Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations

Journal of Computational physics, 2019

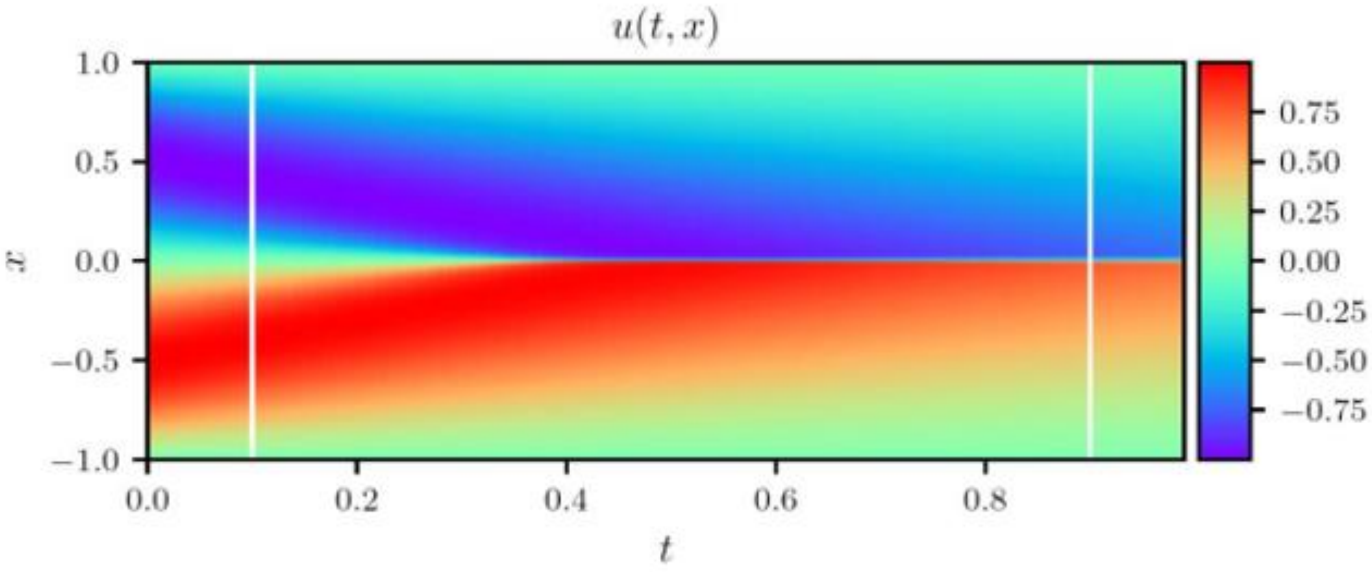
Contributions

- Data-driven solution of PDEs using –
  - Continuous time models
  - Discrete time models
- Data-driven discovery of PDEs (Inverse problem) using –
  - Continuous time models
  - Discrete time models
- All the above approaches based on neural network with innovative loss schemes
- Large step size made possible for discrete time models
- Works very well even in highly non-linear solution space

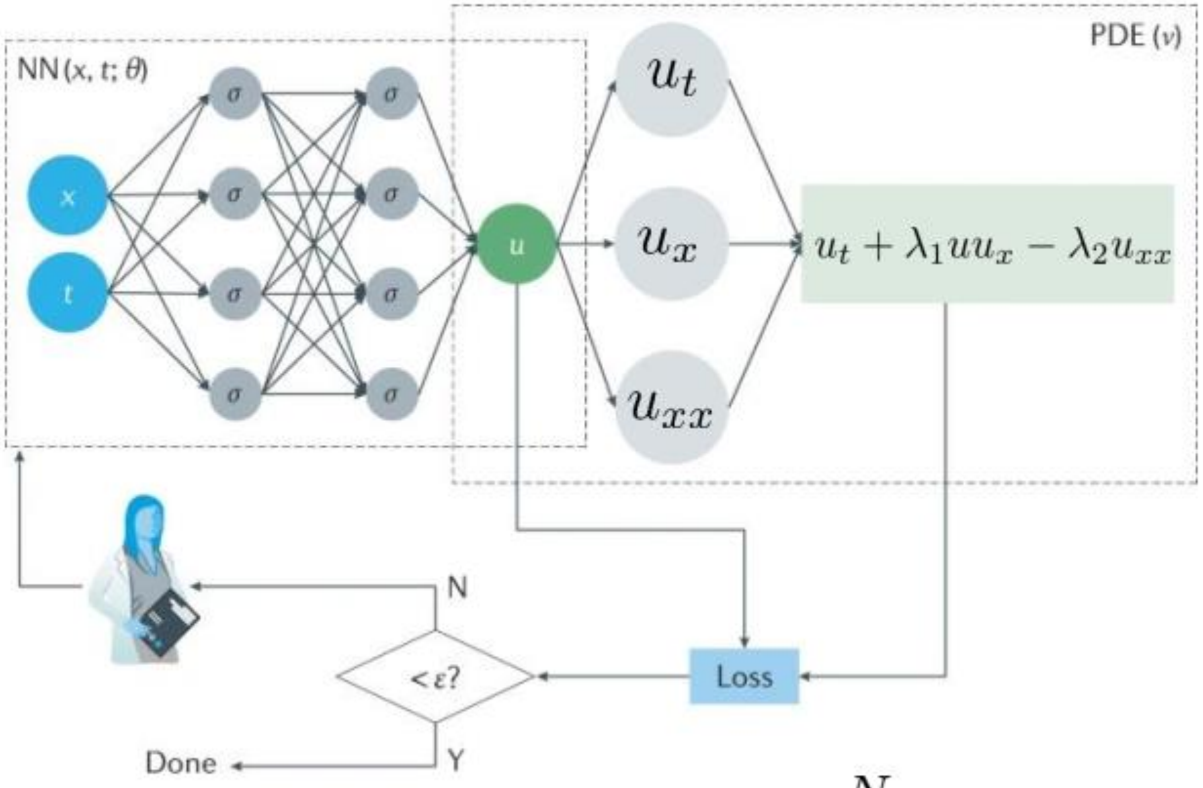
$$u_t + \mathcal{N}[u; \lambda] = 0, \ x \in \Omega, \ t \in [0, T],$$

Data Driven Solution: Continuous Time Models

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1, 1], \quad t \in [0, 1],$$
$$u(0, x) = -\sin(\pi x),$$
$$u(t, -1) = u(t, 1) = 0.$$



Data Driven Solution: Continuous Time Models (contd.)



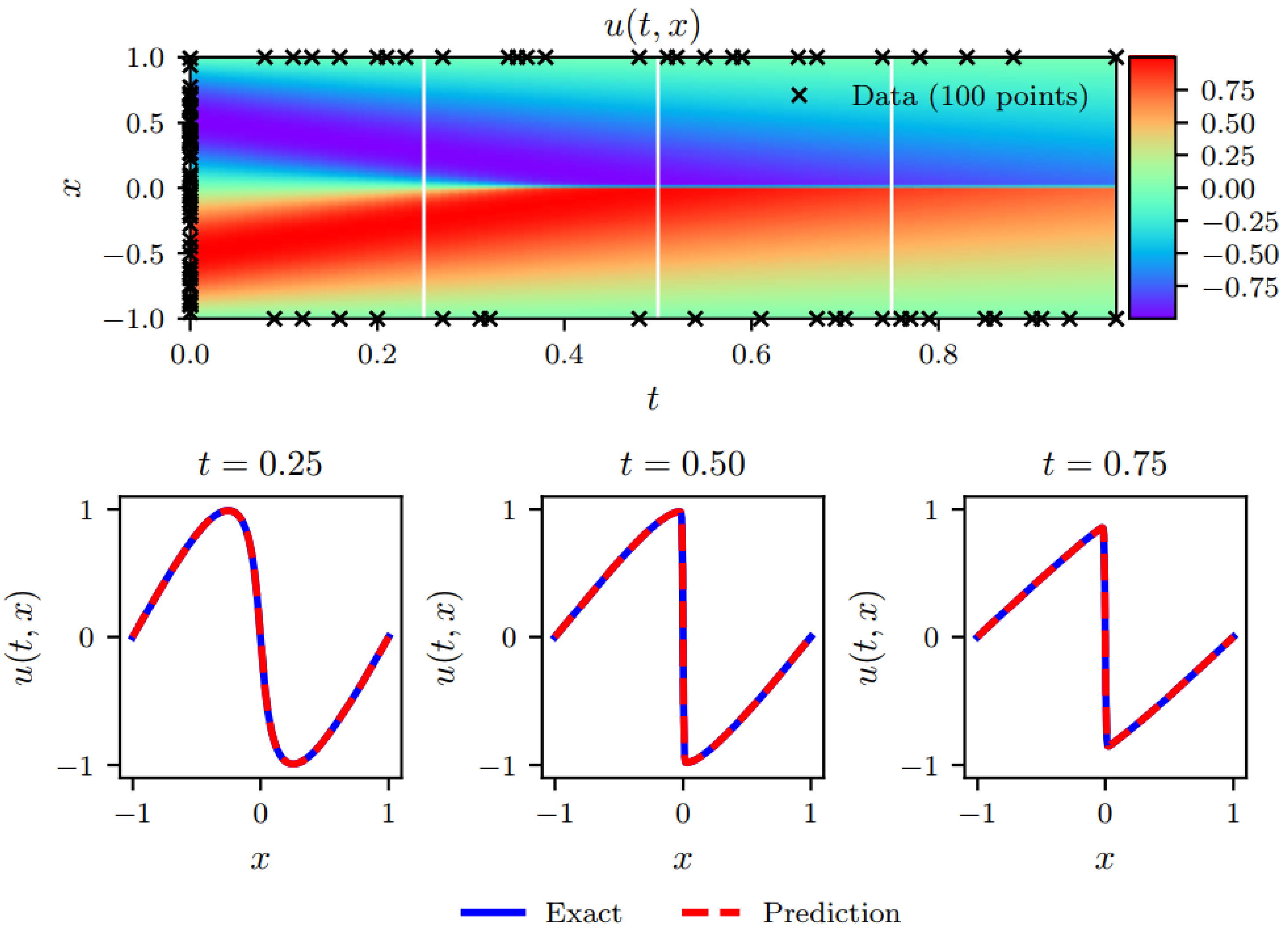
Minimize

$$\mathcal{L} = \mathcal{L}_u + \mathcal{L}_f$$

$$\mathcal{L}_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2$$

$$L_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2$$

Data Driven Solution: Continuous Time Models (contd.)



Data Driven Solution: Continuous Time Models (contd.)

There needs to sufficient number of boundary points

N_u \ N_f	2000	4000	6000	7000	8000	10000
20	2.9e-01	4.4e-01	8.9e-01	1.2e+00	9.9e-02	4.2e-02
40	6.5e-02	1.1e-02	5.0e-01	9.6e-03	4.6e-01	7.5e-02
60	3.6e-01	1.2e-02	1.7e-01	5.9e-03	1.9e-03	8.2e-03
80	5.5e-03	1.0e-03	3.2e-03	7.8e-03	4.9e-02	4.5e-03
100	6.6e-02	2.7e-01	7.2e-03	6.8e-04	2.2e-03	6.7e-04
200	1.5e-01	2.3e-03	8.2e-04	8.9e-04	6.1e-04	4.9e-04

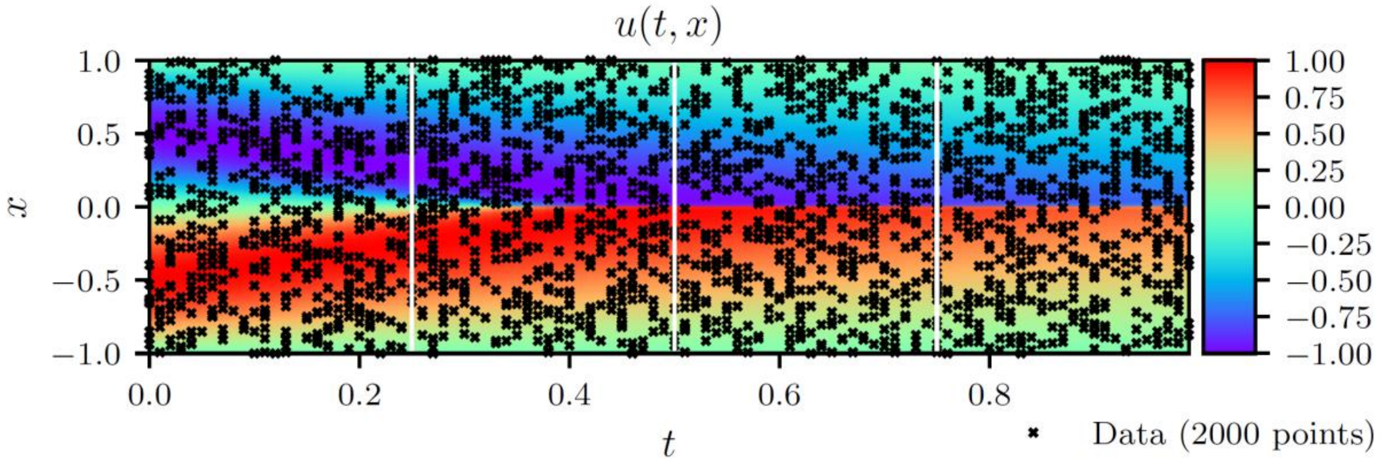
Data Driven Discovery: Continuous Time Models

$$u_t + \lambda_1 u u_x = \lambda_2 u_{xx}, \;\; x \in [-1, 1], \;\; t \in [0, 1]$$

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

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

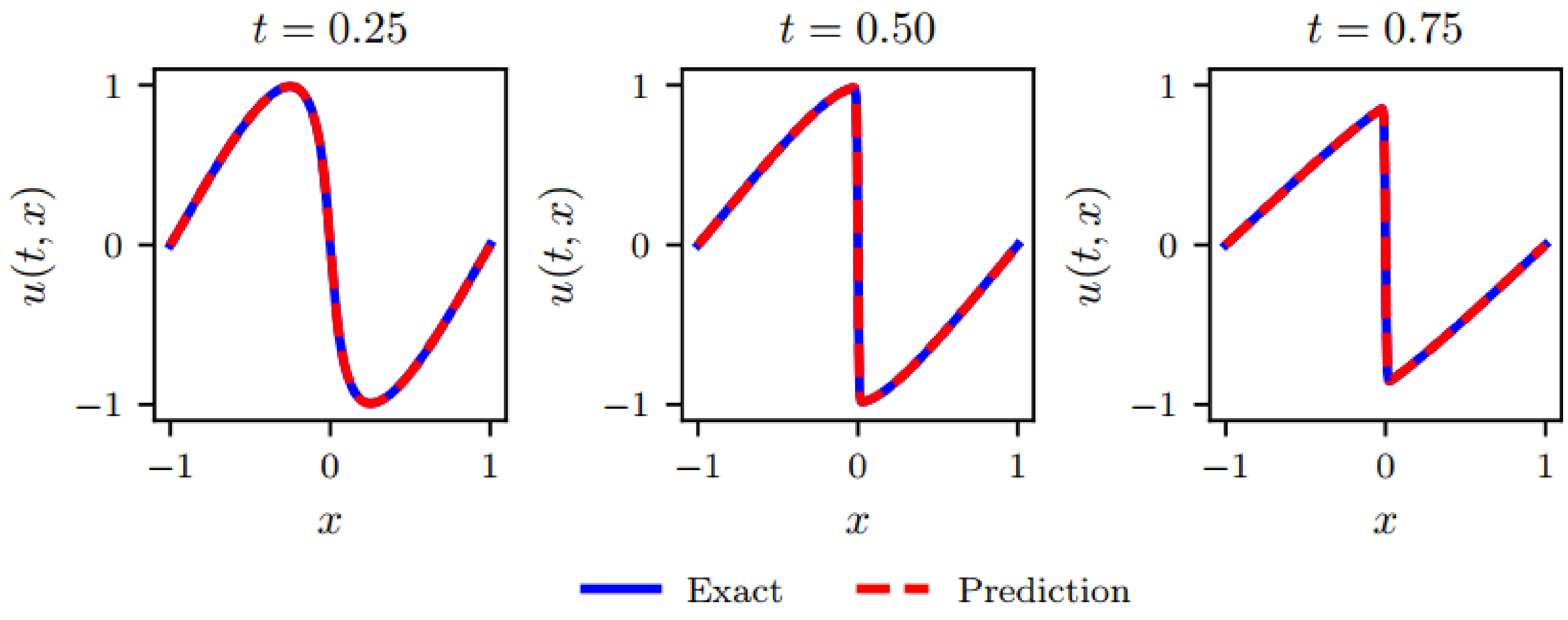
$$\mathcal{N}[u; \lambda] = \lambda_1 u u_x - \lambda_2 u_{xx}$$



Interesting Points:

- The 2000 points require ground truth for the latent variable *u*
- The other points are unsupervised collocation points
- What would happen if we did not use any ground truth points? What if we simply used the forward problem setting consisting of initial, boundary and collocation points??

Data Driven Discovery: Continuous Time Models (contd.)



Correct PDE	$u_t + uu_x - 0.0031831u_{xx} = 0$
Identified PDE (clean data)	$u_t + 0.99915uu_x - 0.0031794u_{xx} = 0$
Identified PDE (1% noise)	$u_t + 1.00042uu_x - 0.0032098u_{xx} = 0$

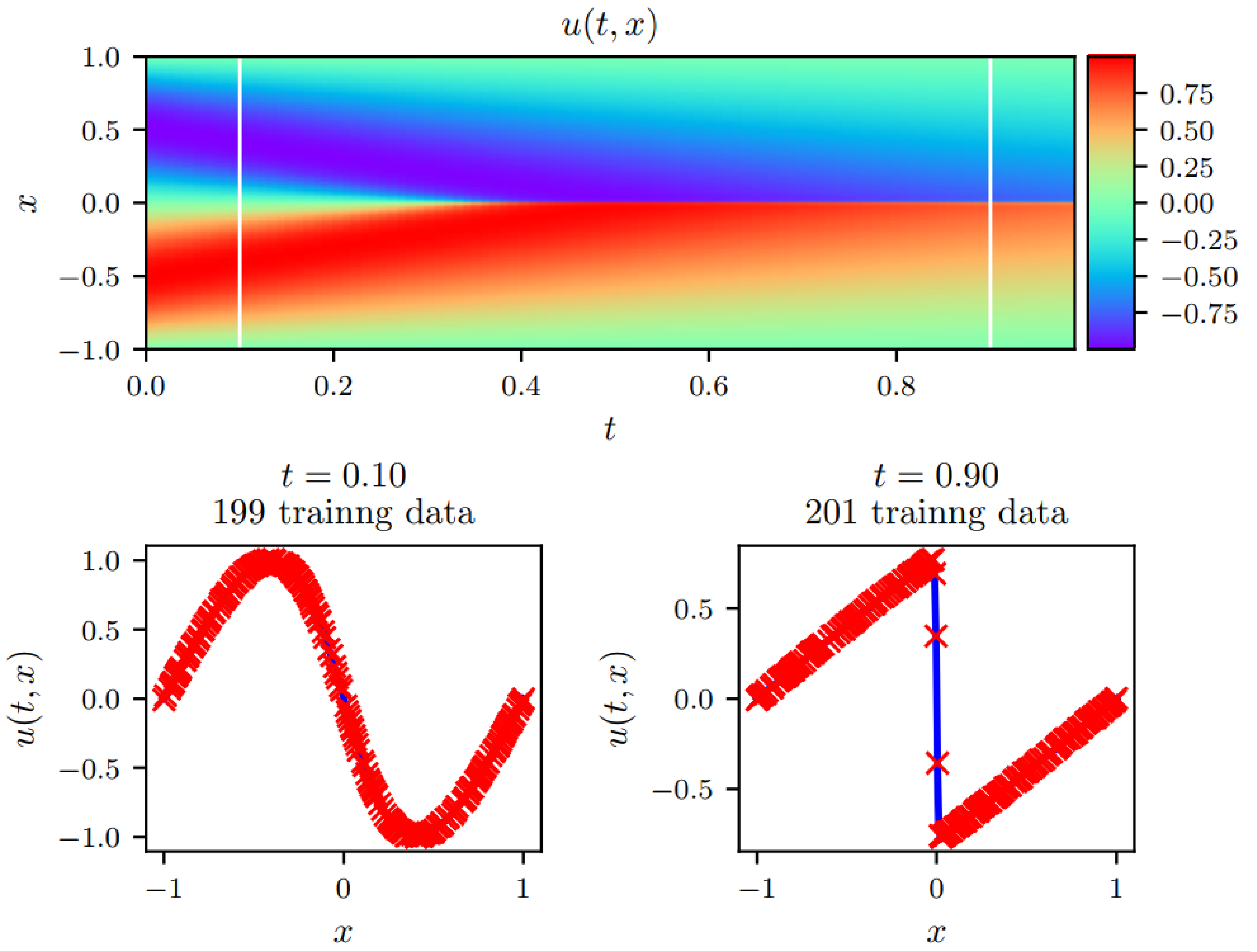
Data Driven Discovery: Continuous Time Models (contd.)

very little consistency of error % with increasing training points

	% error in λ_1				% error in λ_2			
N_u \ noise	0%	1%	5%	10%	0%	1%	5%	10%
500	0.131	0.518	0.118	1.319	13.885	0.483	1.708	4.058
1000	0.186	0.533	0.157	1.869	3.719	8.262	3.481	14.544
1500	0.432	0.033	0.706	0.725	3.093	1.423	0.502	3.156
2000	0.096	0.039	0.190	0.101	0.469	0.008	6.216	6.391

Data Driven Discovery: Discrete Time Models

- In continuous time models, need training data from all over the space
- In practical scenarios, can only observe ground truth points at distinct time points in space
- Discrete time models require only two such time point data for inverse problem solving
- The temporal gap between these two time points are allowed to be large



Data Driven Discovery: Discrete Time Models (contd.)

$$u^{n+c_i} = u^n - \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}; \lambda], \quad i = 1, \ldots, q,$$
$$u^{n+1} = u^n - \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}; \lambda].$$

$$u^{n+c_j}(x) = u(t^n + c_j \Delta t, x)$$
$$u^n = u_i^n, \quad i = 1, \ldots, q,$$
$$u^{n+1} = u_i^{n+1}, \quad i = 1, \ldots, q.$$

$$\Downarrow$$

$$u_i^n := u^{n+c_i} + \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}; \lambda], \quad i = 1, \ldots, q,$$
$$u_i^{n+1} := u^{n+c_i} + \Delta t \sum_{j=1}^{q} (a_{ij} - b_j) \mathcal{N}[u^{n+c_j}; \lambda], \quad i = 1, \ldots, q.$$

Output neurons of the neural network

$$\left[u^{n+c_1}(x), \ldots, u^{n+c_q}(x)\right] \Longrightarrow \begin{array}{l} \left[u_1^n(x), \ldots, u_q^n(x), u_{q+1}^n(x)\right] \\ \left[u_1^{n+1}(x), \ldots, u_q^{n+1}(x), u_{q+1}^{n+1}(x)\right] \end{array}$$

Data Driven Discovery: Discrete Time Models (contd.)

$$SSE = SSE_n + SSE_{n+1},$$

time step n and (n+1) which has delta t difference

$$SSE_n := \sum_{j=1}^{q} \sum_{i=1}^{N_n} |u_j^n(x^{n,i}) - u^{n,i}|^2,$$

j: stage, i: training sample

Food for thought:
Why are we considering all u's to have the same ground truth output for same n irrespective of difference in stage no.??

$$SSE_{n+1} := \sum_{j=1}^{q} \sum_{i=1}^{N_{n+1}} |u_j^{n+1}(x^{n+1,i}) - u^{n+1,i}|^2.$$

Correct PDE	$u_t + uu_x + 0.003183u_{xx} = 0$
Identified PDE (clean data)	$u_t + 1.000uu_x + 0.003193u_{xx} = 0$
Identified PDE (1% noise)	$u_t + 1.000uu_x + 0.003276u_{xx} = 0$

What About Data Driven Solution Using Discrete Time Models??

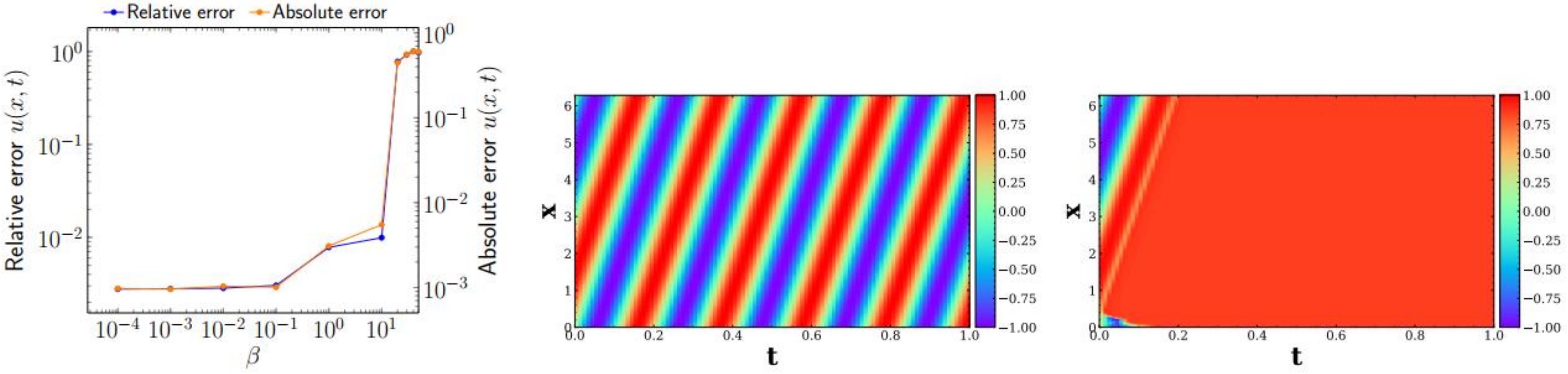
- In discrete time model, you will have collocation points taken from one time stamp and you predict solution for another time stamp
- Essentially, you are not discovering the hidden variable *u* after training
- You are only managing to know about the *u* values for a particular time step after one round of training
- Potentially low possibility of wide adoption for PDE solving

Limitations of Continuous Time Models and High Level Idea of Solutions

Problem with Soft Constraint Regularization

1D convection equation with analytical solution

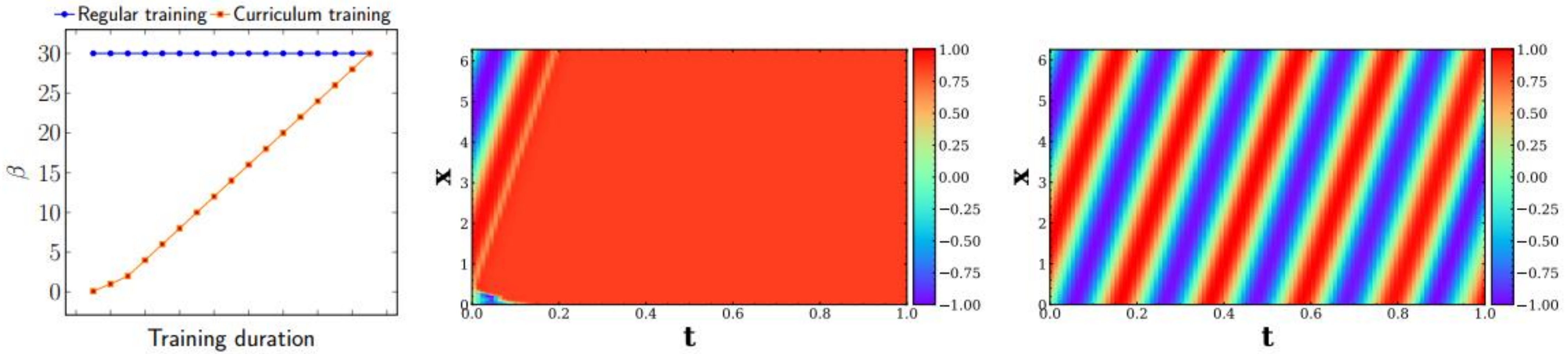
$$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0, \quad x \in \Omega, \ t \in [0, T], \qquad \begin{aligned} u(x,0) &= sin(x), \\ u(0,t) &= u(2\pi, t) \end{aligned}$$



(a) *Error for different* β **(b)** *Exact solution for* $\beta = 30$ **(c)** *PINN solution for* $\beta = 30$

Characterizing possible failure modes in physics-informed neural networks, NIPS, 2021

Possible Solution (Curriculum Learning)



(a) *Curriculum regularization schematic*

(b) *Regular training PINN solution for* $\beta = 30$

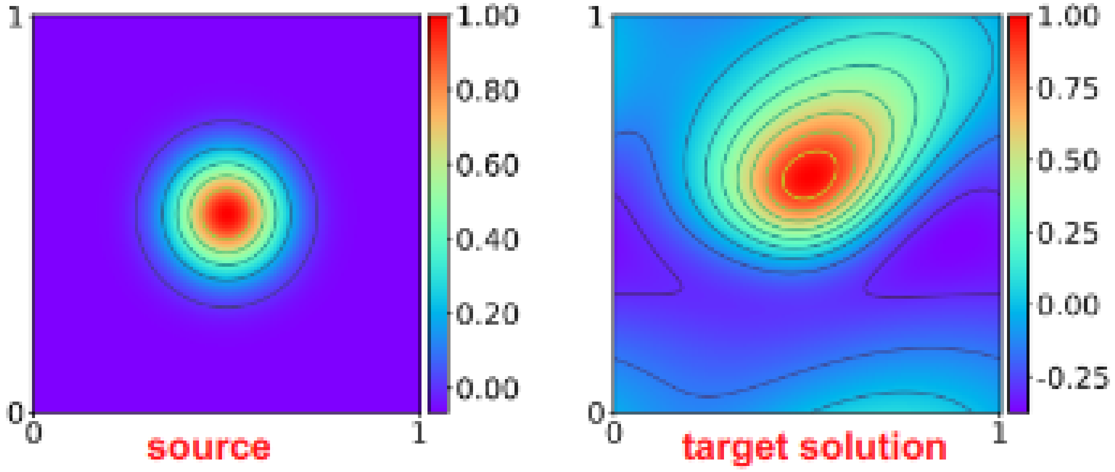
(c) *Curriculum training PINN solution for* $\beta = 30$

Problem with Collocation Point Sampling

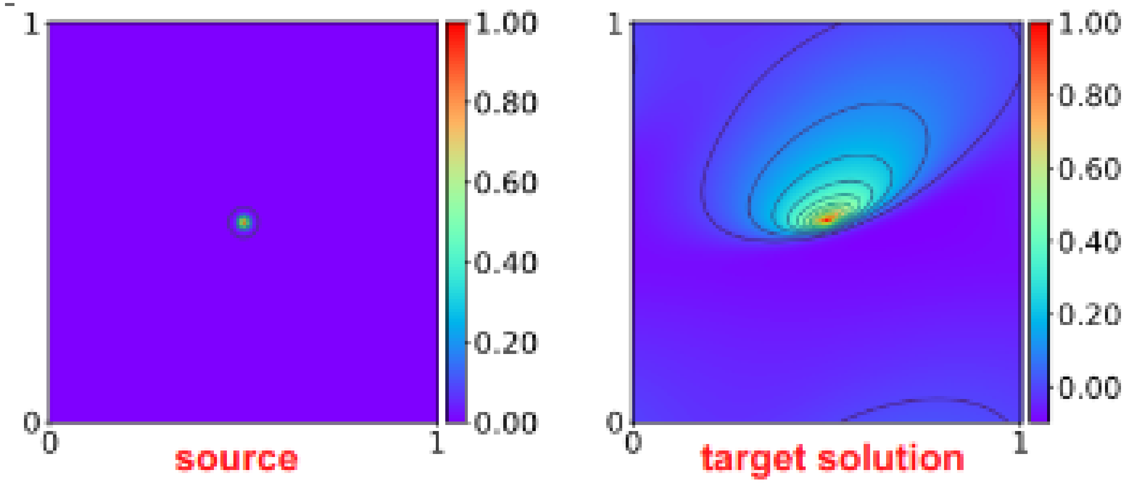
2D diffusion-advection equation

$$-\boldsymbol{v} \cdot \nabla u + \boldsymbol{K} \nabla u = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega.$$

Diffusion tensor: ***K***
Velocity vector: ***v***
Source function: ***f (x)***



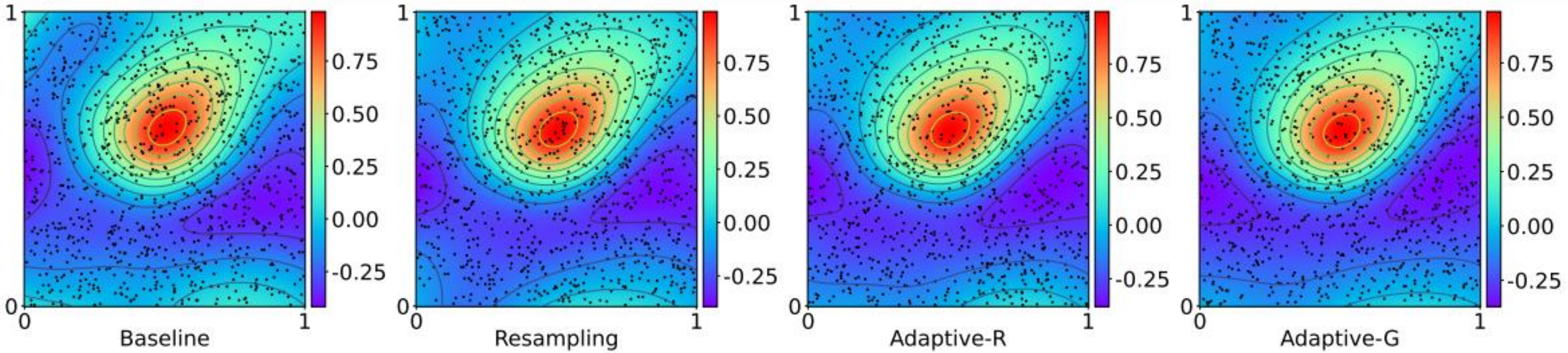
Smooth Source Function



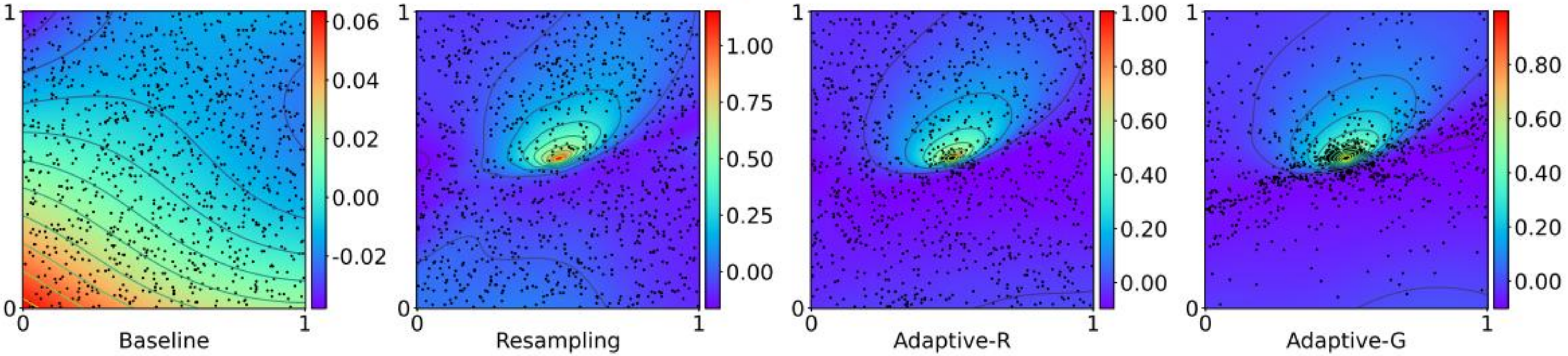
Sharp Source Function

Adaptive Self-supervision Algorithms for Physics-informed Neural Networks, arXiv, 2022

Collocation Point Sampling Strategy



Smooth Source



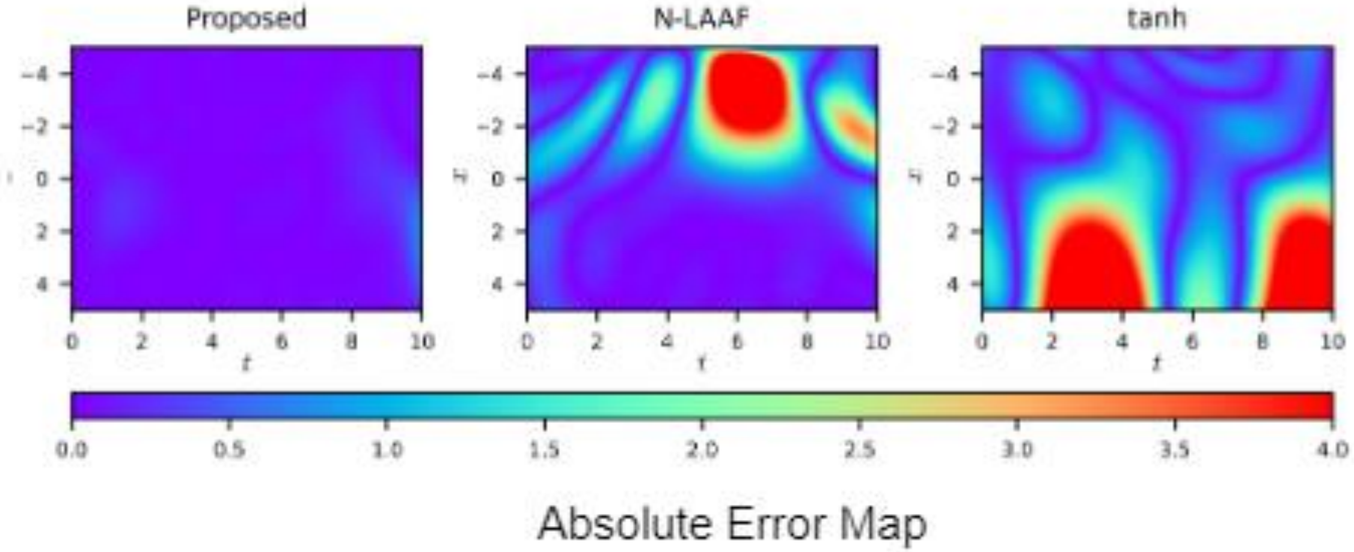
Sharp Source

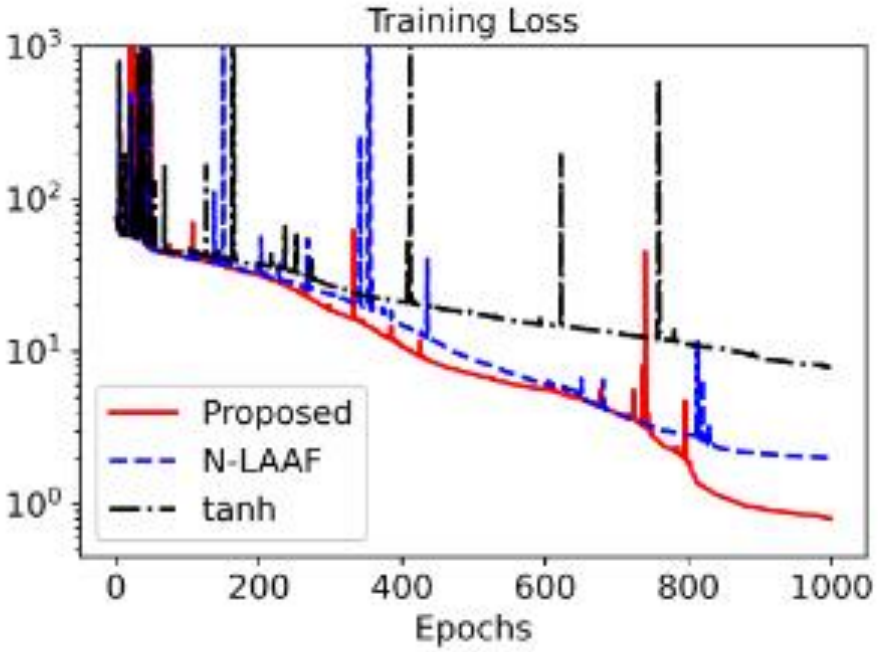
Vanishing Gradient and Output Magnitude Problem

Klein-Gordon Equation

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} + u^2 = -x\cos(t) + x^2\cos^2(t),\ \ x \in [-5,5], t > 0,$$

$$u(x,0) = x;\ \ u(-5,t) = -5\cos(t);\ \ u(5,t) = 5\cos(t).$$



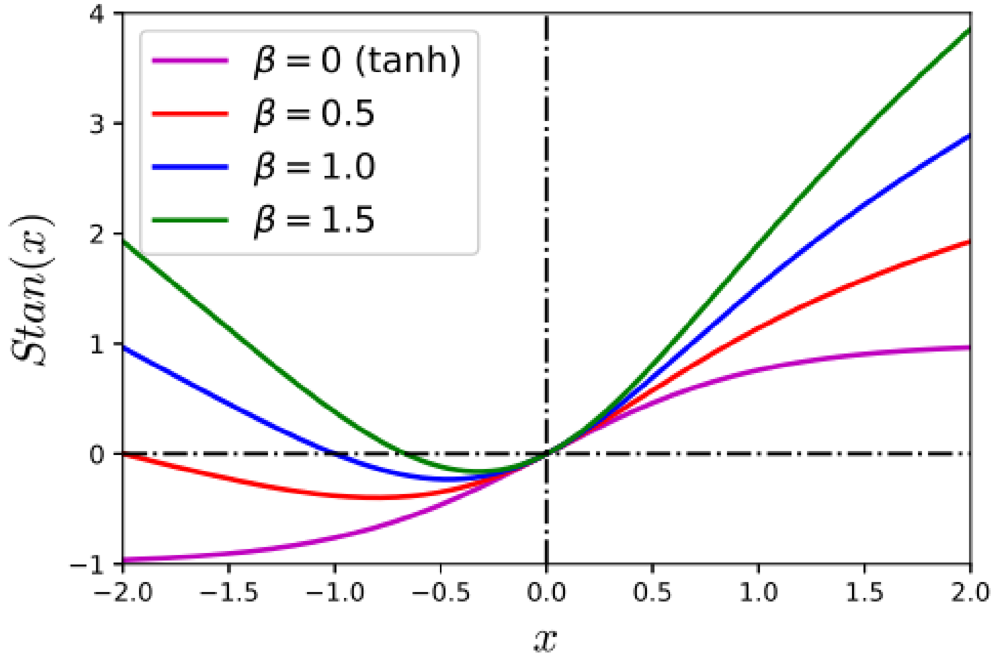


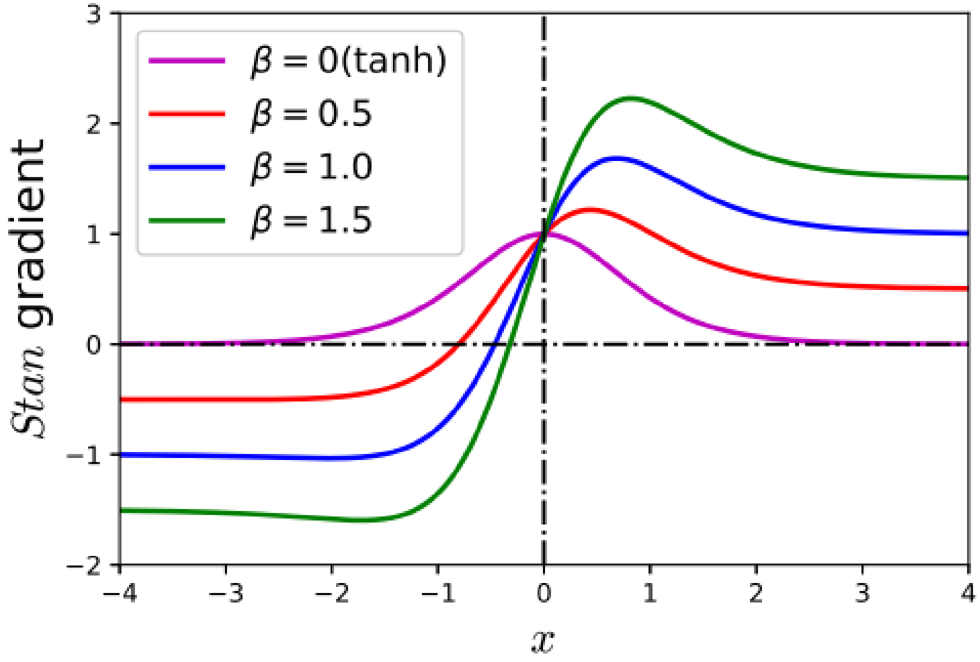
Self-scalable Tanh (Stan): Faster Convergence and Better Generalization in Physics-informed Neural Networks, arXiv, 2022

Solution Idea

$$\sigma_k^i(x) = \tanh(x) + \beta_k^i x \cdot \tanh(x)$$

still may face problem for higher order gradient



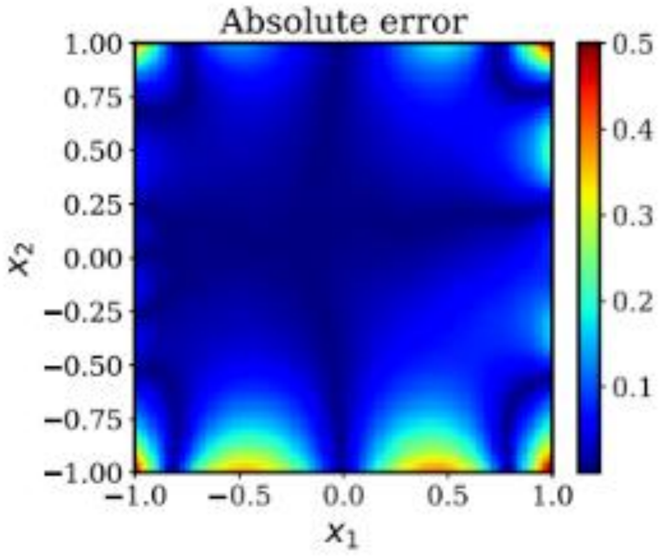


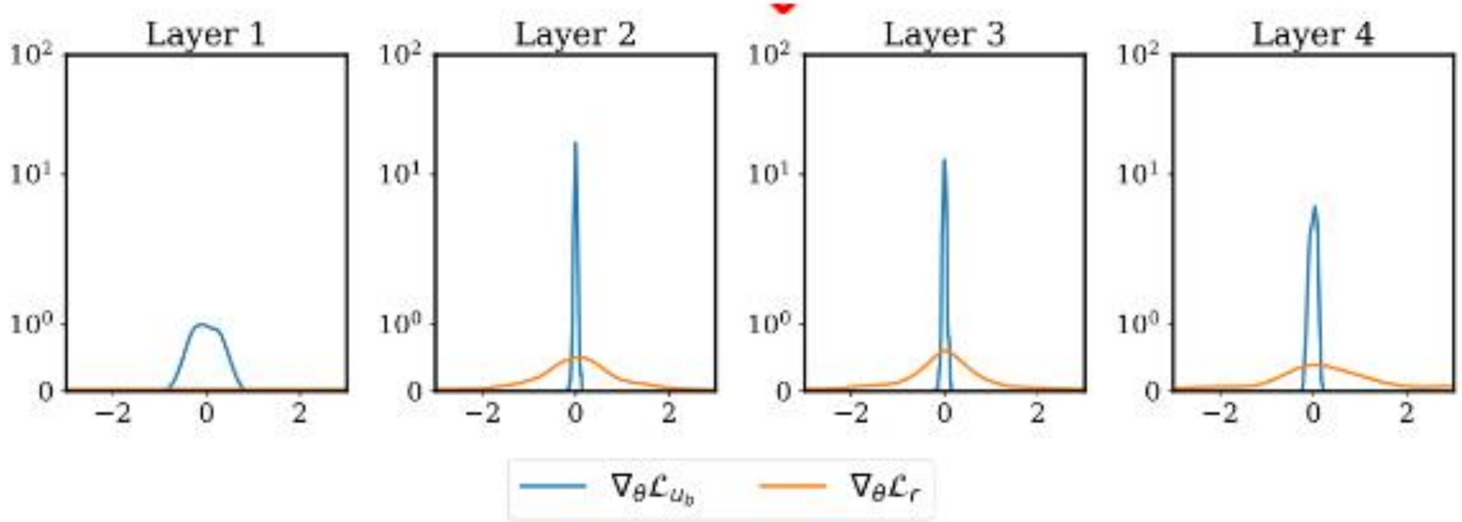
Boundary and Residual Loss Balancing Problem

2D Helmholtz equation

$$\Delta u(x,y) + k^2 u(x,y) = q(x,y), \quad (x,y) \in \Omega := (-1,1) \times (-1,1),$$
$$u(x,y) = h(x,y), \quad (x,y) \in \partial\Omega,$$

$$q(x,y) = -(a_1\pi)^2 \sin(a_1\pi x)\sin(a_2\pi y) - (a_2\pi)^2 \sin(a_1\pi x)\sin(a_2\pi y)$$
$$+ k^2 \sin(a_1\pi x)\sin(a_2\pi y),$$ k=1, a1=1, a2=4
$$h(x,y) = 0.$$





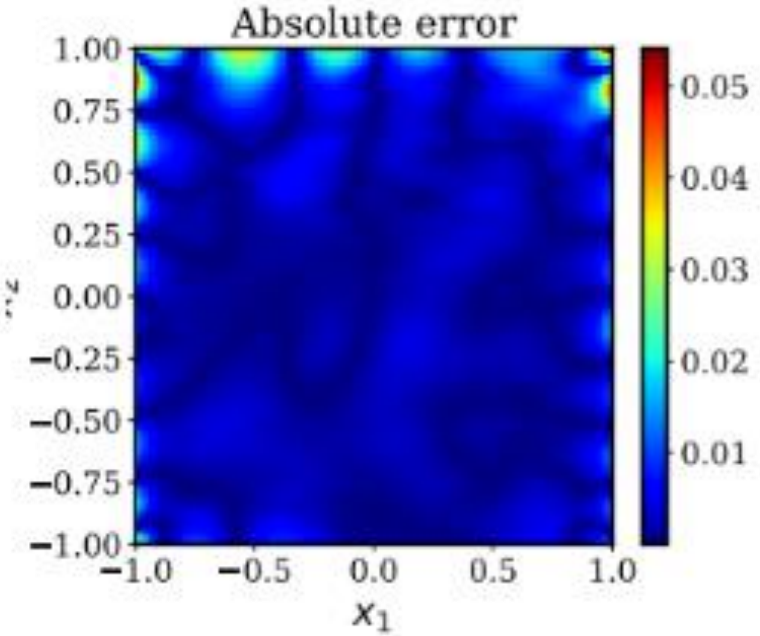
UNDERSTANDING AND MITIGATING GRADIENT FLOW PATHOLOGIES IN PHYSICS-INFORMED NEURAL NETWORKS, SIAM, 2021

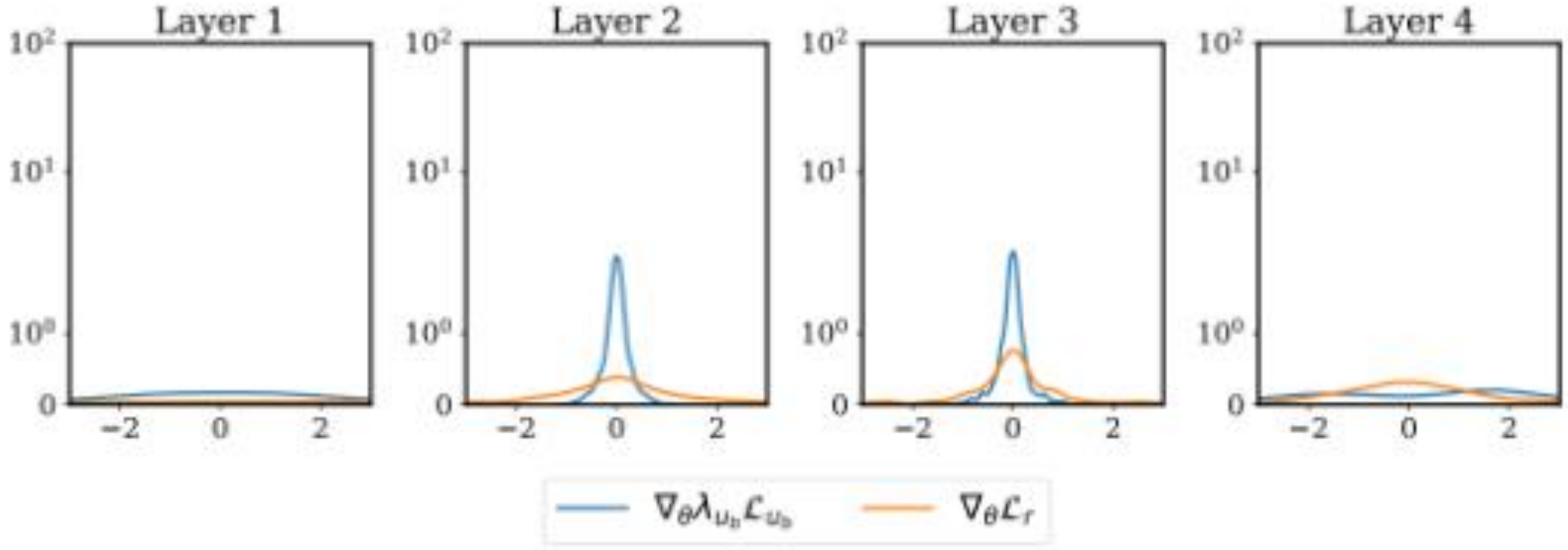
Proposed Solution

$$\mathcal{L}(\theta) := \mathcal{L}_r(\theta) + \sum_{i=1}^{M} \lambda_i \mathcal{L}_i(\theta)$$

$$\hat{\lambda}_i = \frac{\max_{\theta_n}\{|\nabla_\theta \mathcal{L}_r(\theta_n)|\}}{\overline{|\nabla_\theta \lambda_i \mathcal{L}_i(\theta_n)|}}, \quad i = 1, \dots, M,$$

$$\lambda_i = (1-\alpha)\underbrace{\lambda_i}_{\text{old}} + \alpha \underbrace{\hat{\lambda}_i}_{\text{new}}, \quad i = 1, \dots, M.$$





Thank You!