Toward combining principled scientific models and principled machine learning models

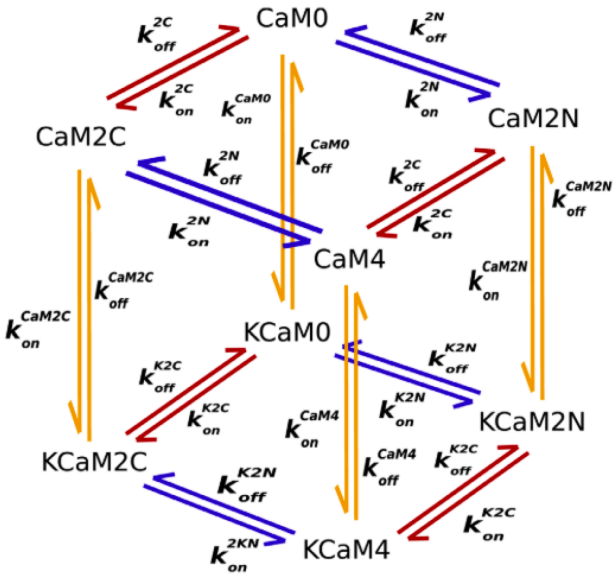
Michael W. Mahoney

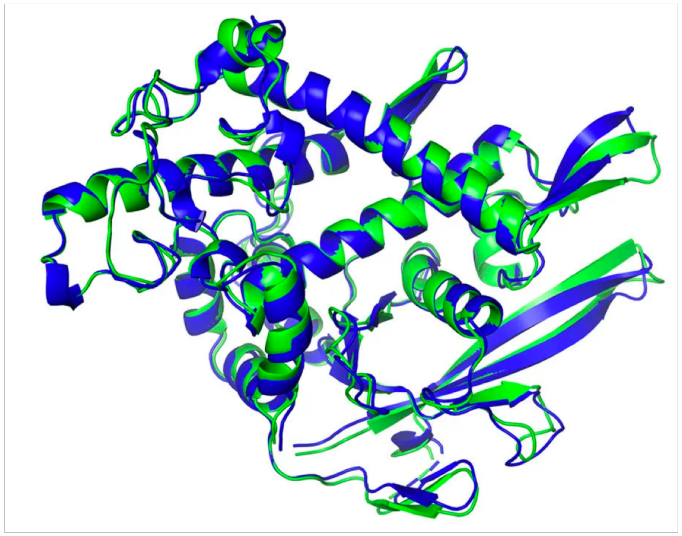
ICSI and Department of Statistics, UC Berkeley

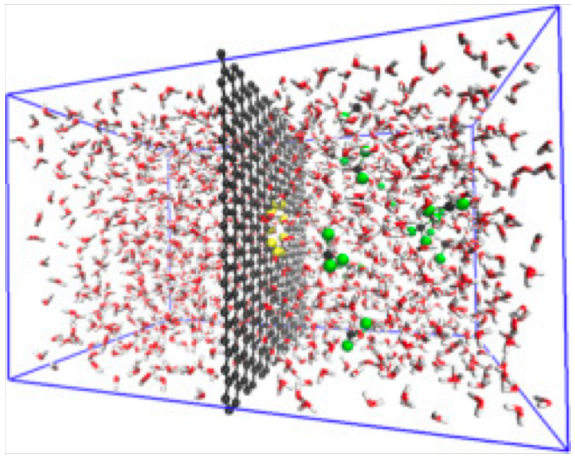
November 2021

With: Aditi Krishnapriyan (LBNL and UC Berkeley) Amir Gholami (UC Berkeley), Shandian Zhe (Univ. of Utah), and Mike Kirby (Univ. of Utah); and Benjamin Erichson (Pittsburgh), Alejandro Queiruga (Google), Dane Taylor (Buffalo); and others.

Modeling spatial and/or temporal behavior is crucial for progress in science







How can we integrate domain insight with data-driven methods to accelerate and improve spatial and temporal modeling?

Increasing data in computational science, and traditional modeling approaches have limitations (computational efficiency, etc.)

+

Neural networks have an ability to approximate high-dimensional functions

Limitations and open questions:

- We still don't usually have as much data as computer vision, natural language processing, etc.
- What does it mean to respect physical/domain information?
- For example, incorporate constraints to respect spatial invariance through topological methods (in addition to other physics-informed approaches)

Topics

- Characterizing possible failure modes in physics-informed neural networks (think: optimization)
- Meaningfully continuous-in-depth neural networks (think: continuity)
- Other related developments (think: temporal/sequential modeling)
- Conclusion

Topics

- Characterizing possible failure modes in physics-informed neural networks (think: optimization)
- Meaningfully continuous-in-depth neural networks (think: continuity)
- Other related developments (think: temporal/sequential modeling)
- Conclusion

Physics-informed neural networks (PINNs): construct a loss function that should "respect" the physics, and model differential equations

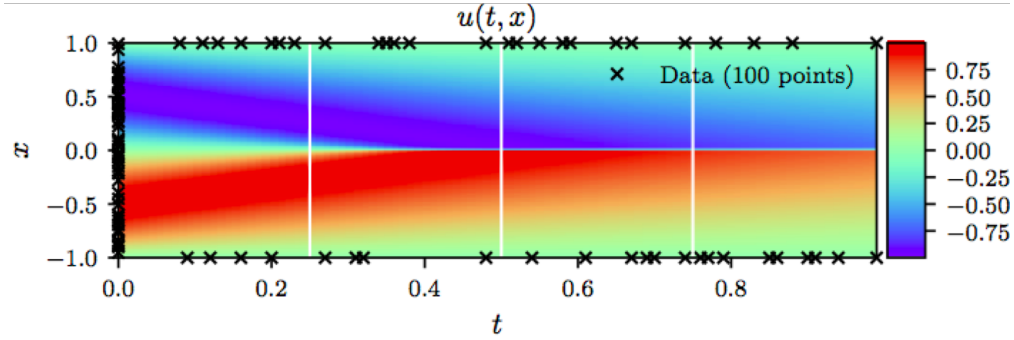
Burger's equation describing fluid flow:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}; x \in [-1,1], t \in [0,1]$$

$$u(0,x) = -sin(\pi x)$$

$$u(t,-1) = u(t,1) = 0$$

$$\nu = 0.01/\pi$$



The solution is modeled on a mesh (but not discretized!), consisting of "space, time" points

Optimization problem:

$$\min_\theta \mathcal{L}(u) = L_{u_0} + L_{u_b}$$

$$\min_\theta \mathcal{L}(u) \quad \text{s.t.} \quad \mathcal{F}(u) = 0$$

$$\mathcal{F}(u) = \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2}$$

$$\min_\theta \mathcal{L}(u) + \lambda_{\mathcal{F}}\mathcal{F}(u)$$

Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comp. Physics. M. Raissi, P. Perdikaris, G. Karniadakis (2019)

Physics-Informed Deep Learning (Part I). Arxiv: 1711.10561 M. Raissi, P. Perdikaris, G. Karniadakis (2017)

Physics-Informed Deep Learning (Part 2). Arxiv:1711.10566 M. Raissi, P. Perdikaris, G. Karniadakis (2017)

Investigating PINNs

- We systematically investigated common scientific phenomena with PINNs:
  - Convection (transport phenomena) $$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0$$
  - Reaction $$\frac{\partial u}{\partial t} - \rho u(1-u) = 0$$
  - Reaction-diffusion $$\frac{\partial u}{\partial t} - \nu \frac{\partial^2 u}{\partial x^2} - \rho u(1-u) = 0$$

Investigating PINNs

A. D. Jagtap, K. Kawaguchi, G. E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, Journal of Computational Physics, 2020.

S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient-flow pathologies in physics-informed neural networks, SIAM Journal on Scientific Computing, 2021.

S. Wang, X. Yu, P. Perdikaris, When and why PINNs fail to train: A neural tangent kernel perspective, Journal of Computational Physics, 2021.

L. McClenny, U. Braga-Neto, Self-adaptive physics-informed neural networks using a soft attention mechanism, arXiv:2009.04544, 2020.

A. D. Jagtap, G. E. Karniadakis, Extended Physics-Informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition Based Deep Learning Framework for Nonlinear Partial Differential Equations, Commun. Comput. Phys., 2020.

… and more.

Investigating PINNs

- We systematically investigated common scientific phenomena with PINNs:
  - Convection (transport phenomena) $$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0$$
  - Reaction $$\frac{\partial u}{\partial t} - \rho u(1-u) = 0$$
  - Reaction-diffusion $$\frac{\partial u}{\partial t} - \nu \frac{\partial^2 u}{\partial x^2} - \rho u(1-u) = 0$$

Learning convection with PINNs

$$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial t} = 0, x \in \Omega, t \in [0, T],$$
$$u(x, 0) = h(x), x \in \Omega$$

We have an analytical solution for this system:

$$u_{\text{analytical}}(x, t) = F^{-1}\left(F(h(x))e^{-i\beta k t}\right)$$

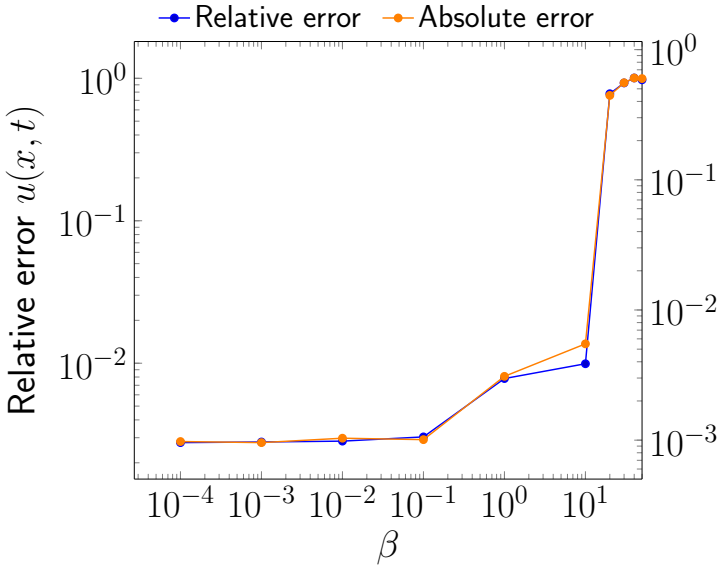
Initial conditions: $u(x, 0) = sin(x),$

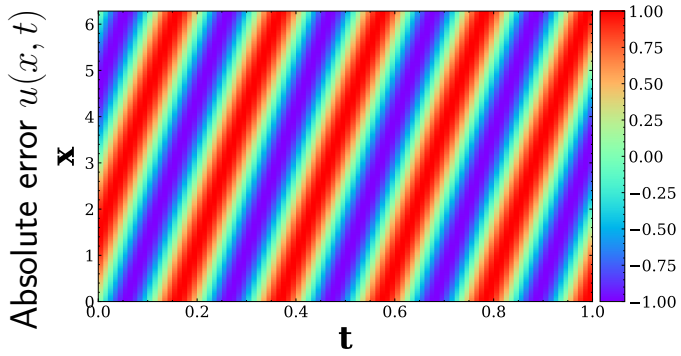
Periodic boundary conditions: $u(0, t) = u(2\pi, t)$

$$\mathcal{L}(\theta) = \frac{1}{N_u}\sum_{i=1}^{N_u}\left((\hat{u}(\theta, x_0^i, 0) - u_0^i)^2\right) + \frac{1}{N_f}\sum_{i=1}^{N_f}\lambda_i\left(\frac{\partial \hat{u}(\theta, x_f^i, t_f^i)}{\partial t} + \beta\frac{\partial \hat{u}(\theta, x_f^i, t_f^i)}{\partial x} - q\right)^2 + \mathcal{L}_{\mathcal{B}}$$

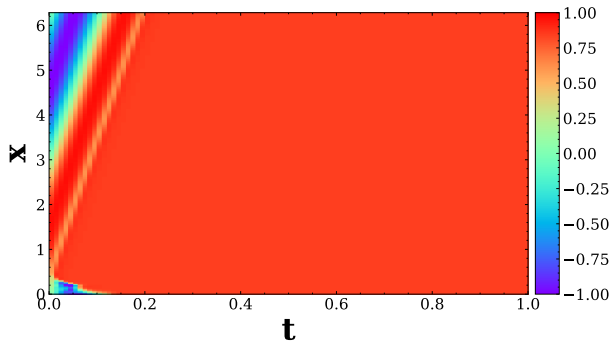
$$\mathcal{L}_{\mathcal{B}} = \frac{1}{N_b}\sum_{i=1}^{N_b}\left(\hat{u}(\theta, 0, t) - \hat{u}(\theta, 0, 2\pi, t)\right)^2$$

At low speeds of propagation, PINNs learns good models but error increases as wave speed increases





Exact solution: β = 30



Predicted PINN solution: β = 30

Convection: $\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0$

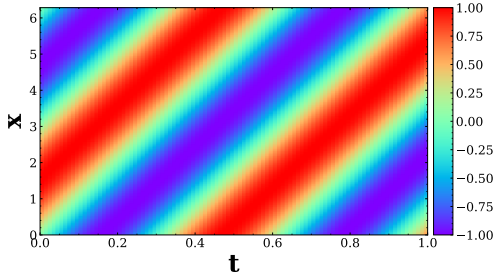
At low speeds of propagation, PINNs learns good models but error increases as wave speed increases

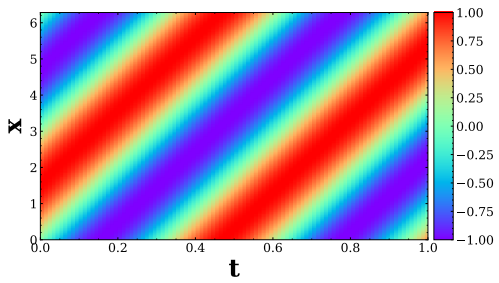
Exact solution

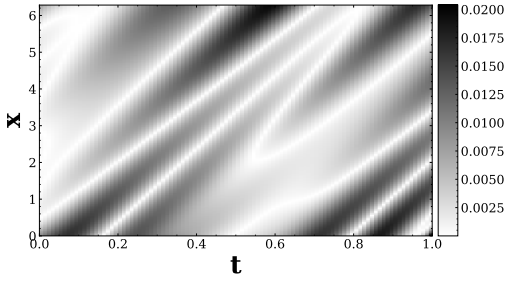
Predicted PINN solution

Difference in exact and predicted solution

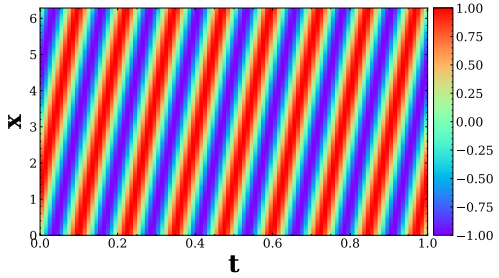
β = 10

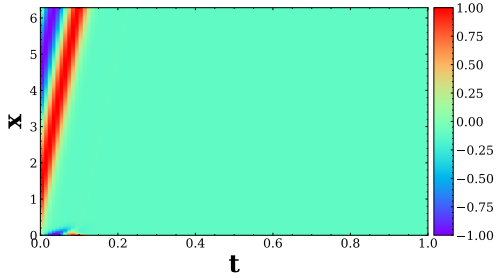


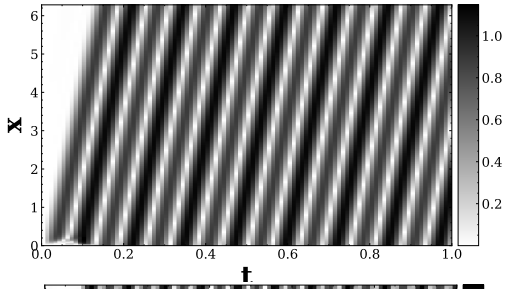




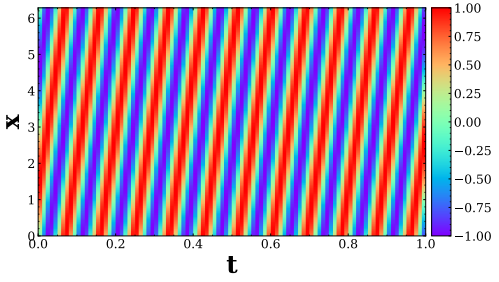
β = 50

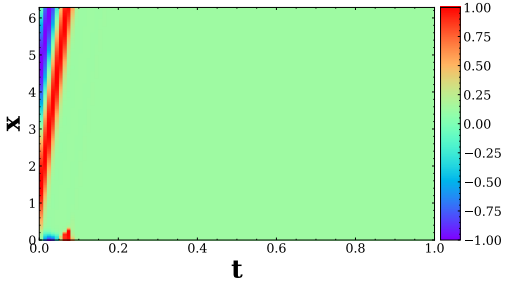


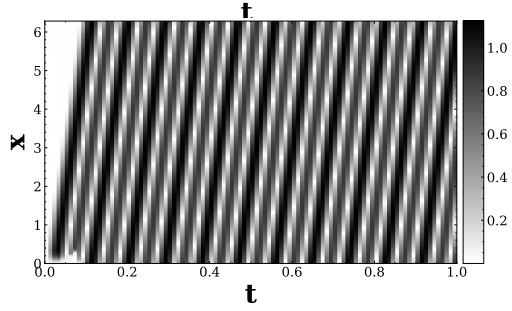




β = 70







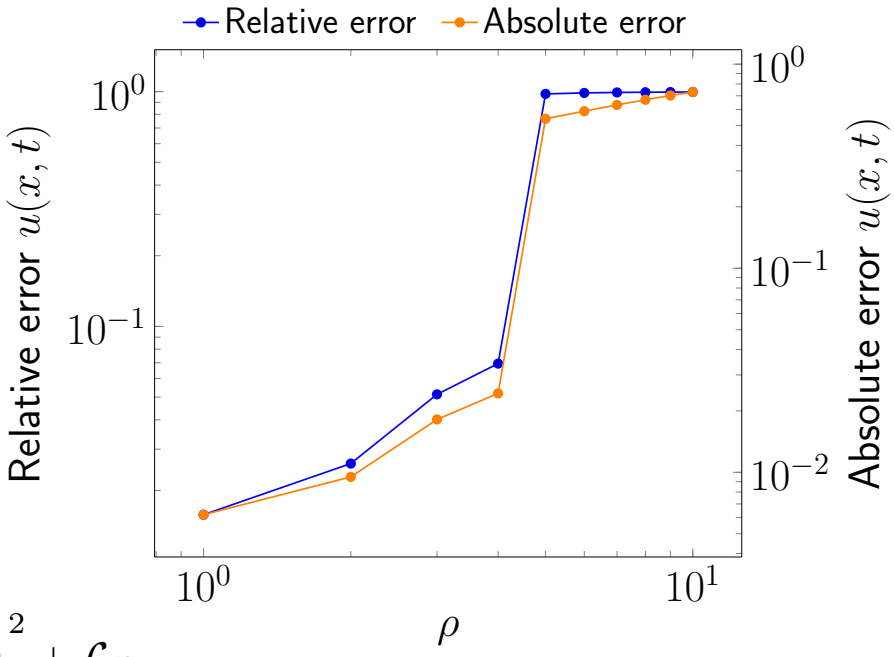
Learning reaction with PINNs

$$\frac{\partial u}{\partial t} - \rho u(1-u) = 0, x \in \Omega, t \in [0, T],$$

$$u(x, 0) = h(x), x \in \Omega$$

Initial conditions: $u(x,0) = e^{-\frac{(x-\pi)^2}{2(\pi/4)^2}},$

Periodic boundary conditions: $u(0,t) = u(2\pi, t)$



$$\mathcal{L}(\theta) = \frac{1}{N_u}\sum_{i=1}^{N_u}\left(\hat{u} - u_0^i\right)^2 + \frac{1}{N_f}\sum_{i=1}^{N_f}\lambda_i\left(\frac{\partial \hat{u}}{\partial t} - \rho\hat{u}(1-\hat{u})\right)^2 + \mathcal{L}_{\mathcal{B}}$$

$$\mathcal{L}_{\mathcal{B}} = \frac{1}{N_b}\sum_{i=1}^{N_b}\left(\hat{u}(\theta, 0, t) - \hat{u}(\theta, 0, 2\pi, t)\right)^2$$

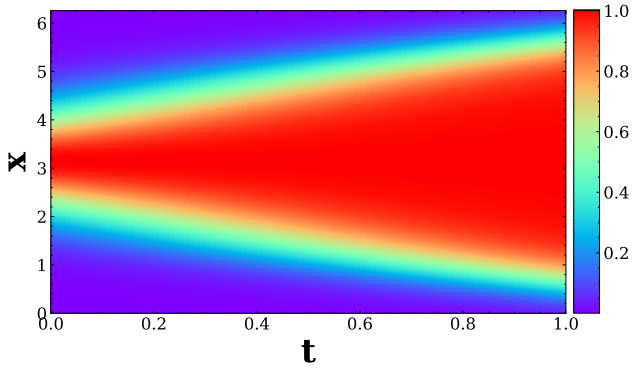
Training a PINN is sensitive to the value of the reaction coefficient

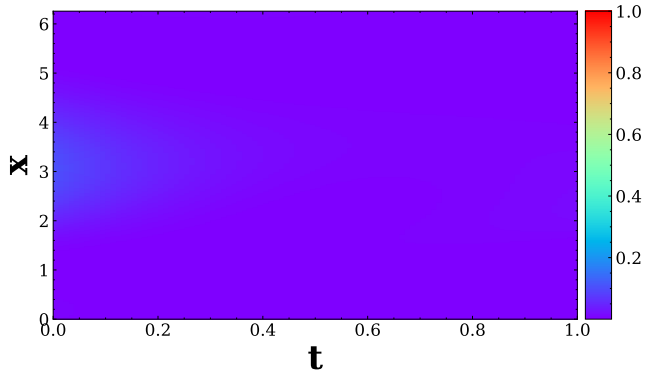
$$\frac{\partial u}{\partial t} - \rho u(1-u) = 0$$

Exact solution

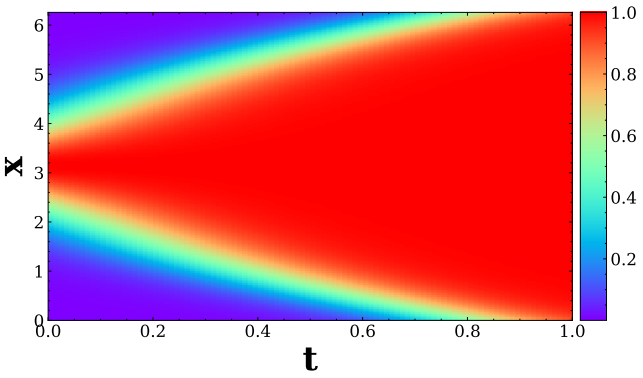
Predicted PINN solution

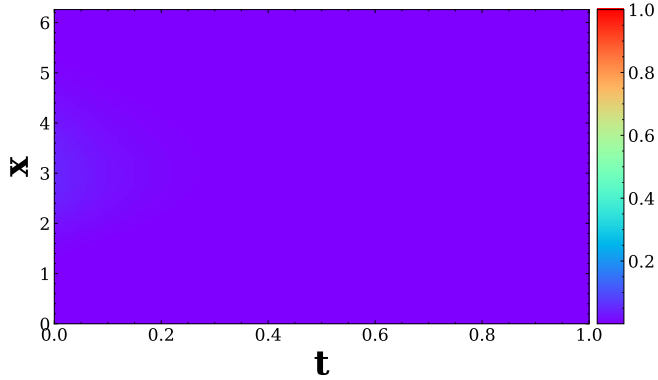
ρ = 5





ρ = 10





Also looked at modeling a reaction-diffusion system with PINNs

$$\frac{\partial u}{\partial t} - \nu \frac{\partial^2 u}{\partial x^2} - \rho u(1-u) = 0, \quad x \in \Omega,\ t \in (0, T],$$
$$u(x, 0) = h(x), \quad x \in \Omega$$

Initial conditions: $u(x,0) = e^{-\frac{(x-\pi)^2}{2(\pi/4)^2}}$,

Periodic boundary conditions: $u(0,t) = u(2\pi, t)$

$$\mathcal{L}(\theta) = \frac{1}{N_u} \sum_{i=1}^{N_u} \left(\hat{u} - u_0^i\right)^2 +$$
$$\frac{1}{N_f} \sum_{i=1}^{N_f} \lambda_i \left(\frac{\partial \hat{u}}{\partial t} - \nu \frac{\partial^2 \hat{u}}{\partial x^2} - \rho \hat{u}(1-\hat{u})\right)^2 + \mathcal{L}_{\mathcal{B}}$$

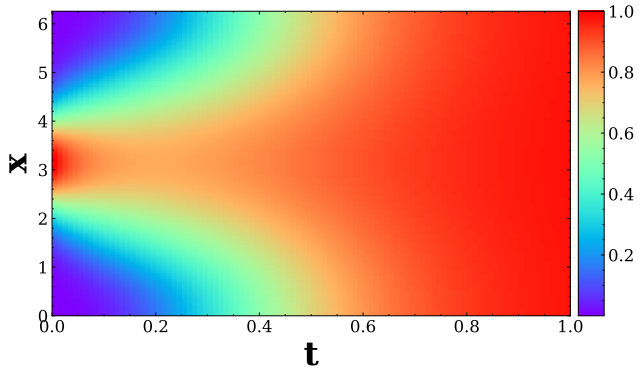
$$\mathcal{L}_{\mathcal{B}} = \frac{1}{N_b} \sum_{i=1}^{N_b} \left(\hat{u}(\theta, 0, t) - \hat{u}(\theta, 0, 2\pi, t)\right)^2$$

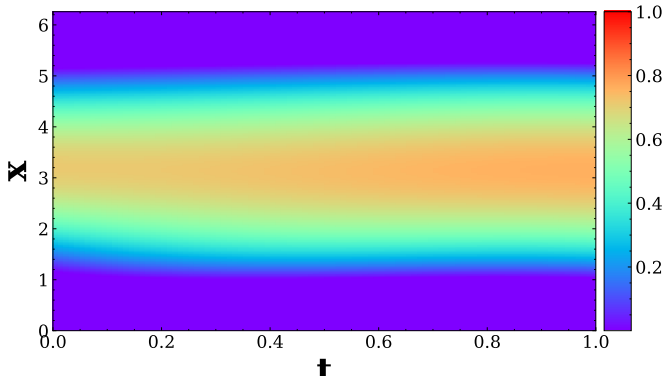
Reaction-diffusion examples with PINNs

Exact solution

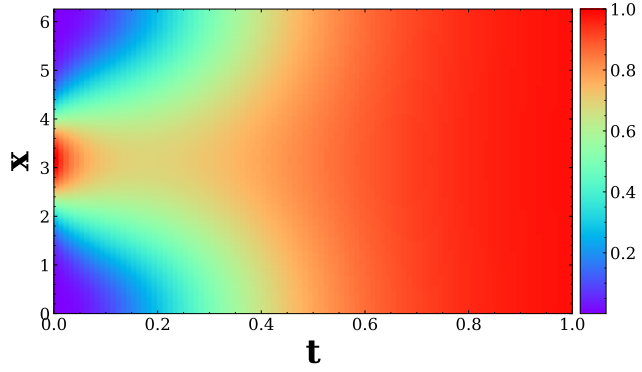
Predicted PINN solution

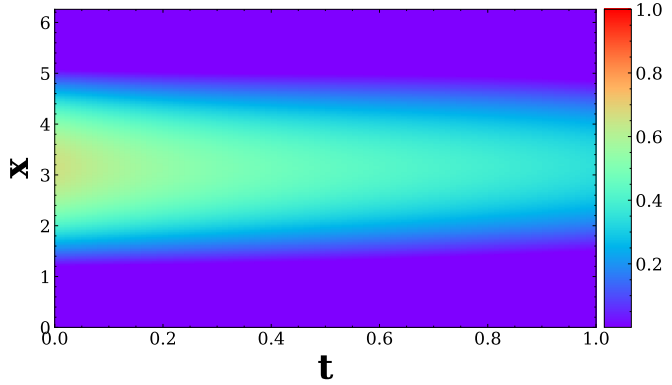
ρ = 5, ν=3





ρ = 5, ν=5



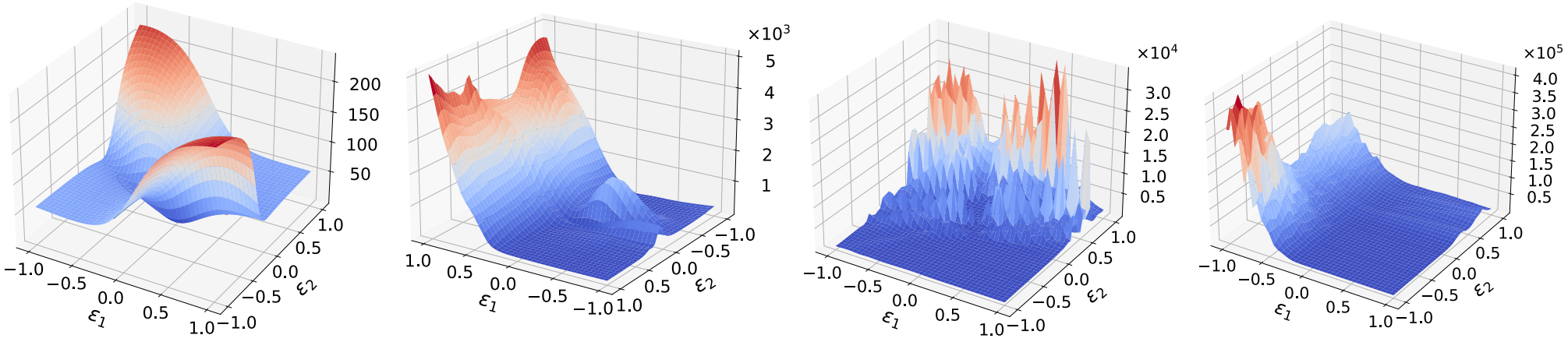


Challenges with PINNs

- We study common differential equations, including ones with convection, reaction, and diffusion operators
- PINNs can be sensitive to the ODE/PDE coefficients (particularly if we have “sharp” features in the solution)
- We characterize some of the challenges, and then change the learning paradigm (as compared to the standard ML training)

To better characterize why PINNs fails, we can look at the loss landscapes in “easy” to learn regimes and “hard” to learn regimes

$$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0$$



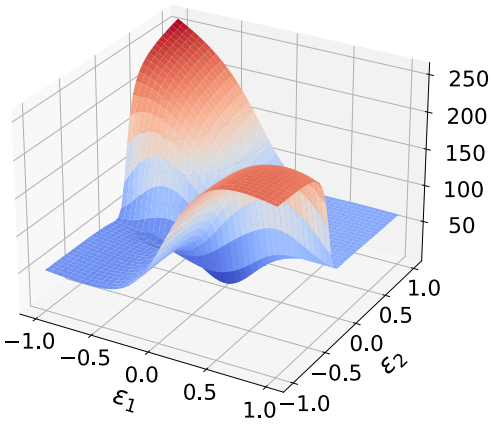
β = 1 β = 20 β = 30 β = 40

β	1	20	30	40
Relative error	8 x 10^{-3}	7.5 x 10^{-1}	9 x 10^{-1}	9.6 x 10^{-1}

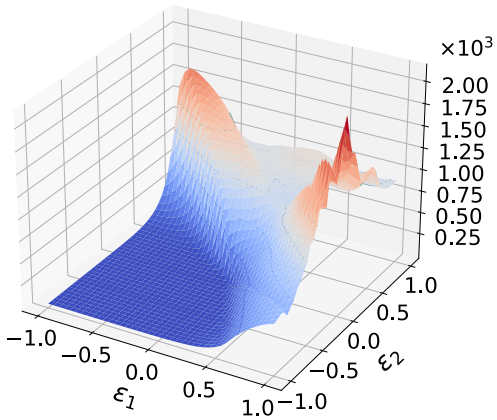
The PINN loss landscape changes as the “amount” of PDE constraint increases

Trade-off between low regularization, smooth landscapes, and error

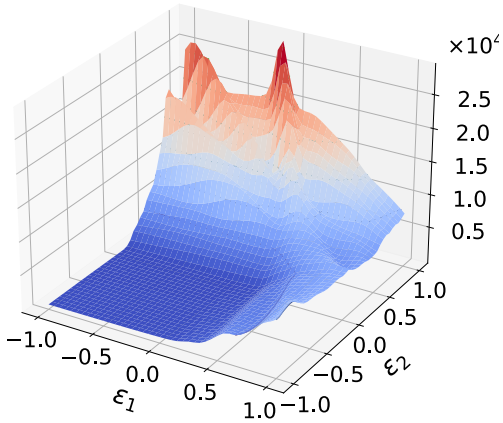
$$\min_{\theta} \mathcal{L}(u) + \lambda_{\mathcal{F}} \mathcal{F}(u)$$



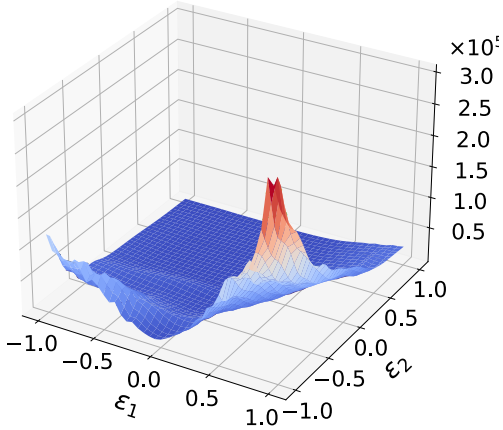
$\lambda = 1\text{x}10^{-6}$



$\lambda = 1\text{x}10^{-3}$



$\lambda = 1\text{x}10^{-1}$



$\lambda = 1\text{x}10^{1}$

λ	$1\text{x}10^{-6}$	$1\text{x}10^{-3}$	$1\text{x}10^{-1}$	$1\text{x}10^{1}$
Relative error	1.69	1.00	1.08	0.982

Building on sight gained for the challenges in training PINNs, we can address these “failure” modes by changing the learning paradigm

- Curriculum regularization:
  - start training a NN with a simple ODE/PDE, and then slowly make the problem harder

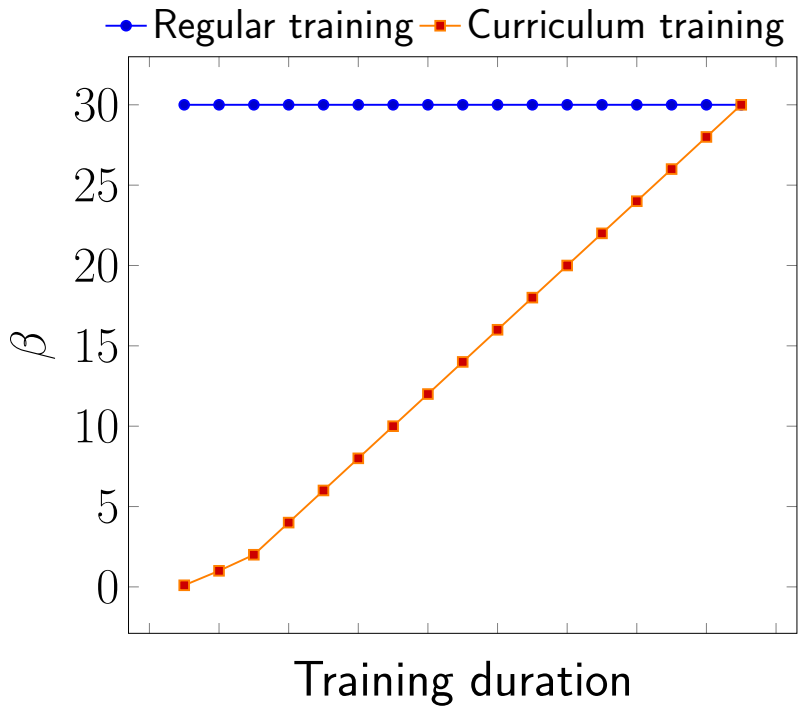
- Sequence-to-sequence learning:
  - Instead of predicting the entire state space, predict for one temporal segment (”time block”) at a time (“time marching”)

Curriculum regularization starts by training the neural network with a simple ODE/PDE, and then making the problem harder and harder

$$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0, x \in \Omega, t \in [0, T],$$

$$u(x, 0) = h(x), x \in \Omega$$

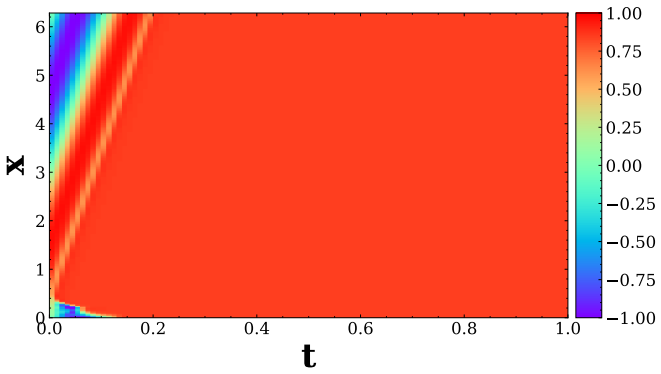
PINNs performed poorly at higher β coefficients



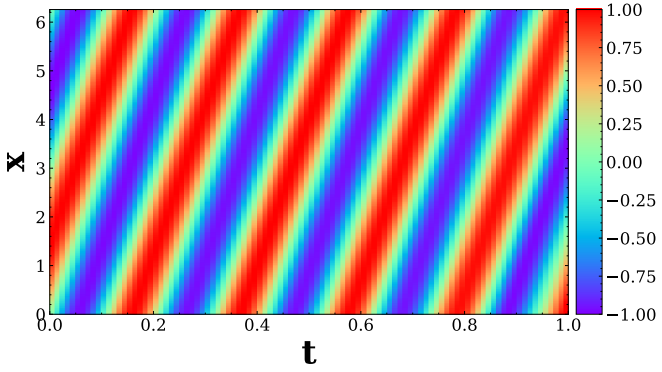
Convection: Curriculum regularization decreases error by 1-2 orders of magnitude

Convection:

$$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0$$



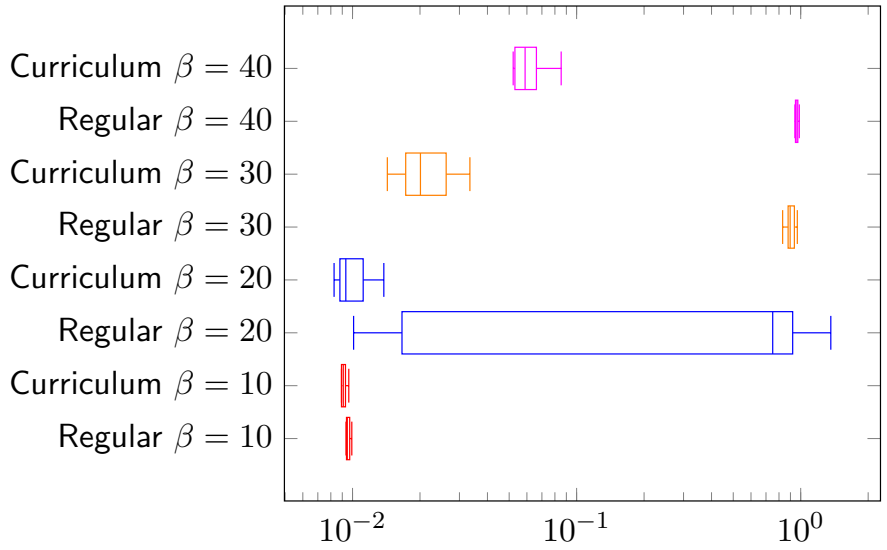
Regular PINN solution, β = 30



Curriculum regularization solution, β = 30

Convection: Curriculum regularization decreases error by 1-2 orders of magnitude

		Regular PINN	Curriculum training
1D convection: $\beta = 20$	Relative error	7.50×10^{-1}	$\mathbf{9.84 \times 10^{-3}}$
	Absolute error	4.32×10^{-1}	$\mathbf{5.42 \times 10^{-3}}$
1D convection: $\beta = 30$	Relative error	8.97×10^{-1}	$\mathbf{2.02 \times 10^{-2}}$
	Absolute error	5.42×10^{-1}	$\mathbf{1.10 \times 10^{-2}}$
1D convection: $\beta = 40$	Relative error	9.61×10^{-1}	$\mathbf{5.33 \times 10^{-2}}$
	Absolute error	5.82×10^{-1}	$\mathbf{2.69 \times 10^{-2}}$



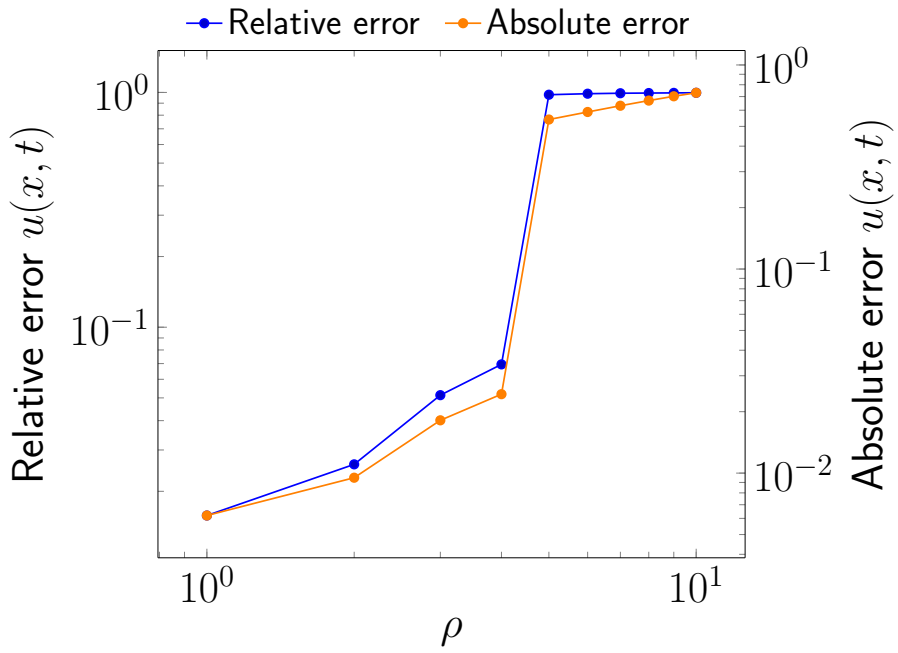
PINN error over 10 different initializations (seeds)

Curriculum regularization for the reaction case

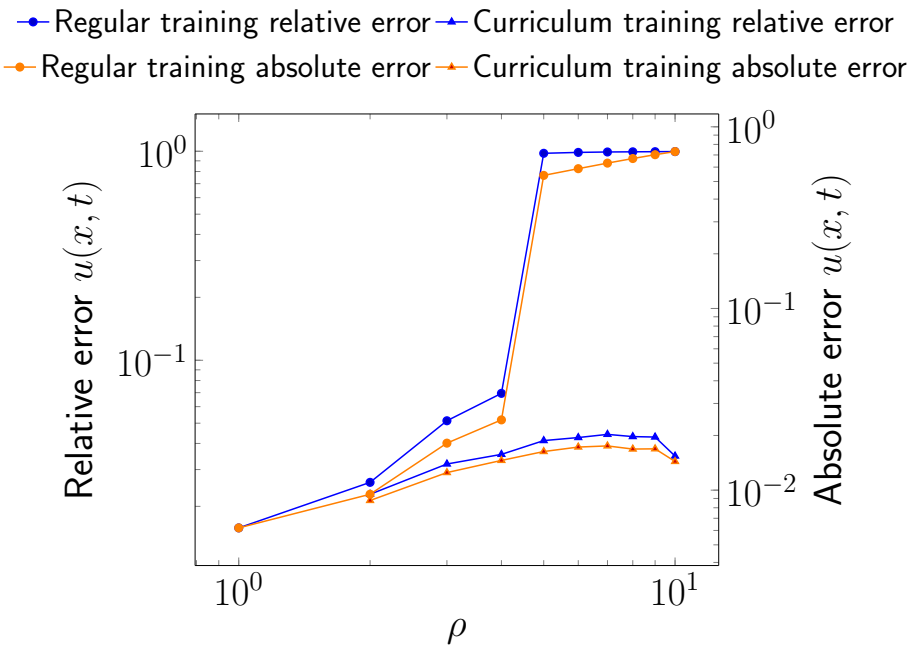
$$\frac{\partial u}{\partial t} - \rho u(1-u) = 0, x \in \Omega, t \in [0, T],$$

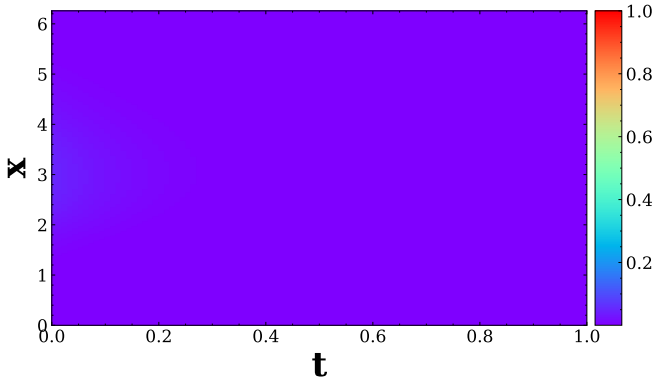
$$u(x, 0) = h(x), x \in \Omega$$

PINNs performed poorly at higher ρ coefficients

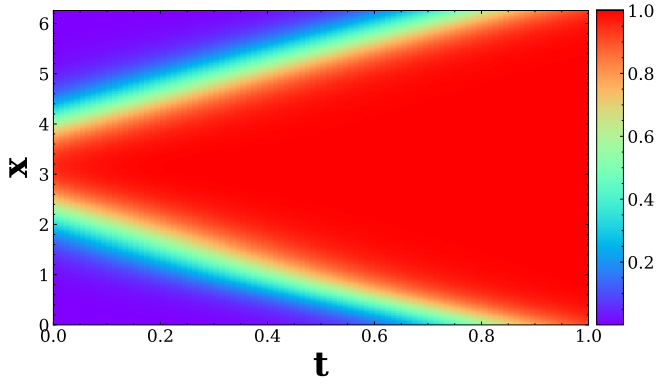


Reaction: Curriculum regularization decreases error by 1-2 orders of magnitude





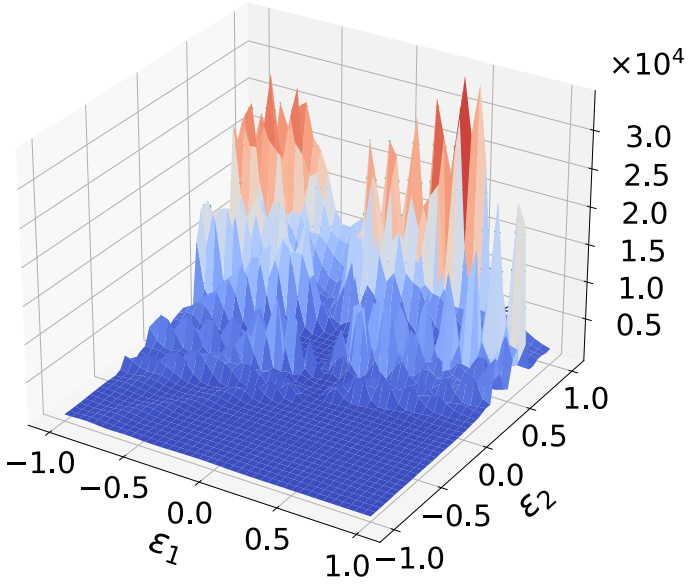
Regular PINN training, ρ = 10

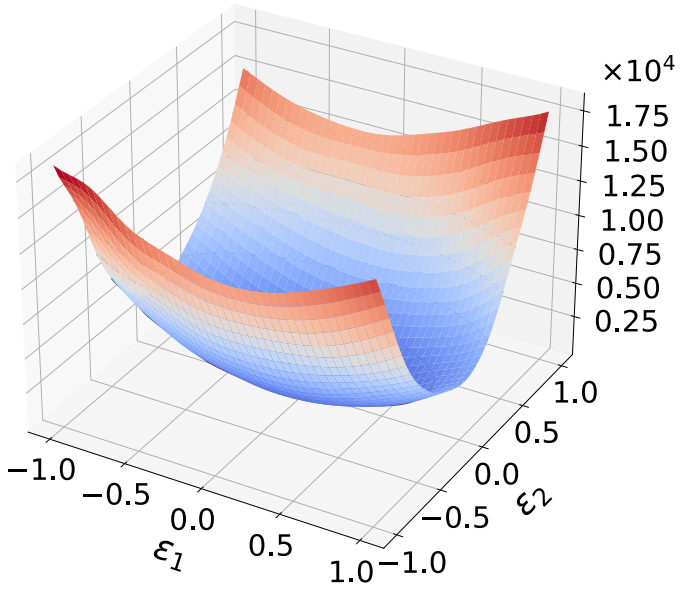


Curriculum PINN training, ρ = 10

The loss landscape provides further insight, and is much smoother when training via curriculum regularization

$$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0$$ β = 30

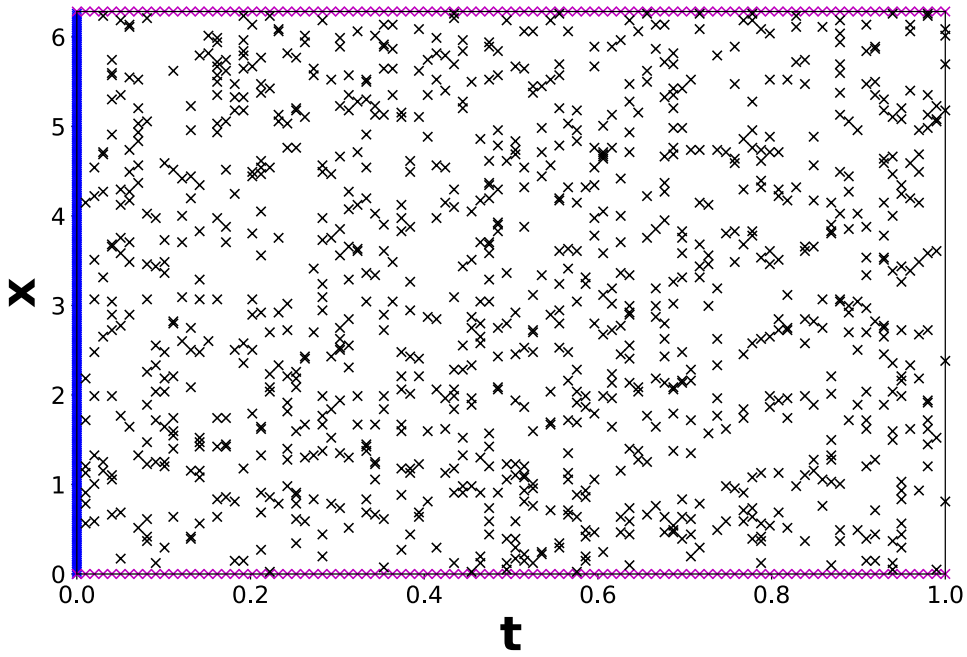




Regular training. Error = 90%

Curriculum training. Error = 0.1%

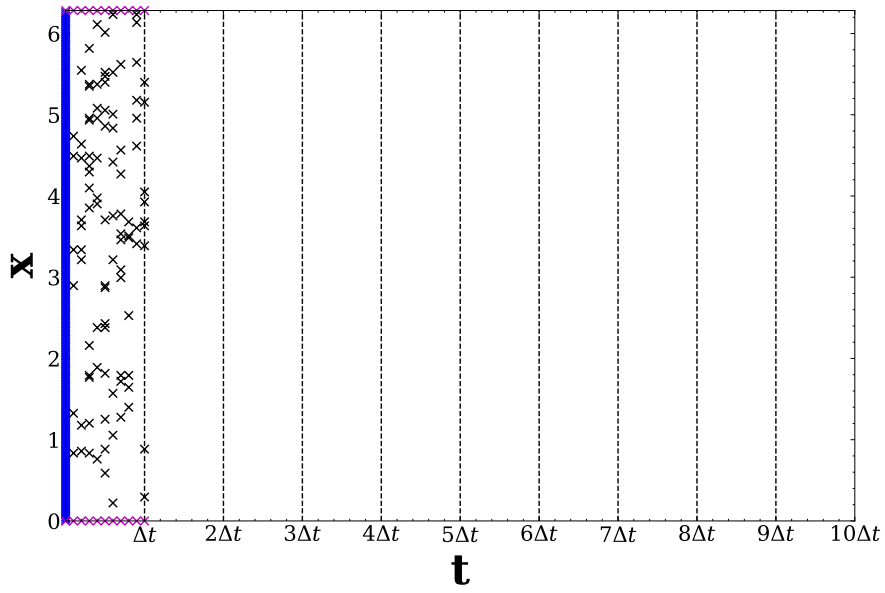
Regular PINN training tries to learn the solution for the whole state space



× Initial condition points × Boundary points × Collocation points

We still assume that we only have the exact solution at t=0 (initial condition).

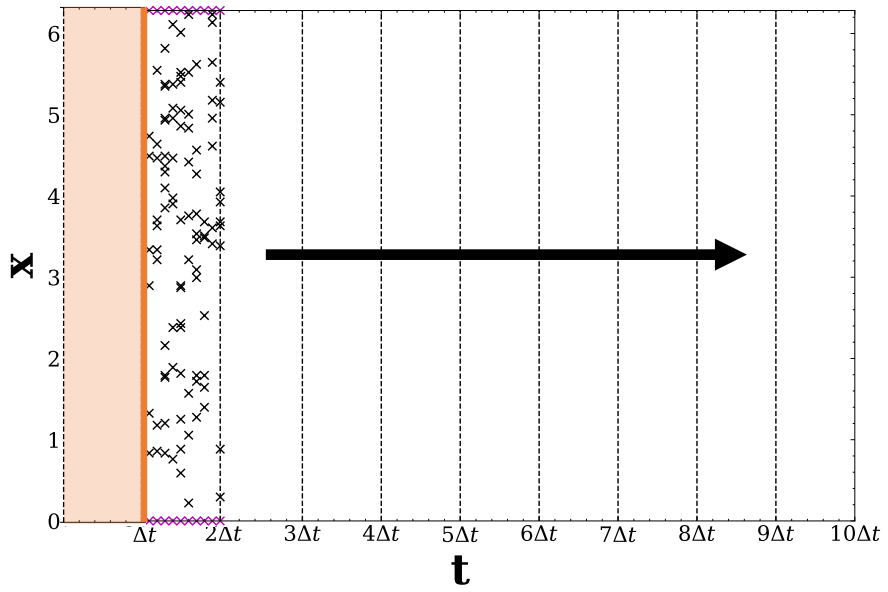
Changing the learning paradigm to sequence-to-sequence learning can address failure modes: We solve one “time block” at a time



Exact initial condition

We still assume that we only have the exact solution at t=0 (initial condition).

Changing the learning paradigm to sequence-to-sequence learning can address failure modes: We solve one “time block” at a time



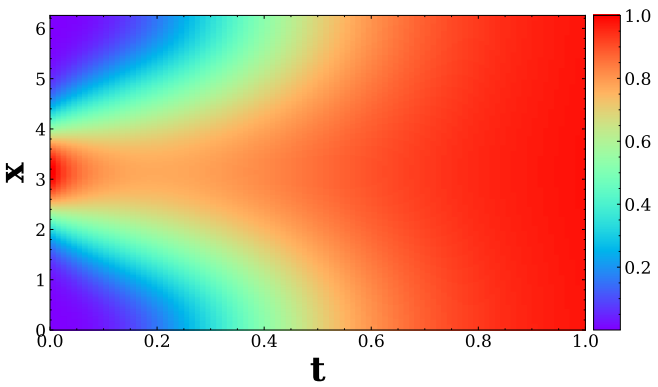
Predicted initial condition for the next time block

Already predicted solution

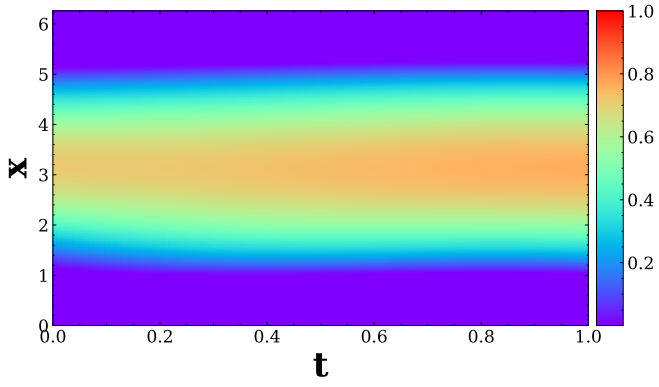
We still assume that we only have the exact solution at t=0 (initial condition).

Sequence-to-sequence learning greatly decreases error for all systems of study

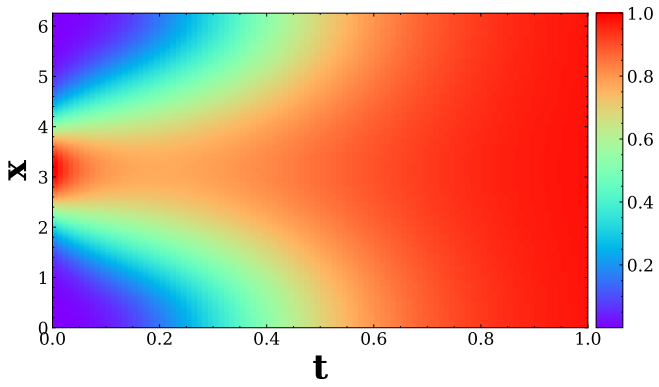
Example: Reaction-diffusion systems



Exact solution, ρ = 5, ν=3



Regular PINN solution, ρ = 5, ν=3



Seq2seq PINN solution, ρ = 5, ν=3

Reaction-diffusion: sequence-to-sequence learning greatly decreases error

		Entire state space	$\Delta t = 0.05$	$\Delta t = 0.1$
$\nu = 2,\ \rho = 5$	Relative error	5.07×10^{-1}	2.04×10^{-2}	$\mathbf{1.18 \times 10^{-2}}$
	Absolute error	2.70×10^{-1}	1.06×10^{-2}	$\mathbf{6.41 \times 10^{-3}}$
$\nu = 3,\ \rho = 5$	Relative error	7.98×10^{-1}	1.92×10^{-2}	$\mathbf{1.56 \times 10^{-2}}$
	Absolute error	4.79×10^{-1}	1.01×10^{-2}	$\mathbf{8.17 \times 10^{-3}}$
$\nu = 4,\ \rho = 5$	Relative error	8.84×10^{-1}	2.37×10^{-2}	$\mathbf{1.59 \times 10^{-2}}$
	Absolute error	5.74×10^{-1}	1.15×10^{-2}	$\mathbf{8.01 \times 10^{-3}}$
$\nu = 5,\ \rho = 5$	Relative error	9.35×10^{-1}	$\mathbf{2.36 \times 10^{-2}}$	2.39×10^{-2}
	Absolute error	6.46×10^{-1}	$\mathbf{1.09 \times 10^{-2}}$	1.15×10^{-2}
$\nu = 6,\ \rho = 5$	Relative error	9.60×10^{-1}	2.81×10^{-2}	$\mathbf{2.69 \times 10^{-2}}$
	Absolute error	6.84×10^{-1}	$\mathbf{1.17 \times 10^{-2}}$	1.28×10^{-2}

Reaction: sequence-to-sequence learning greatly decreases error

		Entire state space	$\Delta t = 0.05$	$\Delta t = 0.1$
$\rho = 5$	Relative error	9.79×10^{-1}	$\mathbf{7.06 \times 10^{-2}}$	7.09×10^{-2}
	Absolute error	5.40×10^{-1}	2.52×10^{-2}	$\mathbf{2.39 \times 10^{-2}}$
$\rho = 6$	Relative error	9.88×10^{-1}	8.25×10^{-2}	7.78×10^{-2}
	Absolute error	5.88×10^{-1}	3.02×10^{-2}	$\mathbf{2.65 \times 10^{-2}}$
$\rho = 7$	Relative error	9.92×10^{-1}	8.16×10^{-2}	$\mathbf{7.56 \times 10^{-2}}$
	Absolute error	6.31×10^{-1}	3.03×10^{-2}	$\mathbf{2.69 \times 10^{-2}}$
$\rho = 8$	Relative error	9.94×10^{-1}	8.19×10^{-2}	$\mathbf{7.44 \times 10^{-2}}$
	Absolute error	6.69×10^{-1}	3.10×10^{-2}	$\mathbf{2.73 \times 10^{-2}}$
$\rho = 9$	Relative error	9.95×10^{-1}	$\mathbf{7.02 \times 10^{-2}}$	8.63×10^{-2}
	Absolute error	7.02×10^{-1}	$\mathbf{2.83 \times 10^{-2}}$	3.21×10^{-2}
$\rho = 10$	Relative error	9.96×10^{-1}	$\mathbf{6.88 \times 10^{-2}}$	7.47×10^{-2}
	Absolute error	7.31×10^{-1}	$\mathbf{2.85 \times 10^{-2}}$	$\mathbf{2.85 \times 10^{-2}}$

Summary on the analysis for physics-informed neural networks

- PINNs is a promising and popular method that works for certain cases – we analyze PINNs for common common scientific problems (convection, reaction, reaction-diffusion)
- The “regularization” term (the PDE constraint) in PINNs can make the loss landscape very hard to optimize
- Rethinking “standard” machine learning training: changing the learning paradigm can greatly decrease error with curriculum regularization, sequence-to-sequence learning

A. S. Krishnapriyan, A. Gholami, S. Zhe, R. M. Kirby, M. W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Neural Information Processing Systems (NeurIPS)*, 2021.

Any More Failure Modes?

- If we try to learn a continuous model from discrete data, do we?
- What does it even mean to learn a continuous model?

Topics

- Characterizing possible failure modes in physics-informed neural networks (think: optimization)
- Meaningfully continuous-in-depth neural networks (think: continuity)
- Other related developments (think: temporal/sequential modeling)
- Conclusion

Continuous-in-Depth Neural Networks*

Joint with Alejandro Queiruga (LBNL -> Google Research), N. Benjamin Erichson (ICSI and UC Berkeley -> Pitt), Dane Taylor (U Buffalo), and Liam Hodgkinson (ICSI and UC Berkeley)

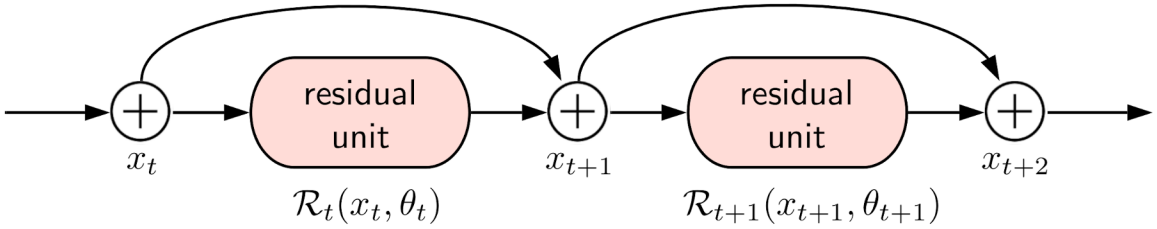
"Continuous-in-Depth Neural Networks," Queiruga, Erichson, Taylor, and Mahoney, arXiv:2008.02389.

"Compressing Deep ODE-Nets using Basis Function Expansions," Queiruga, Erichson, Hodgkinson, and Mahoney, arXiv:2106.10820 and NeurIPS21.

“Learning Continuous Models for Continuous Physics,” Krishnapriyan, Queiruga, and Mahoney, to be submitted.

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:

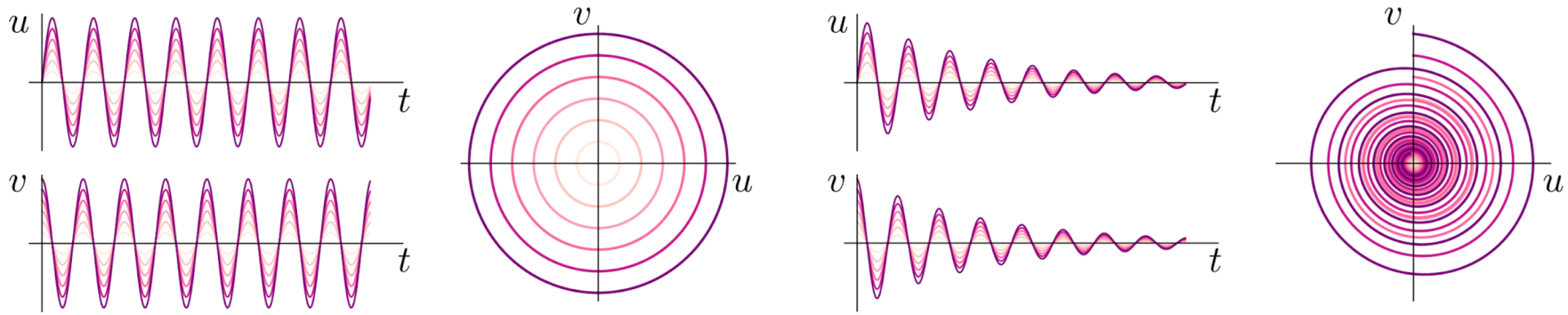
$$x_{k+1} = x_k + \Delta t \mathcal{R}(x_k, \theta_k) \longrightarrow \frac{\partial x(t)}{\partial t} = \mathcal{R}(x(t), t, \theta)$$

=1 sneak it in

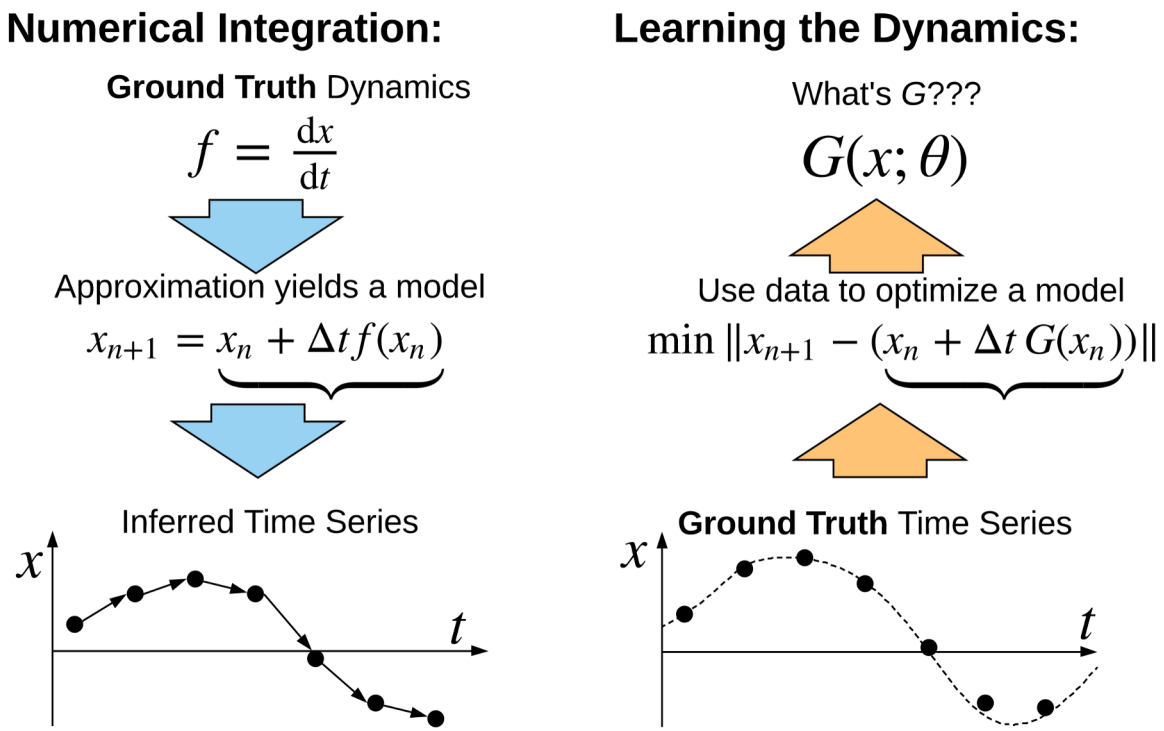
- Spoiler: we show that ResNets are not forward Euler discretizations of a dynamical system in a meaningful way due to overfitting.

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



Revisiting a Simple Dynamical System

- Learning a residual makes sense in many contexts:

  Future = Now + Update

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

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

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

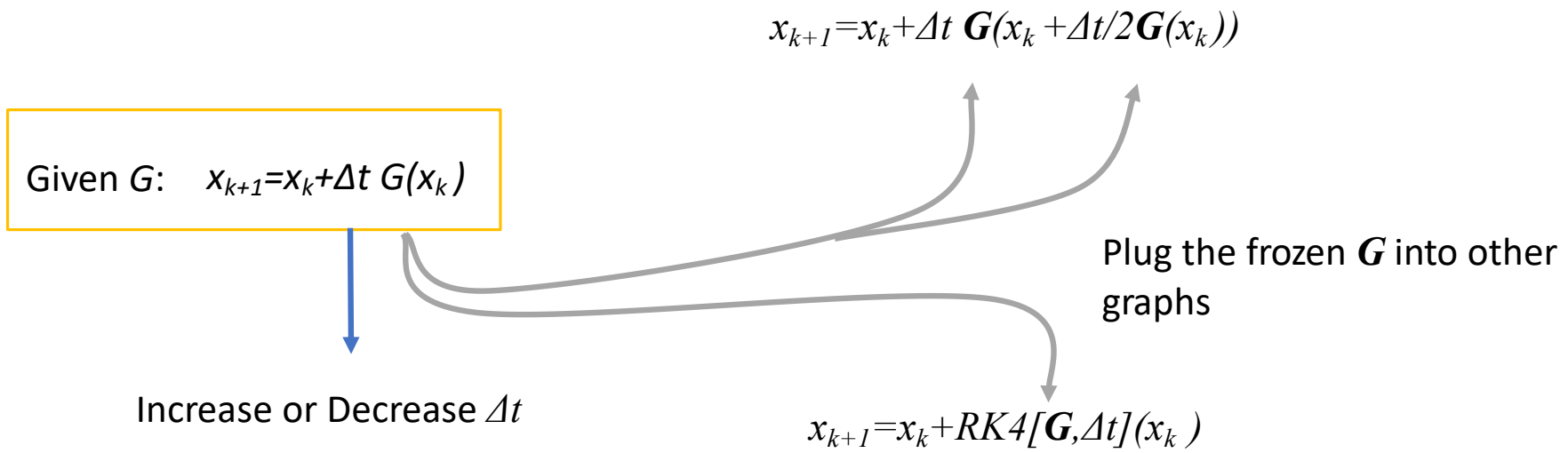
$$\begin{aligned} x(t+\Delta t) &= x(t) + \int_t^{t+\Delta t} f(x,t)\mathrm{d}t \\ &\approx x(t) + scheme[f, x, t, \Delta t] \end{aligned}$$

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

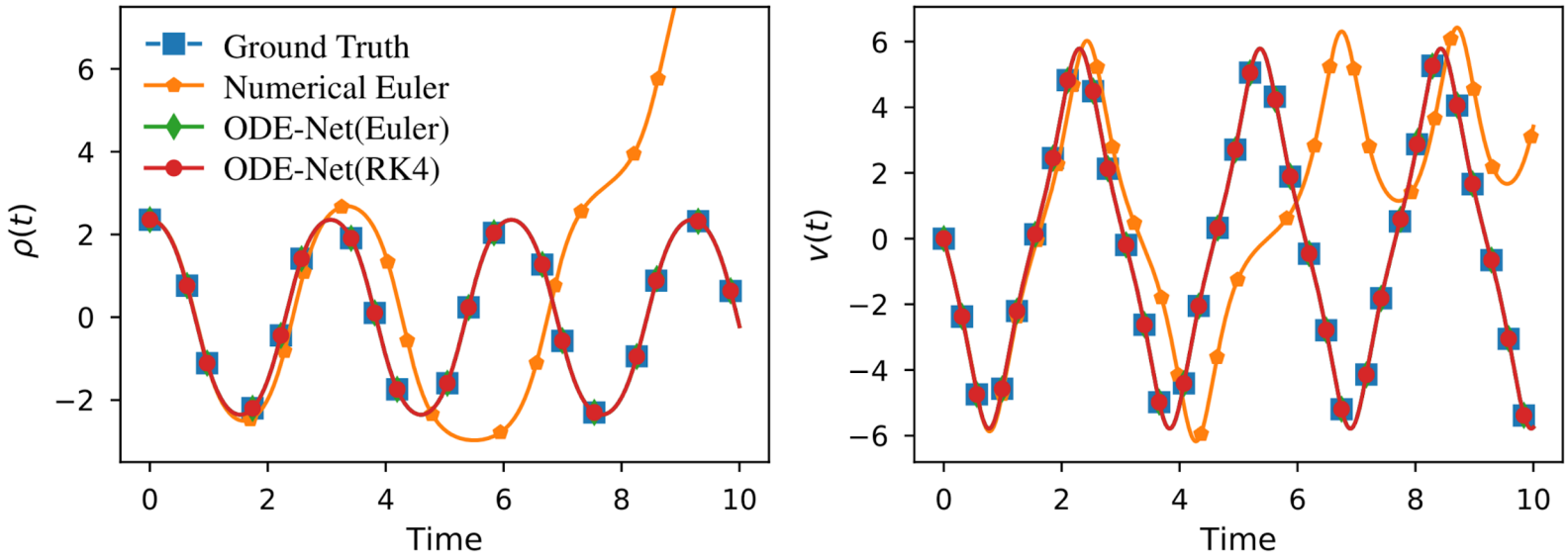
We train three models:

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

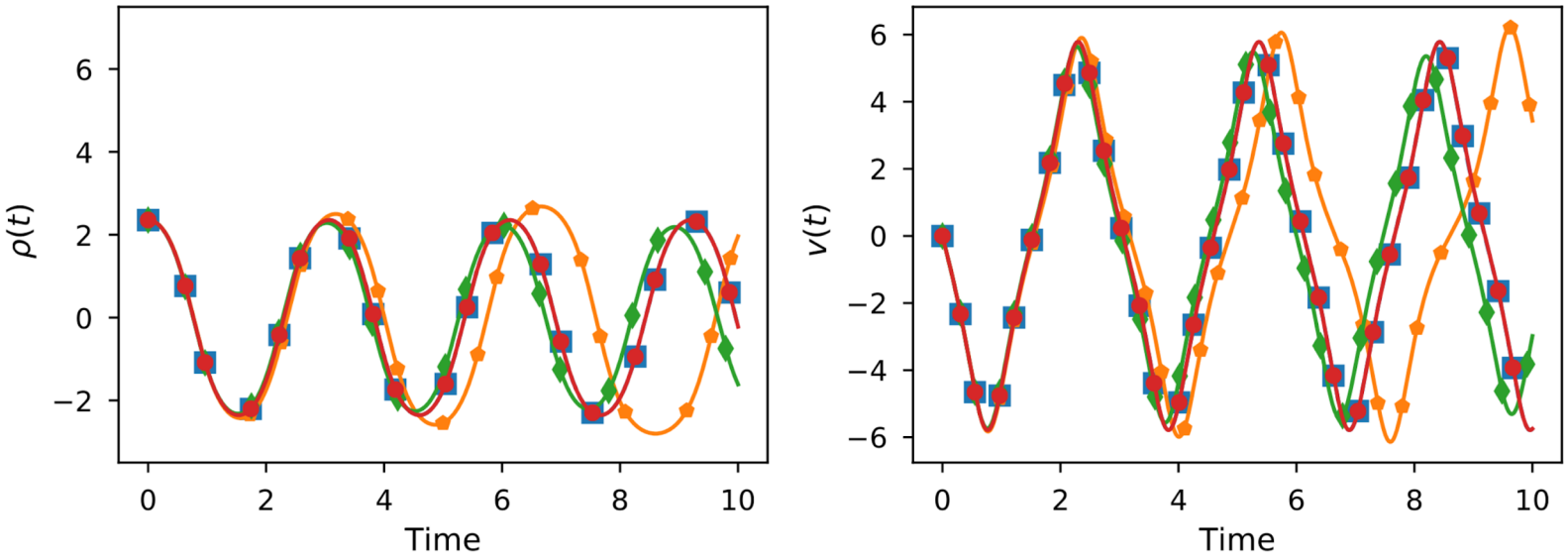
- They are good **discrete models** without changing *Δt.*
- Note how using Euler as a numerical method is inaccurate.



(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.

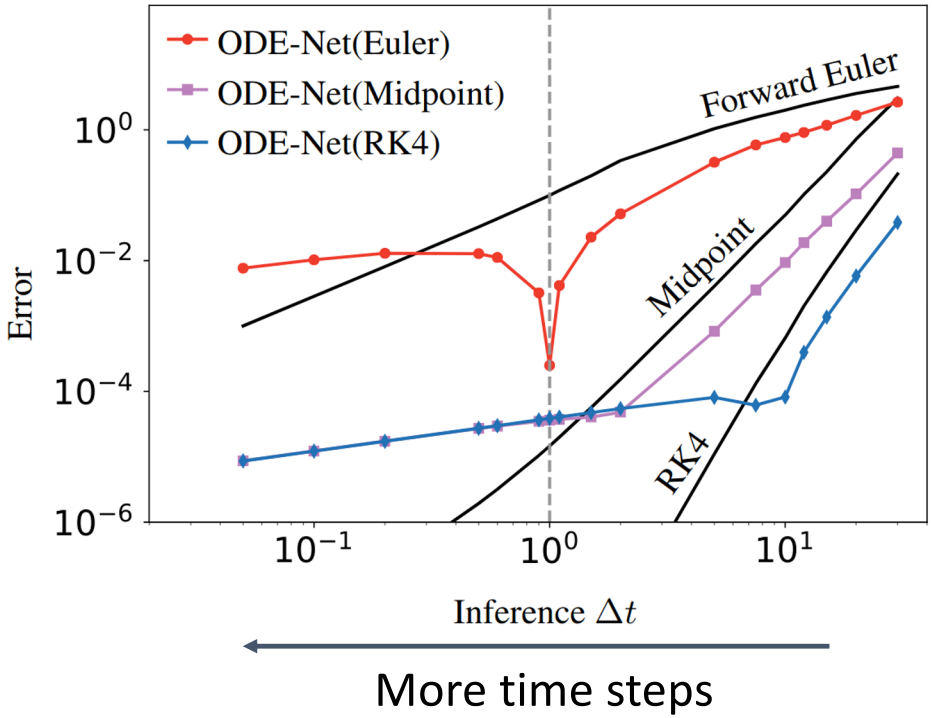


(b) $\Delta t = 0.5\Delta t_{data}$

One-off plots aren't sufficient

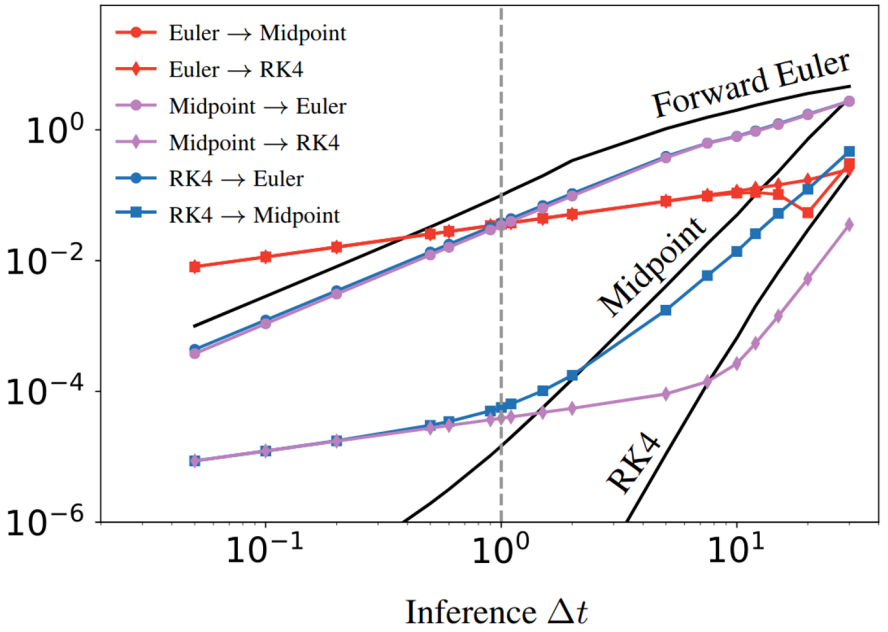
- Sweep Δt and calculate errors to do the full convergence test.

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



More time steps

Part 2: Try different 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).

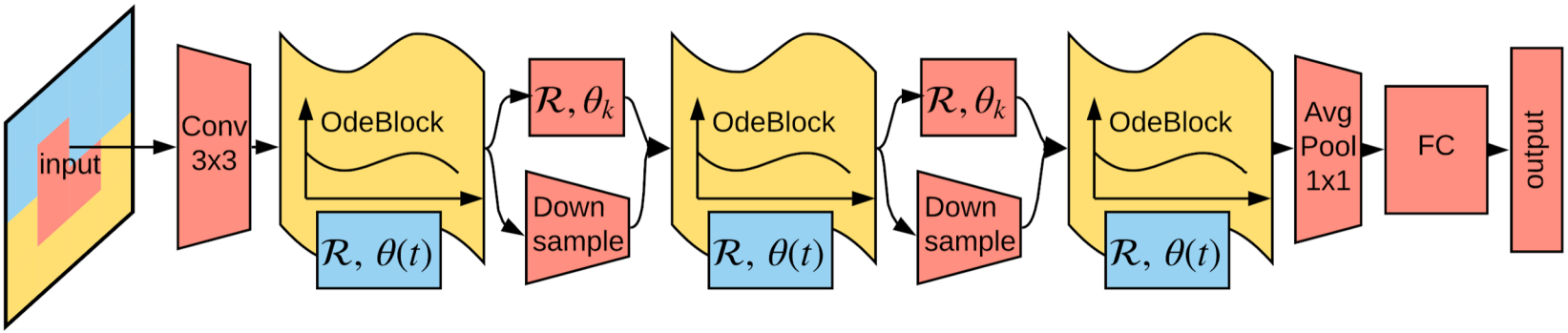
- Our analysis can be seen as a diagnostic tool to assess if the model has overfit: how well does it represent a continuous system, and how well does it exhibit the numerical properties of a continuous operator?

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.



Experiments for Image Classification

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

Updated hypothesis:

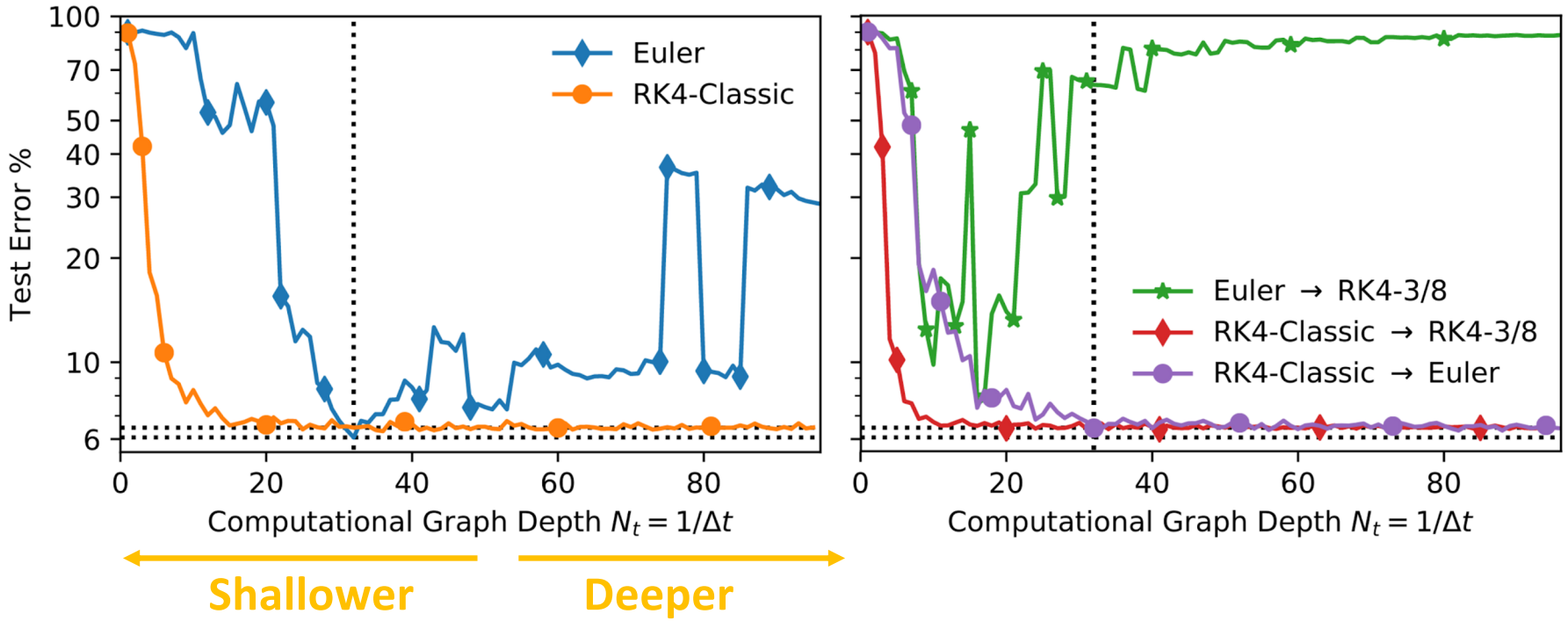
- Expect forward Euler (ResNet) to overfit
- Expect training with Midpoint or RK4 to enable transfer between depths and graph modules

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



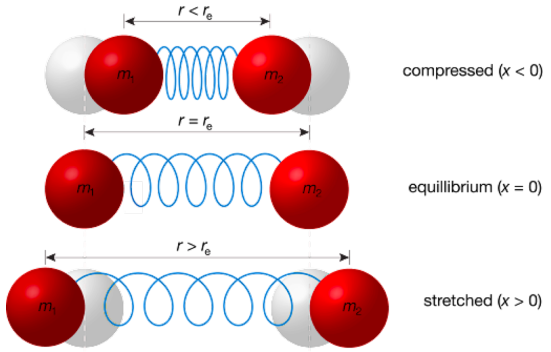
Why ContinuousNet?

- 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.

Looking at the harmonic oscillator example with ODE-Nets

- The harmonic oscillator can be written as two coupled first-order differential equations:

$$\frac{dx}{dt} = y$$
$$\frac{dy}{dt} = -x$$



Train an ODE-Net on:

$$x(0), x(\Delta t), x(2\Delta t) \ldots x(t_{max})$$
$$y(0), y(\Delta t), y(2\Delta t) \ldots y(t_{max})$$

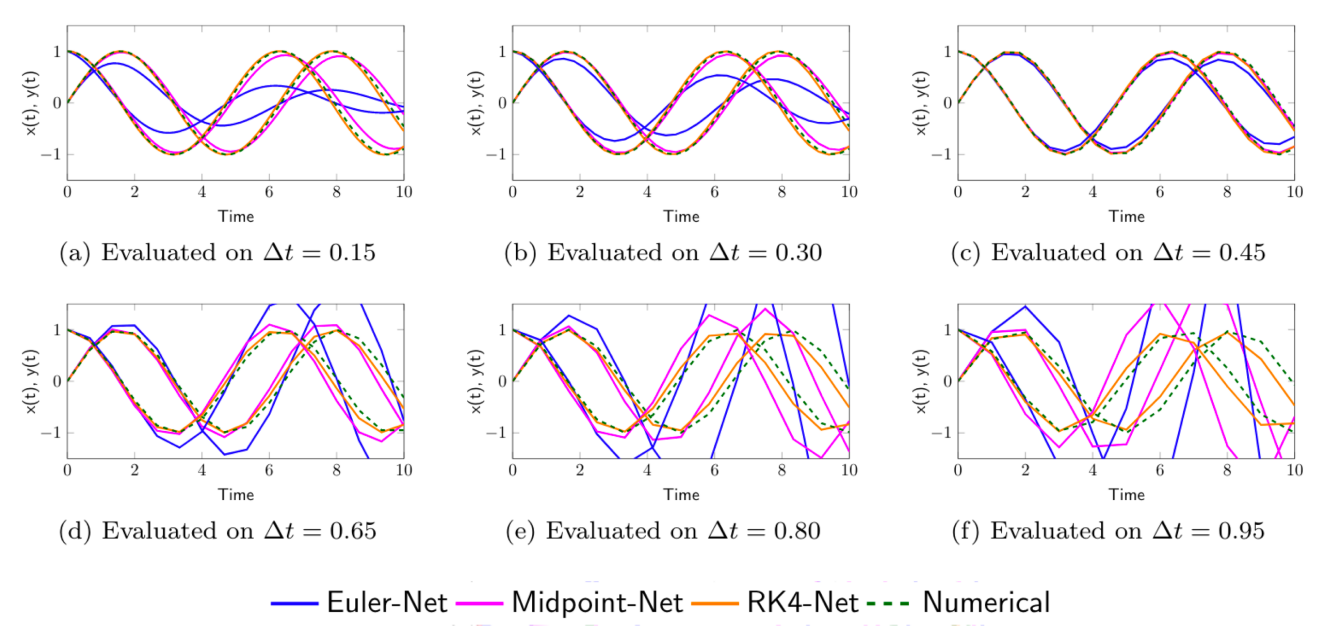
$$x(t + \Delta t) = x(t) + G(x(t)) = \text{Numerical Method}[G(x(t)]$$

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

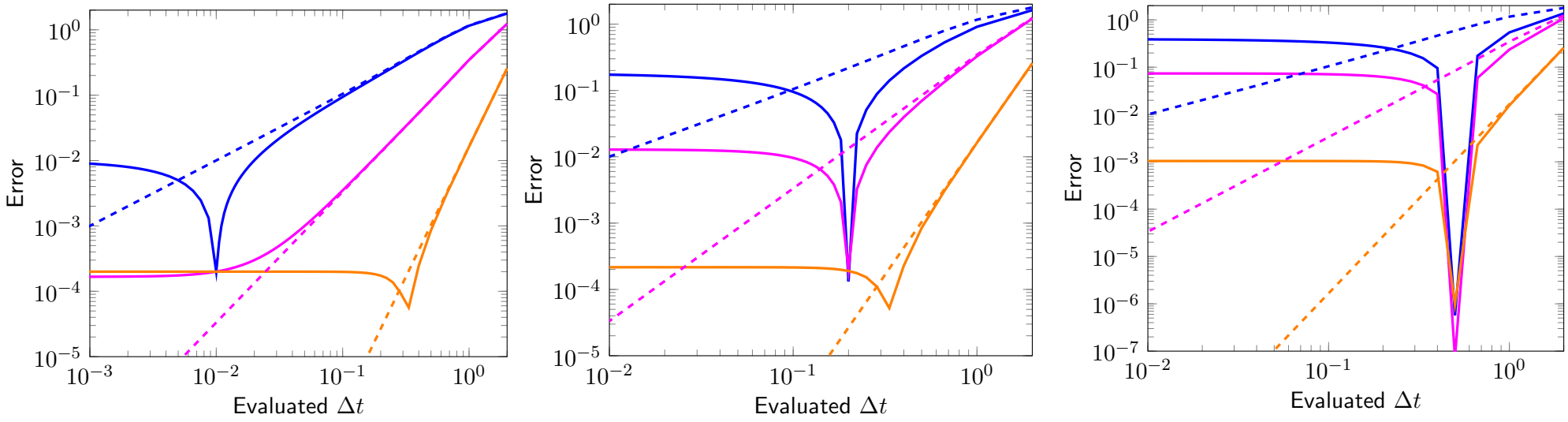
Scheme: Euler, Midpoint, RK4, etc.

Harmonic oscillator: Training at one Δt and evaluating at another Δt gives incorrect results – contrary to how numerical integrators behave (where error goes down as Δt decreases)

All models are trained with a Δt = 0.5 between data points: Euler-Net and Midpoint-Net get worse when they are evaluated away from the trained Δt

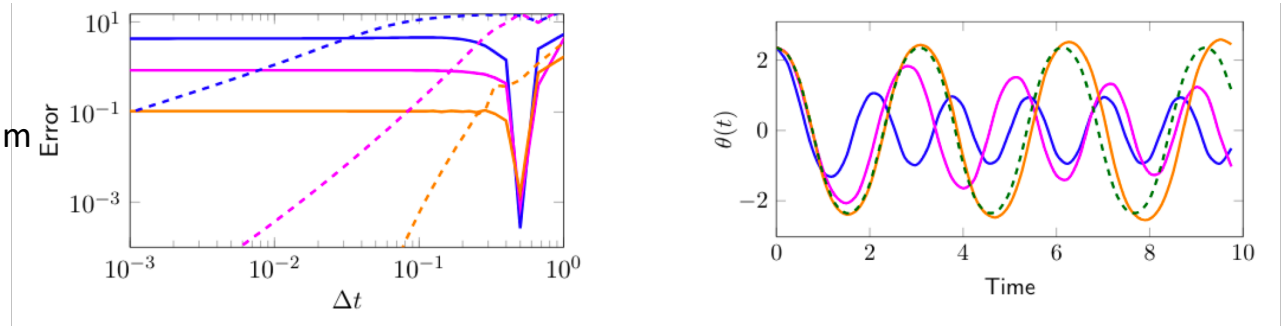


Error as a function of Δt shows that some of the ODE-Nets diverge from the behavior of a numerical integrator

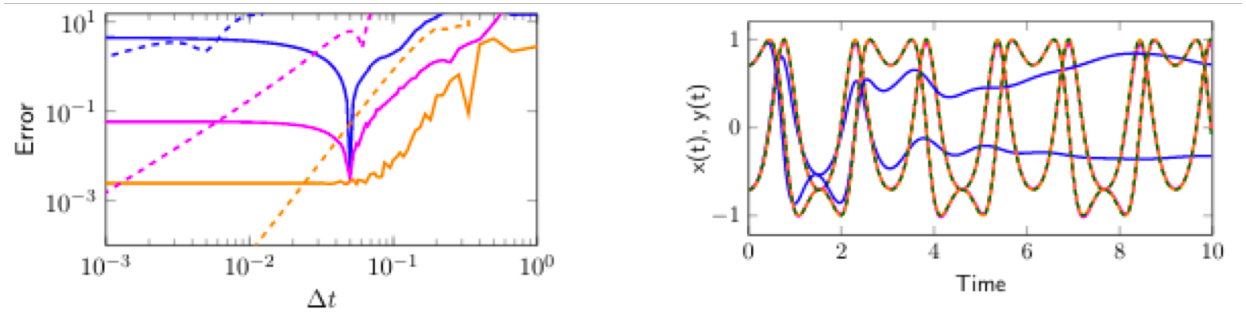


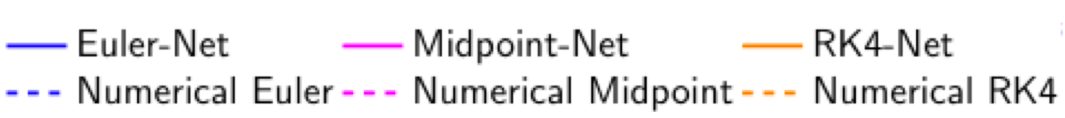
We have a diagnostic method for whether or not we learned a continuous model, which is applicable across many systems

Non-linear pendulum

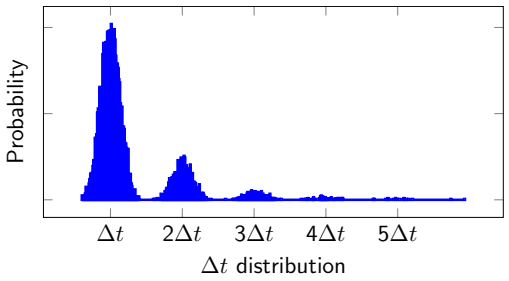


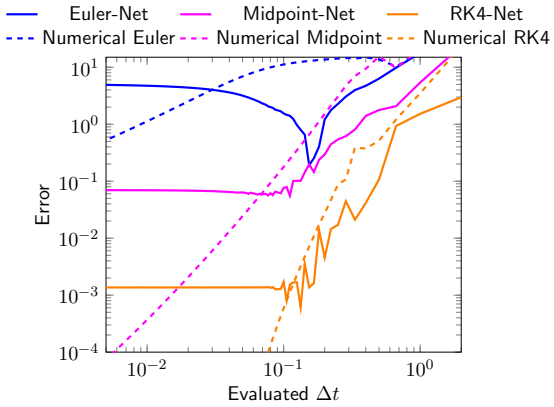
Cartesian pendulum (xy-space): *stiff*

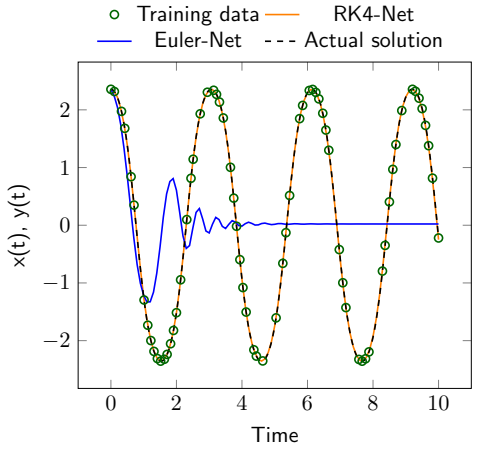


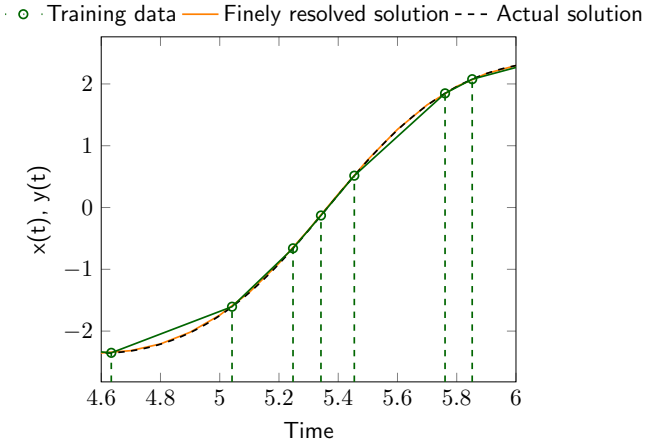


Using this method, once we have learned the underlying continuous dynamics, we can evaluate a problem in multiple scenarios









Topics

- Characterizing possible failure modes in physics-informed neural networks (think: optimization)
- Meaningfully continuous-in-depth neural networks (think: continuity)
- Other related developments (think: temporal/sequential modeling)
- Conclusion

Lipshitz RNNs

Lipschitz Recurrent Neural Network (ICLR 2021)[2]

- We can view RNNs as dynamical systems whose temporal evolutions are governed by an abstract system of differential equations with an external input:

$$\begin{cases} \dot{h}(t) &= \sigma(Wh + Ux + b), & (5) \\ y &= Dh. & (6) \end{cases}$$

- We propose a continuous-time recurrent unit that describes the hidden state's evolution with two parts: a well-understood linear component plus a Lipschitz nonlinearity.

$$\dot{h} = Ah + \sigma(Wh + Ux + b)$$

We assume that the nonlinearity σ is an M-Lipschitz function.

"Lipschitz Recurrent Neural Networks," Erichson, Azencot, Queiruga, Hodgkinson, and Mahoney, arXiv:2006.12070, ICLR21.

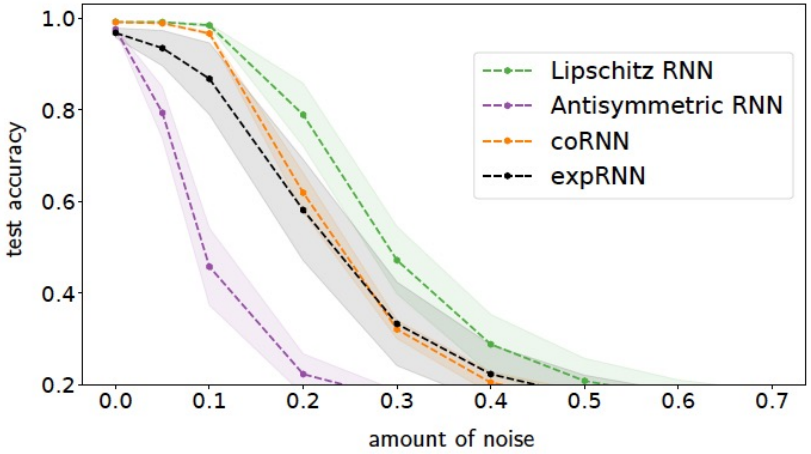
Lipshitz RNNs

Empirical Evaluation

Table 1: Evaluation accuracy on ordered and permuted pixel-by-pixel MNIST.

Name	ordered	permuted	N	# params
LSTM baseline by (Arjovsky et al., 2016)	97.3%	92.7%	128	≈68K
MomentumLSTM (Nguyen et al., 2020)	99.1%	94.7%	256	≈270K
Unitary RNN (Arjovsky et al., 2016)	95.1%	91.4%	512	≈9K
Full Capacity Unitary RNN (Wisdom et al., 2016)	96.9%	94.1%	512	≈270K
Soft orth. RNN (Vorontsov et al., 2017)	94.1%	91.4%	128	≈18K
Kronecker RNN (Jose et al., 2018)	96.4%	94.5%	512	≈11K
Antisymmteric RNN (Chang et al., 2019)	98.0%	95.8%	128	≈10K
Incremental RNN (Kag et al., 2020)	98.1%	95.6%	128	≈4K/8K
Exponential RNN (Lezcano-Casado & Martinez-Rubio, 2019)	98.4%	96.2%	360	≈69K
Sequential NAIS-Net (Ciccone et al., 2018)	94.3%	90.8%	128	≈18K
Lipschitz RNN using Euler (ours)	99.0%	94.2%	64	≈9K
Lipschitz RNN using RK2 (ours)	99.1%	94.2%	64	≈9K
Lipschitz RNN using Euler (ours)	**99.4%**	**96.3%**	128	≈34K
Lipschitz RNN using RK2 (ours)	99.3%	96.2%	128	≈34K

Robustness with Respect to Input Perturbations



"Lipschitz Recurrent Neural Networks," Erichson, Azencot, Queiruga, Hodgkinson, and Mahoney, arXiv:2006.12070, ICLR21.

Noisy RNNs

Noisy Recurrent Neural Networks (NeurIPS 2021)

Let x be an input signal, we consider the following (Itô) SDE model

$$\mathrm{d}\, h_t = f(h_t, x_t)\,\mathrm{d}\, t + \sigma(h_t, x_t)\,\mathrm{d}\, B_t, \qquad y_t = V h_t, \tag{13}$$

where $(B_t)_{t\geq 0}$ is an r-dimensional Brownian motion.

- The functions f and σ are referred to as the *drift* and *diffusion* coefficients, respectively.
  - $$f(h, x) = Ah + a(Wh + Ux + b), \tag{14}$$
    where $a : \mathbb{R} \to \mathbb{R}$ is a Lipschitz continuous scalar activation function.
  - $$\sigma(h, x) = \epsilon(\sigma_1 I + \sigma_2 \mathrm{diag}(f(h, x))), \tag{15}$$
    where the noise level $\epsilon > 0$ is small, and $\sigma_1 \geq 0$ and $\sigma_2 \geq 0$ are tunable parameters
- Noise injection can be viewed as a stochastic learning strategy used to improve robustness of the learning model against data perturbations.

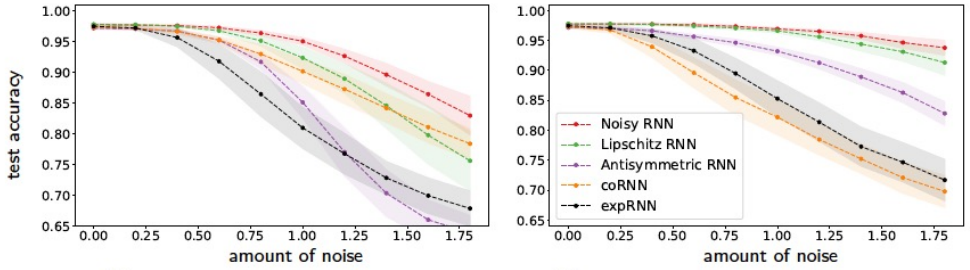
"Noisy Recurrent Neural Networks," Lim, Erichson, Hodgkinson, M. W. Mahoney, arXiv:2102.04877, NeurIPS21.

Noisy RNNs

Results for Electrocardiogram (ECG) Classification

Table 2: Robustness w.r.t. white (σ) and multiplicative (σ_M) noise perturbations on the ECG task.

Name	clean	$\sigma = 0.4$	$\sigma = 0.8$	$\sigma = 1.2$	$\sigma_M = 0.4$	$\sigma_M = 0.8$	$\sigma_M = 1.2$
Antisymmetric RNN	97.1%	96.6%	91.6%	77.0%	96.6%	94.6%	91.2%
CoRNN	97.5%	96.8%	92.9%	87.2%	93.9%	85.4%	78.4%
Exponential RNN	97.4%	95.6%	86.4%	76.7%	95.7%	89.4%	81.3%
Lipschitz RNN	**97.7%**	97.4%	95.1%	88.9%	97.6%	97.0%	95.6%
Noisy RNN (mult./add. noise: 0.03/0.06)	**97.7%**	**97.5%**	**96.3%**	**92.6%**	**97.7%**	**97.3%**	**96.5%**



(a) Additive white noise perturbations. (b) Multiplicative white noise perturbations.

Regularization Induced by Noise Injection

Theorem 2: Implicit regularization induced by noise injection

Under mild assumption on f and σ,

$$\mathbb{E}\ell(h_M^\delta) = \ell(\bar{h}_M^\delta) + \frac{\epsilon^2}{2}[\hat{Q}(\bar{h}^\delta) + \hat{R}(\bar{h}^\delta)] + \mathcal{O}(\epsilon^3), \tag{16}$$

as $\epsilon \to 0$, where the terms $\hat{Q}$ and $\hat{R}$ are given by

$$\hat{Q}(\bar{h}^\delta) = \nabla l(\bar{h}_M^\delta)^T \sum_{k=1}^{M} \delta_{k-1} \hat{\Phi}_{M-1,k} \sum_{m=1}^{M-1} \delta_{m-1} \operatorname{vec} v_m, \tag{17}$$

$$\hat{R}(\bar{h}^\delta) = \sum_{m=1}^{M} \delta_{m-1} tr(\sigma_{m-1}^T \hat{\Phi}_{M-1,m}^T H_{\bar{h}^\delta} l \, \hat{\Phi}_{M-1,m} \sigma_{m-1}), \tag{18}$$

with $\operatorname{vec} v_m$ a vector with the pth component ($p = 1, \ldots, d_h$):

$$[v_m]^p = tr(\sigma_{m-1}^T \hat{\Phi}_{M-2,m}^T H_{\bar{h}^\delta} [f_M]^p \hat{\Phi}_{M-2,m} \sigma_{m-1}). \tag{19}$$

Moreover,

$$|\hat{Q}(\bar{h}^\delta)| \leq C_Q \Delta^2, \quad |\hat{R}(\bar{h}^\delta)| \leq C_R \Delta, \tag{20}$$

for $C_Q, C_R > 0$ independent of Δ.

"Noisy Recurrent Neural Networks," Lim, Erichson, Hodgkinson, M. W. Mahoney, arXiv:2102.04877, NeurIPS21.

LEMs

From Multi-Resolution to Long Expressive Memory Units

- A simple example of a system of *two-scale ODEs* is given by

$$\frac{d\boldsymbol{y}}{dt} = \tau_y \left(\sigma\left(\mathbf{W}_y \boldsymbol{z} + \mathbf{V}_y \mathbf{x} + \mathbf{b}_y\right) - \boldsymbol{y}\right), \quad \frac{d\boldsymbol{z}}{dt} = \tau_z \left(\sigma\left(\mathbf{W}_z \boldsymbol{y} + \mathbf{V}_z \mathbf{x} + \mathbf{b}_z\right) - \boldsymbol{z}\right). \tag{21}$$

- Here τ_y and τ_z are the two time scales,
- $\boldsymbol{y}(t) \in \mathbb{R}^{d_y}$, and $\boldsymbol{z}(t) \in \mathbb{R}^{d_z}$ are the vectors of *slow* and *fast* variables.

- Two scales (one fast and one slow) are not enough for for complicate problems, in practice.

- We can generalize this idea to a *multiscale* version, provided by the following set of ODEs,

$$\begin{aligned} \frac{d\boldsymbol{y}}{dt} &= \hat{\sigma}\left(\mathbf{W}_2 \boldsymbol{y} + \mathbf{V}_2 \mathbf{u} + \mathbf{b}_2\right) \odot \left(\sigma\left(\mathbf{W}_y \boldsymbol{z} + \mathbf{V}_y \mathbf{u} + \mathbf{b}_y\right) - \boldsymbol{y}\right), \\ \frac{d\boldsymbol{z}}{dt} &= \hat{\sigma}\left(\mathbf{W}_1 \boldsymbol{y} + \mathbf{V}_1 \mathbf{u} + \mathbf{b}_1\right) \odot \left(\sigma\left(\mathbf{W}_z \boldsymbol{y} + \mathbf{V}_z \mathbf{u} + \mathbf{b}_z\right) - \boldsymbol{z}\right). \end{aligned} \tag{22}$$

"Long Expressive Memory for Sequence Modeling," Rusch, Mishra, Erichson, and Mahoney, arXiv:2110.04744.

LEMs

Training Long Expressive Memory Units

- We discretize the system of ODEs with using an implicit-explicit time-stepping scheme

$$\begin{aligned}
\mathbf{\Delta t}_n &= \Delta t\hat{\sigma}(\mathbf{W}_1 y_{n-1} + \mathbf{V}_1 \mathbf{u}_n + \mathbf{b}_1),\\
\overline{\mathbf{\Delta t}}_n &= \Delta t\hat{\sigma}(\mathbf{W}_2 y_{n-1} + \mathbf{V}_2 \mathbf{u}_n + \mathbf{b}_2),\\
z_n &= (1 - \mathbf{\Delta t}_n) \odot z_{n-1} + \mathbf{\Delta t}_n \odot \sigma(\mathbf{W}_z y_{n-1} + \mathbf{V}_z \mathbf{u}_n + \mathbf{b}_z),\\
y_n &= (1 - \overline{\mathbf{\Delta t}}_n) \odot y_{n-1} + \overline{\mathbf{\Delta t}}_n \odot \sigma(\mathbf{W}_y z_n + \mathbf{V}_y \mathbf{u}_n + \mathbf{b}_y).
\end{aligned} \tag{23}$$

- The particular model mitigates the exploding and vanishing gradients problem.
- We can show that this model is a universal approximation of general dynamical systems,
- and also a universal approximation of multiscale dynamical systems.

Results

Table 3: Test accuracies on Google12 (benchmark for speech recognition).

Model	test accuracy	# units	# params
tanh-RNN	73.4%	128	27k
LSTM	94.9%	128	107k
GRU	95.2%	128	80k
expRNN	92.3%	128	19k
coRNN	94.7%	128	44k
LEM	**95.7%**	128	107k

Table 4: Test L^2 error on heart-rate prediction.

Model	test L^2 error	# units	# params
LSTM	9.93	128	67k
expRNN	1.63	256	34k
coRNN	1.61	128	34k
UnICORNN (3 layers)	1.31	128	34k
LEM	**0.85**	128	67k

"Long Expressive Memory for Sequence Modeling," Rusch, Mishra, Erichson, and Mahoney, arXiv:2110.04744.

Topics

- Characterizing possible failure modes in physics-informed neural networks (think: optimization)
- Meaningfully continuous-in-depth neural networks (think: continuity)
- Other related developments (think: temporal/sequential modeling)
- Conclusion

Conclusion

- Grand challenge: Combining domain-driven scientific models and data-driven machine learning models
- Fundamental foundational question for data science
- Lots of obvious, but even more non-obvious, challenges
  - We looked at several, but many others
- Failure to address the challenge = failure to deliver on the promise …