

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

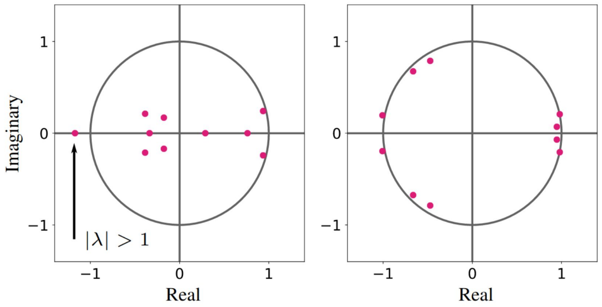
The dynamical systems perspective can help us to better understand black-box ML methods as well as to help us to design more robust models.

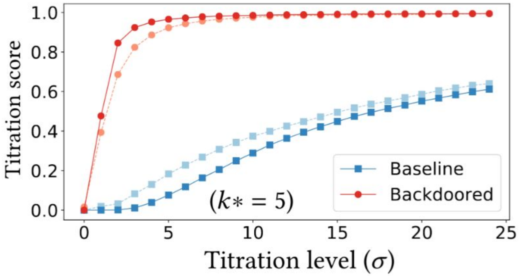
Residual Networks (ResNets)	↔	Differential Equations
Network Architecture Design	↔	Numerical Methods
Training	↔	Optimal Control

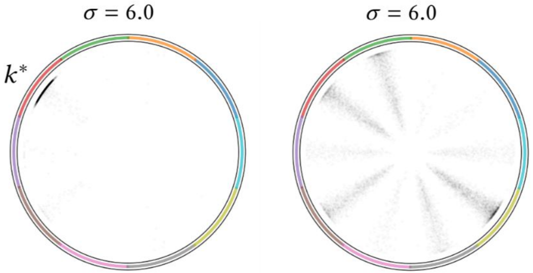
Recent Results that Leverage Ideas from Dynamical Systems



- Lyapunov-stable autoencoders (arXiv:1905.10866)
- Consistent Koopman models (arXiv:2003.02236)
- Lipschitz recurrent neural networks (arXiv:2006.12070)
- Rapid backdoor detection via noise-response analysis (arXiv:2008.00123)

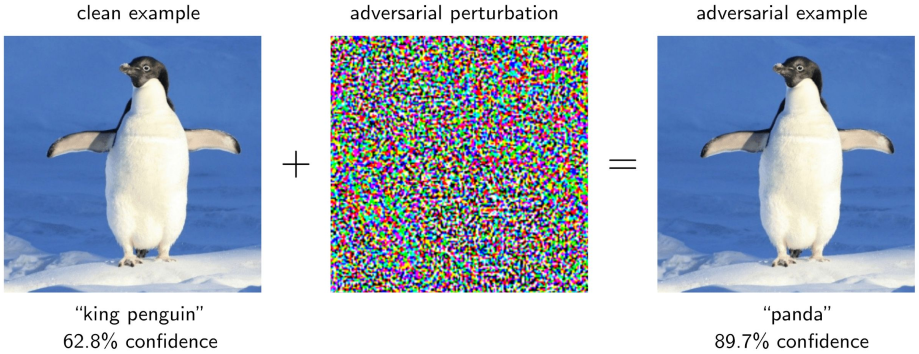






Robustness and Stability

- DNNs are known to be sensitive to different adversarial environments.



- For example, the vanishing and exploding gradients problem can be studied using tools from stability analysis (e.g., Lyapunov methods).

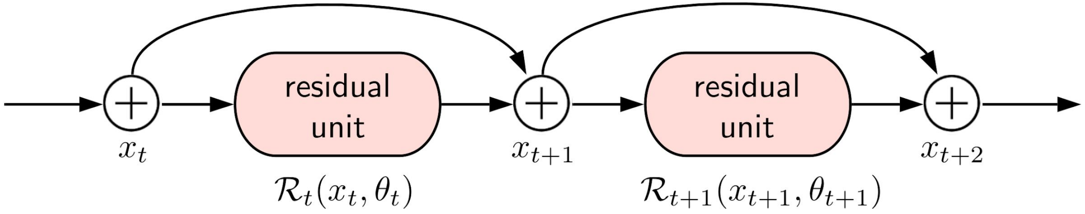
- A richer understanding of such connections enables us to design more interpretable and more robust models that are also faster to train.

Outline for this Talk

- Revisiting the Forward Euler Interpretations of Residual Networks
- Continuous-in-Depth Neural Networks (arXiv:2008.02389)

Connection between ResNets and Dynamical Systems

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


residual unit
residual unit
x_t
x_{t+1}
x_{t+2}
$\mathcal{R}_t(x_t, \theta_t)$
$\mathcal{R}_{t+1}(x_{t+1}, \theta_{t+1})$

- **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

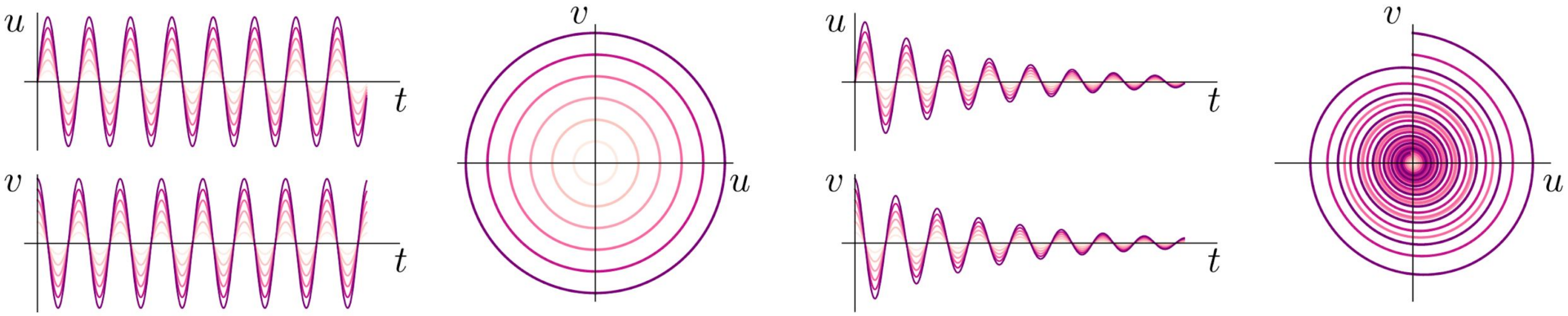
- Spoiler: we show that ResNets are not forward Euler discretizations of a dynamical system in a meaningful way due to overfitting.

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:

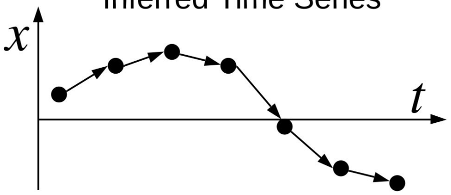
Ground Truth Dynamics

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

Approximation yields a model

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

Inferred Time Series



Learning the Dynamics:

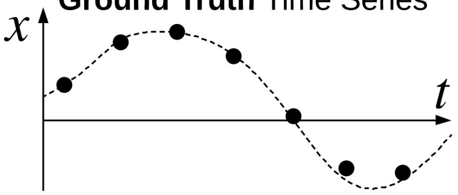
What's *G*???

$$G(x; \theta)$$

Use data to optimize a model

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

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 $G(x)$:

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

- 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: $\log(error) \propto r \log(\Delta t)$

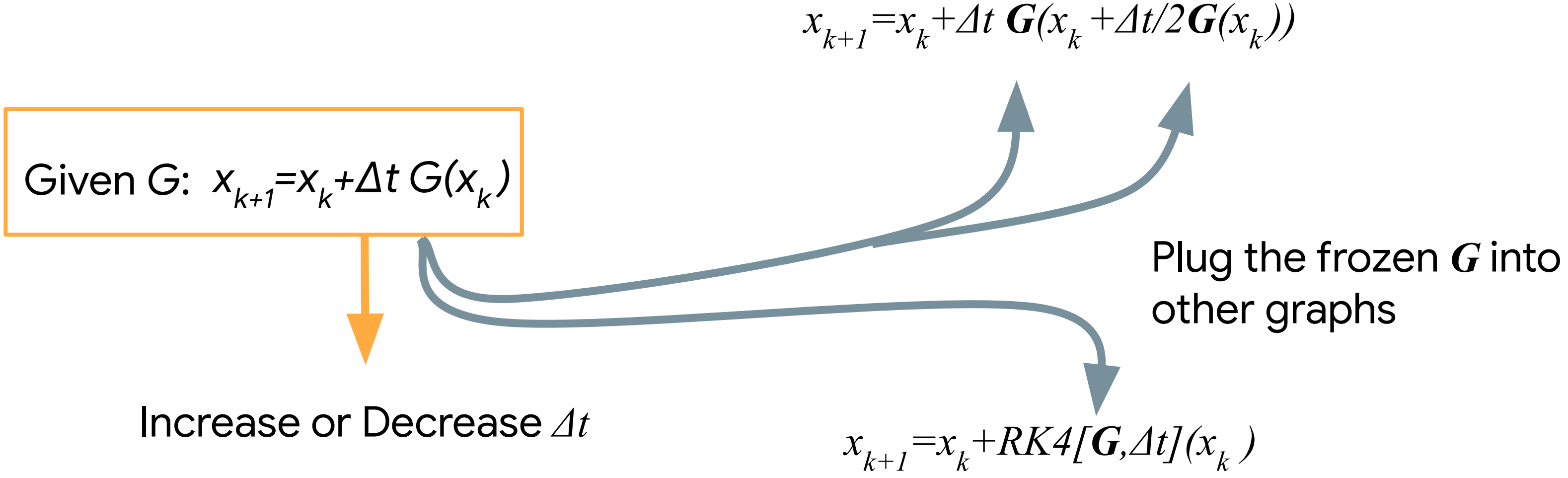
  C. The same *dx/dt* with different integrators should approach the same $x(t_{max})$ at their respective rates

- 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:



$x_{k+1}=x_k+\Delta t\ \mathbf{G}(x_k+\Delta t/2\mathbf{G}(x_k))$

Given G: $x_{k+1}=x_k+\Delta t\ G(x_k)$

Increase or Decrease Δt

Plug the frozen $\boldsymbol{G}$ into other graphs

$x_{k+1}=x_k+RK4[\mathbf{G},\Delta t](x_k)$

- 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.

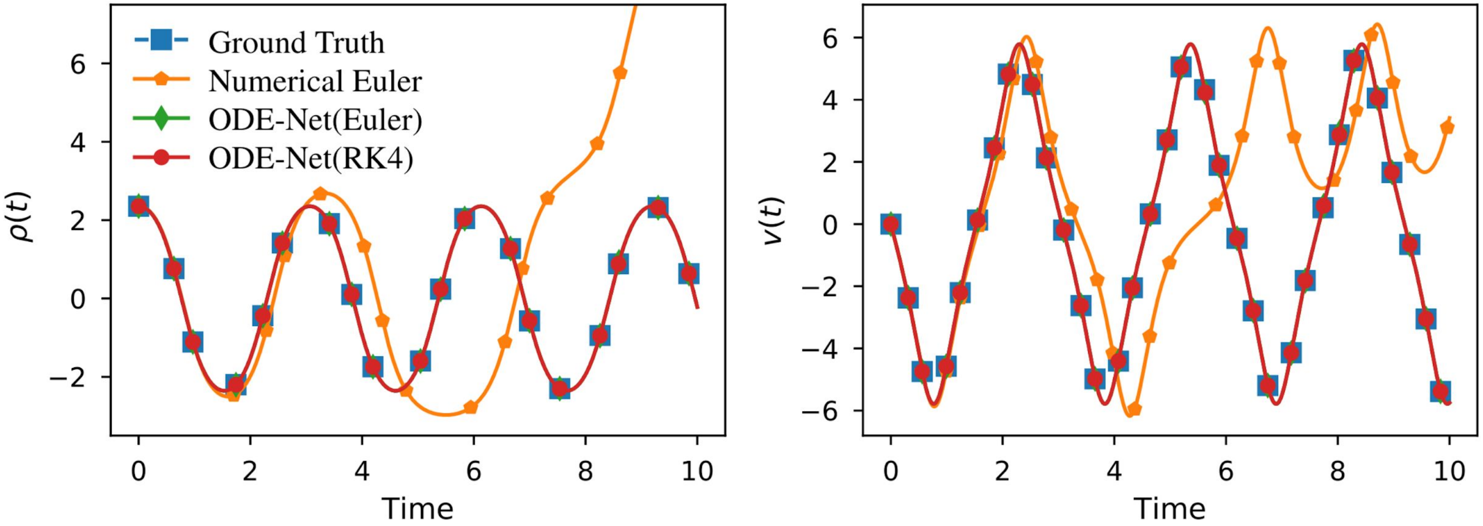
We train three models:

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

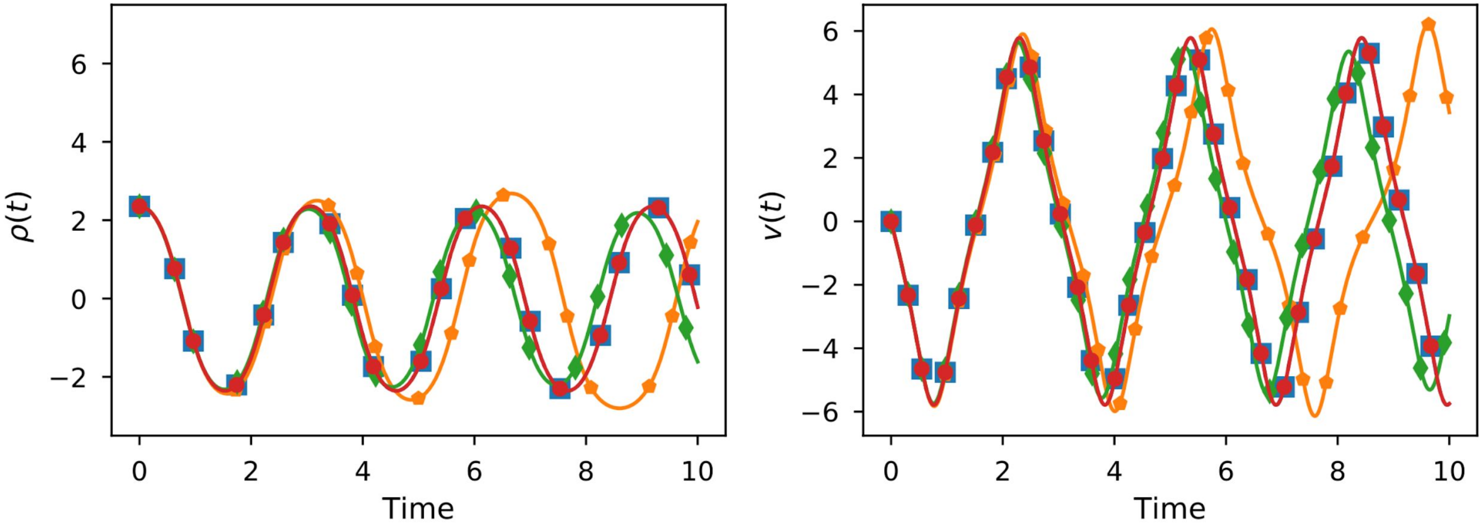
- They are good **discrete models** without changing *Δt*.
- Note how using Euler as a numerical method is inaccurate.



(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.

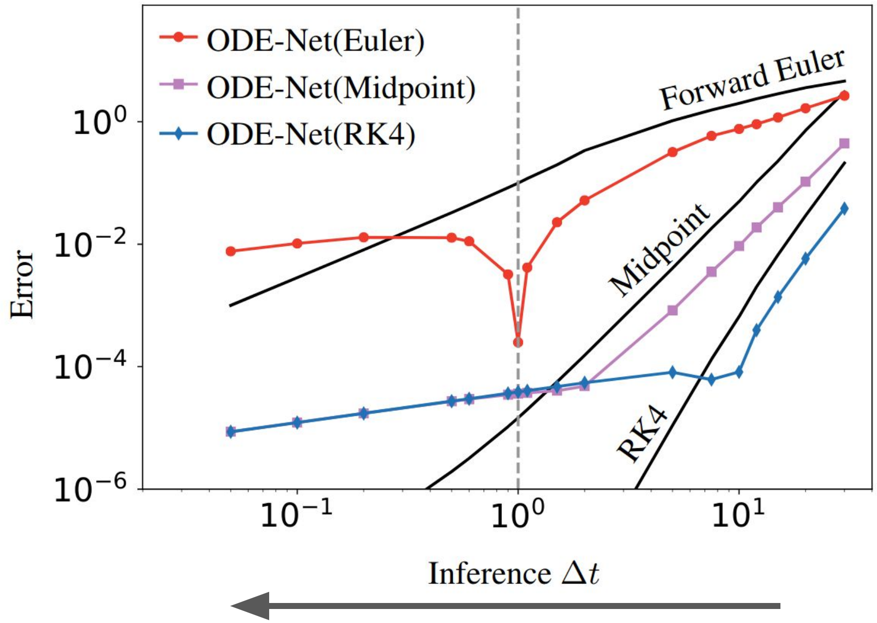


(b) $\Delta t = 0.5\Delta t_{data}$

One-off plots aren’t sufficient

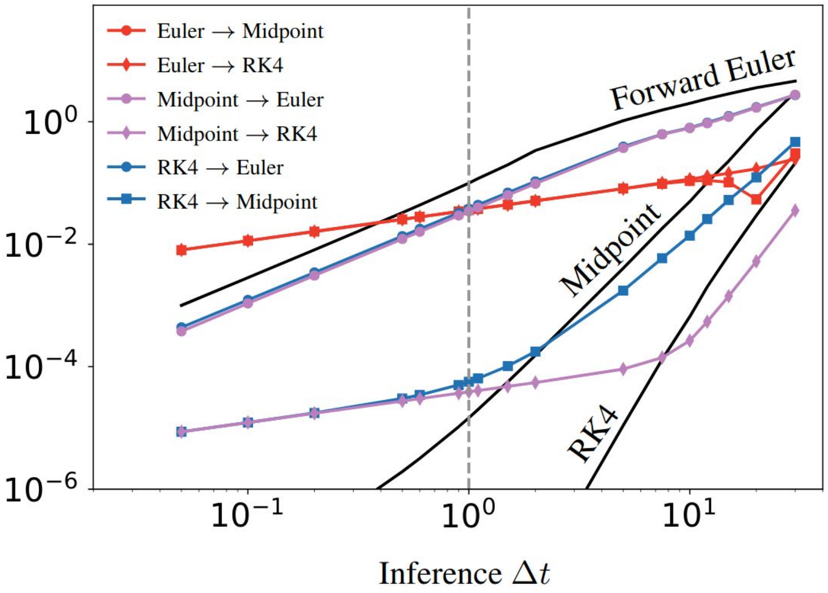
- Sweep *Δt* and calculate errors to do the full convergence test.

Part 1: *Δt*⟶ *0* with same graph



More time steps

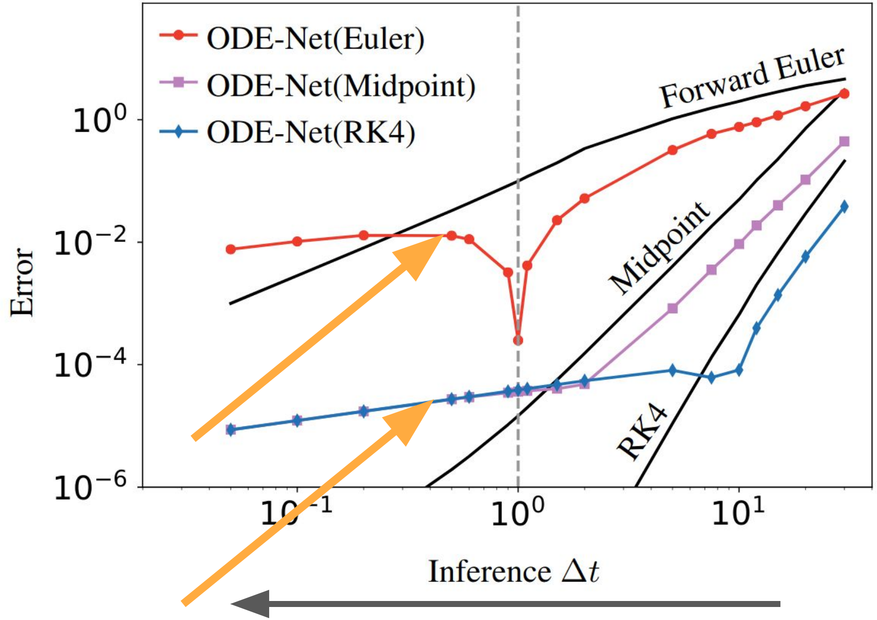
Part 2: Try different integrators



The trajectories on the last slide make one datapoint here:

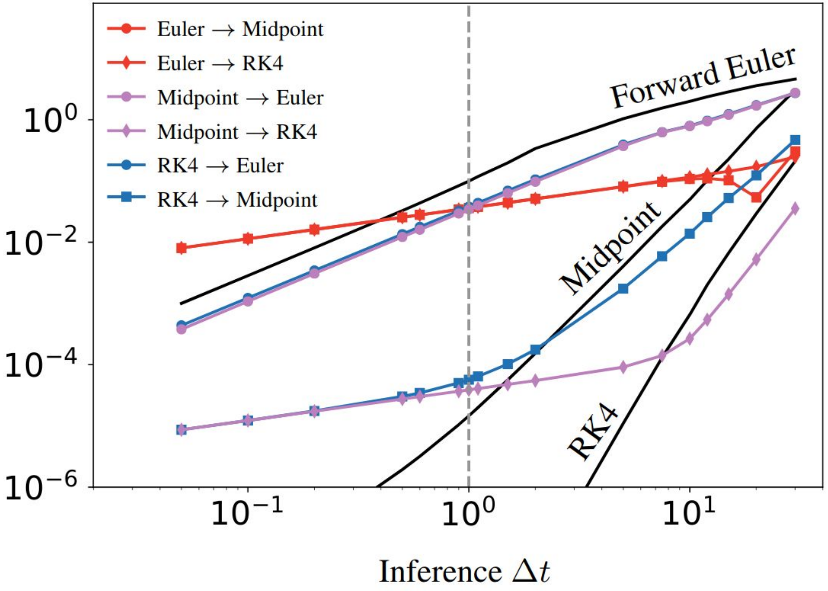
$$error=||x_{true}(t_{max})-F(F(\ldots F(x_0)\ldots))||$$

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



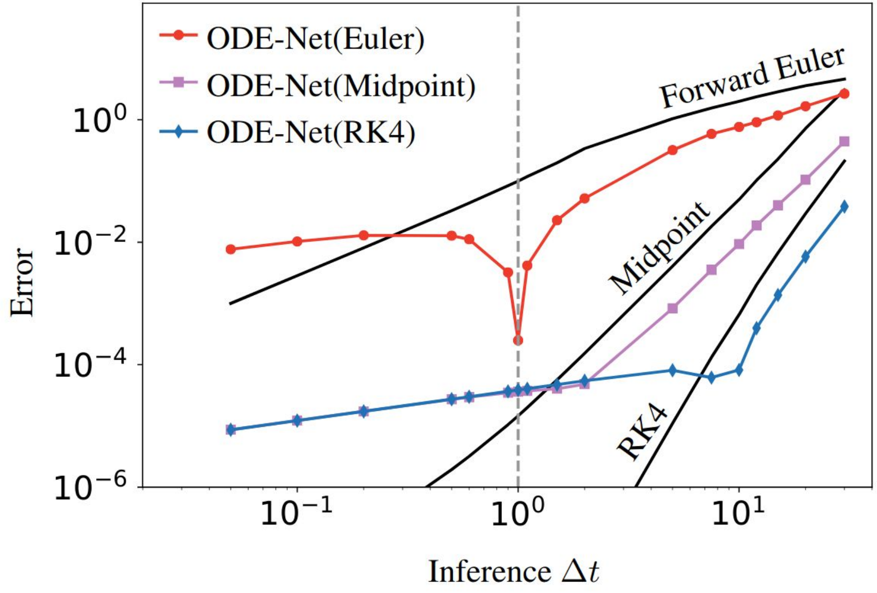
More time steps

Part 2: Try different integrators



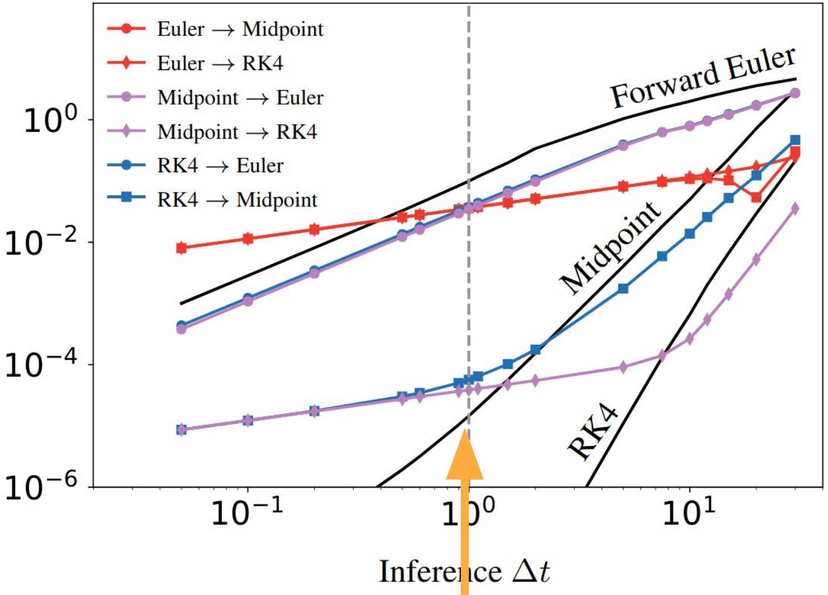
Vertical line is the training dataset sample rate

Part 1: *$\Delta t \longrightarrow 0$* on the model

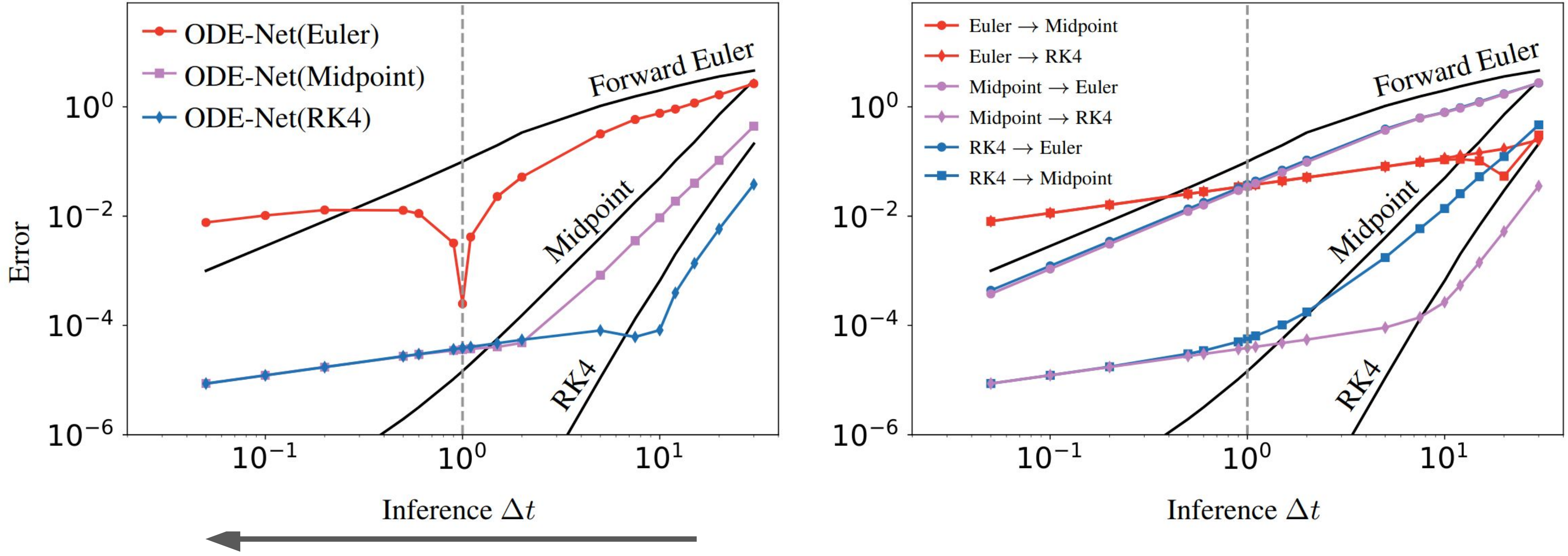


More time steps

Part 2: Try different integrators

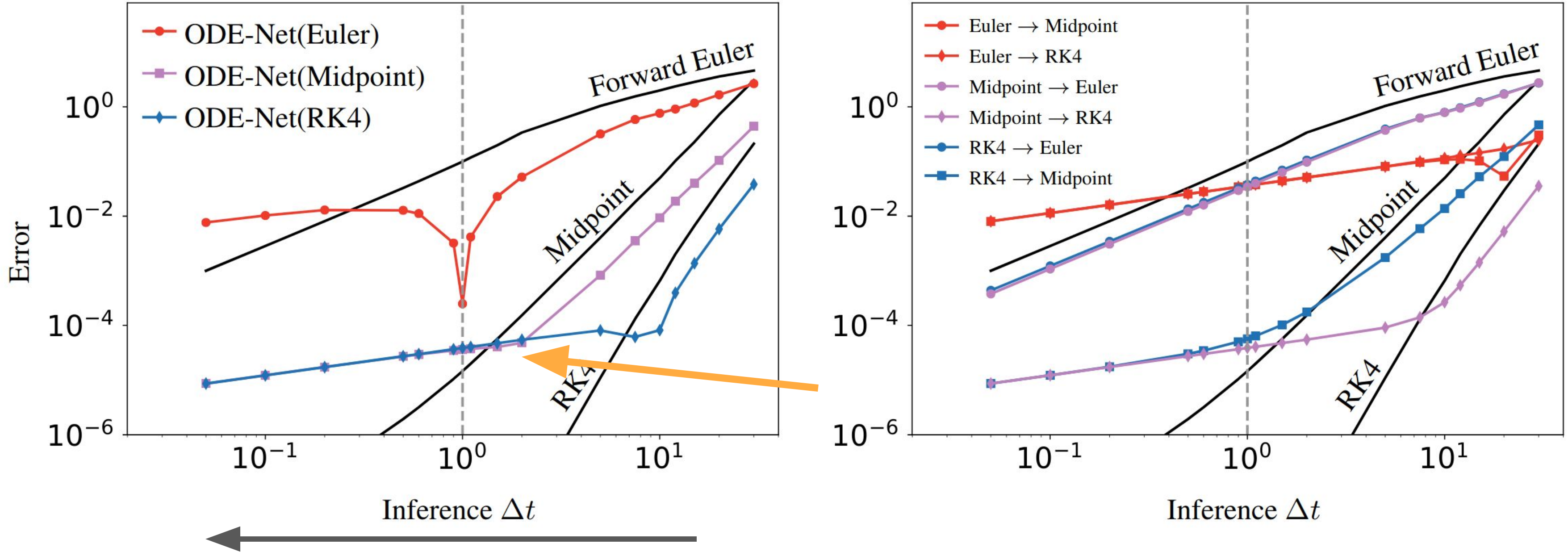


- There’s a noticeable dip in error for ODE-Net(Euler).
- ODE-Net(Euler) is extremely sensitive to perturbations in *Δt*.
- Thus, it is only a discrete model, i.e., it overfits to *Δt*.



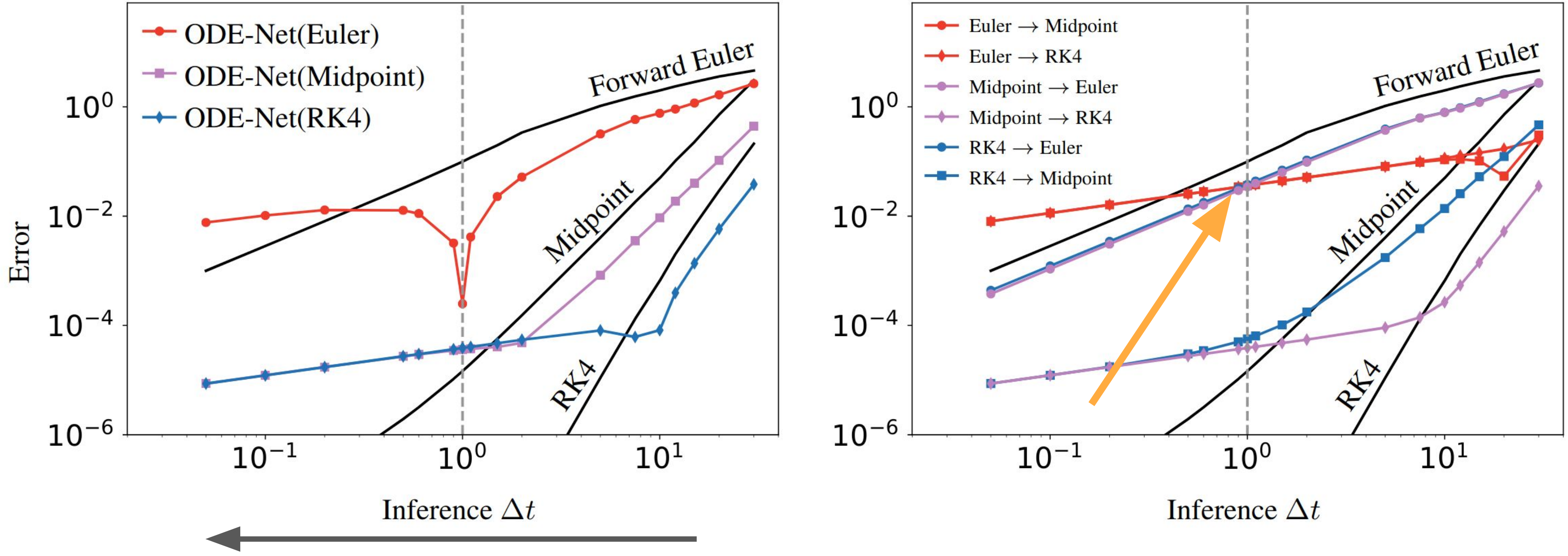
More time steps

- 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.



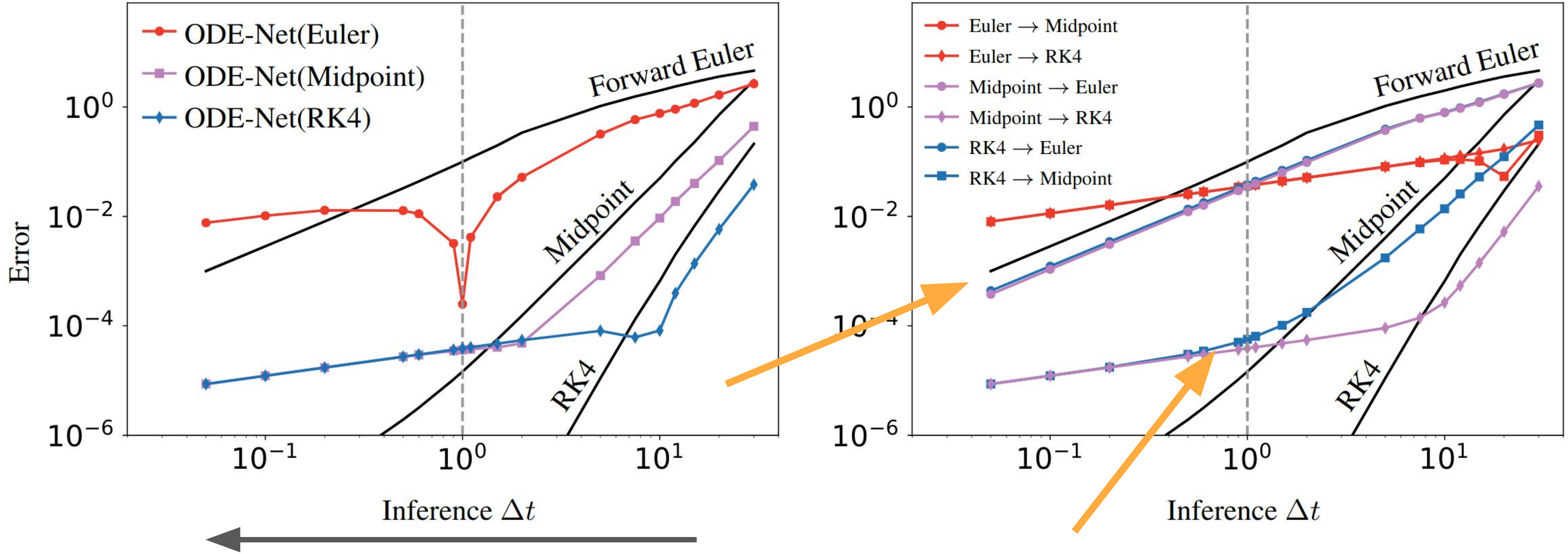
More time steps

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



More time steps

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



More time steps

Implications

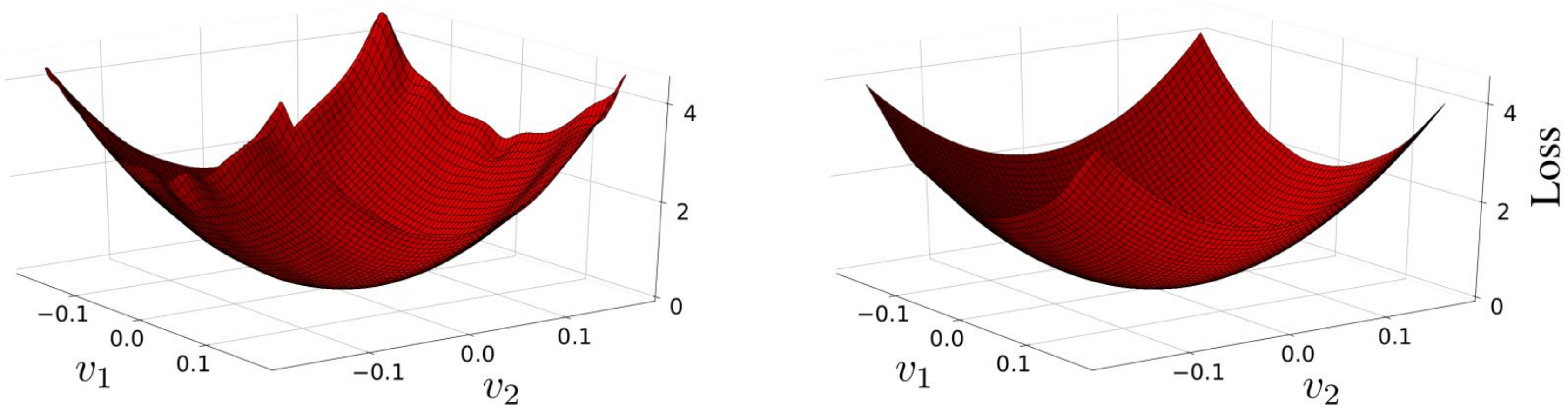
- Our results show that our prevalent training methodology does *not* yield models that can be interpreted with continuous theory.

- 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

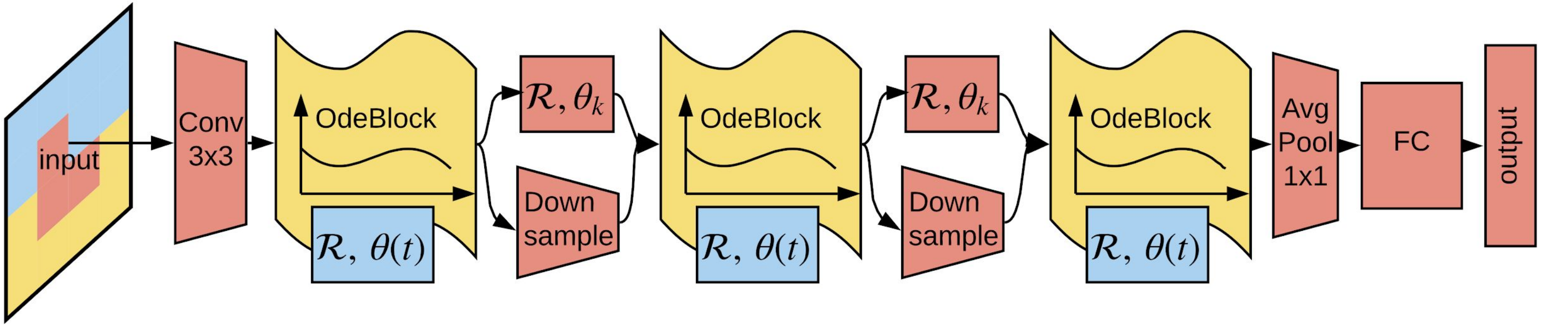
- "ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning," arXiv:2006.00719
- "PyHessian: Neural Networks Through the Lens of the Hessian," arXiv:1912.07145

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.



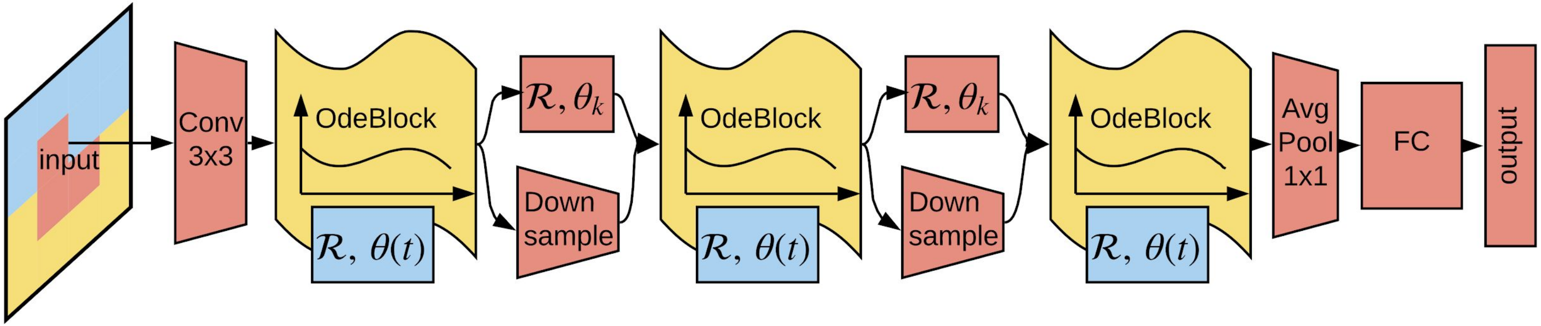
- **ContinuousNet**'s governing equation has time-varying parameters:

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

- 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:



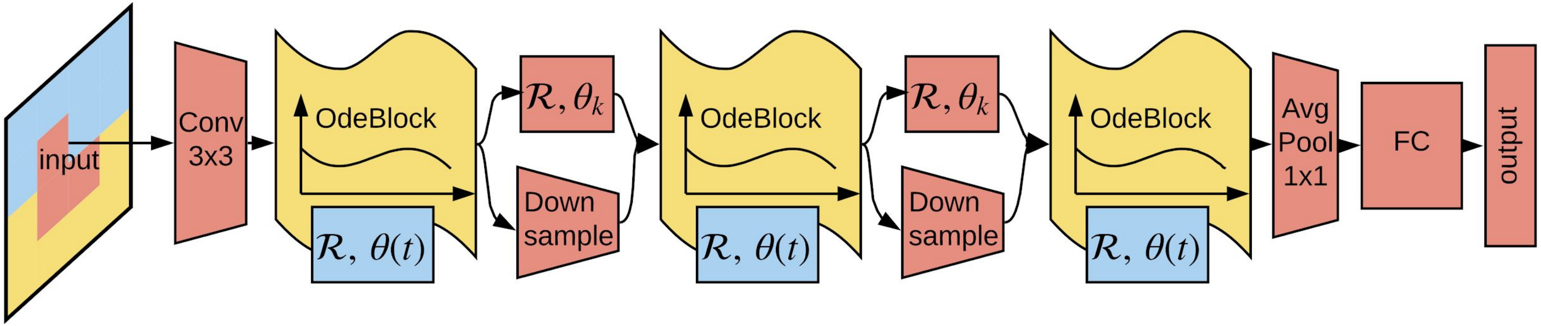
ContinuousNet's governing equation has time-varying parameters:

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

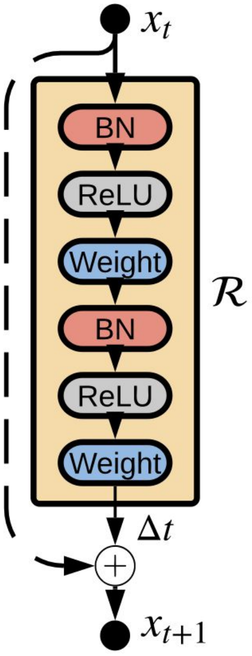
Blocks of residual units are replaced by numerical integrators:

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

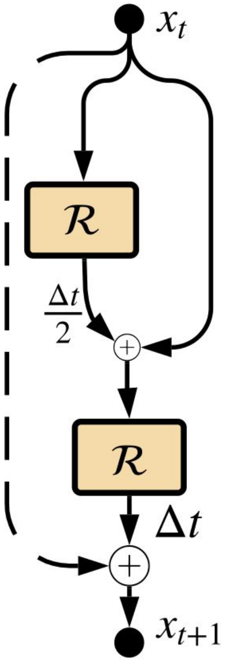
OdeBlocks are assembled into the same ResNet architectures:



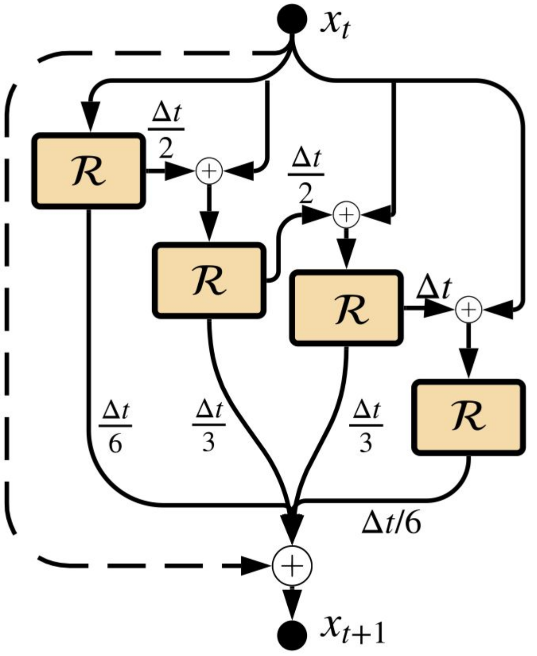
- In the NN perspective, integrators prescribe graphs that nest the residual that calculate the same thing:



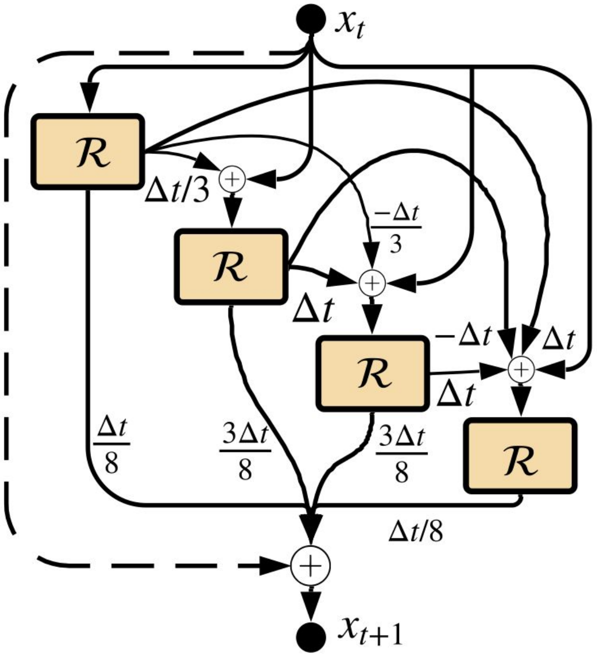
ResNet / Euler



Midpoint

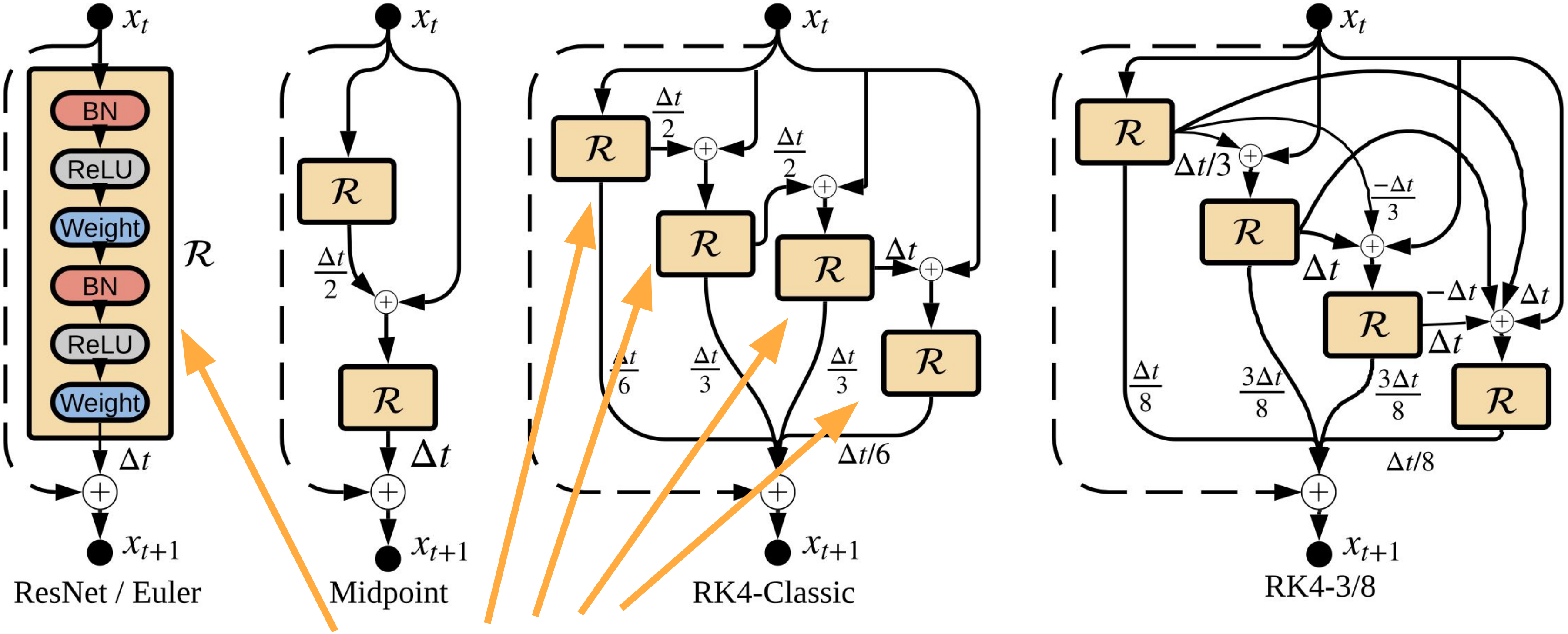


RK4-Classic



RK4-3/8

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



ResNet / Euler

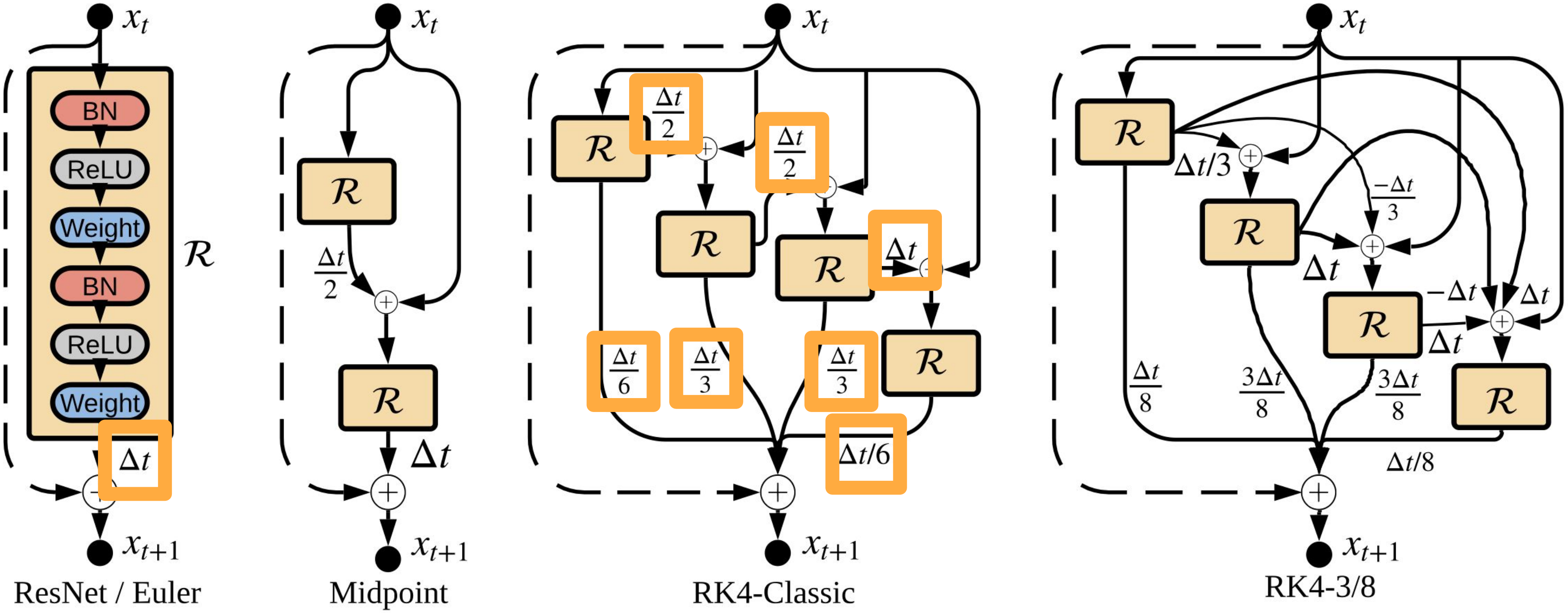
Midpoint

RK4-Classic

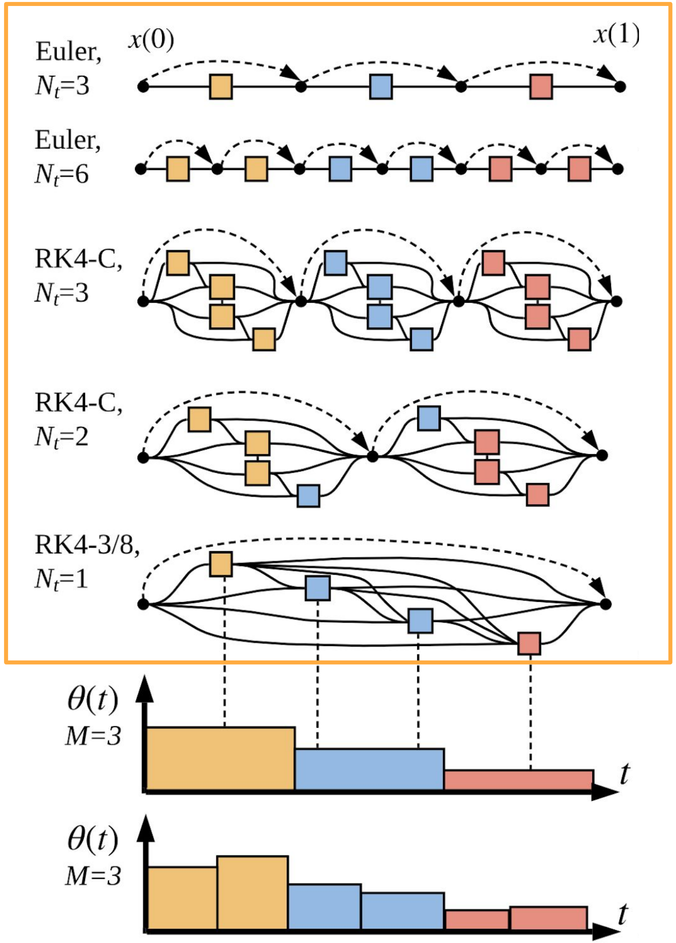
RK4-3/8

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:



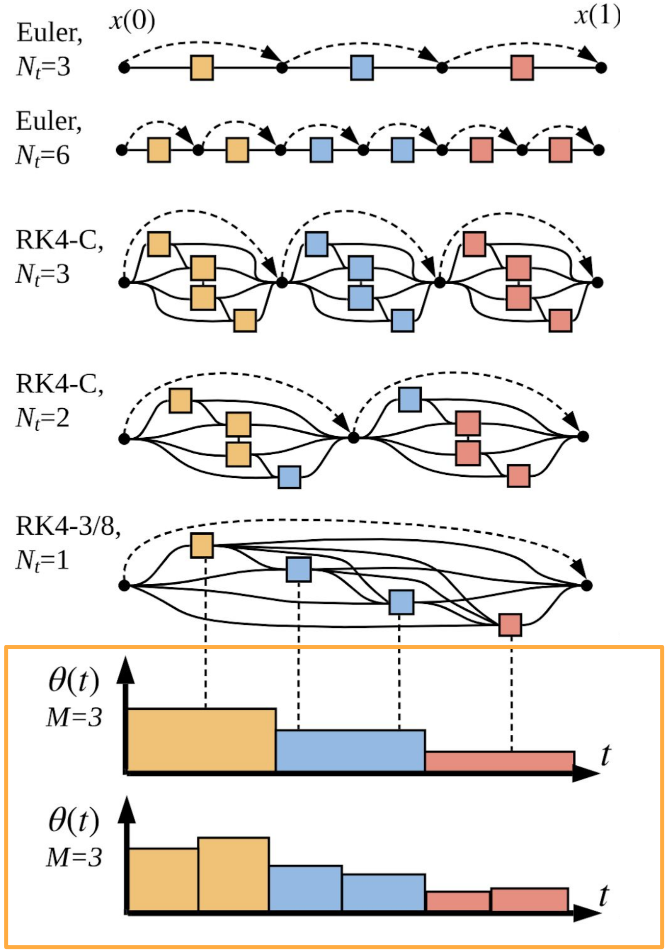
- We use edge weights that are well-known in numerical analysis.



- 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.



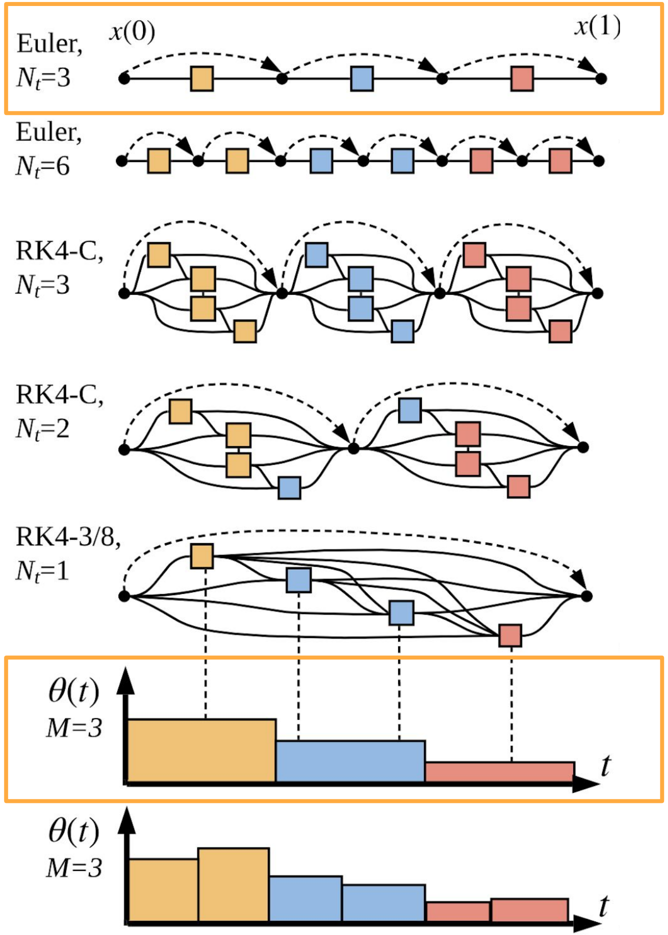
- In ResNet and NNs, parameters are glued to computation nodes in the graph.

- **ContinuousNet**'s computations are assigned weights by evaluating a continuous *θ(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 $[t_k, t_{k+1}) \rightarrow \theta_k$ to define values between steps:

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

Experiments for Image Classification

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

Updated hypothesis:

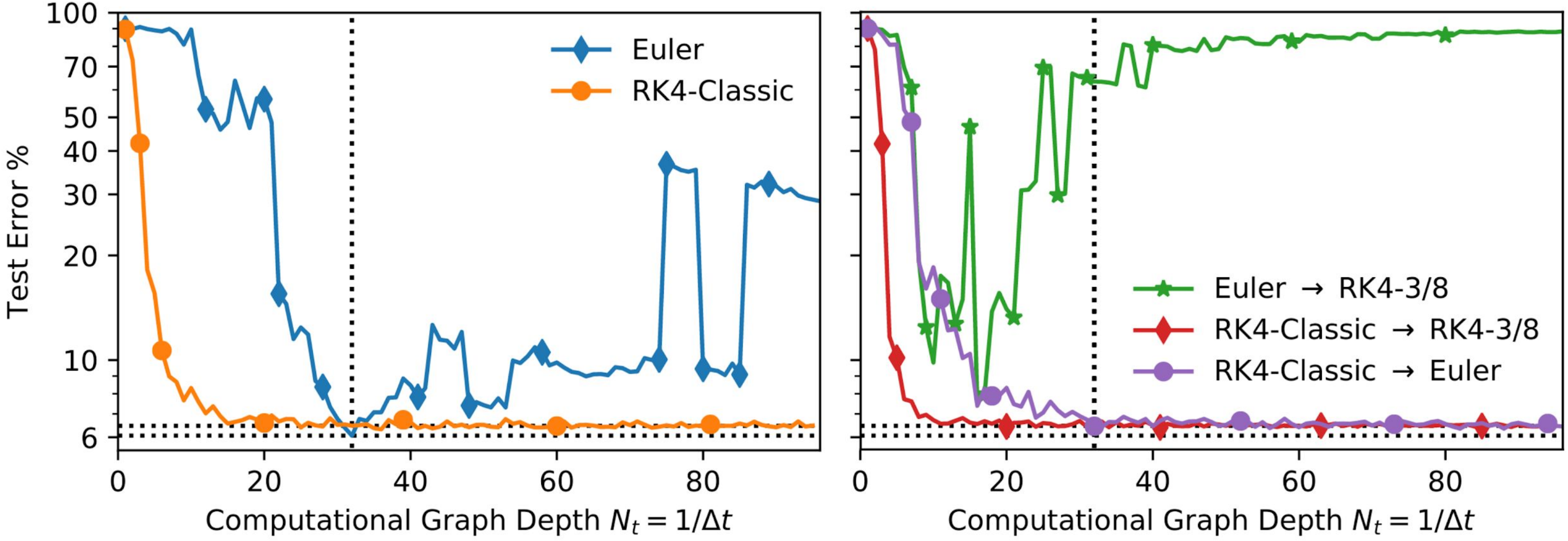
- Expect forward Euler (ResNet) to overfit
- Expect training with Midpoint or RK4 to enable transfer between depths and graph modules

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

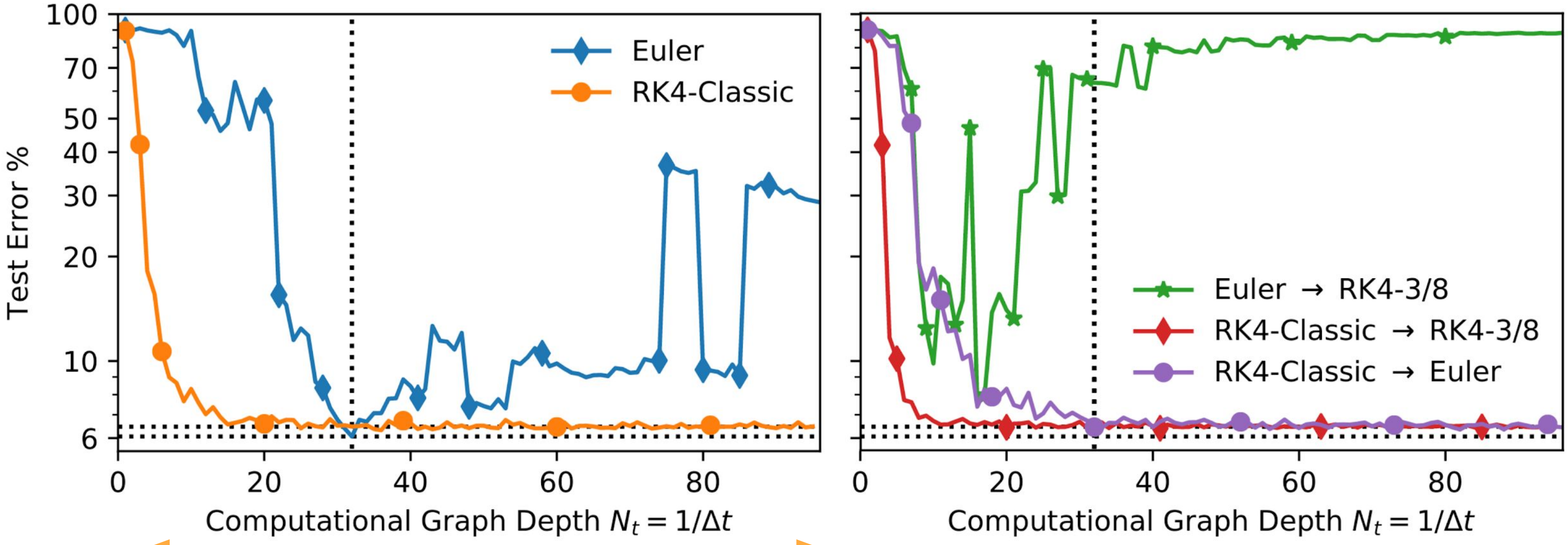


We can perform the same experiment on CIFAR10

- 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



Shallower

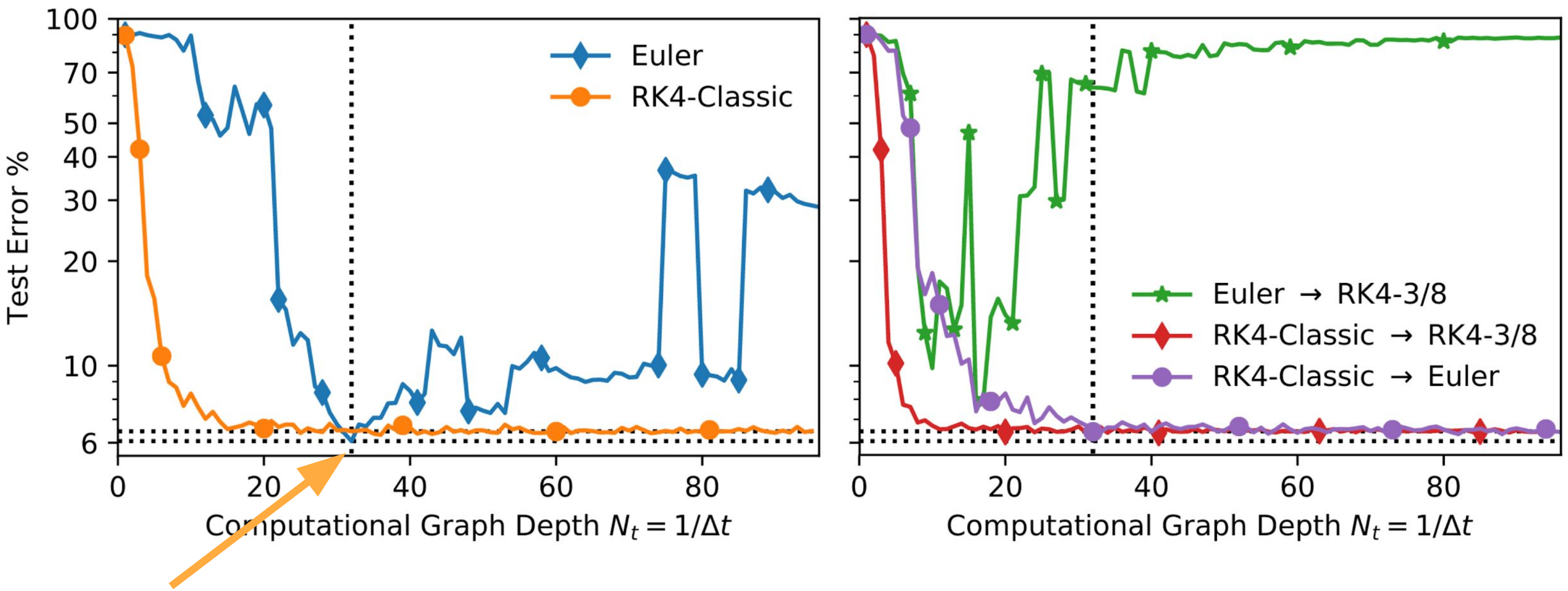
Deeper

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

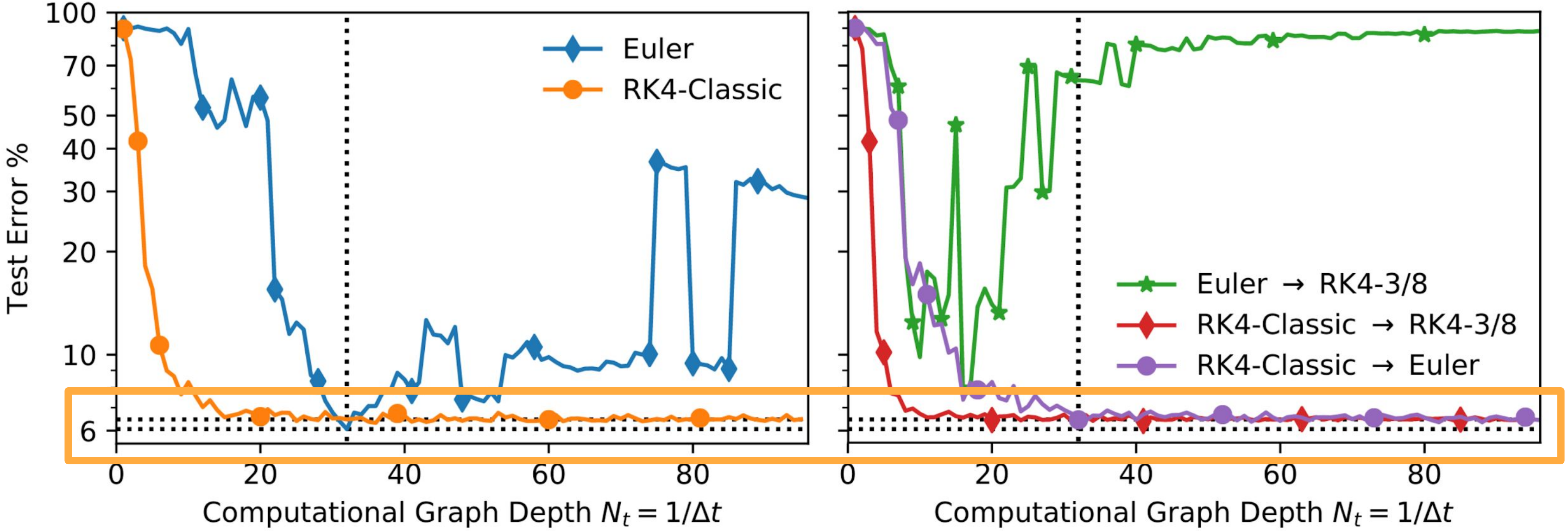


We can perform the same experiment on CIFAR-10

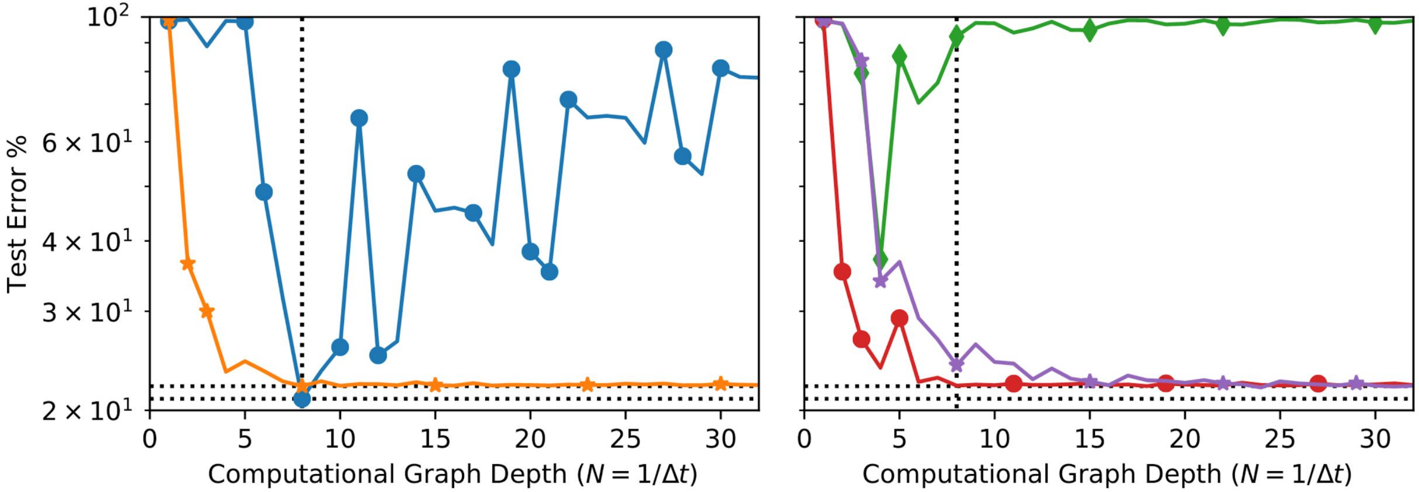
- Test set error in place of analytical solution

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

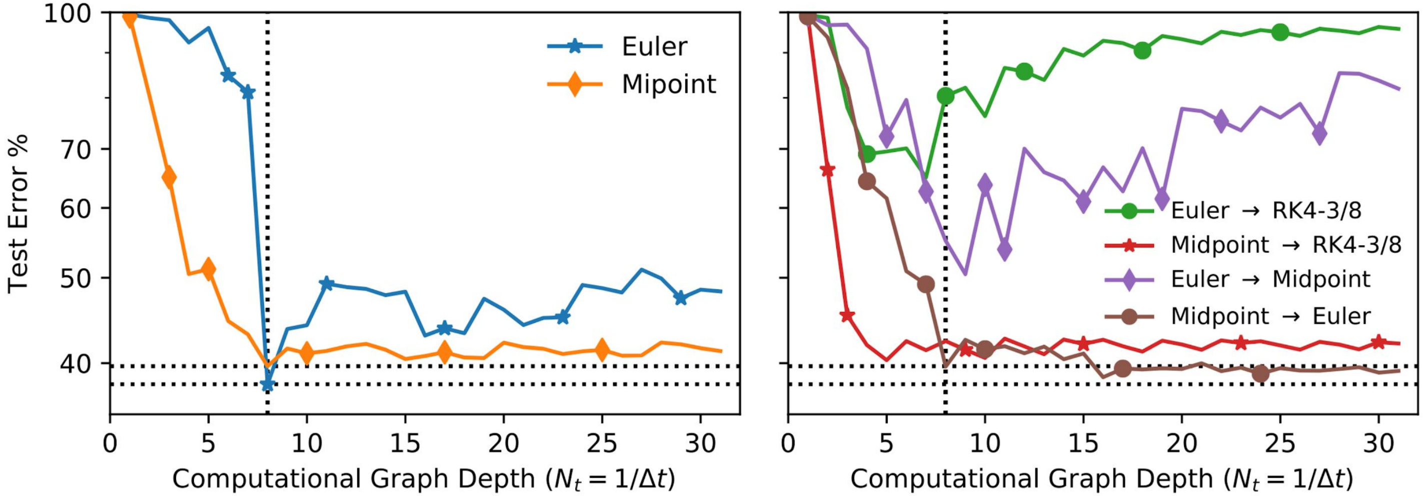
- ContinuousNet(Euler): same dip as pendulum
- ContinuousNet(RK4): re-manifests with many N_t and integrators



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:

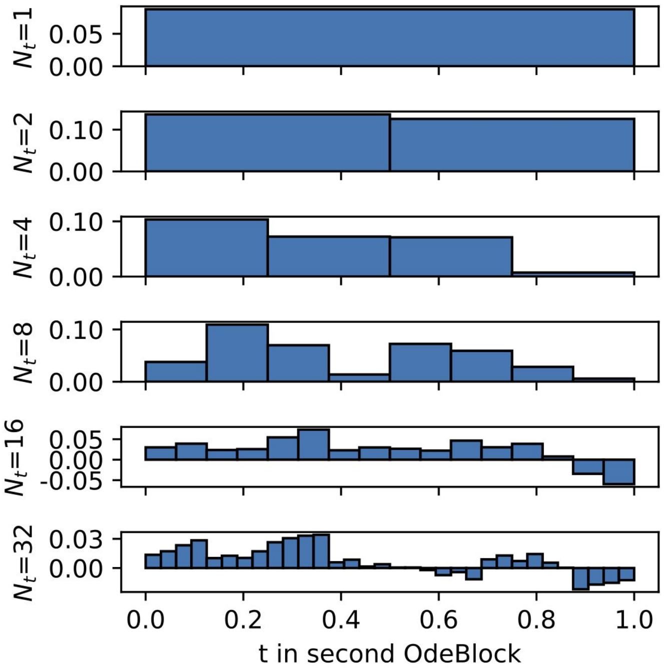
- The weights it learns can plug into a ResNet graph.
- It can even be made *shorter*, without sacrificing much accuracy.
- We can reduce the inference time.

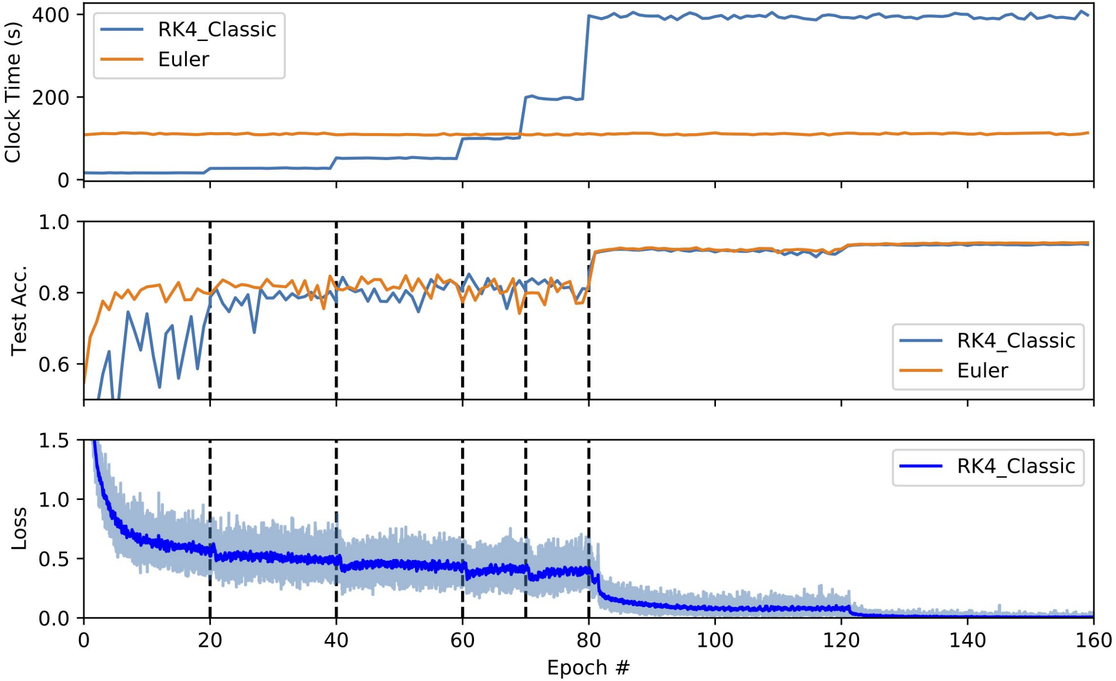
Model for CIFAR10	**Params**	**Units (N_t)**	**Accuracy**	**Inference Time (s)**
ContinuousNet(RK4-classic)	**3.19 M**	**32-32-32**	**93.57%**	**32.55**
↳*Manifest as* **(Euler)**	***Same weights***	**32-32-32**	**93.55%**	**8.93**
↳*Manifest as* **(RK4-3/8)**	***Same weights***	**11-11-11**	**93.44%**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

`OdeBlock2:`$\mathcal{R}$`:conv1:w[0,1,2,0]`(t)

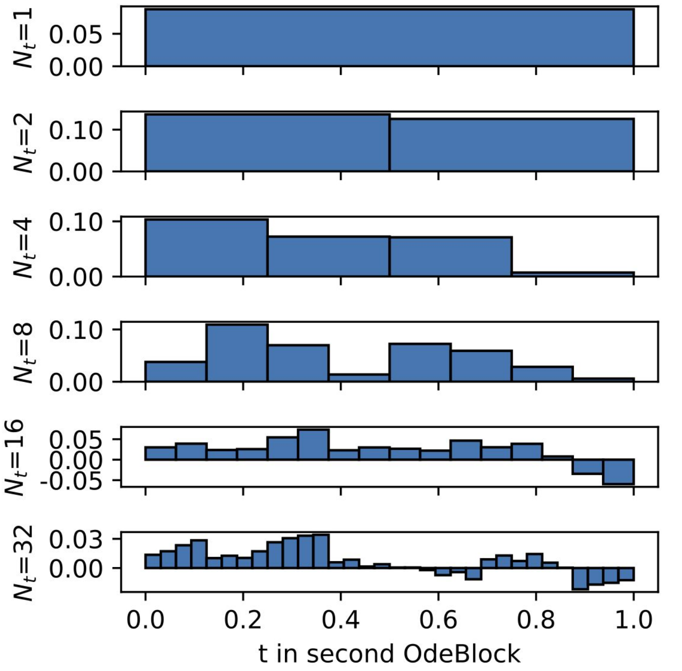


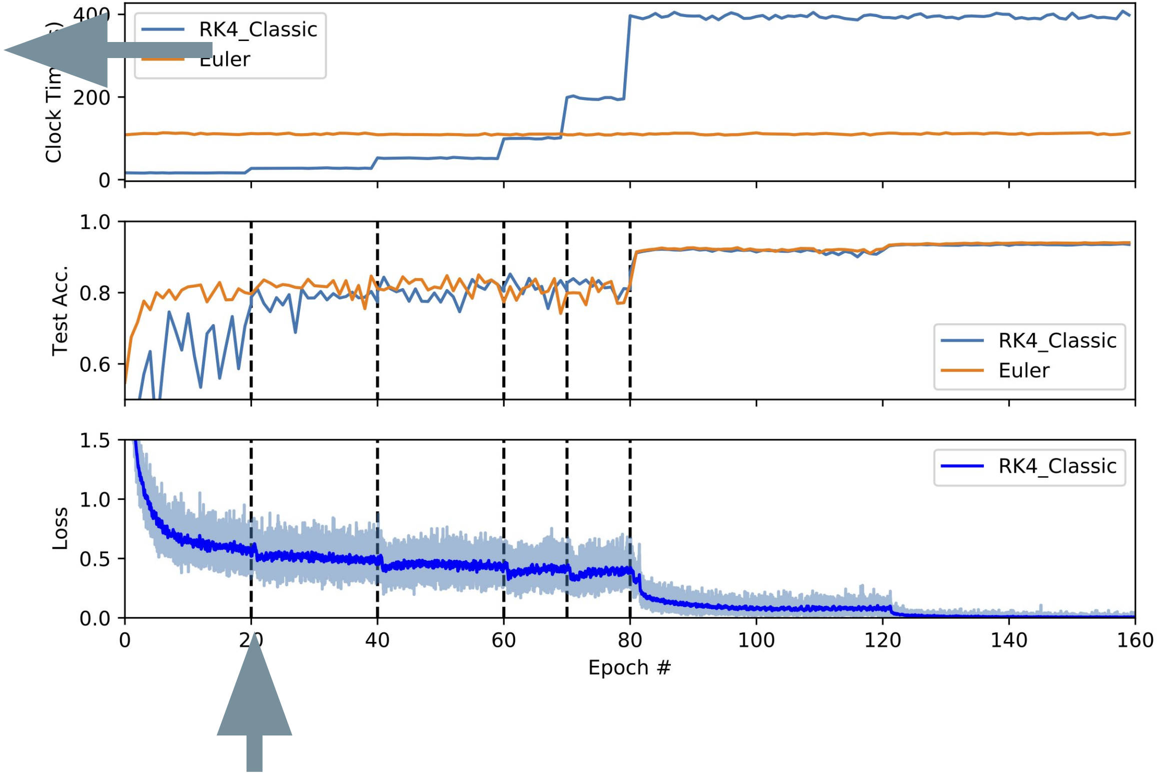


ContinuousNet can also iteratively deepen its graph during training:

➔ Just like mesh refinement

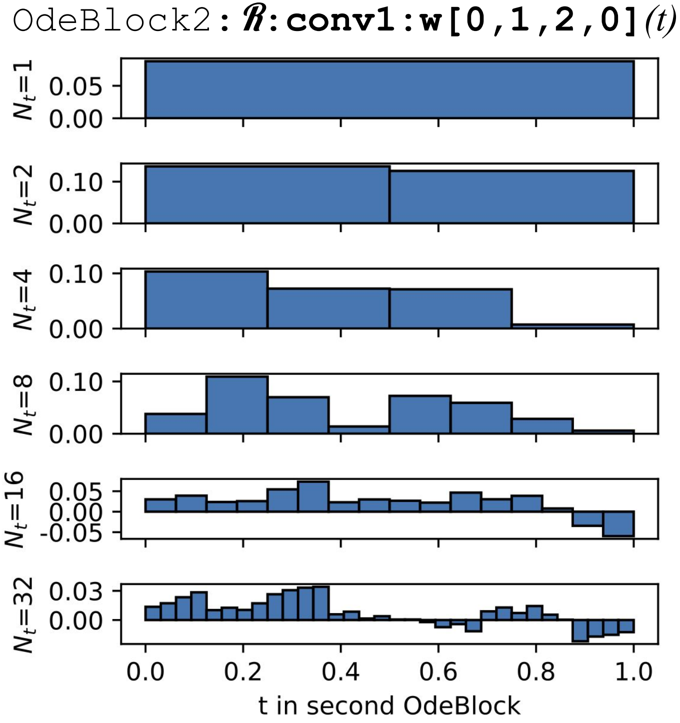
OdeBlock2:$\mathscr{R}$:**conv1:w[0,1,2,0]**(t)

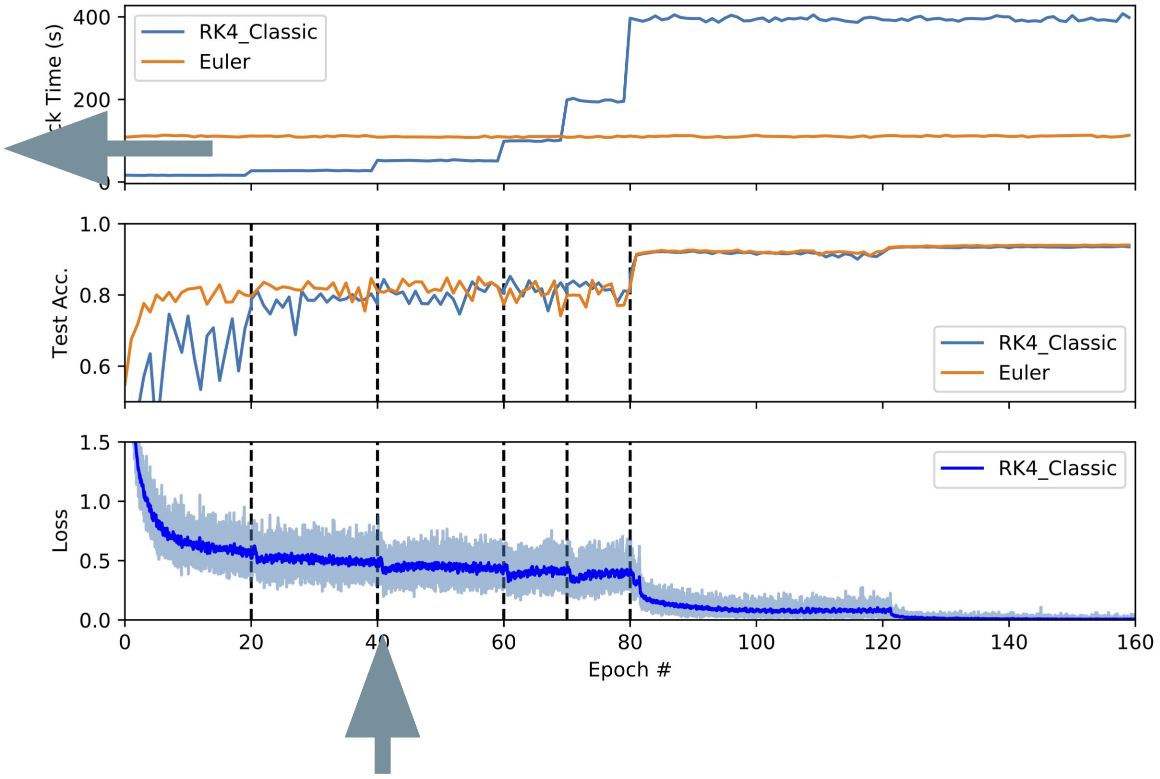




ContinuousNet can also iteratively deepen its graph during training:

➔ Just like mesh refinement

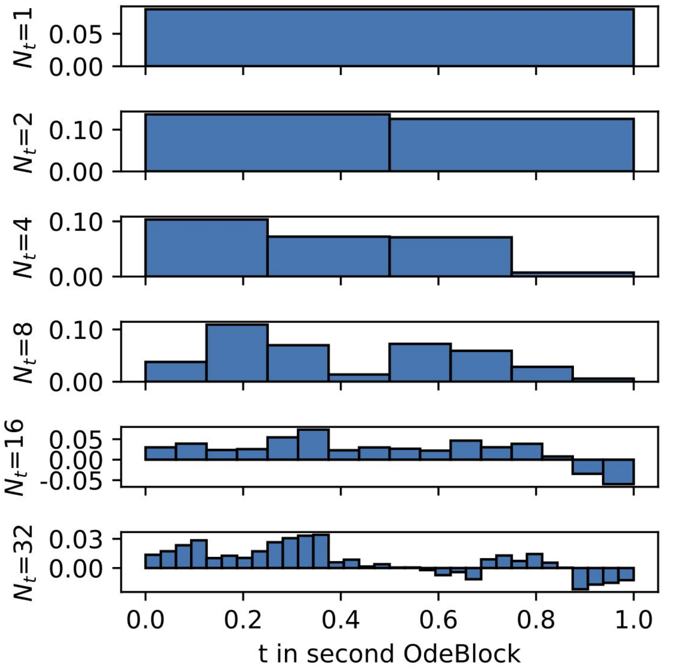


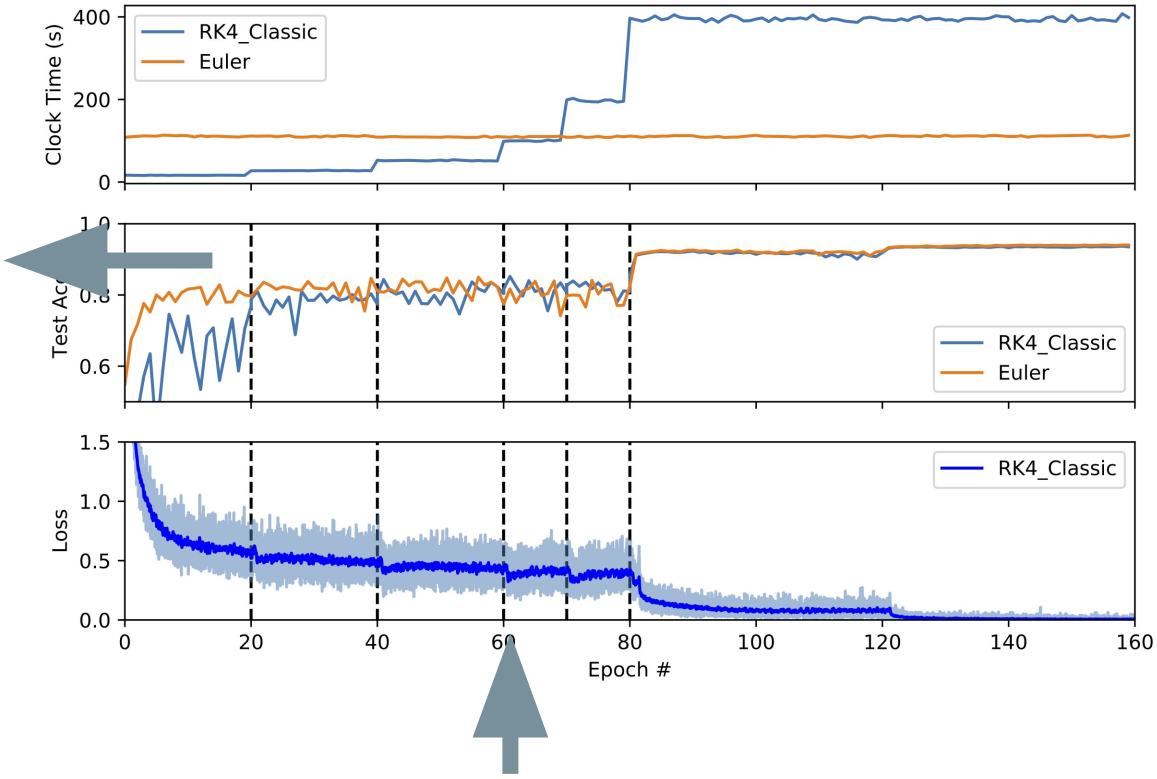


ContinuousNet can also iteratively deepen its graph during training:

➔ Just like mesh refinement

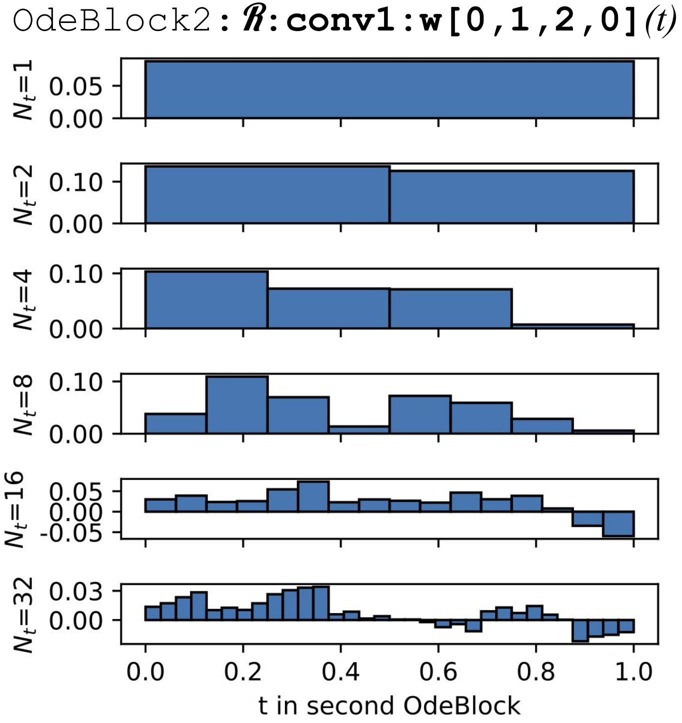
`OdeBlock2:`$\mathscr{R}$`:conv1:w[0,1,2,0]`(t)

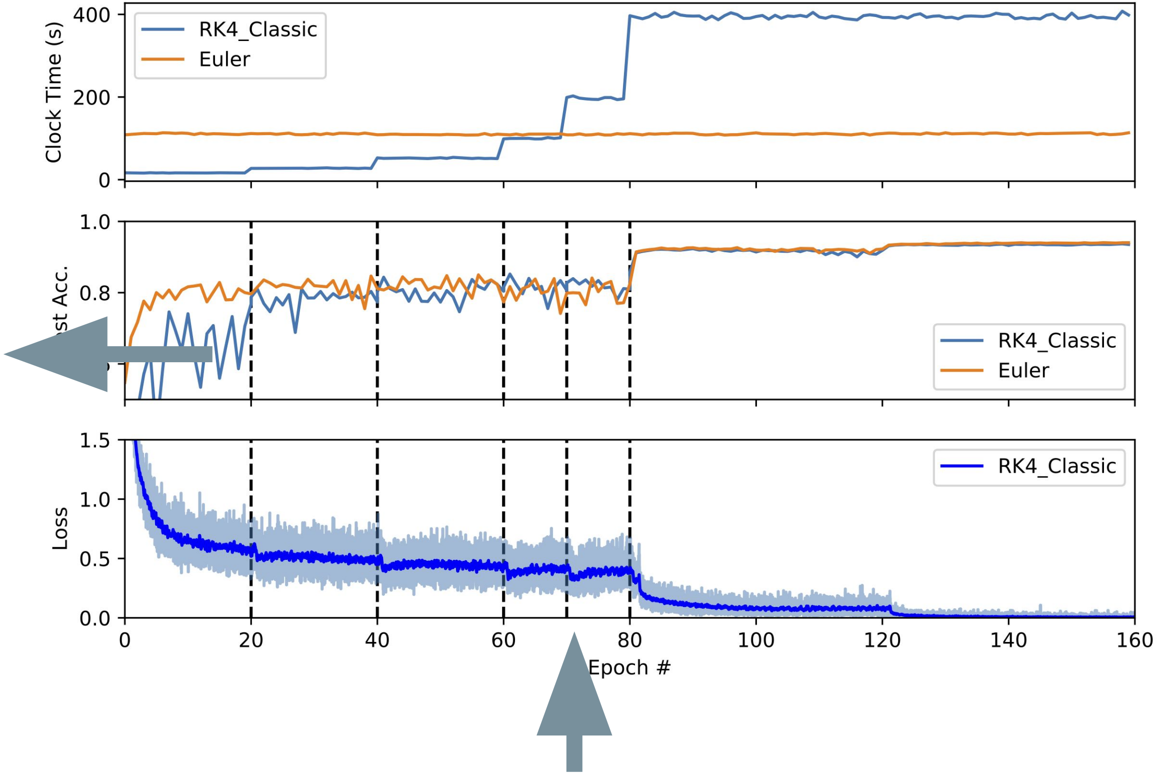




ContinuousNet can also iteratively deepen its graph during training:

➔ Just like mesh refinement

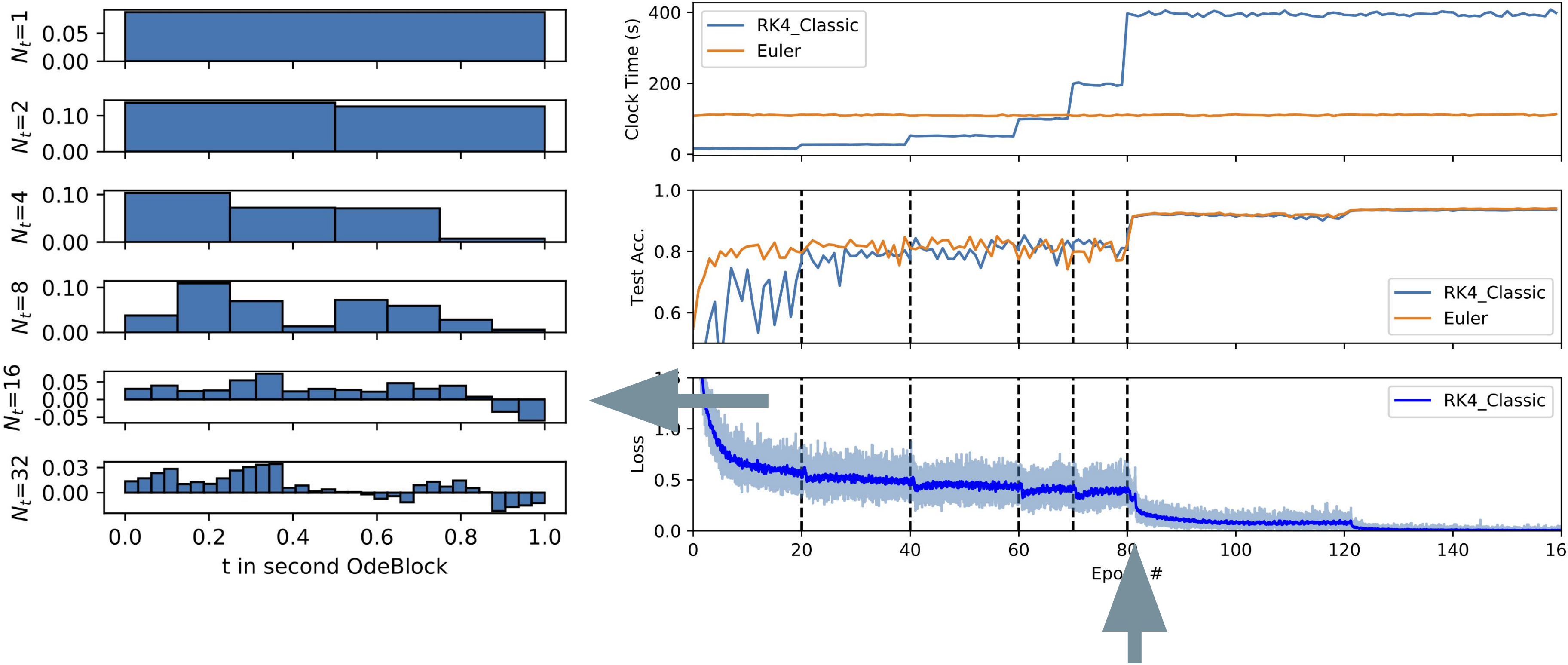




ContinuousNet can also iteratively deepen its graph during training:

➔ Just like mesh refinement

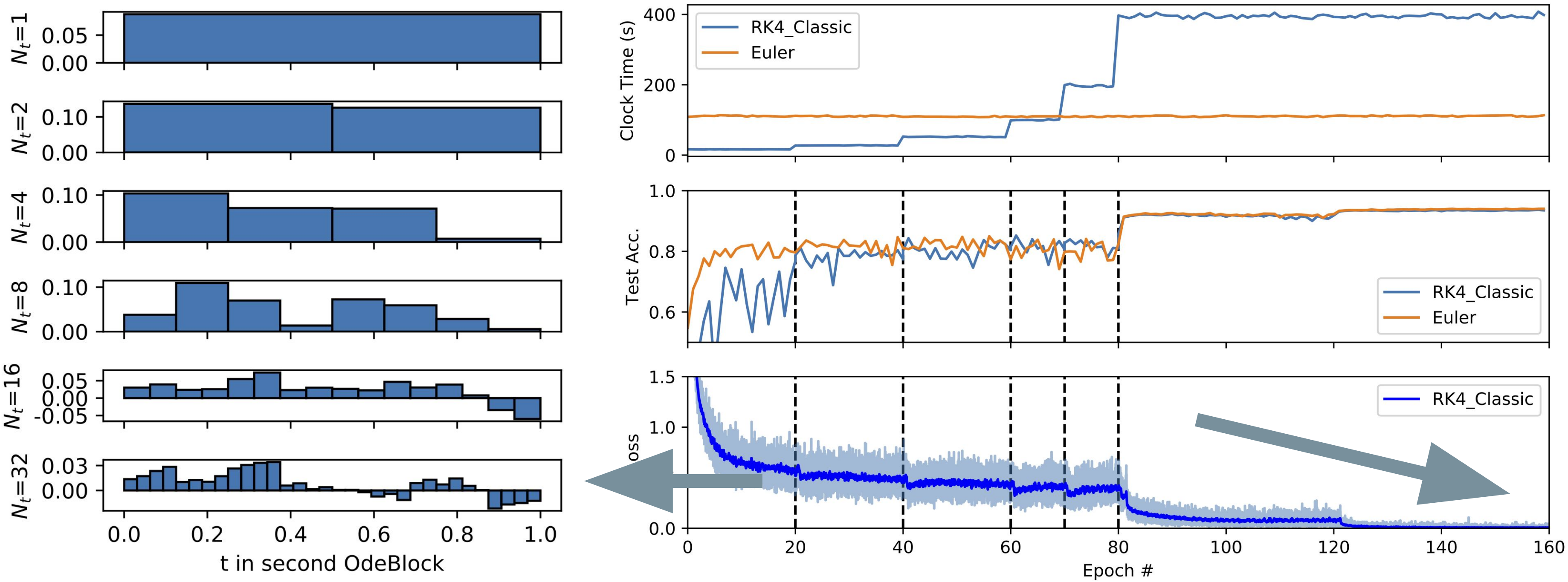
OdeBlock2:$\mathscr{R}$:**conv1:w[0,1,2,0]**(t)



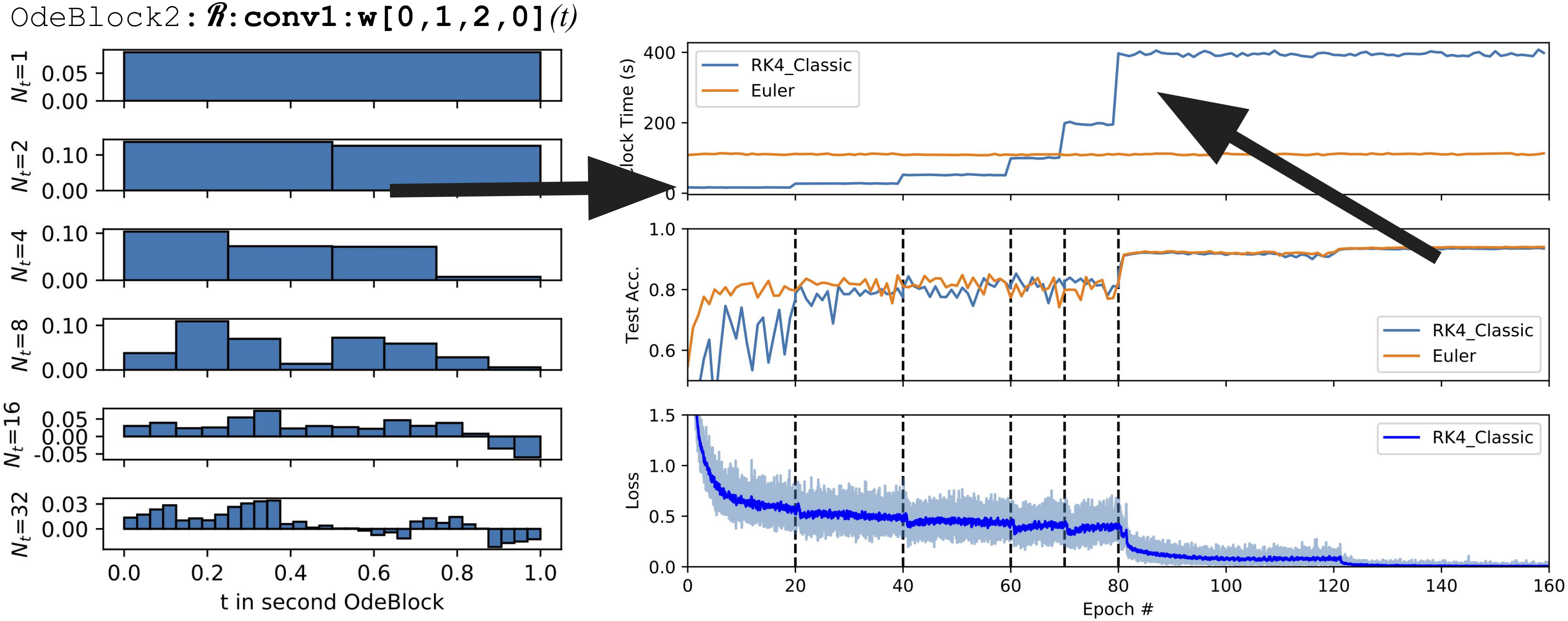
ContinuousNet can also iteratively deepen its graph during training:

➔ Just like mesh refinement

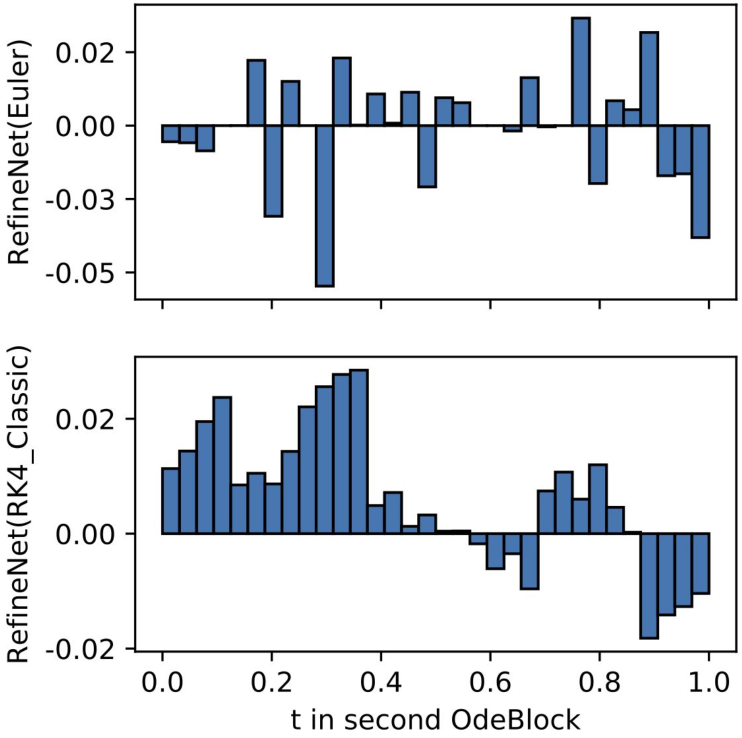
`OdeBlock2:`$\mathcal{R}$`:conv1:w[0,1,2,0]`(t)



Save time by training on shorter graphs initially



OdeBlock2:$\mathcal{R}$:**conv1:w[0,1,2,0]**(t)



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

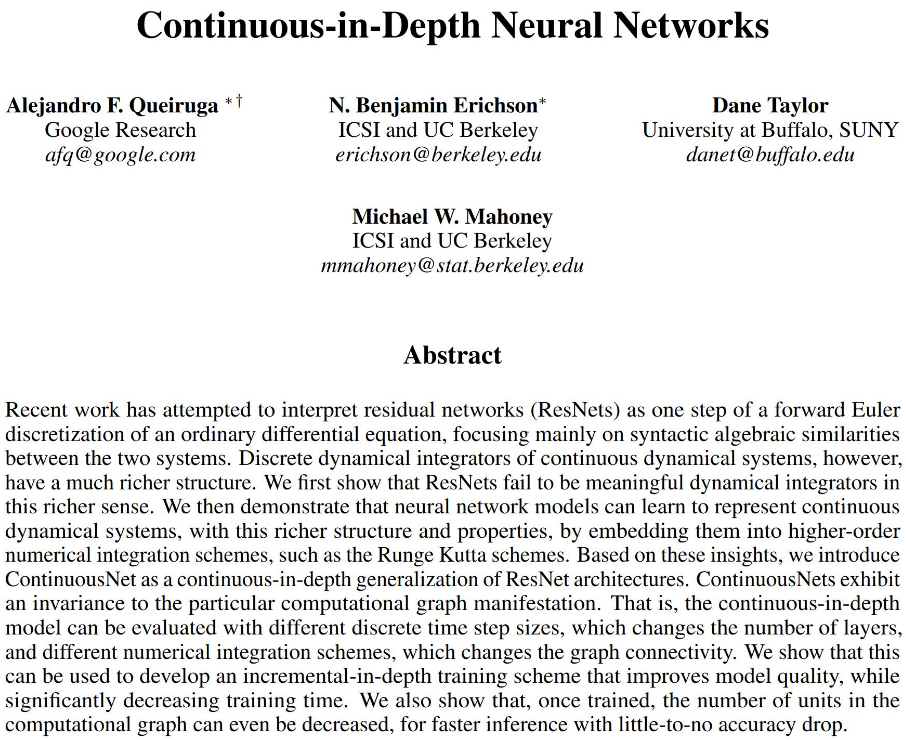
1. Reimplement in JAX

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

In ResNet and NNs, parameters are glued to nodes in the graph.

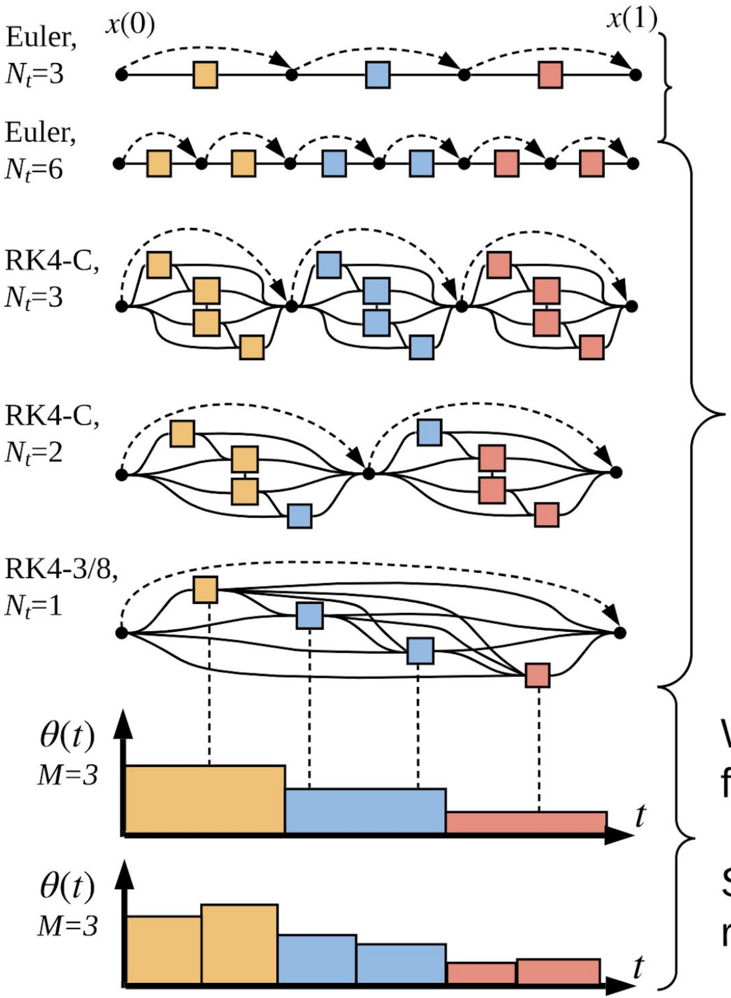
ContinuousNet’s computations are assigned weights by evaluating a *θ(t)*.

To reconstruct ResNet, we map a continuous map $[t_k, t_{k+1}) \rightarrow \theta_k$:

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

To generalize further, use basis functions (piecewise constant here):

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



ResNet *looks like* forward Euler with different parameters at each layer.

Numerical integrators all approximate the same function.

A model that does represent a dynamical system, has a whole family of interchangeable graphs.

Weights are assigned to nodes with shape functions.

Shape functions have well-defined refinements, too.