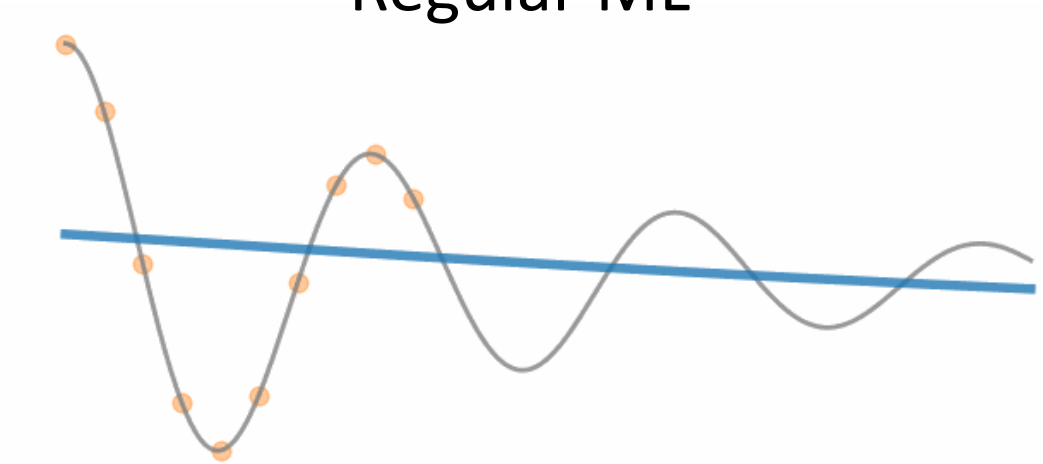
Physics-informed neural networks

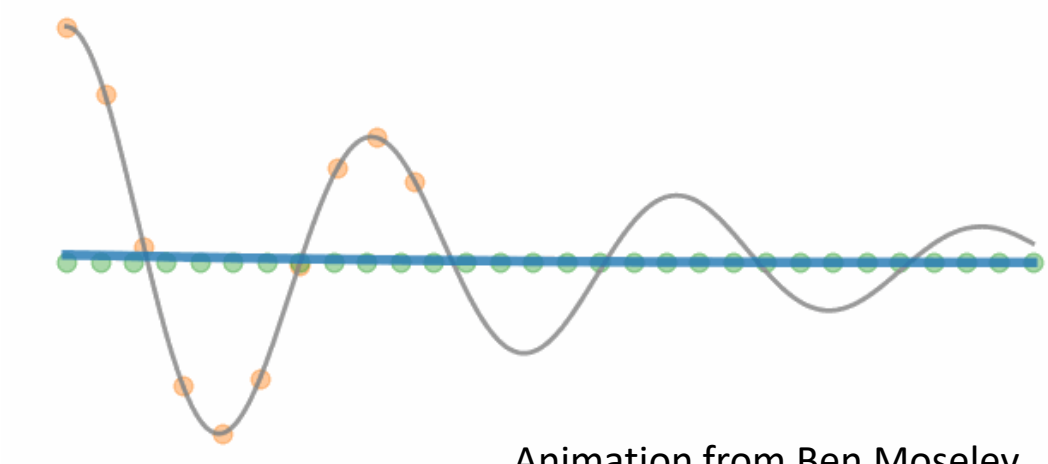
by C. Yao Lai for 2025 Princeton summer school

Better extrapolation beyond training data

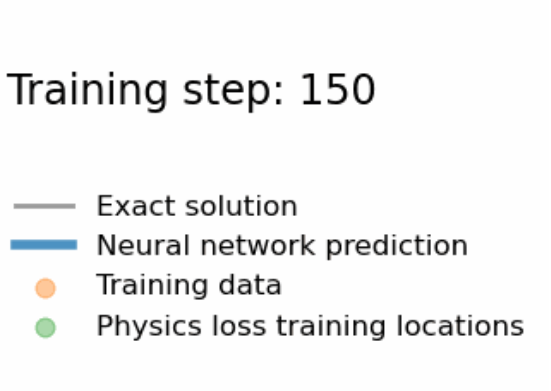
Regular ML



Physics-informed ML



Animation from Ben Moseley



Data loss: fit the data

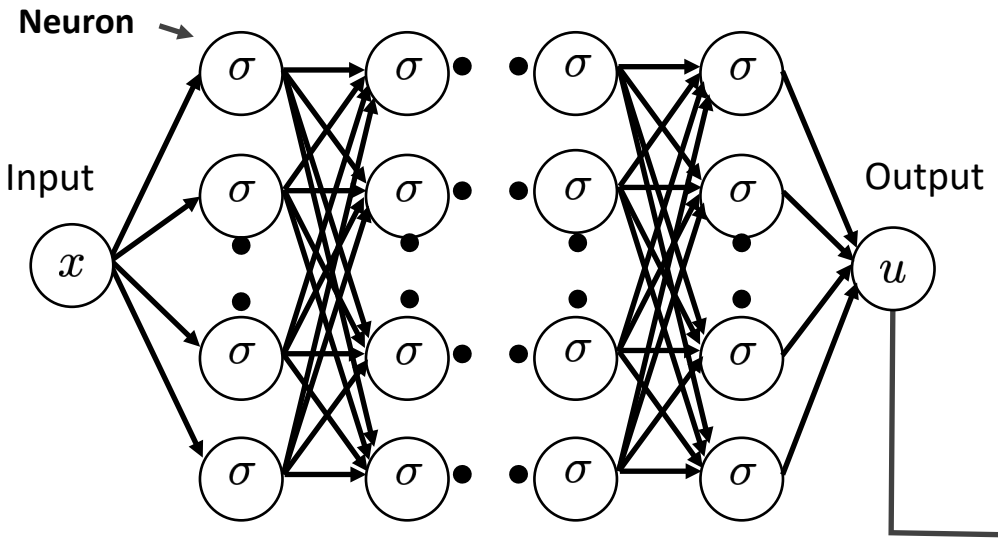
Eqn loss: fit the physics equations

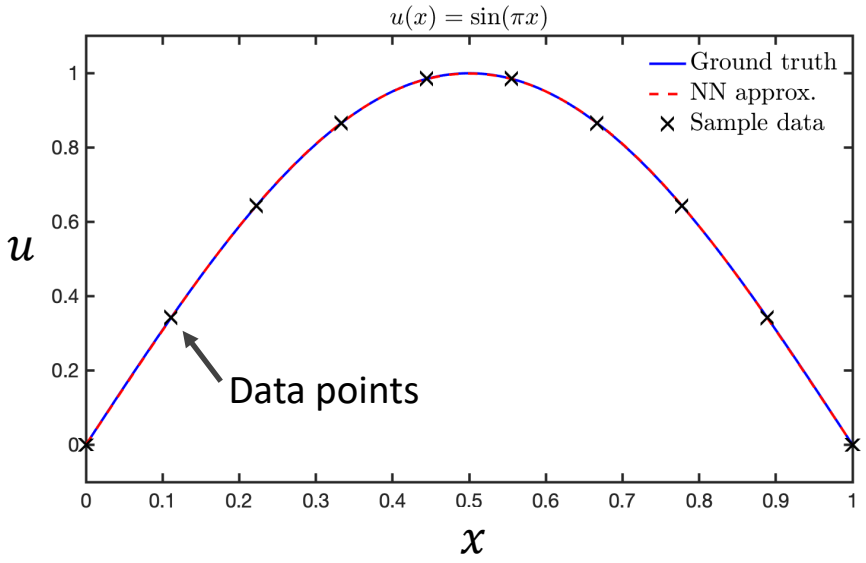
$$\text{Loss} = \text{Loss}_{\text{DATA}} + \text{Loss}_{\text{EQN}}$$

Neural network for curve fitting

Karniadakis *et. al.* (2021), *Nat. Rev. Phys.*

Fully-connected Neural network





$$u(x) = \sum_{j=1} w_{lk}^{(n)} \sigma \left(\dots \sigma \left(\sum_{i=1} w_{ji}^{(1)} \, \sigma \left(w_i^{(0)} x + b_i^{(0)} \right) + b_j^{(1)} \right) \dots \right) + b_l^{(n)}$$

Updating variables: $\mathbf{w}$: weights $\mathbf{b}$: biases

Optimization ← Loss

Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \, \nabla_{\mathbf{w}} J(x, \mathbf{w}^{(i)}, \mathbf{b}^{(i)})$$

$$\mathbf{b}^{(i+1)} = \mathbf{b}^{(i)} - \eta \, \nabla_{\mathbf{b}} J(x, \mathbf{w}^{(i)}, \mathbf{b}^{(i)})$$

$\mathbf{w}^{(i)}, \mathbf{b}^{(i)}$: value at the i-th iteration η : learning rate

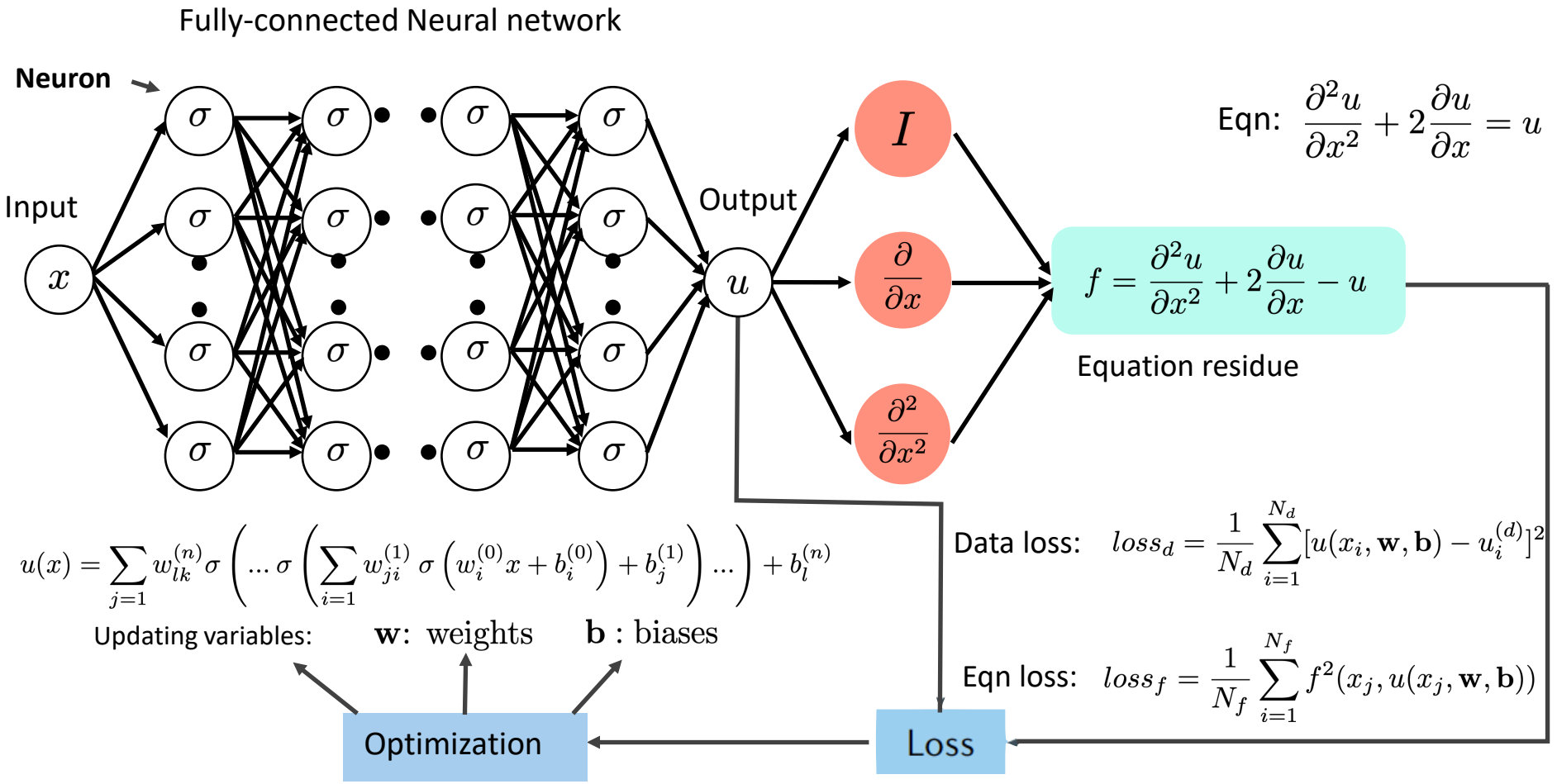
Cost function: *mean squared error*

data of u at $x = x_i$

$$J(x, \mathbf{w}, \mathbf{b}) = loss_d = \frac{1}{N_d} \sum_{i=1}^{N_d} [u(x_i, \mathbf{w}, \mathbf{b}) - u_i^{(d)}]^2$$

Physics-informed neural networks

Karniadakis *et. al.* (2021), *Nat. Rev. Phys.*



Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \nabla_{\mathbf{w}} J(x, \mathbf{w}^{(i)}, \mathbf{b}^{(i)})$$

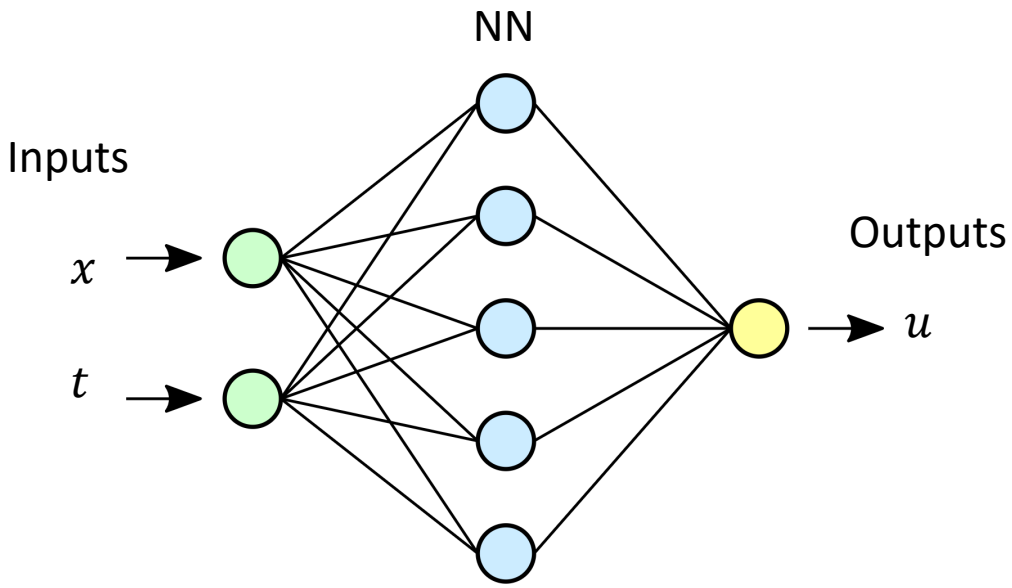
$$\mathbf{b}^{(i+1)} = \mathbf{b}^{(i)} - \eta \nabla_{\mathbf{b}} J(x, \mathbf{w}^{(i)}, \mathbf{b}^{(i)})$$

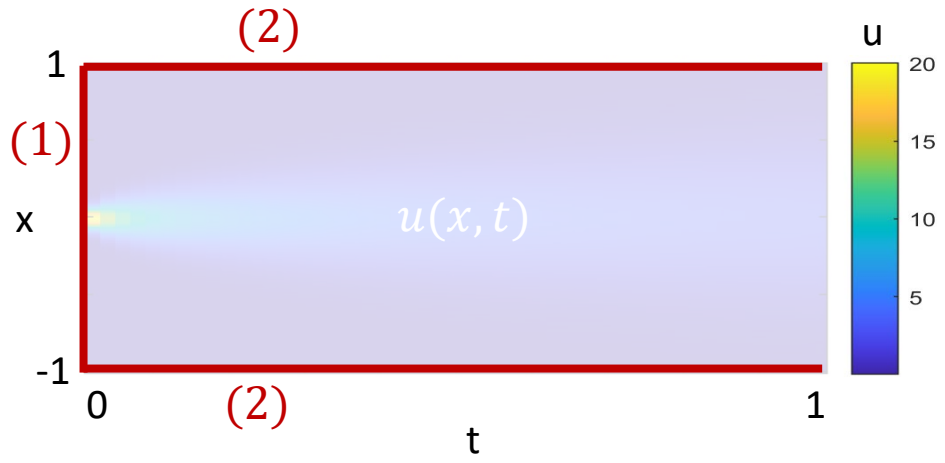
$\mathbf{w}^{(i)}, \mathbf{b}^{(i)}$: value at the i-th iteration η : learning rate

Cost function: *data + equation loss*

$$J(x, \mathbf{w}, \mathbf{b}) = loss_d + loss_f$$

Learn u with physics equation + ICs + BCs





Use NN to search for a surface that satisfies (1) ICs + (2) BCs + (3) physics equation.

1. Initial NN(x,t) = u(x,t). x,t are inputs, u is output

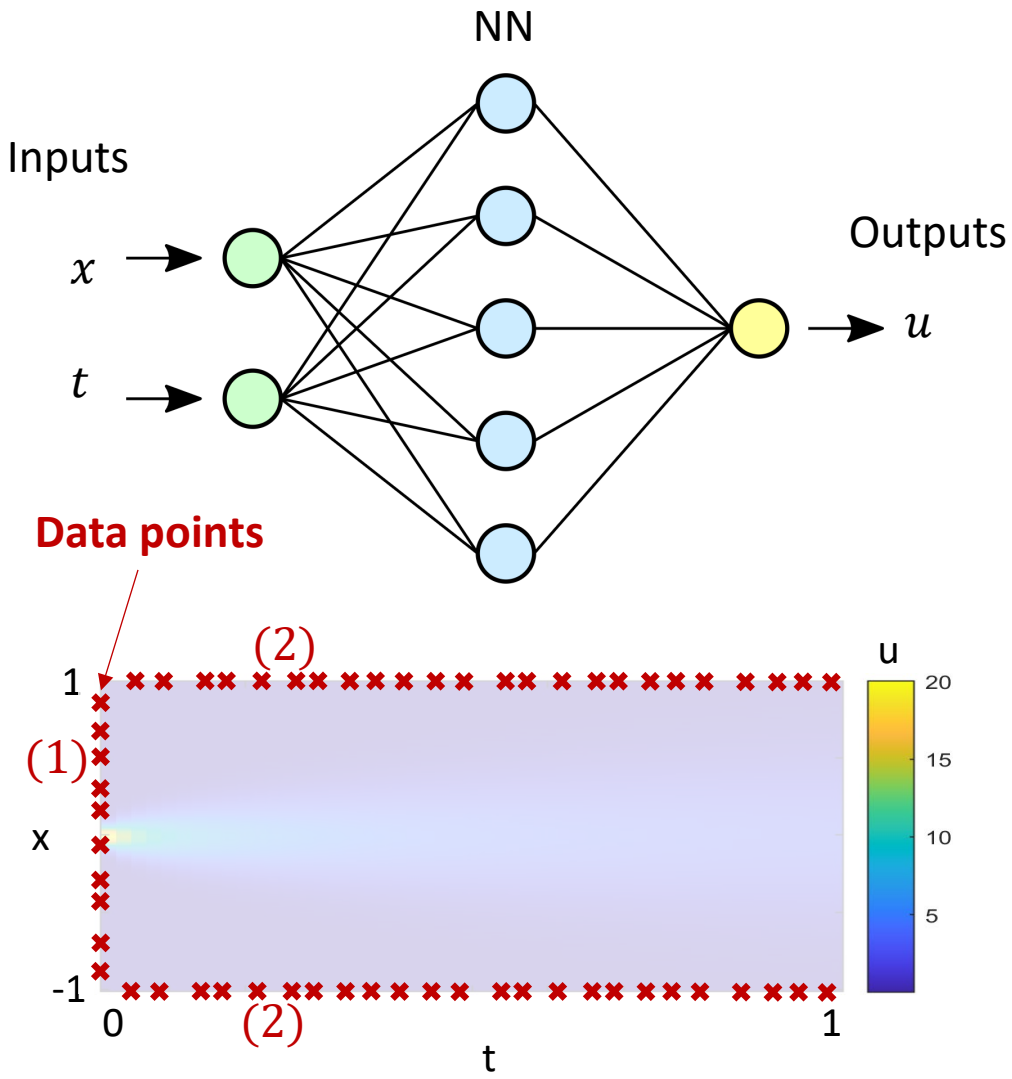
2. For a NN, all derivatives of u w.r.t x and t can be calculated analytically because u(x,t) is exact!

3. We can calculate $\frac{\partial u}{\partial t}, \frac{\partial^2 u}{\partial x^2}$

4. In the cost function, minimize $\left[\frac{\partial u}{\partial t} - a\frac{\partial^2 u}{\partial x^2}\right]^2$

Turn the problem into an optimization problem!

Physics-informed NN



Training data (ground truth):

(1) *Initial condition:*

$$u_0^i \; at \; \{t = 0, x_i\}, \; i = 1 \ldots . N$$

(2) *Boundary conditions*:

$$u_{lb}^j \; at \; \{t_j, x = -1\} \;\; \& \;\; u_{ub}^j \; at \; \{t_j, x = 1\}, j = 1 \ldots . M$$

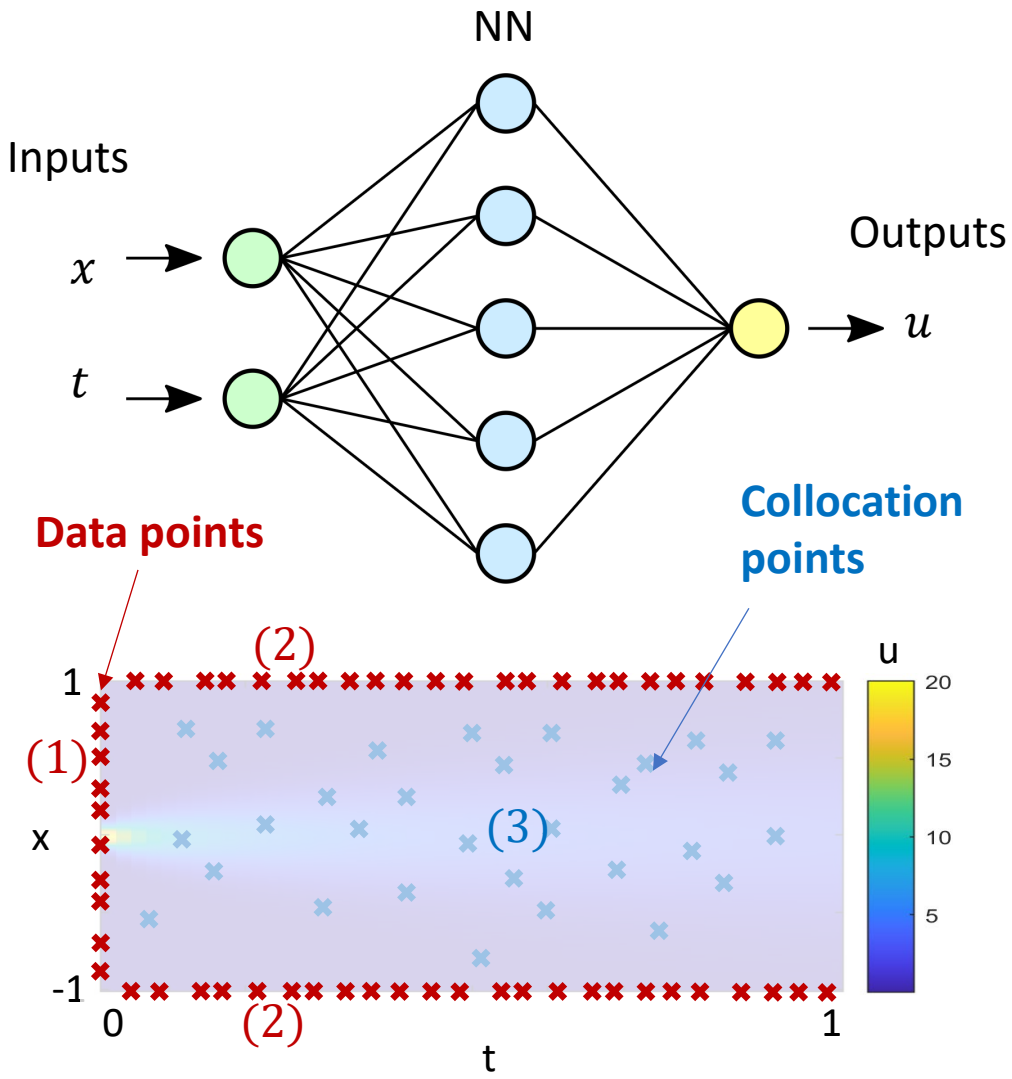
What would be the loss function?

Data loss: minimize $\frac{1}{N}\sum_i^N \left|u_0^i - u_{pred}(t = 0, x_i)\right|^2$ (1) IC

minimize $\frac{1}{M}\sum_j^M \left|u_{lb}^j - u_{pred}(t_j, x = -1)\right|^2 + \frac{1}{M}\sum_j^M \left|u_{ub}^j - u_{pred}(t_j, x = 1)\right|^2$ (2) BC

GOAL: Find NN that minimizes the loss function

Physics-informed NN



Q: How to incorporate physics equation in the loss function?

$$(3)\quad \frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} \longrightarrow \frac{\partial u}{\partial t} - a\frac{\partial^2 u}{\partial x^2} = 0$$

Recall: NN(x,t) = u(x,t) is a smooth, analytical function

$\frac{\partial u}{\partial t}, \frac{\partial^2 u}{\partial x^2}$ can be directly calculated at the collocation points.

What would be the loss function?

Equation loss:

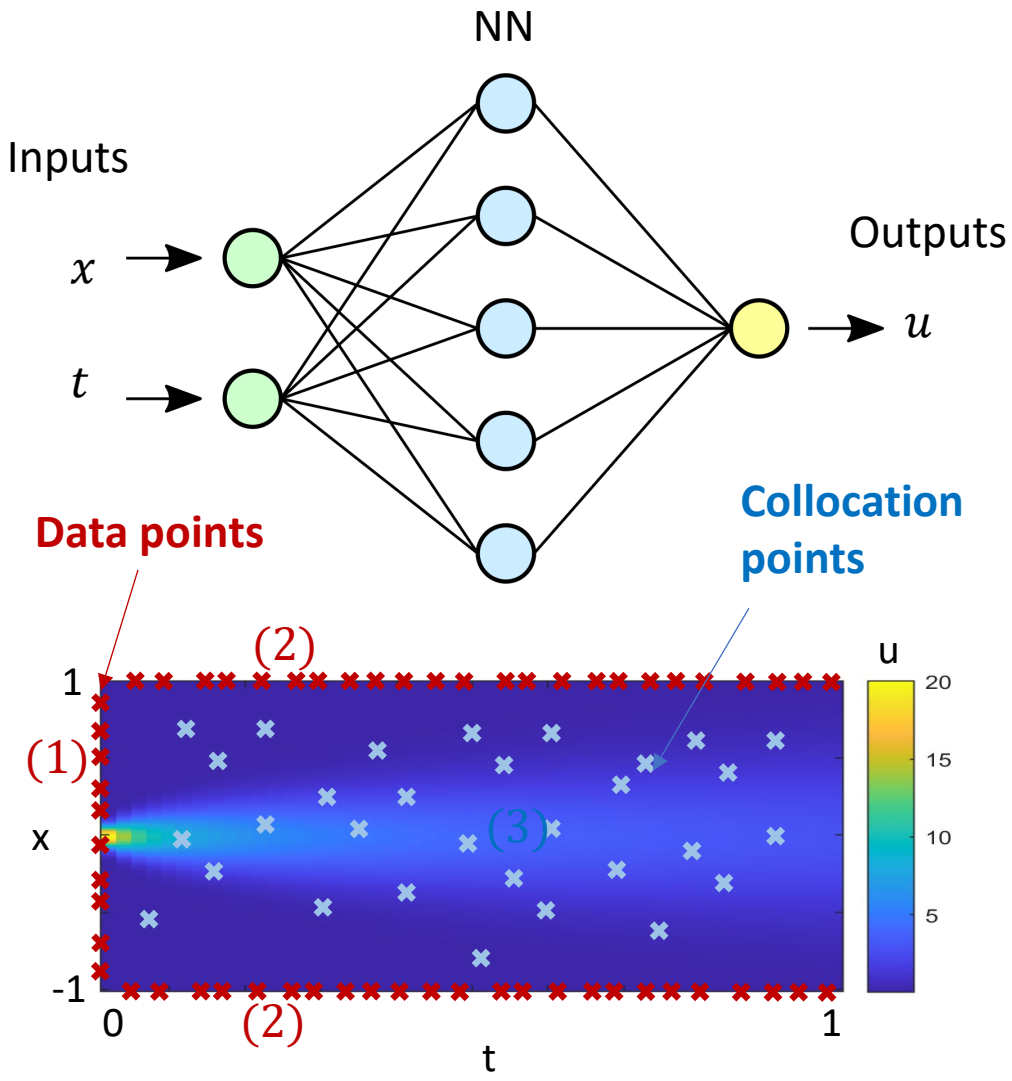
minimize $$\frac{1}{N_f}\sum_{k}^{N_f}\left|\frac{\partial u_{pred}^k}{\partial t} - a\frac{\partial^2 u_{pred}^k}{\partial x^2}\right|^2$$ (3) Eqn

GOAL: Find NN that minimizes the loss function

→ *solving for* $u(x,t)$ *satisfying* **(1) ICs + (2) BCs + (3) Eqn**

Raissi et al (2019)

Physics-informed NN



Given a partial differential equation of a general form:

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

where $\mathcal{N}[\cdot]$ is a nonlinear differential operator.
Define equation residue f as

$$f := u_t + \mathcal{N}[u]$$

Cost function: (MSE: mean squared error)

$$MSE = \underbrace{MSE_u}_{\text{Data loss}} + \underbrace{MSE_f}_{\text{Equation loss}},$$

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

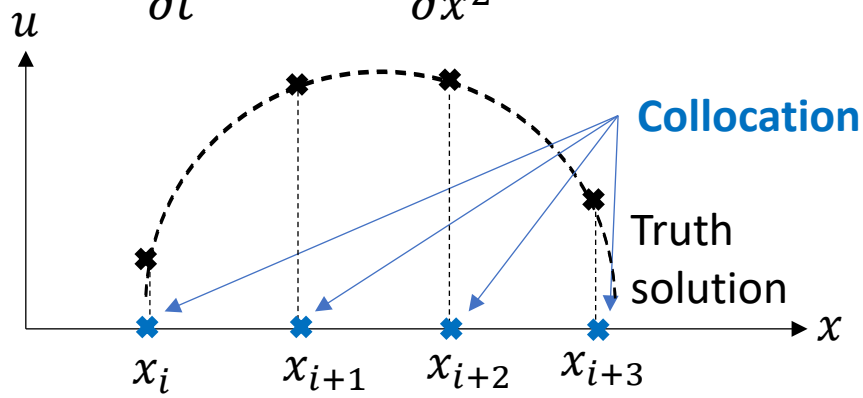
Data points

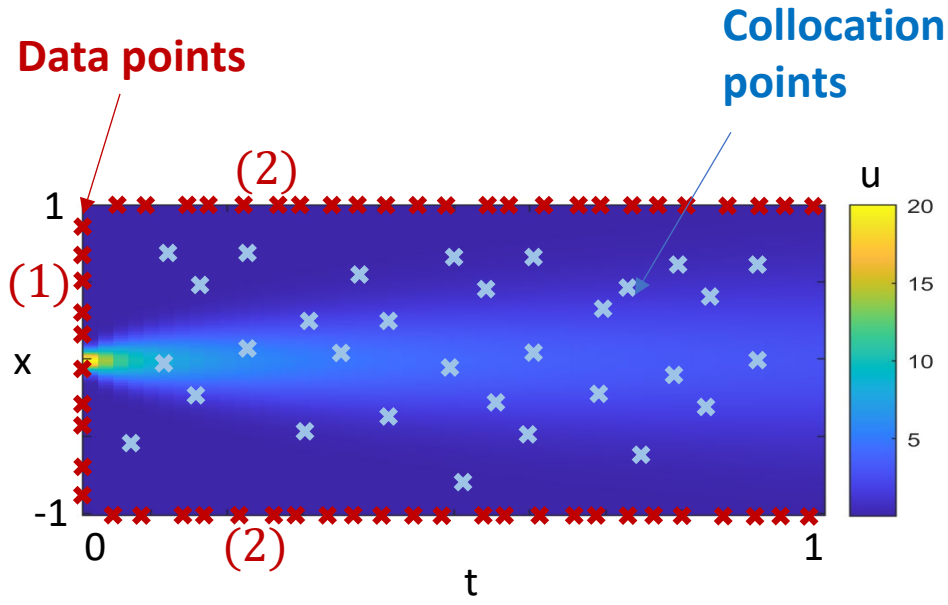
$$\frac{1}{N_f}\sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2$$

Collocation points

PINN: searches for a curve that satisfies

$$\frac{\partial u(x_i)}{\partial t} - a\frac{\partial^2 u(x_i)}{\partial x^2} \approx 0 \text{ at the collocation pts}$$

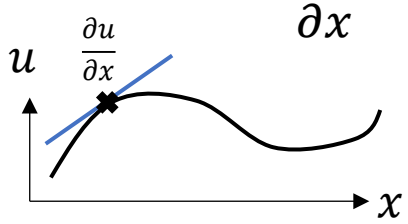




Automatic differentiation: differentiate NN output with respect to their input coordinates.

e.g. $\mathrm{NN(x)} = u = \sigma(w_2\sigma(w_1 x + b_1) + b_2)$,

$$\frac{\partial u}{\partial x} = \sigma'(z_2)w_2\sigma'(z_1)w_1$$



Raissi et al (2019)

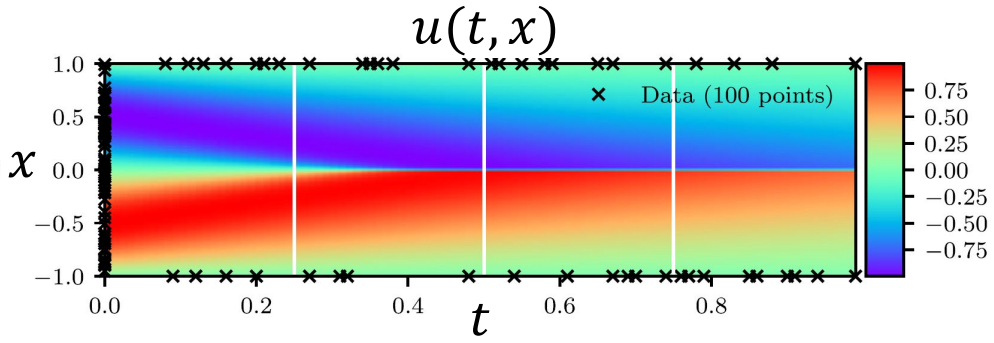
E.g., Burgers' equation

Problem statement

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

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

$$BC: \; u(t, -1) = u(t, 1) = 0.$$



Training data (from ground truth):

$$\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u} \qquad N_u = 100$$

Collocation points:

$$\{t_f^i, x_f^i\}_{i=1}^{N_f} \qquad N_f = 10{,}000$$

Physics equations:

$$f := u_t + uu_x - (0.01/\pi)u_{xx}$$

Loss function:

Data loss

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

Data points

Equation loss

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

Collocation points

Raissi et al (2019)

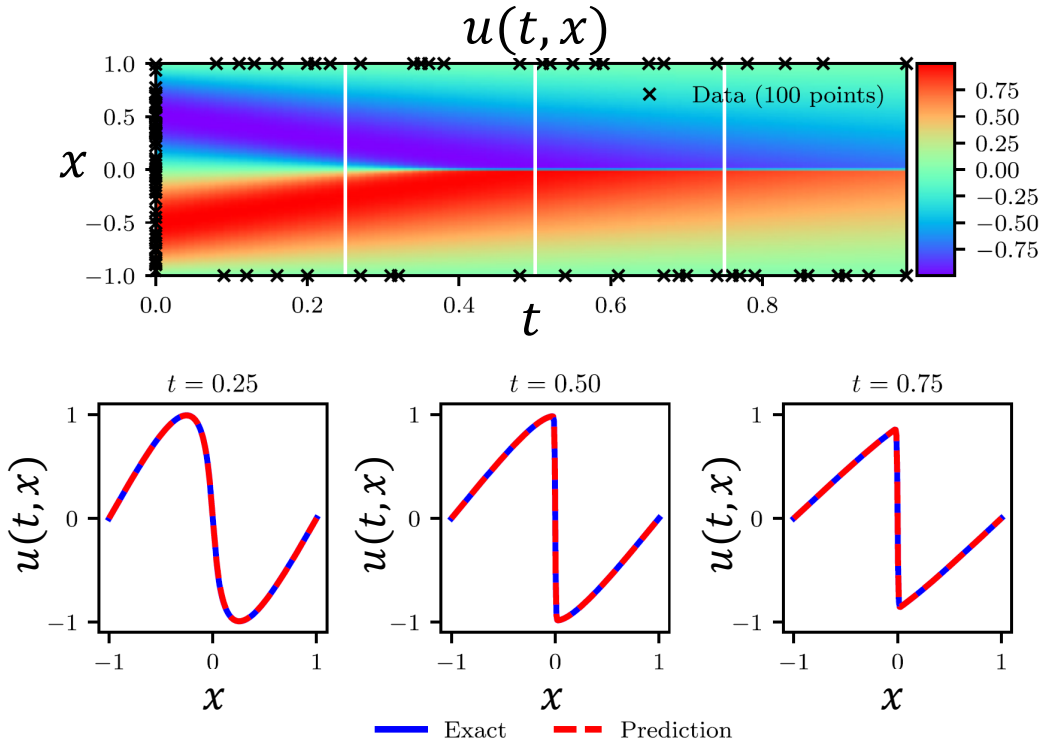
E.g., Burgers' equation

Problem statement

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

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

$$BC: \; u(t, -1) = u(t, 1) = 0.$$



Training data (from ground truth):

$$\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u} \qquad N_u = 100$$

Collocation points:

$$\{t_f^i, x_f^i\}_{i=1}^{N_f} \qquad N_f = 10{,}000$$

Physics equations:

$$f := u_t + uu_x - (0.01/\pi)u_{xx}$$

Loss function:

Data loss

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

Data points

Equation loss

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

Collocation points

Activation function: tanh or sin, cos?

Ex1: Sin activation, Iter: 4800

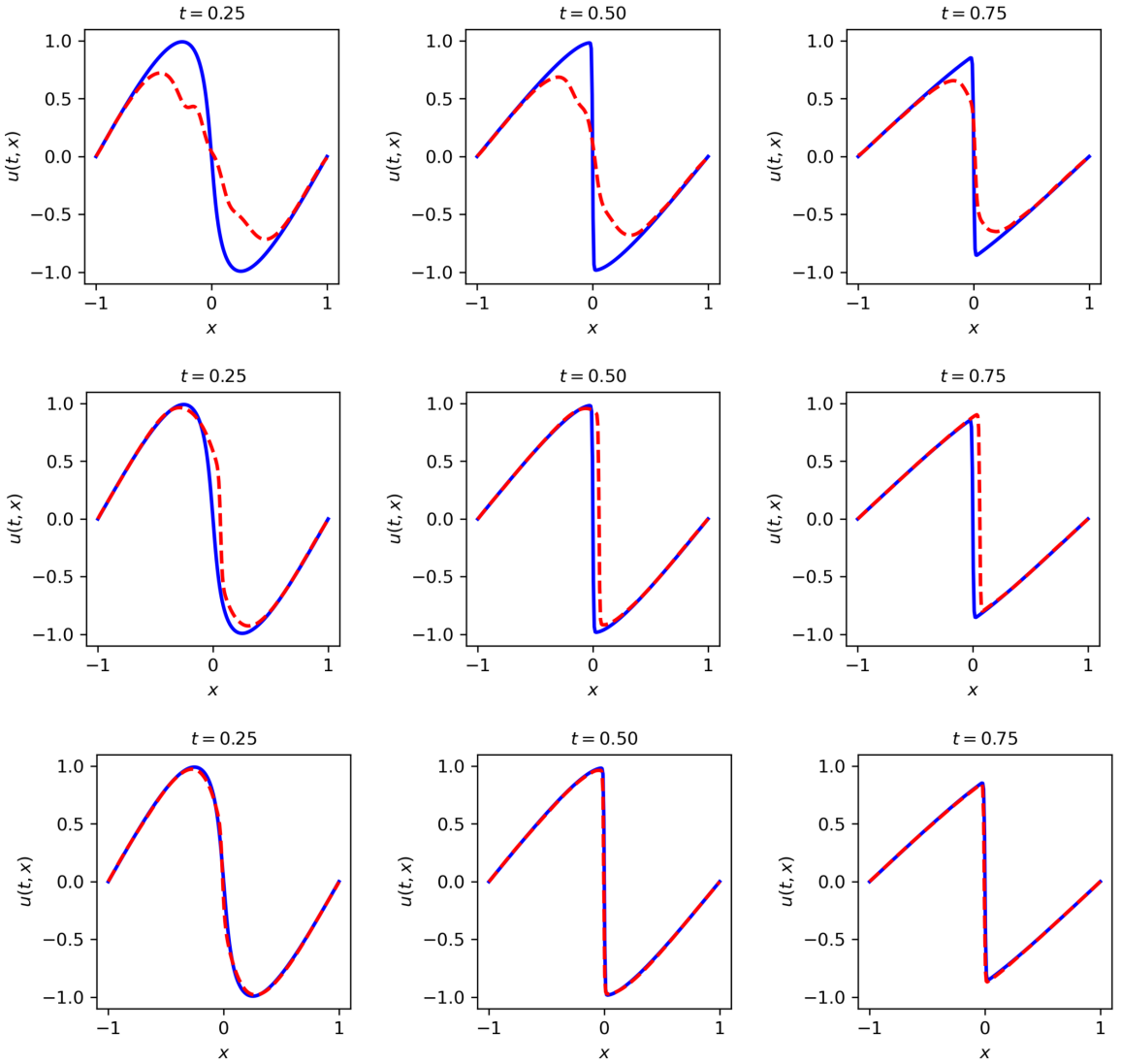
loss: 0.0319006
Training time: 173.6166 Error u: 3.554990e-01

Ex2: Sin activation, Iter: 24680

loss: 0.0029452811
Training time: 1126.4058 Error u: 3.946169e-01

Ex3: Tanh activation, Iter: 4800

loss: 0.000854328566
Training time: 163.6977 Error u: 6.638371e-02



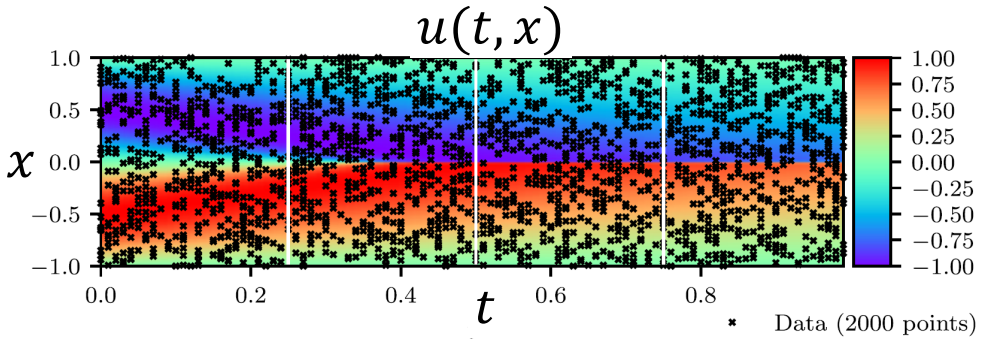
Raissi et al (2019)

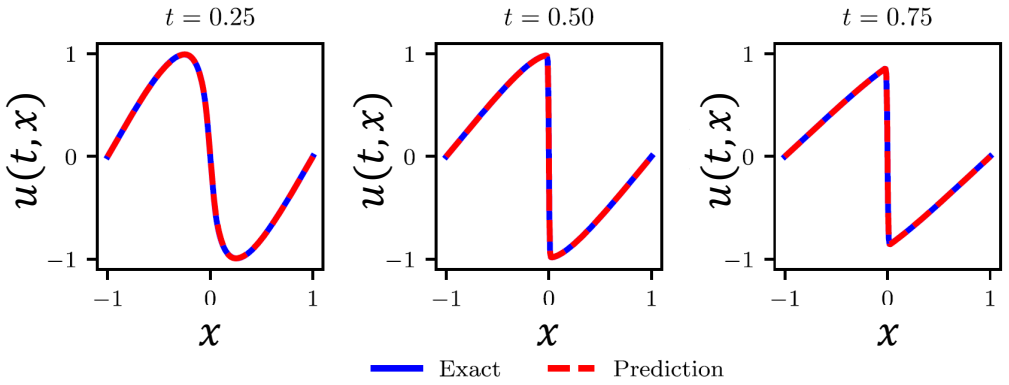
E.g., Burgers' equation

Given training data of u, t, x find λ_1, λ_2

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

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





Training data (from ground truth):

$$\{t_u^i, x_u^i, u^i\}_{i=1}^N \qquad N = 2{,}000$$

Collocation points:

$$\{t_u^i, x_u^i\}_{i=1}^N \qquad N = 2{,}000$$

Physics equations:

$$f := u_t + \lambda_1 u u_x - \lambda_2 u_{xx}$$

Loss function:

Data loss

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

Data points

Equation loss

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

Collocation points

What would happen if using ReLU activation?

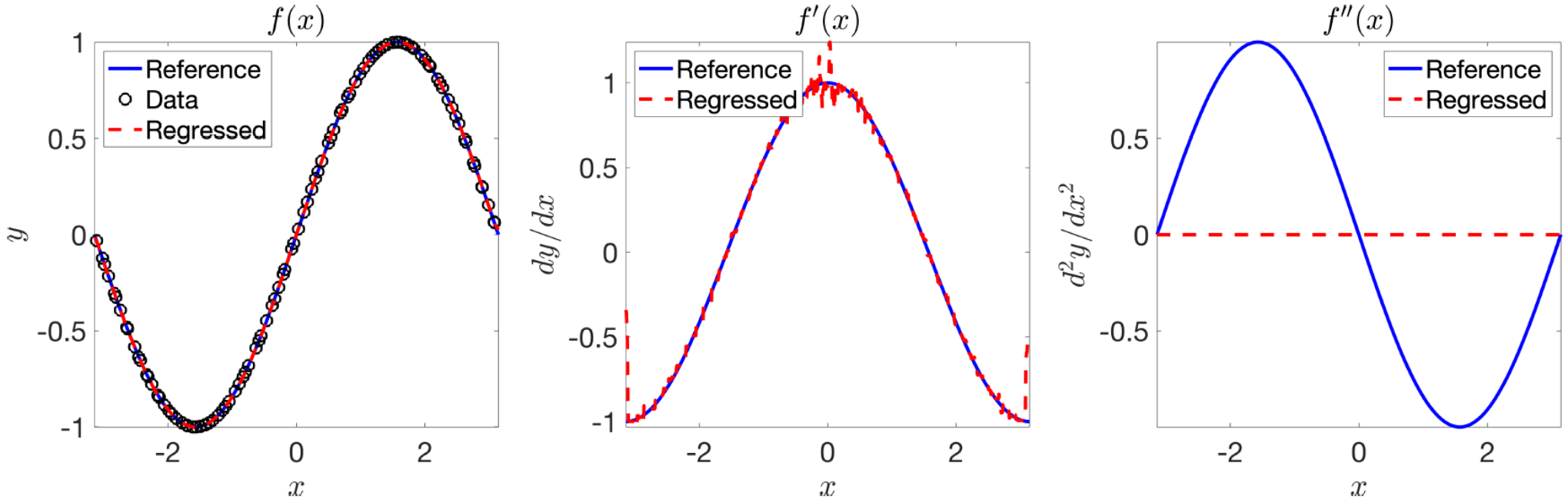


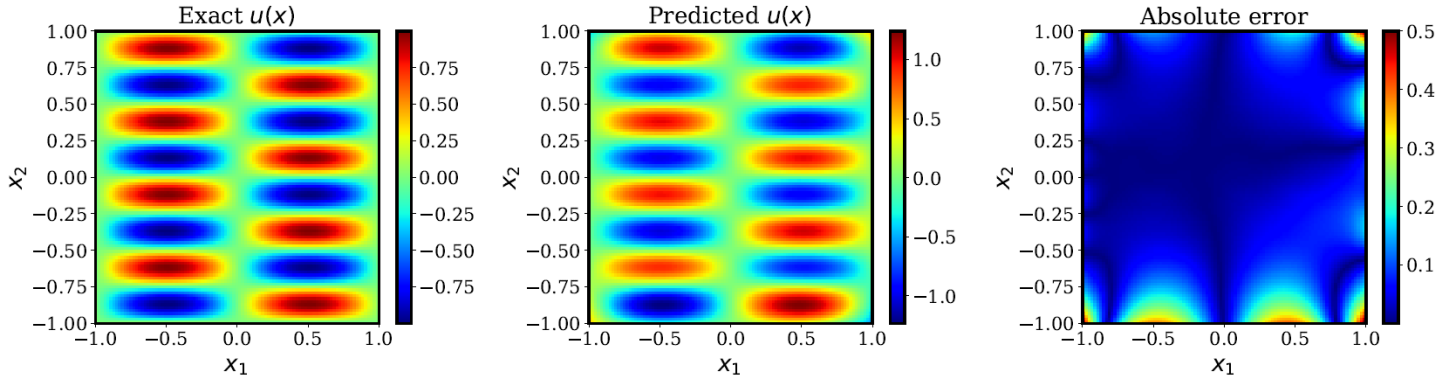
Fig. S2. Regressing data generated from $f(x) = \sin(x)$ using the ReLU activation function. A neural network with ReLU activation can accurately approximate $f(x)$ (left panel), while it leads to a non-smooth regression for the first derivative of f (middle panel) and a vanishing second derivative of f (right panel).

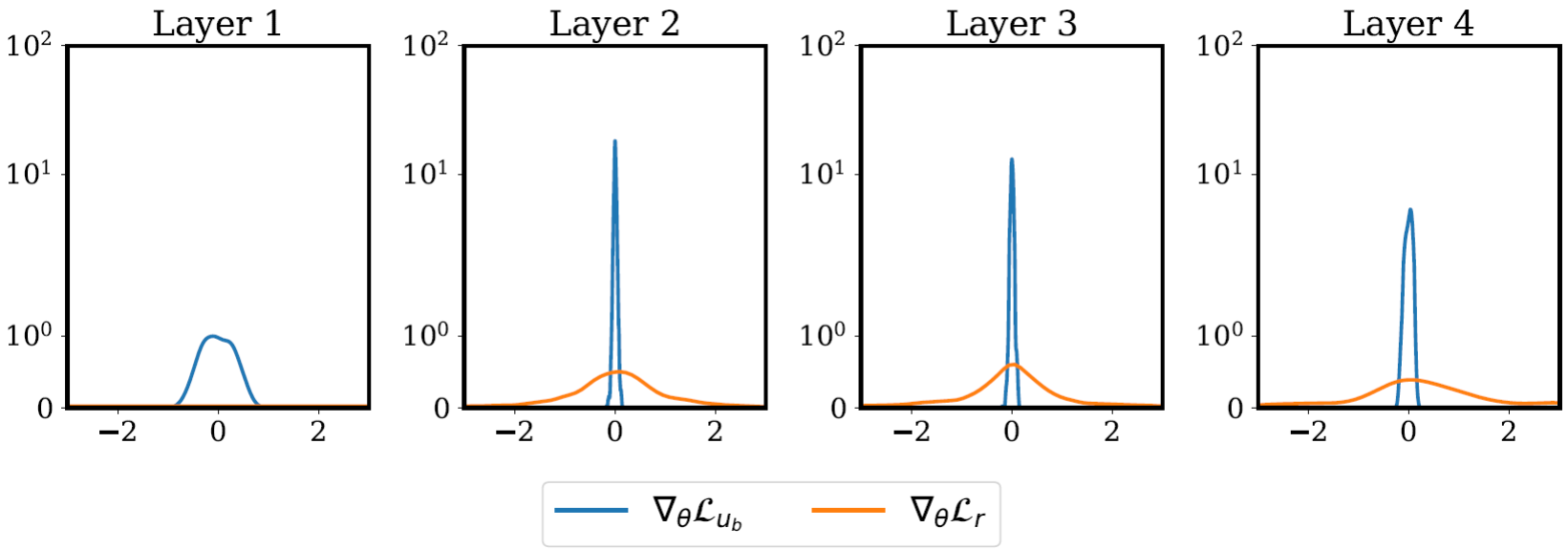
How to balance different terms in the loss

Learning rate annealing: Wang et al. SIAM J. Sci. Comput. (2021)

$$\mathcal{L}(\theta) = \mathcal{L}_r(\theta) + \mathcal{L}_{u_b}(\theta)$$

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



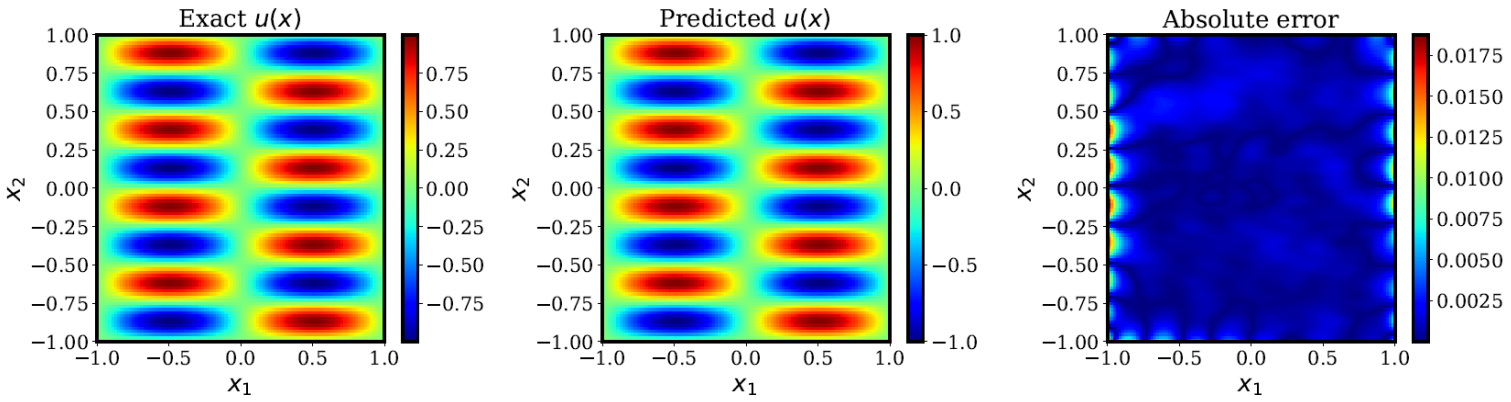


Learning rate annealing: Wang et al. SIAM J. Sci. Comput. (2021)

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

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

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}_r(\theta_n) - \eta \sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta_n)$$



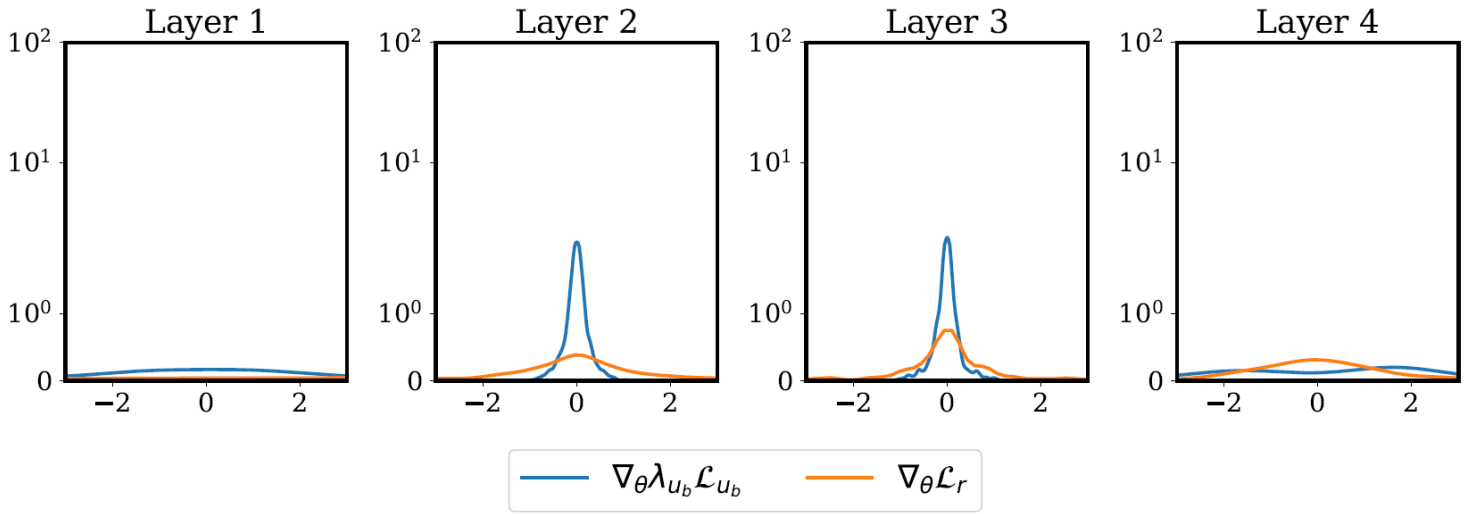


FIG. 7. Helmholtz equation: *Histograms of back-propagated gradients of* $\nabla_\theta \mathcal{L}_r$ *and* $\nabla_\theta \mathcal{L}_{u_b}$ *at each layer at the end of training a PINN with the proposed Algorithm* 2.1 *to solve the Helmholtz equation.*

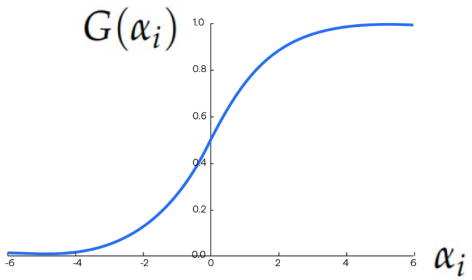
Adaptive Blending Unit ABU-PINN, Wang et al 2023 https://arxiv.org/pdf/2308.04073.pdf

Learning Activation Functions for PINN

$$f(x) = \sum_{i=1}^{N} G(\alpha_i)\sigma_i(\beta_i x).$$

where $\sigma_i(\cdot)$ and α_i denote a candidate activation function and a learnable parameter learnable scaling factor β_i and $G(\alpha_i)$ is a softmax function

Softmax Function



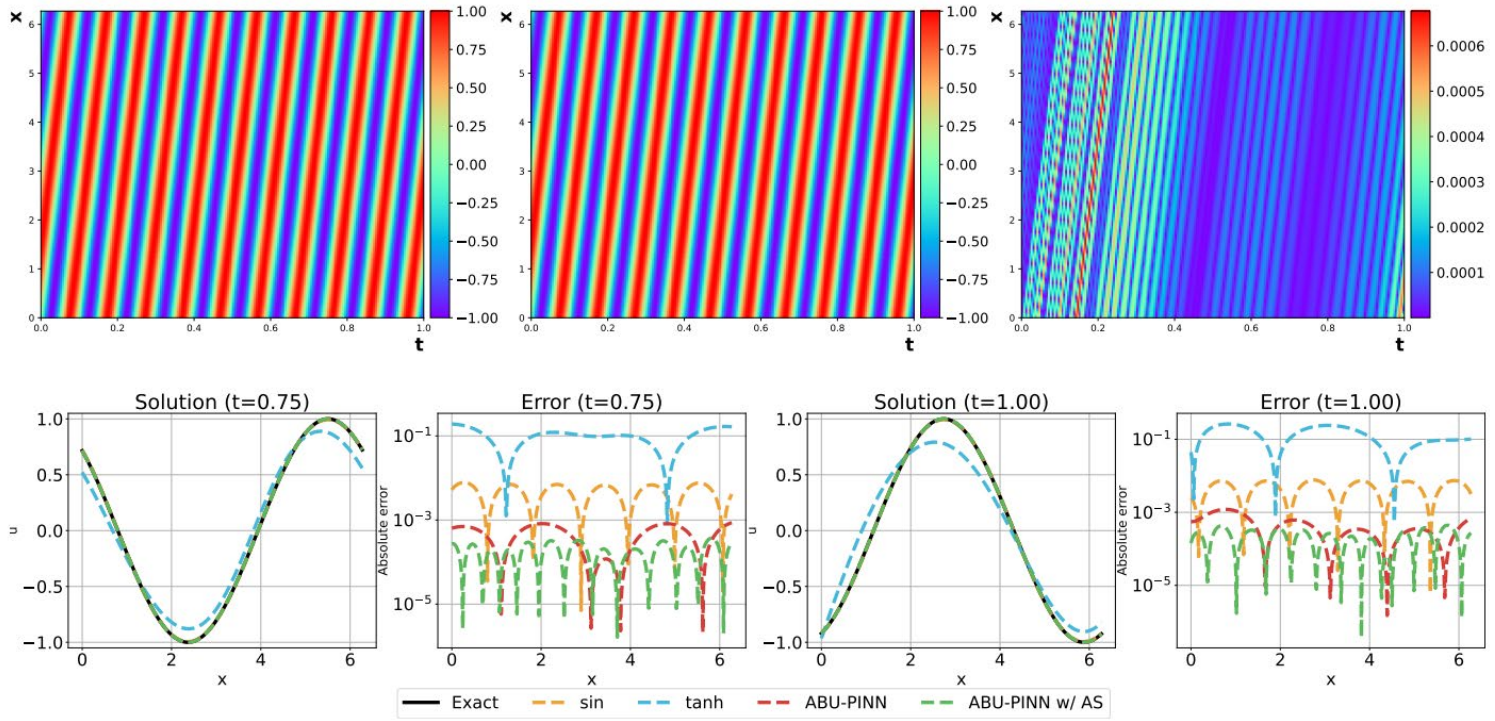


Figure 2: The convection equation. Top: the exact solution (left), the predictions of ABU-PINN (middle), and the absolute error between them (right). Bottom: predicted solutions at different time snapshots.

Adaptive Blending Unit ABU-PINN, Wang et al 2023 https://arxiv.org/pdf/2308.04073.pdf

Learning Activation Functions for PINN

$$f(x) = \sum_{i=1}^{N} G(\alpha_i)\sigma_i(\beta_i x).$$

where $\sigma_i(\cdot)$ and α_i denote a candidate activation function and a learnable parameter learnable scaling factor β_i and $G(\alpha_i)$ is a softmax function

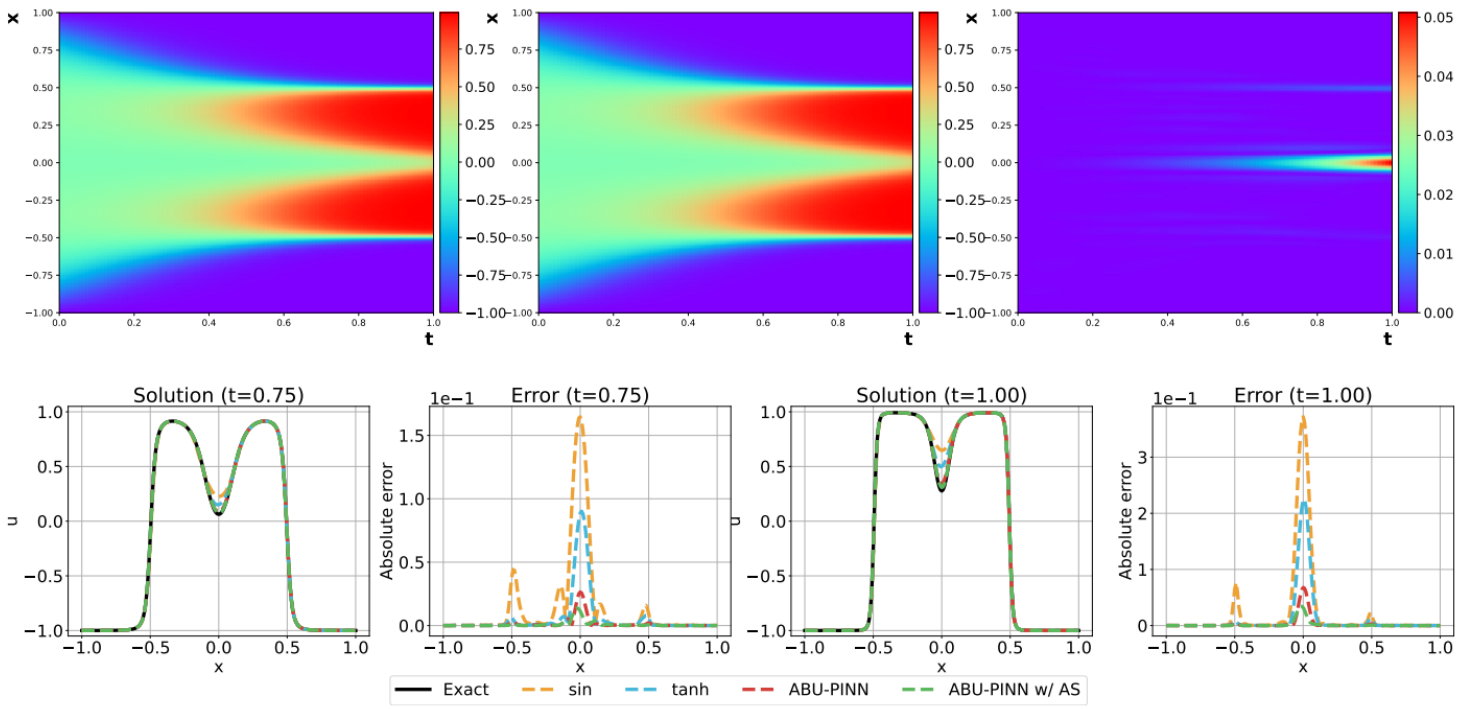


Figure 3: The Allen-Cahn equation. Top: the exact solution (left), the predictions of ABU-PINN (middle), and the absolute error between them (right). Bottom: predicted solutions at different time snapshots.

Adaptive Blending Unit ABU-PINN, Wang et al 2023 https://arxiv.org/pdf/2308.04073.pdf

Learning Activation Functions for PINN

$$f(x) = \sum_{i=1}^{N} G(\alpha_i)\sigma_i(\beta_i x).$$

where $\sigma_i(\cdot)$ and α_i denote a candidate activation function and a learnable parameter learnable scaling factor β_i and $G(\alpha_i)$ is a softmax function

Table 5: Comparisons of standard activation functions, adaptive activation functions and their counterparts with adaptive slopes (AS) on 1D time-dependent PDEs. We report the L_2 relative error (%) for each problem and the average error rate over all problems. The better results are **bold-faced**.

Method	Convection equation	Burgers' equation	Allen-Cahn equation	KdV equation	Cahn-Hilliard equation	Average error
sin	0.36±0.15	5.18±3.73	3.57±0.64	0.63±0.17	1.72±0.61	2.29
tanh	6.83±4.79	0.26±0.13	1.34±0.54	1.32±1.12	4.02±4.56	2.75
sigmoid	70.38±2.98	1.24±1.05	1.63±0.13	2.34±0.53	3.12±2.72	15.74
GELU	39.29±35.51	4.43±2.87	3.93±0.78	1.21±0.41	1.01±1.27	9.97
Swish	5.59±2.18	8.27±4.35	5.56±1.28	1.73±0.10	2.22±2.60	4.67
Softplus	55.39±2.46	17.75±8.11	17.72±6.04	6.23±0.44	9.67±2.57	21.35
ELU	6.67±0.96	46.34±2.36	52.55±2.95	78.95±2.57	90.77±2.07	55.66
SLAF	0.36±0.18	43.93±0.66	33.93±9.97	25.23±1.28	52.57±27.93	31.20
PAU	45.78±35.47	48.31±8.21	43.83±15.18	68.11±13.49	115.59±3.79	64.32
ACON	3.55±1.66	1.18±1.55	3.88±1.82	1.52±0.35	2.46±1.96	2.52
ABU-PINN	**0.06±0.03**	**0.21±0.11**	**0.76±0.26**	**0.34±0.05**	**0.50±0.15**	**0.37**
With AS						
sin	0.28±0.08	1.82±1.58	3.57±0.46	0.57±0.19	1.73±0.73	1.59
tanh	3.06±1.65	0.16±0.09	0.81±0.21	1.71±1.53	2.11±0.49	1.57
GELU	38.00±39.53	0.71±0.38	1.99±0.63	0.79±0.17	0.96±0.57	8.49
ABU-PINN	**0.05±0.02**	**0.15±0.10**	**0.58±0.15**	**0.34±0.08**	**0.35±0.07**	**0.29**

Wu, Chenxi, Min Zhu, Qinyang Tan, Yadhu Kartha, and Lu Lu. "A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks." Computer Methods in Applied Mechanics and Engineering 403 (2023)

Where should I place the collocation points?

$$\text{residual } \varepsilon(\mathbf{x}) = |f(\mathbf{x}; \widehat{u}(\mathbf{x}))|$$

$$p(\mathbf{x}) \propto \varepsilon(\mathbf{x}), \qquad \text{i.e.,} \qquad p(\mathbf{x}) = \frac{\varepsilon(\mathbf{x})}{A},$$

where $A = \int_\Omega \varepsilon(\mathbf{x})\, dx$ is a normalizing constant. Then all the residual points are sampled according to $p(\mathbf{x})$.

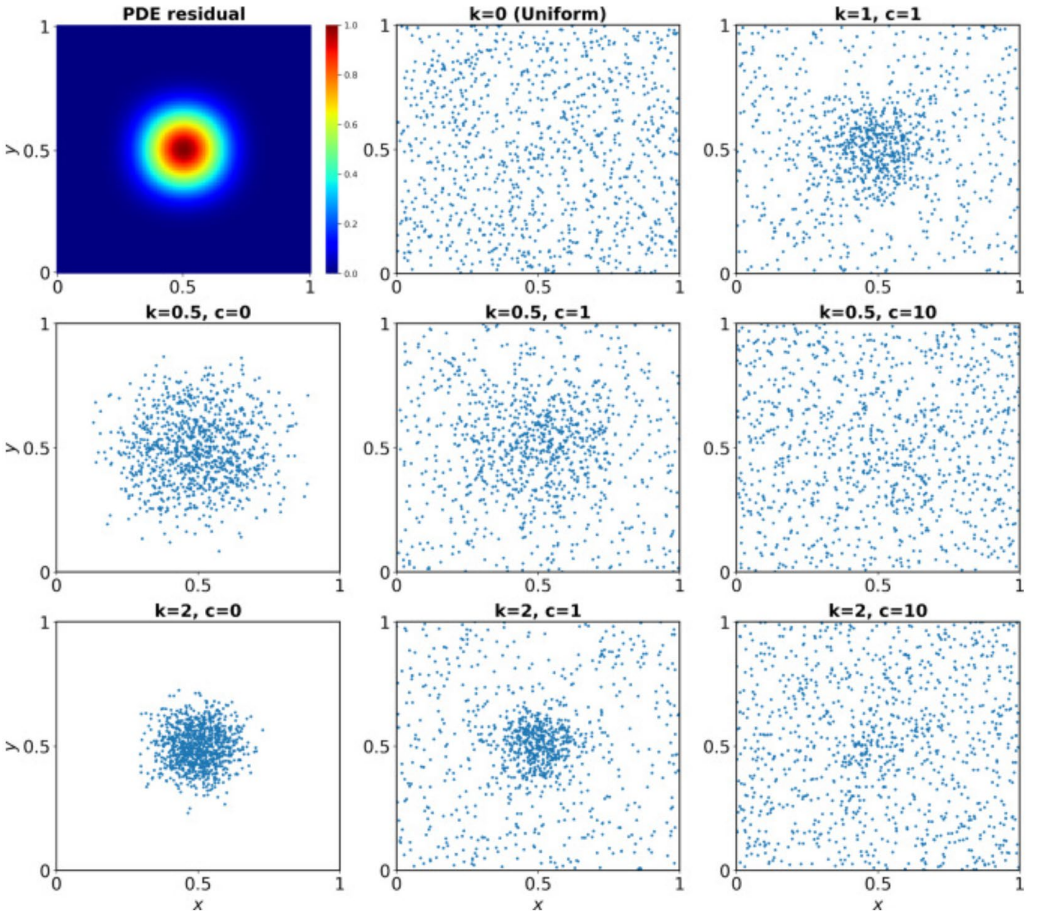
This approach works for certain PDEs, but as we show in our numerical examples, it does not work well in some cases. Following this idea, we propose an improved version called the residual-based adaptive distribution (RAD) method (Algorithm 2), where we use a new PDF defined as

$$p(\mathbf{x}) \propto \frac{\varepsilon^k(\mathbf{x})}{\mathbb{E}\left[\varepsilon^k(\mathbf{x})\right]} + c. \tag{2}$$

Here, $k \geq 0$ and $c \geq 0$ are two hyperparameters. $\mathbb{E}\left[\varepsilon^k(\mathbf{x})\right]$ can be approximated by a

Algorithm 2: RAD.

1 Sample the initial residual points $\mathcal{T}$ using one of the methods in Section 2.2.1;
2 Train the PINN for a certain number of iterations;
3 **repeat**
4 $\quad \mathcal{T} \leftarrow$ A new set of points randomly sampled according to the PDF of Eq. (2);
5 $\quad$ Train the PINN for a certain number of iterations;
6 **until** *the total number of iterations reaches the limit*;



So how do NNs find the finite-time blow-up solutions?

Benchmark example: Burgers equation

Burgers' equation

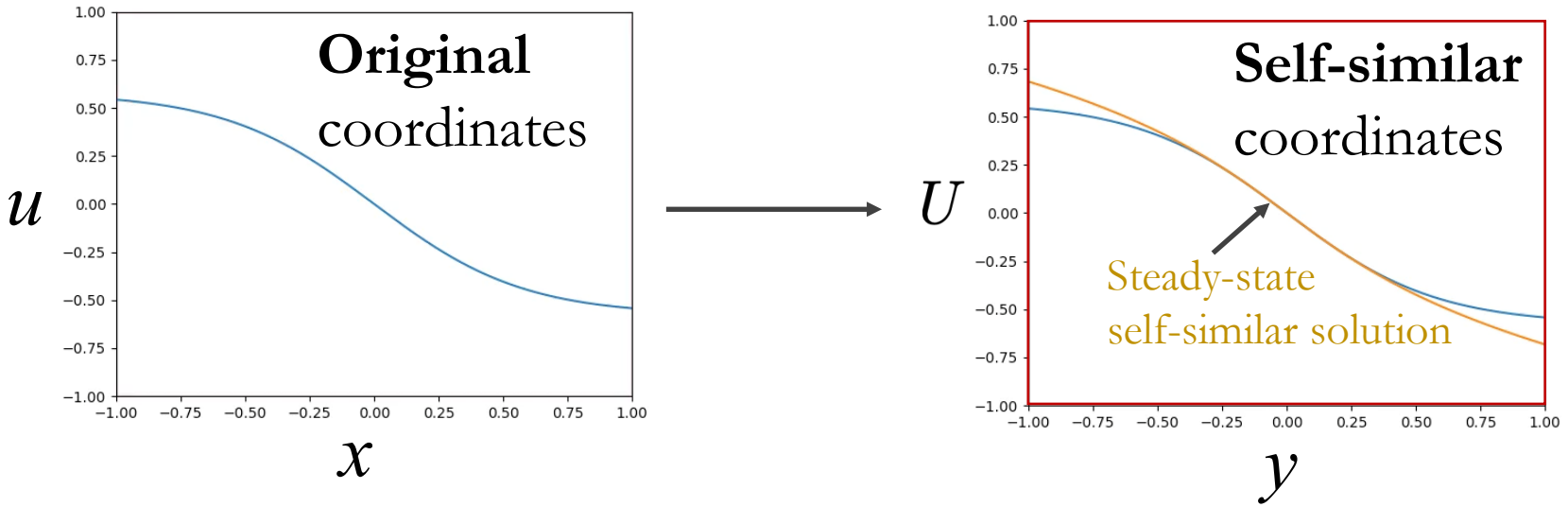
$$u_t + uu_x = 0$$

Shock wave

Assuming the self-similar ansatz

$$u = (1-t)^{\lambda} U\left(\frac{x}{(1-t)^{1+\lambda}}\right)$$ is the solution that blows up at $t = 1$.

$$y = \frac{x}{(1-t)^{1+\lambda}}$$ is the self-similar variable.



Benchmark example: Burgers equation

Burgers' equation

$$u_t + uu_x = 0$$

Shock wave

Assuming the self-similar ansatz

$$u = (1-t)^{\lambda} U\left(\frac{x}{(1-t)^{1+\lambda}}\right)$$ is the solution that blows up at $t = 1$.

$$y = \frac{x}{(1-t)^{1+\lambda}}$$ is the self-similar variable.

we obtain the self-similar Burgers' equation

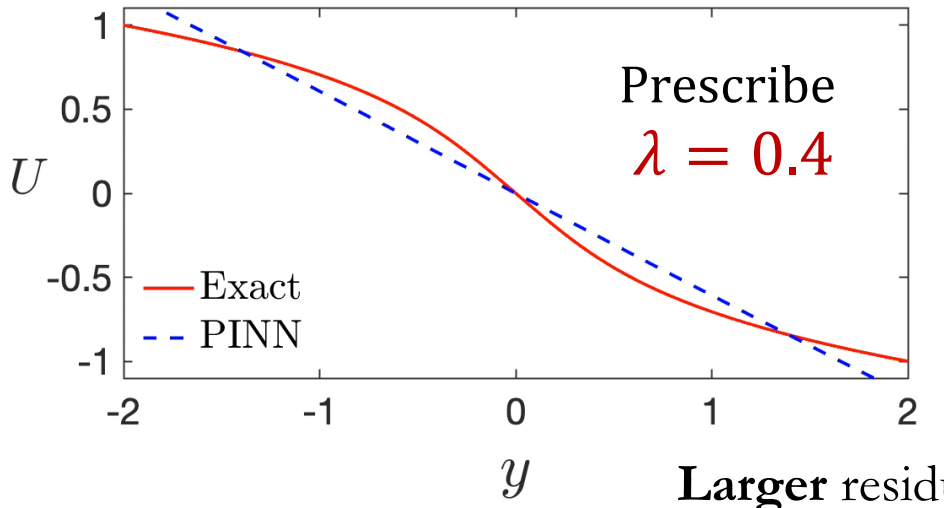
$$-\lambda U + ((1+\lambda)y + U)\partial_y U = 0$$

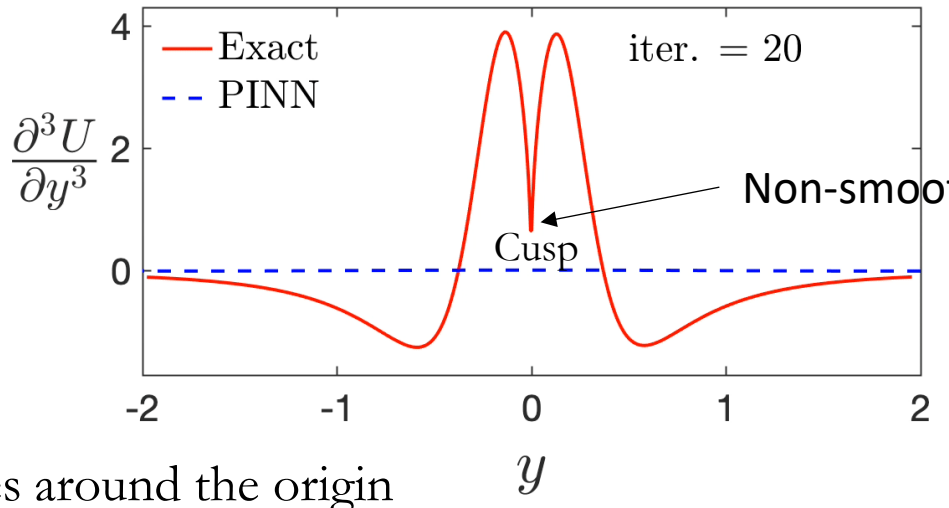
The goal for NN:
1. Find the solution $U(y)$
2. Find the self-similar exponent λ

This is an inverse problem!

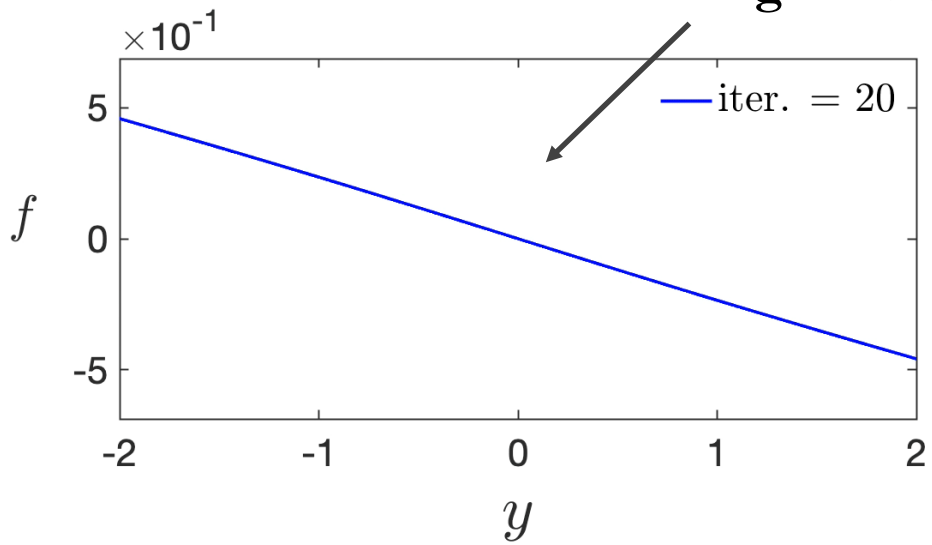
Non-smooth solution found by NN (forward)

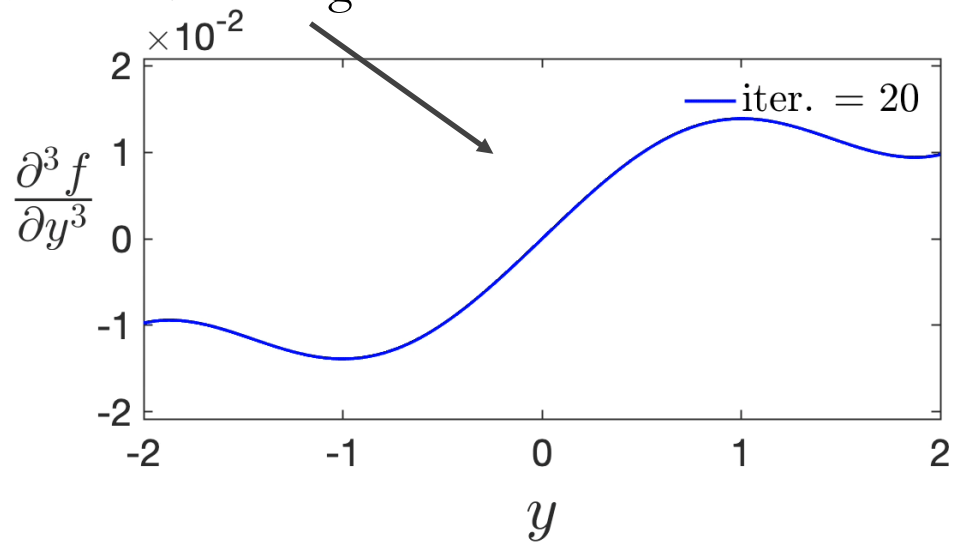
Self-similar equation for Burgers: $f = -\lambda U + ((1+\lambda)y + U)\partial_y U$



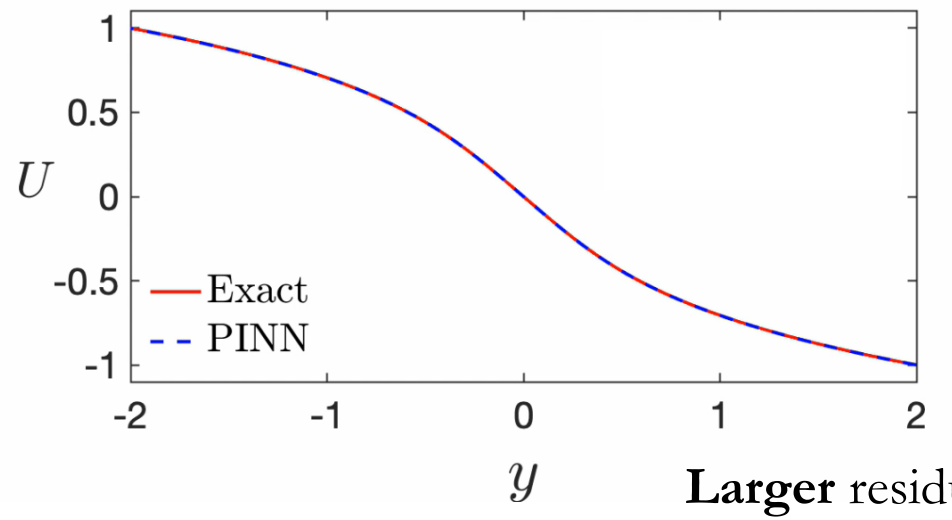


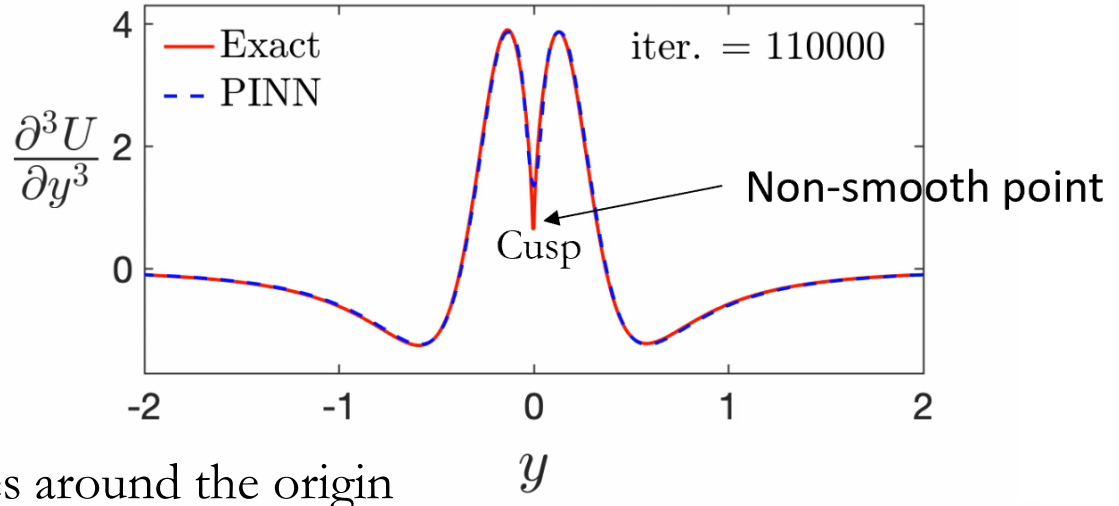
Larger residues around the origin





But we want to find the **smooth** solution and the corresponding λ… (inverse problem)

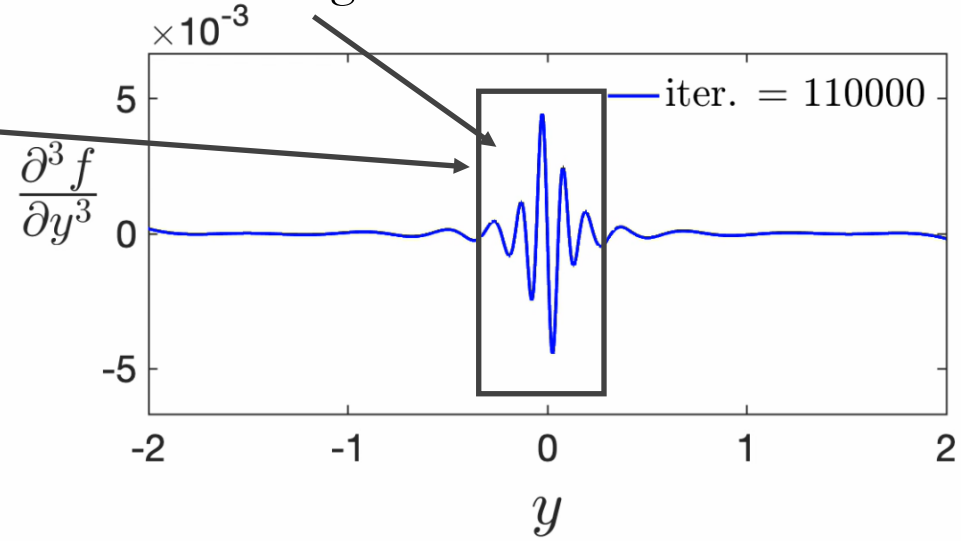




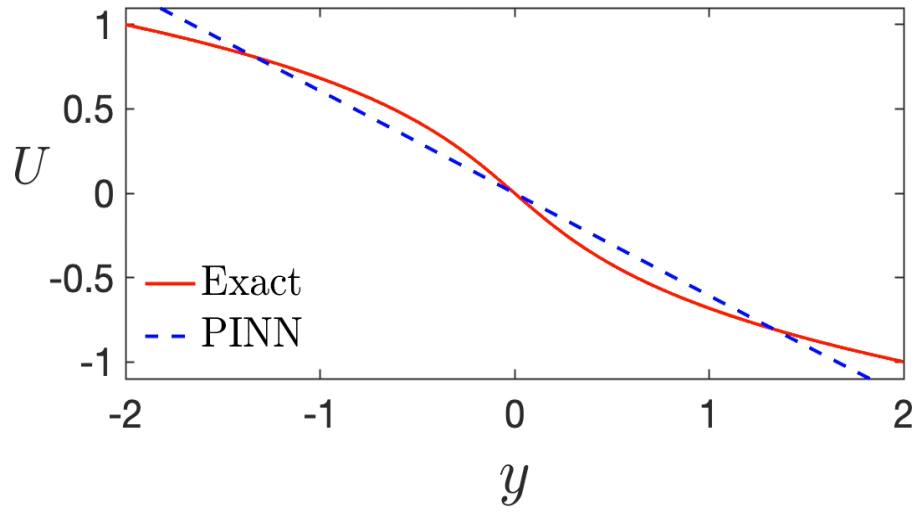
Larger residues around the origin

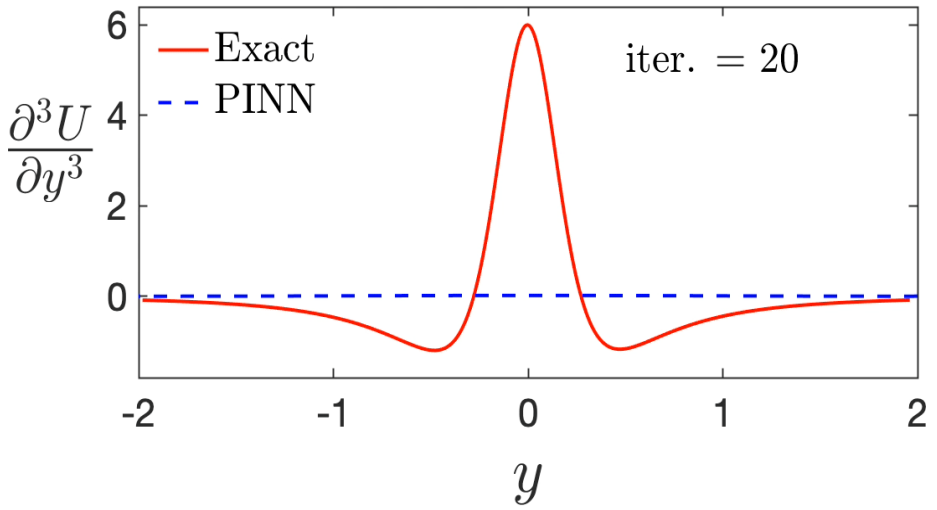
1. **Tunable** λ
2. **Smoothness** constraint

$$loss_s = [\partial_{yyy} f(y)]^2 \to 0$$



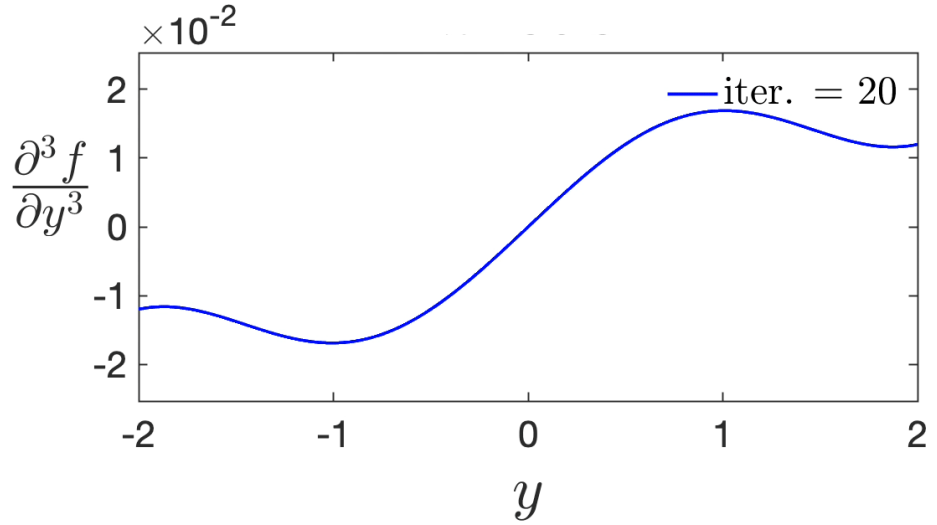
But we want to find the **smooth** solution and the corresponding λ… (inverse problem)





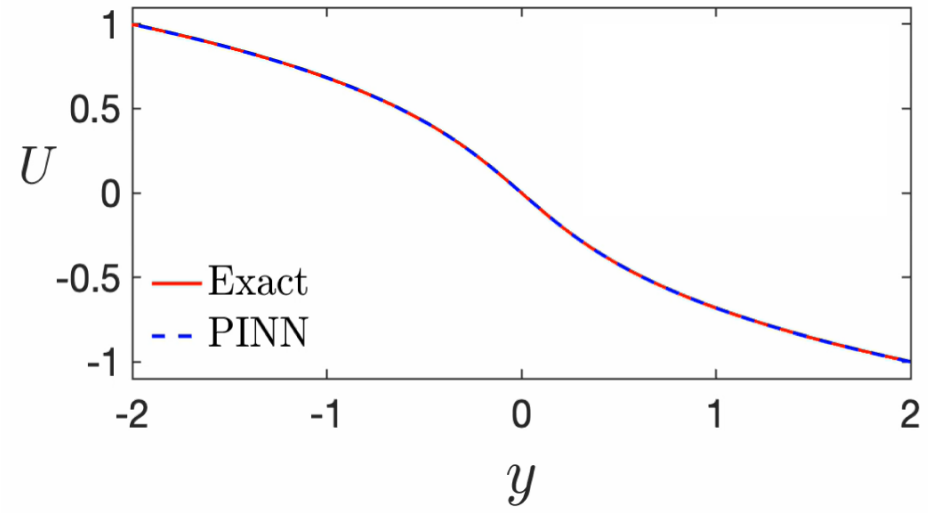
1. **Tunable** λ
2. **Smoothness** constraint

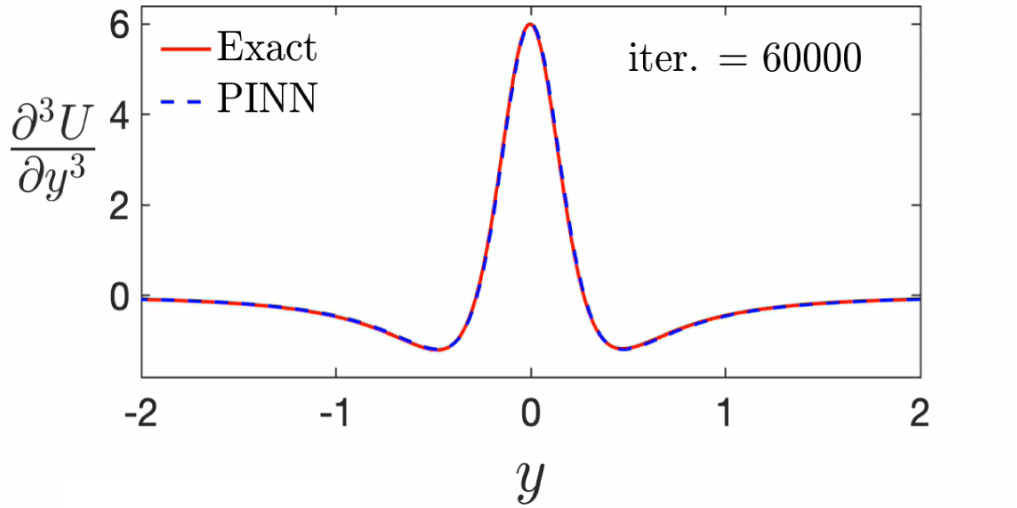
$$loss_s = [\partial_{yyy} f(y)]^2 \rightarrow 0$$



inferred $\lambda = 0.548$

But we want to find the **smooth** solution and the corresponding λ… (inverse problem)

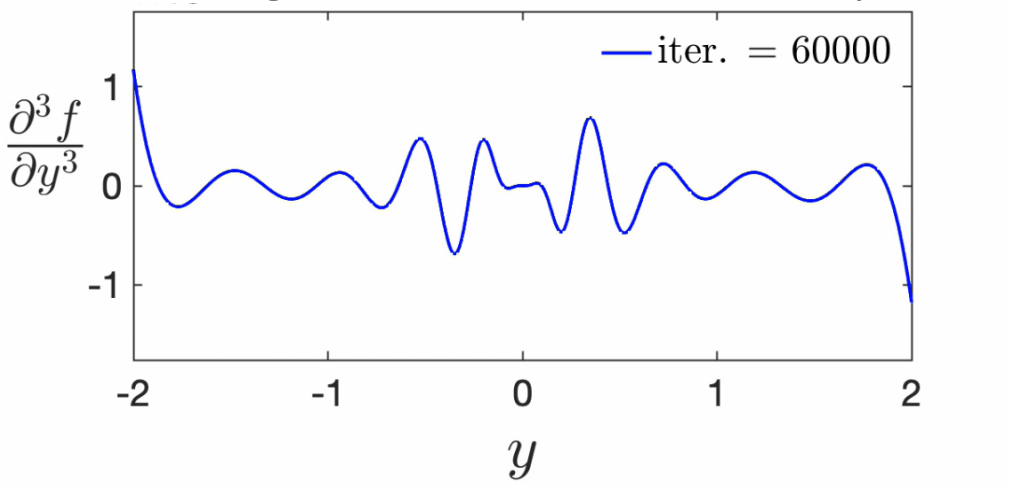




Uniform higher-order derivatives everywhere

1. **Tunable** λ
2. **Smoothness** constraint

$$loss_s = [\partial_{yyy} f(y)]^2 \to 0$$

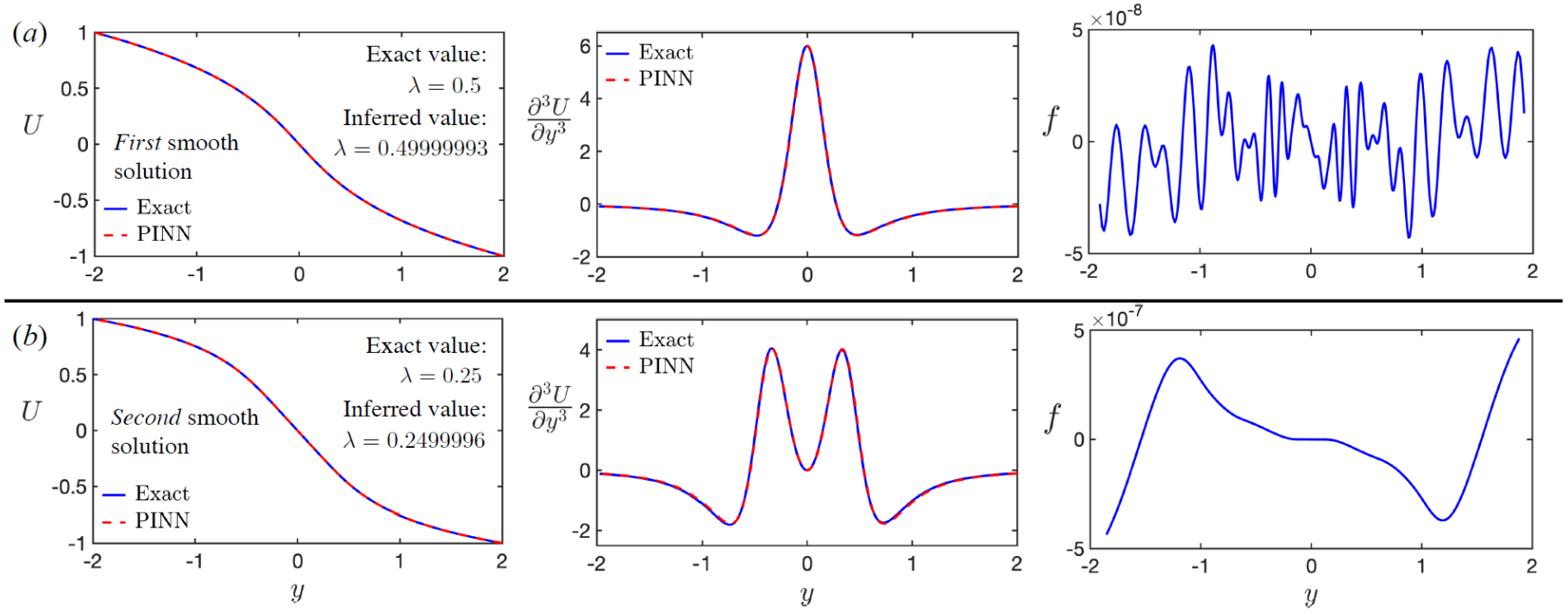


Very precise

theoretical $\lambda = 0.5$

inferred $\lambda = 0.500$

A hierarchy of smooth Burgers solutions



Smoothness constraint

$$loss_{s1} = \frac{1}{N_s}\sum_{i=1}^{N_s}\left|\frac{d^3 f}{dy^3}\left(y_i,\, U(y_i)\right)\right|^2$$

Inferred value:
$\lambda = 0.49999993$

$$loss_{s2} = \frac{1}{N_s}\sum_{i=1}^{N_s}\left|\frac{d^5 f}{dy^5}\left(y_i,\, U(y_i)\right)\right|^2 .$$

Inferred value:
$\lambda = 0.2499996$

Figure 6: The first and second smooth solutions for the Burgers' equation derived from the physics-informed neural network. The error of the inferred λ for the first smooth solution is of order 10^{-8} and the second of order 10^{-7}, which is consistent with the equation residue.

Incompressible Euler equations

The pair $(\mathbf{u}, p)$ solves the incompressible 3-D Euler equations if

$$\partial_t \mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = \cancel{\mu\Delta\mathbf{u}}^{\,0}, \quad \operatorname{div}(\mathbf{u}) = 0, \quad \text{and} \quad \mathbf{u}(\cdot, t) = \mathbf{u_0}$$

for velocity $\mathbf{u}$, pressure p and initial velocity $\mathbf{u_0}$.

Major question:

Does there exist smooth, finite energy initial condition $\mathbf{u_0}$ leading to a solution **blowing up** in finite time?

DNS: Luo and Hou, PNAS (2014), Luo and Hou, MMS (2014)
Proof: Chen and Hou, arXiv:2210.07191 (2022), arXiv:2305.05660 (2023)

Incompressible asymmetric Euler…

The pair (u, p) solves the incompressible 3-D Euler equations if

$$\partial_t u + (u \cdot \nabla)u + \nabla p = 0, \quad \operatorname{div}(u) = 0, \quad \text{and} \quad u(\cdot, t) = u_0$$

for velocity u, pressure p and initial velocity u_0.

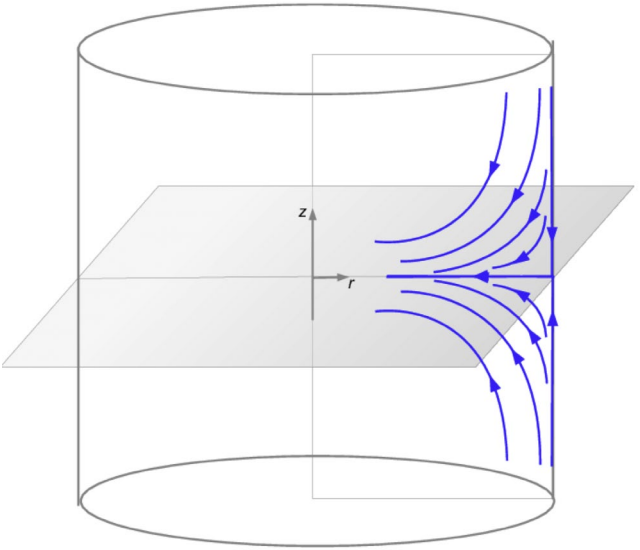
- The **Luo-Hou scenario**: assume asymmetricity with a cylindrical boundary
- Imposing the **self-similar ansatz** for self-similar coordinates

$$\begin{aligned}
\mathbf{y} = (y_1, y_2) &= \frac{(x_3, r-1)}{(1-t)^{1+\lambda}}, \\
(u_r, u_3) &= (1-t)^{\lambda}\mathbf{U}(\mathbf{y}, s) \\
\omega_\theta &= (1-t)^{-1}\Omega(\mathbf{y}, s) \\
\partial_r (r u_\theta)^2 &= (1-t)^{-2}\Psi(\mathbf{y}, s) \\
\partial_{x_3} (r u_\theta)^2 &= (1-t)^{-2}\Phi(\mathbf{y}, s)
\end{aligned}$$

- Assuming $t \to 1$, $s = -\log(1-t) \to \infty$ incompressible 3-D Euler equation becomes

$$\begin{aligned}
\Omega + ((1+\lambda)\mathbf{y} + \mathbf{U}) \cdot \nabla \Omega &= \Phi \\
(2 + \partial_{y_1} U_1)\Phi + ((1+\lambda)\mathbf{y} + \mathbf{U}) \cdot \nabla \Phi &= -\partial_{y_1} U_2 \Psi \\
(2 + \partial_{y_2} U_2)\Psi + ((1+\lambda)\mathbf{y} + \mathbf{U}) \cdot \nabla \Psi &= -\partial_{y_2} U_1 \Phi \\
\Omega = \partial_{y_1} U_2 - \partial_{y_2} U_1 \qquad & \operatorname{div} \mathbf{U} = 0
\end{aligned}$$

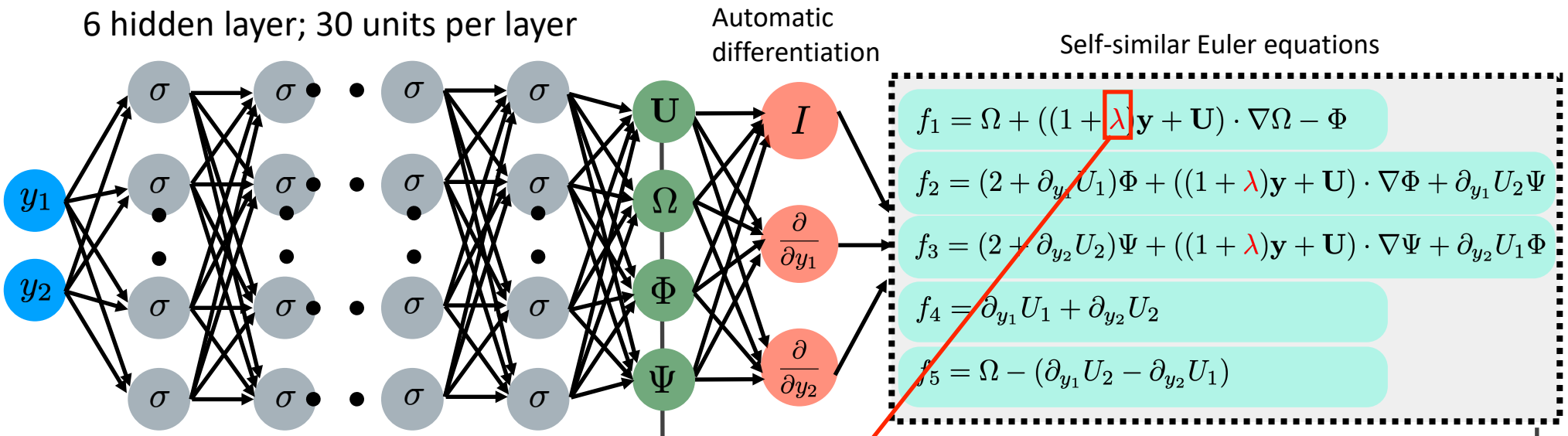
Luo-Hou (2014):
Compelling numerical evidence for blow-up



Their simulation suggests an asymptotic self-similar scaling near blowup

PINN (self-similar coordinate)

PINN: Raissi *et. al.* (2019), *J Comp.Phys*
Wang, Lai, Gómez-Serrano, Buckmaster,
Physical Review Letters (2023)



$$q(\mathbf{y}) = \sum_{j=1} w_{lk}^{(n)} \sigma\left(\sum w_{kj}^{(n-1)}\left(\ldots\left(\sum_{i=1} w_{ji}^{(1)} y_i + b_j^{(1)}\right)\ldots\right) + b_k^{(n-1)}\right) + b_l^{(n)}$$

Updating variables: $\mathbf{w}$: weights, $\mathbf{b}$: biases + λ: unknown in eqns

BC loss:

$$loss_c^{(j)} = \frac{1}{N_c^{(j)}} \sum_{i=1}^{N_c^{(j)}} [q^{(j)}(\mathbf{y}_i, \mathbf{w}, \mathbf{b}) - q_i^{(j)}]^2 \quad (j = 1, 2, \ldots, 5)$$

Eqn loss:

$$loss_f^{(k)} = \frac{1}{N_f^{(k)}} \sum_{i=1}^{N_f^{(k)}} f_k^2(\mathbf{y}_i, q(\mathbf{y}_i, \mathbf{w}, \mathbf{b})) \quad (k = 1, 2, \ldots, 5)$$

Optimization

Loss

$q^{(j)}$: j-th output variable

$q_i^{(j)}$: data of $q^{(j)}$ at $\mathbf{y} = \mathbf{y}_i$

Adding structures to the NN-PDE solver

1. Symmetry constraints

U_1, Φ, Ω are odd in y_1

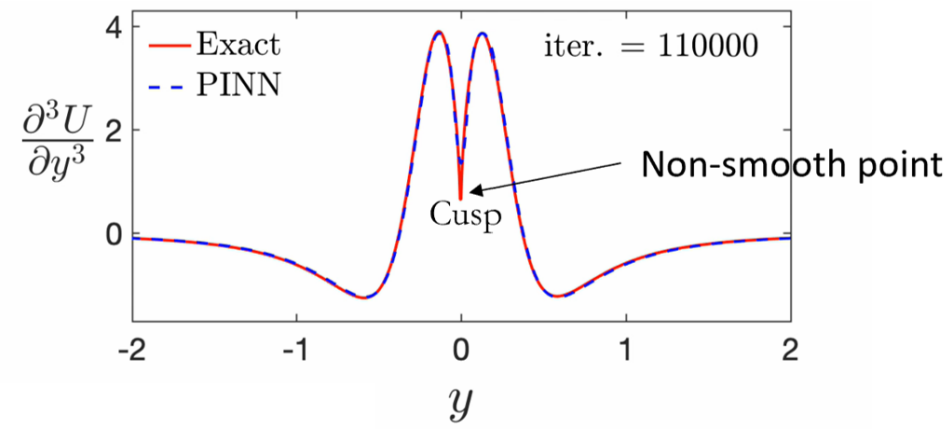
U_2, Ψ are even in y_1

e.g., to impose odd symmetry

$$U = [\mathrm{NN}_u(y) - \mathrm{NN}_u(-y)]/2$$

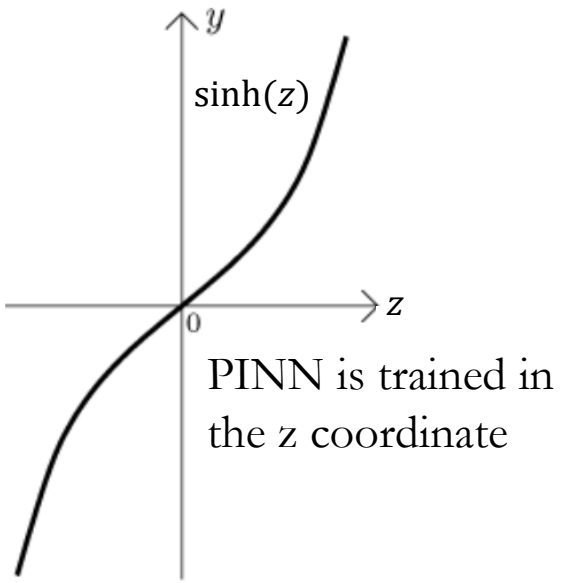
2. Smoothness constraint

e.g., $loss_s = [\partial_{yyy} f(y)]^2 \to 0$



3. Map infinite to finite domain size

Change of coordinate:
e.g. Domain size $\sinh(30) \approx 5 \times 10^{12}$



Approaching infinite domain (domain size $\approx 10^{12}$)

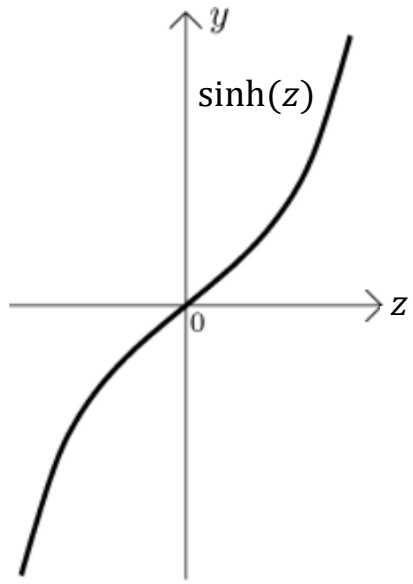
In order to cope with very large domains, we defined a new z-coordinate (z_1, z_2) with relation

$$\mathbf{y} = (y_1, y_2) = (\sinh(z_1),\ \sinh(z_2)) \quad \Longleftrightarrow \quad \mathbf{z} = (z_1, z_2) = (\sinh^{-1} y_1,\ \sinh^{-1} y_2)\,.$$

Change of coordinate: PINN is trained in the z coordinate; domain size $d = \sinh(30) \approx 5 \times 10^{12}$

$$\begin{aligned}
f_1 &= \Omega + ((1+\lambda)(\sinh(z_1), \sinh(z_2)) + \mathbf{U}) \cdot \nabla_y \Omega - \Phi \\
f_2 &= (2 + \operatorname{sech}(z_1)\partial_{z_1} U_1)\Phi + ((1+\lambda)(\sinh(z_1), \sinh(z_2)) + \mathbf{U}) \cdot \nabla_y \Phi + \operatorname{sech}(z_1)\partial_{z_1} U_2 \Psi \\
f_3 &= (2 + \operatorname{sech}(z_2)\partial_{z_2} U_2)\Psi + ((1+\lambda)(\sinh(z_1), \sinh(z_2)) + \mathbf{U}) \cdot \nabla_y \Psi + \operatorname{sech}(z_2)\partial_{z_2} U_1 \Phi \\
f_4 &= \operatorname{sech}(z_1)\partial_{z_1} U_1 + \operatorname{sech}(z_2)\partial_{z_2} U_2, \\
f_5 &= \Omega - [\operatorname{sech}(z_2)\partial_{z_2} U_1 - \operatorname{sech}(z_1)\partial_{z_1} U_2]\,, \\
f_6 &= \operatorname{sech}(z_2)\partial_{z_2}\Phi - \operatorname{sech}(z_1)\partial_{z_1}\Psi\,.
\end{aligned}$$

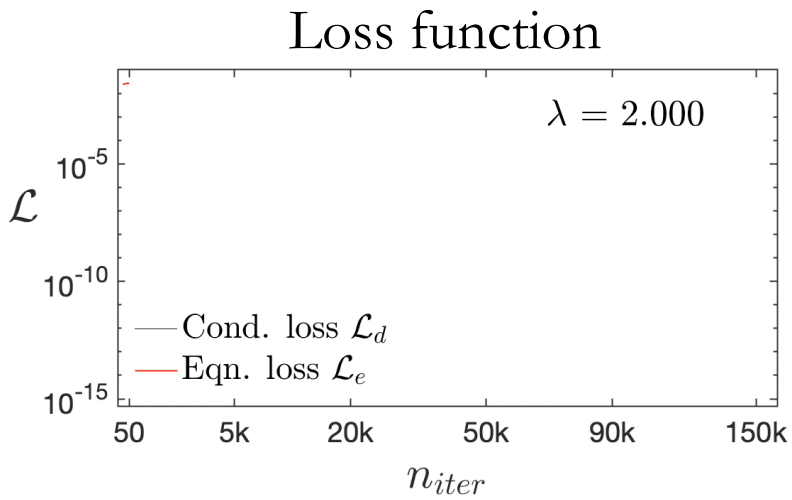
$$\begin{aligned}
g_1 &= U_2(z_1, 0), && \text{(non-penetration condition)} \\
g_2 &= \partial_{z_1}\Omega(0,0) + 1, && \text{(fix scaling symmetry)} \\
g_3 &= \nabla_y \mathbf{U}(|z_1| = \sinh^{-1} d), \quad g_4 = \nabla_y \mathbf{U}(|z_2| = \sinh^{-1} d) && \text{(suitable decay)} \\
g_5 &= \Phi(|z_1| = \sinh^{-1} d), \quad g_6 = \Phi(|z_2| = \sinh^{-1} d), \\
g_7 &= \Psi(|z_1| = \sinh^{-1} d), \quad g_8 = \Psi(|z_2| = \sinh^{-1} d),
\end{aligned}$$



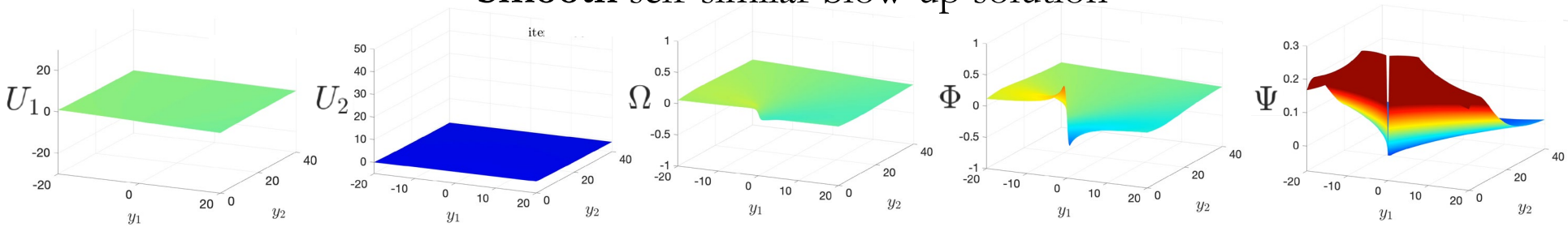
Find self-similar singularities to the Euler equation

Inferred $\lambda = 1.917$

Wang, Lai, Gómez-Serrano, Buckmaster, *PRL* (2023)

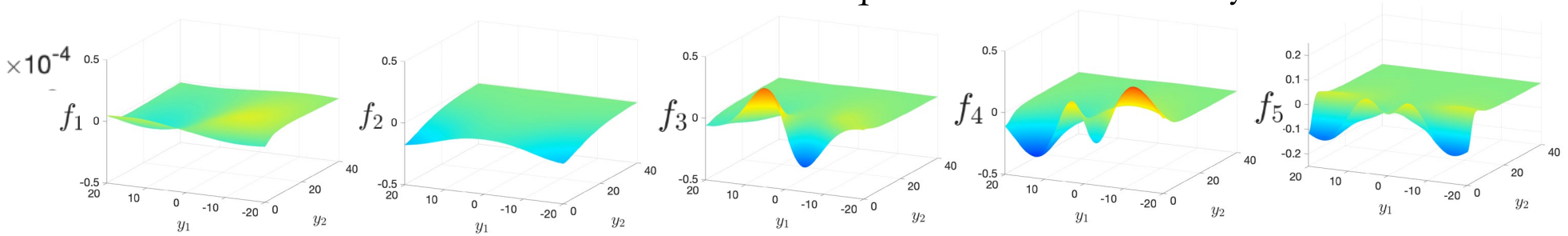


Smooth self-similar blow-up solution



iter. $= 50$

Uniform and small equation residues everywhere



MSNN gives solution close to machine precision

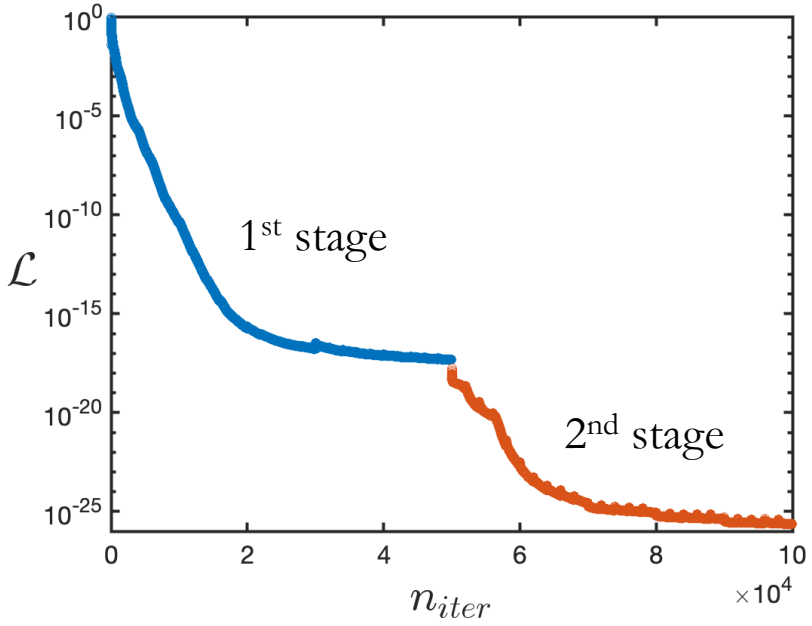
Inferred $\lambda = 1.920560013482733$

Chen and Hou: $\lambda = 1.9205600$

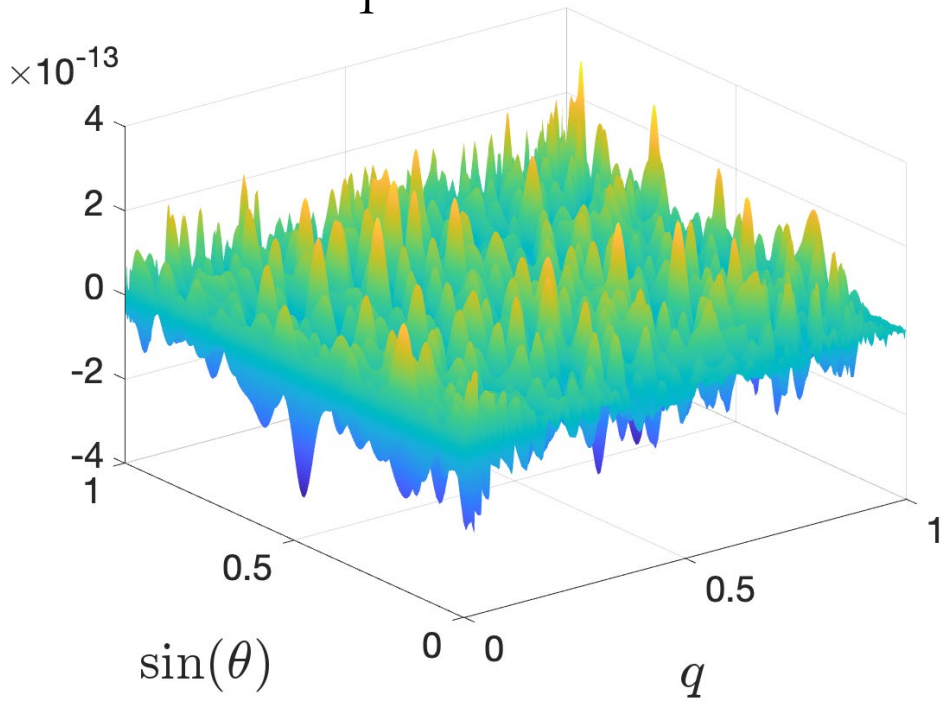
With multistage NN, the error is further reduced.

Proof: Chen and Hou, arXiv:2210.07191 (2022), arXiv:2305.05660 (2023)

Loss function

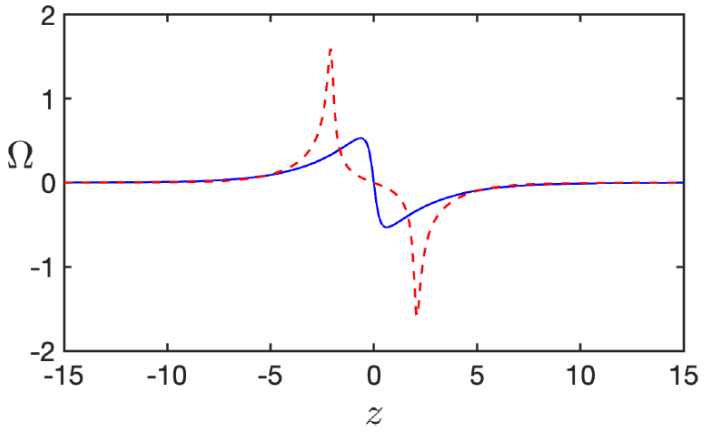


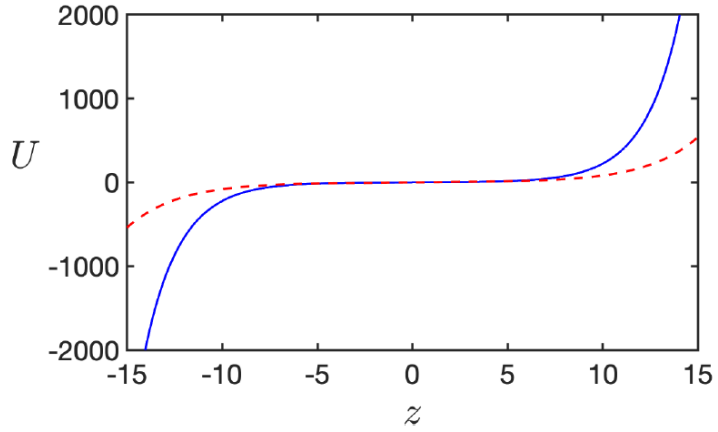
Equation residue

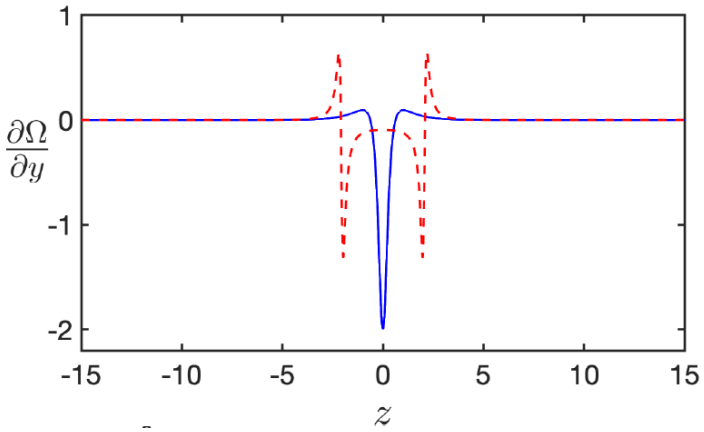


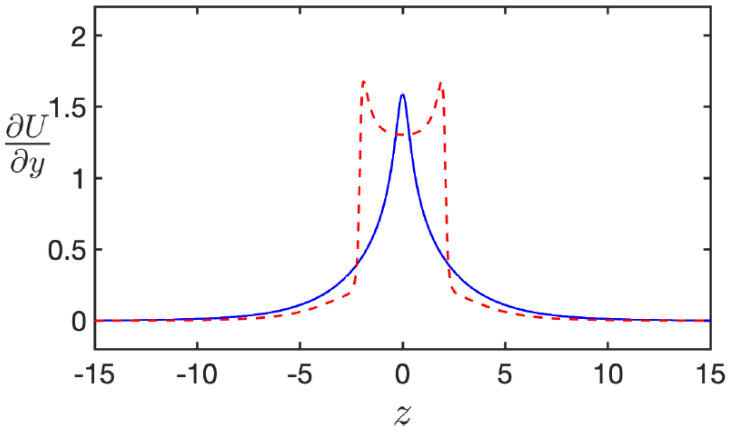
An unstable CCF solution was found

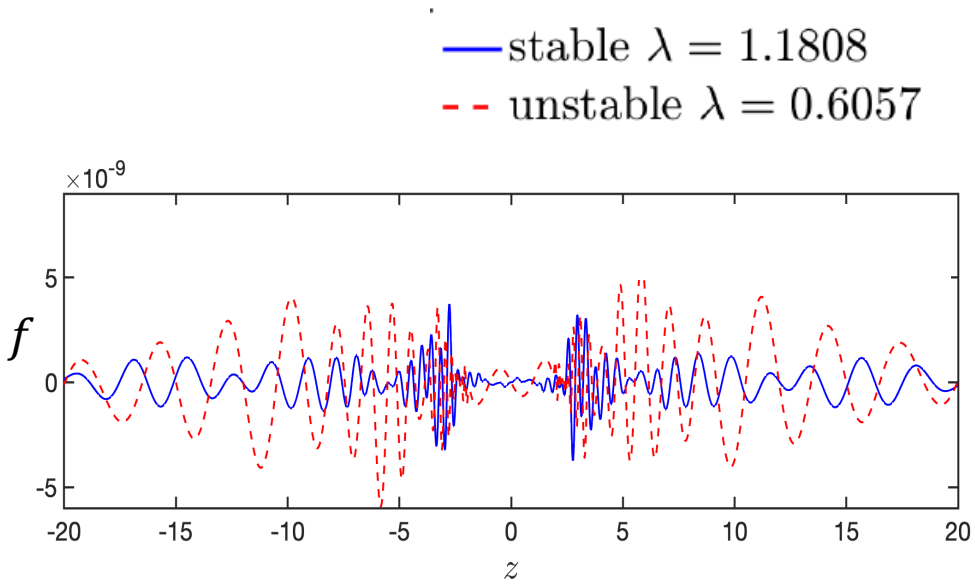
The self-similar CCF equation: $\Omega + ((1+\lambda)y - U)\partial_y\Omega - \Omega\partial_y U = 0, \quad \text{where} \quad U = -\int_0^y H\Omega(s)ds$











Unstable blow-up solutions are generally difficult to find numerically via classical time-stepping simulations.

Some challenges and progress

- NN's function approximation errors

  → Multistage neural network: Wang & Lai, JCP (2024). Faster optimization: Jnini et al. arXiv:2402.10680

- Approximating high-frequency functions → Jagtap et al., JCP (2020) $\sigma(a\,\mathcal{L}_k(z^{k-1}))$,
  Introduce a scaling variable in the activation function, which can be optimized

- Determine the weights for different terms in the cost function

  → Learning rate annealing: Wang et al. SIAM J. Sci. Comput. (2021), Neural Tangent Kernel: Wang et al., JCP (2022), SA-PINN: McClenny & Braga-Neto, JCP (2023)

- Error bounds of the PINN solutions → Mishra & Molinaro, IMA J. Numer. Anal. (2023)
  In addition to saying PDE residues are small, we know that the distance between the PINN approximated solution itself with the true solution, which is not available, would be small

So what have we learned?

- NN is a powerful tool for searching singularities in fluids; it found solution that had not been discovered before
- It requires A TONS of mathematical knowledge to guide the PINN find the right thing
- PINNs are far from being a general method for solving PDEs; however, they can be useful for finding solutions by imposing problem-specific mathematical structures
- Aims and opportunities: deep learning drives deeper understanding

Acknowledgement

Wang, Lai, Gómez-Serrano, Buckmaster, *Physical Review Letters* (2023)

Wang and Lai, "Multi-stage neural networks: Function approximator of machine precision", *J. Comput. Phys.* (2024)

Yongji Wang
Postdoc (NYU)

Prof. Tristan Buckmaster
New York University

Prof. Javier G´omez-Serran
Brown University

Gonzalo Cao-Labora
Student (MIT Math)