Understanding and Mitigating Gradient Pathologies in Physics-Informed Neural Networks

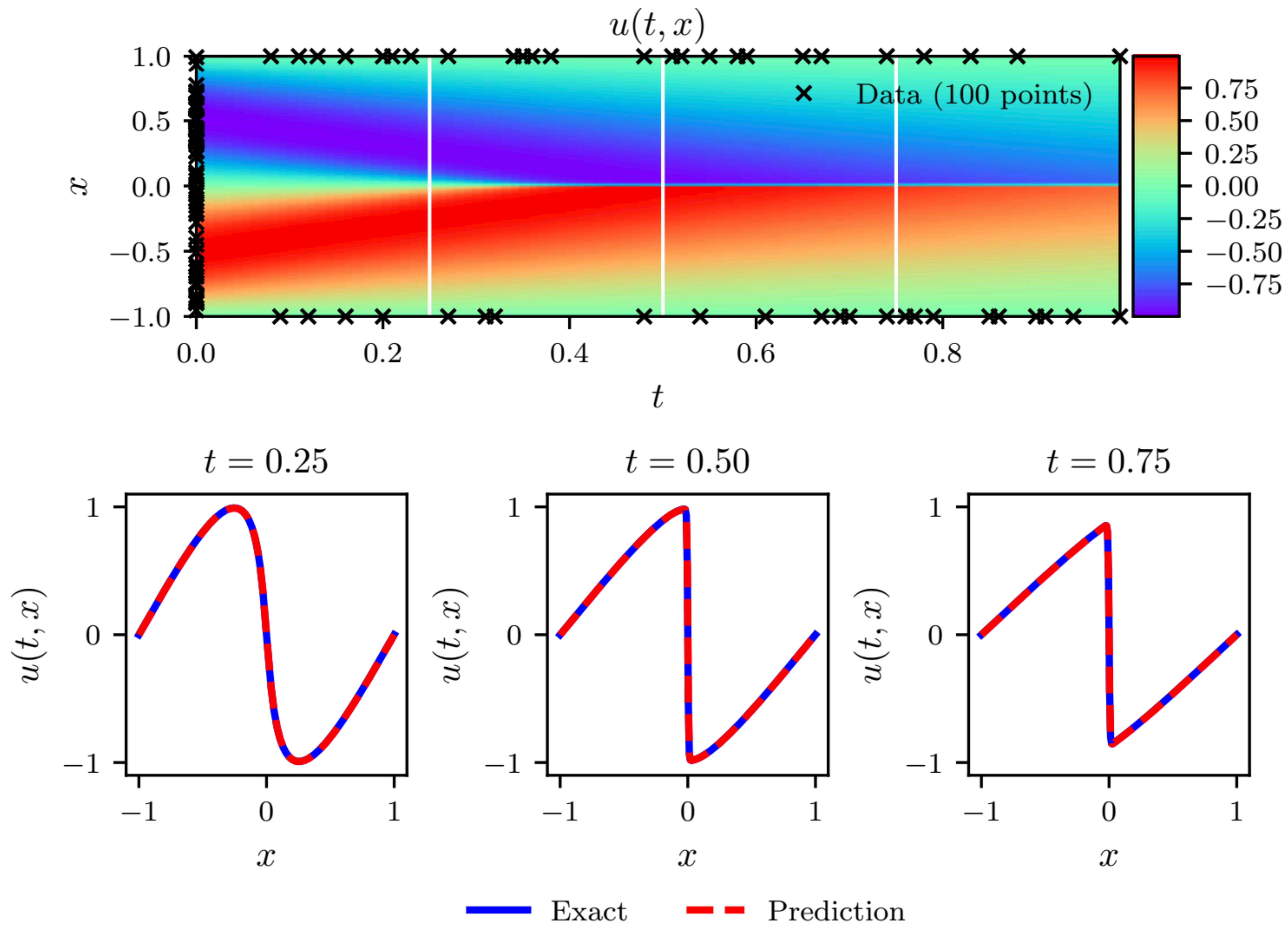
S. Wang, Y. Teng, and P. Perdikaris

Yifei Han, Jonathan Seele - June 01, 2024

Table of Content

Motivation

- PINNs deliver reasonable results
- But in some settings they perform poorly

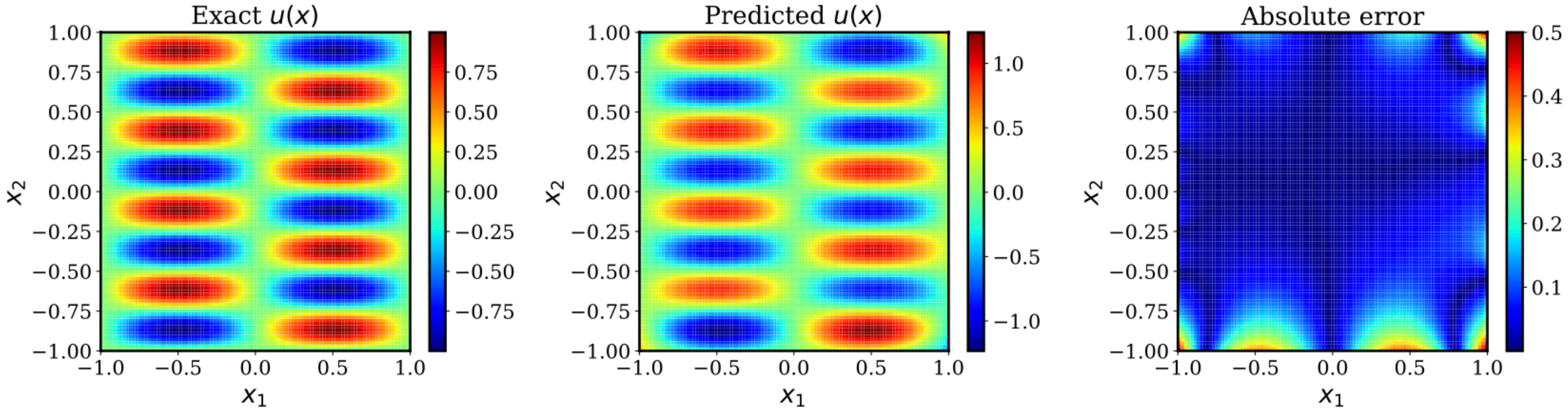


Burger's equation: predicted solution $u(t,x)$ with error $L_2 : 6.7 * 10^{-4}$

Motivation

Example - Helmholtz Equation

- Helmholtz Equation
- Conventional PINN delivers poor results



PINN model with 40 layers, 50 neurons per layer, after 40,000 iterations. Relative L_2 error: 0.181

Table of Content

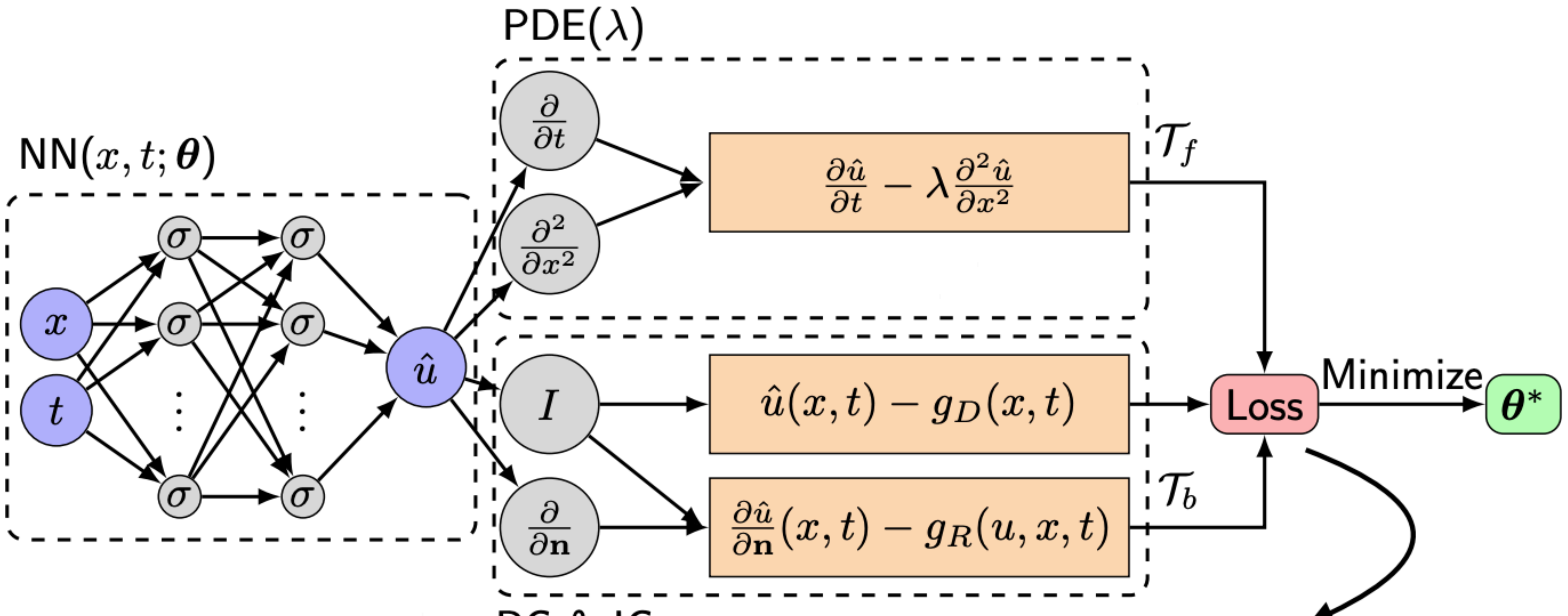
Related Work

- Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
- Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, 19(1):932–955, 2018.
- Luning Sun, Han Gao, Shaowu Pan, and Jian-Xun Wang. Surrogate modeling for fluid flows based on physics- constrained deep learning without simulation data. *arXiv preprint arXiv:1906.02382*, 2019.

Table of Content

Architecture PINN

- Fully Cc
- 4 hidder
- 40,000 i gradient



$$\mathcal{L}(\theta) := \underbrace{\frac{1}{N_u}\sum_{i=1}^{N_u}[\boldsymbol{u}_i - f_\theta(\boldsymbol{x}_i)]^2}_{\text{Data fit}} + \underbrace{\frac{1}{\lambda}\mathcal{R}[f_\theta(\boldsymbol{x})]}_{\text{Physics regularization}}$$

Architecture of a regular PINN

Primer in physics-informed neural networks

- PINNs aim at inferring function $\boldsymbol{u}(\boldsymbol{x}, t)$
- Solution to system of nonlinear partial differential equations:
  1. $\boldsymbol{u}_t + \mathcal{N}_{\boldsymbol{x}}[\boldsymbol{u}] = 0, \;\; \boldsymbol{x} \in \Omega, t \in [0, T]$
  2. Initial condition: $\boldsymbol{u}(\boldsymbol{x}, 0) = h(\boldsymbol{x}), \;\; x \in \Omega$
  3. Boundary condition: $\boldsymbol{u}(\boldsymbol{x}, t) = g(\boldsymbol{x}, t), \;\; t \in [0, T], \;\; \boldsymbol{x} \in \partial\Omega$
- No initial/boundary conditions ➔ infinite solutions

Primer in physics-informed neural networks

- Composite loss function: $\mathcal{L}(\theta) := \mathcal{L}_r(\theta) + \sum_{i=1}^{M} \lambda_i \mathcal{L}_i(\theta)$
- Loss function of residual: $\mathcal{L}_r = \frac{1}{N_r} \sum_{i=1}^{N_r} [\boldsymbol{r}(\boldsymbol{x}_r^i, t_r^i)]^2$

$$\boldsymbol{r}_\theta(\boldsymbol{x}, t) := \frac{\partial}{\partial t} f_\theta(\boldsymbol{x}, t) + \mathcal{N}_{\boldsymbol{x}}[f_\theta(\boldsymbol{x}, t)]$$

- $\mathcal{L}_i$ loss function of data fit terms
  - e.g., boundary loss
  - $\mathcal{L}_{u_b} = \frac{1}{N_b} \sum_{i=1}^{N_b} [\boldsymbol{u}(\boldsymbol{x}_b^i, t_b^i) - g_b^i]^2$

Table of Content

Gradient Pathologies

Using Helmholtz Equation

- Revisit the Helmholtz Equation in 2D
- Remember: PINNs struggle constructing accurate solution
- Now:
  - Fabricated solution causing erroneous prediction
  - Inspect the gradients of loss terms

Gradient Pathologies

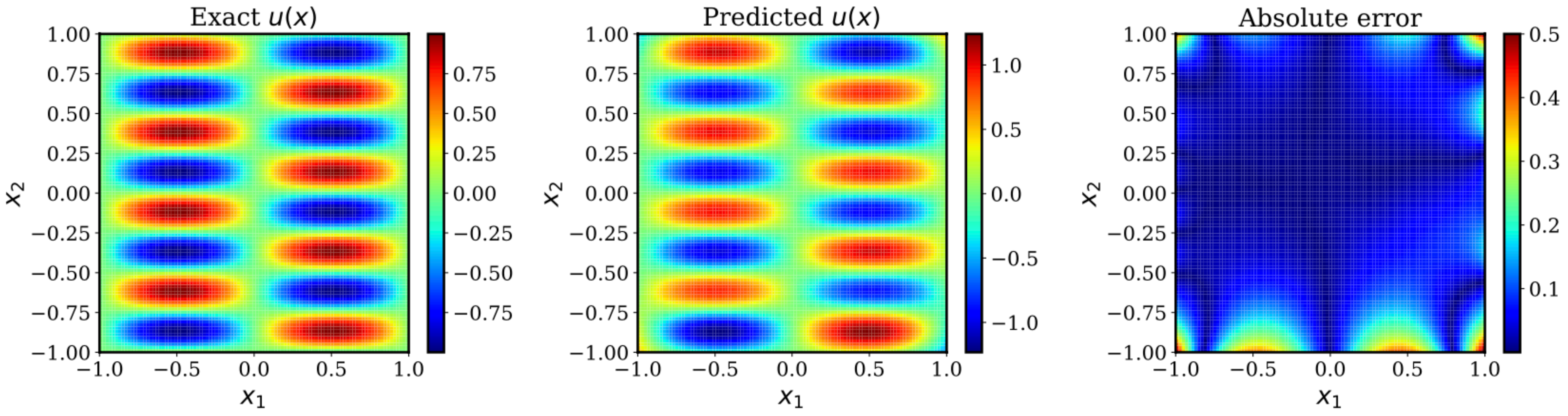
Fabricated Solution

- $\Delta u(x, y) + k^2 u(x, y) = q(x, y), \ (x, y) \in \Omega := (-1, 1)$
- $u(x, y) = h(x, y), \ \ (x, y) \in \partial\Omega$
- Simple exact solution: $u(x, y) = \sin(a_1 \pi x) \sin(a_2 \pi y)$
- Lets choose $a_1 = 1$ and $a_2 = 4$.

Gradient Pathologies

Inspecting gradients

- Revisit the prediction of the PINN
- Fails especially at the boundary

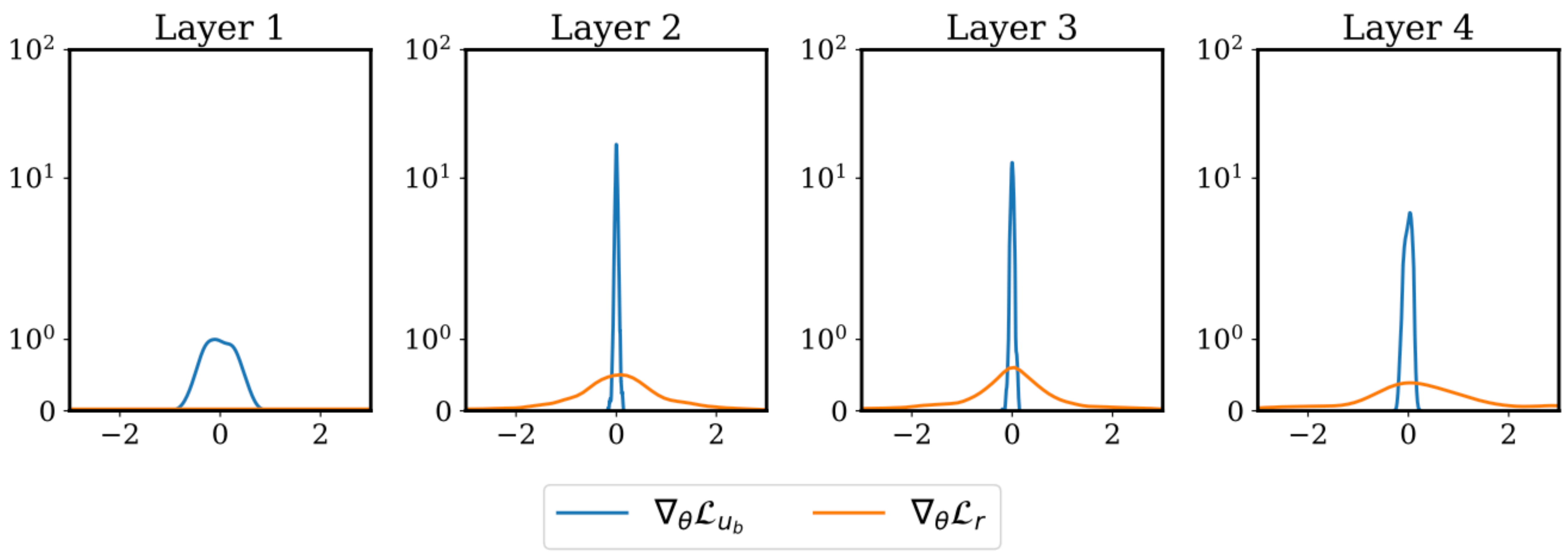


PINN model with 40 layers, 50 neurons per layer, after 40,000 iterations. Relative L^2 error: 0.181

Gradient Pathologies

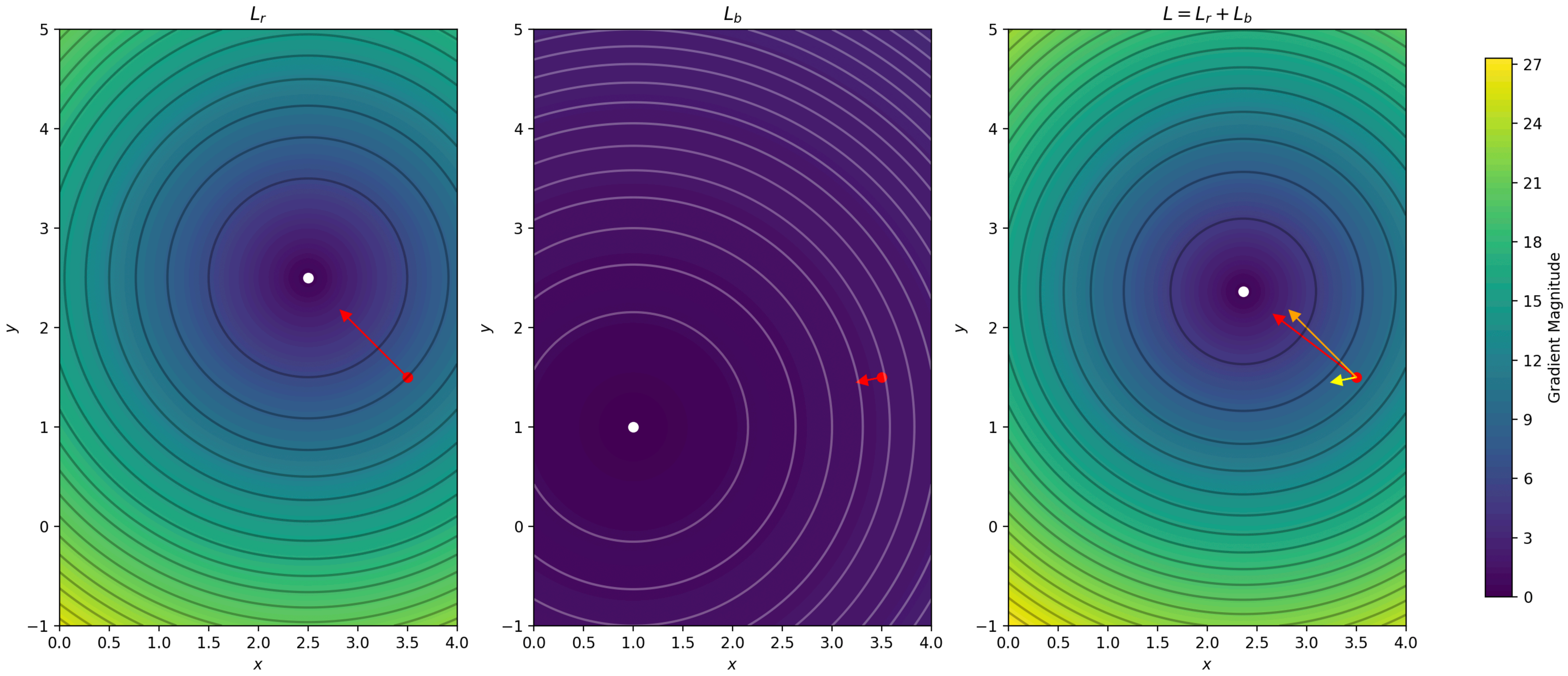
Inspecting gradients

- Imbalanced gradients ➔ Boundary Condition is not enforced
- PDE has multiple solutions ➔ Network finds some (wrong) solution.
- Conclusion: model is biased towards minimizing residual loss $\mathcal{L}_r(\theta)$

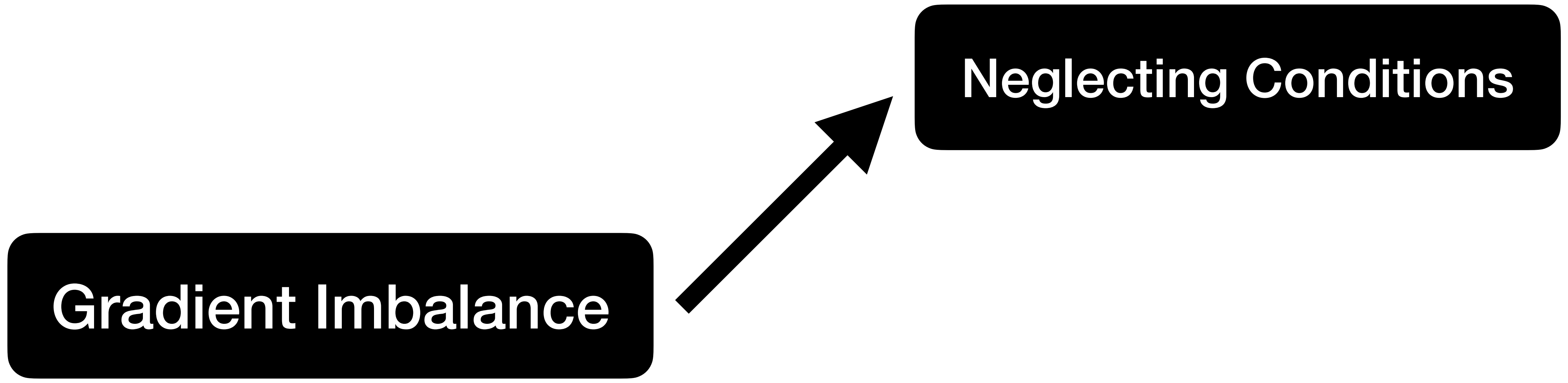


Histograms of back-propagated gradients at each layer, during the 40,000th iteration

Gradient Pathologies



The big picture



Gradient Analysis

What is causing the gradient imbalance ?

- 1D Poisson Equation
  - $\Delta u(x) = g(x), \;\; x \in [0, 1]$
  - $u(x) = h(x), \;\; x = 0 \text{ and } x = 1$
- Exact solution: $u(x) \; = \; \sin(Cx)$
- Use PINN $f_\theta(x)$ approximate $u(x)$

Gradient Analysis

What is causing the gradient imbalance ?

- We can show that:
  - $\|\nabla_\theta \mathcal{L}_{u_b}(\theta)\|_{L^\infty} \leq 2\epsilon \cdot \|\nabla_\theta \epsilon_\theta(x)\|_{L^\infty}$
  - $\|\nabla_\theta \mathcal{L}_r(\theta)\|_{L^\infty} \leq O(C^4) \cdot \epsilon \cdot \|\nabla_\theta \epsilon_\theta(x)\|_{L^\infty}$

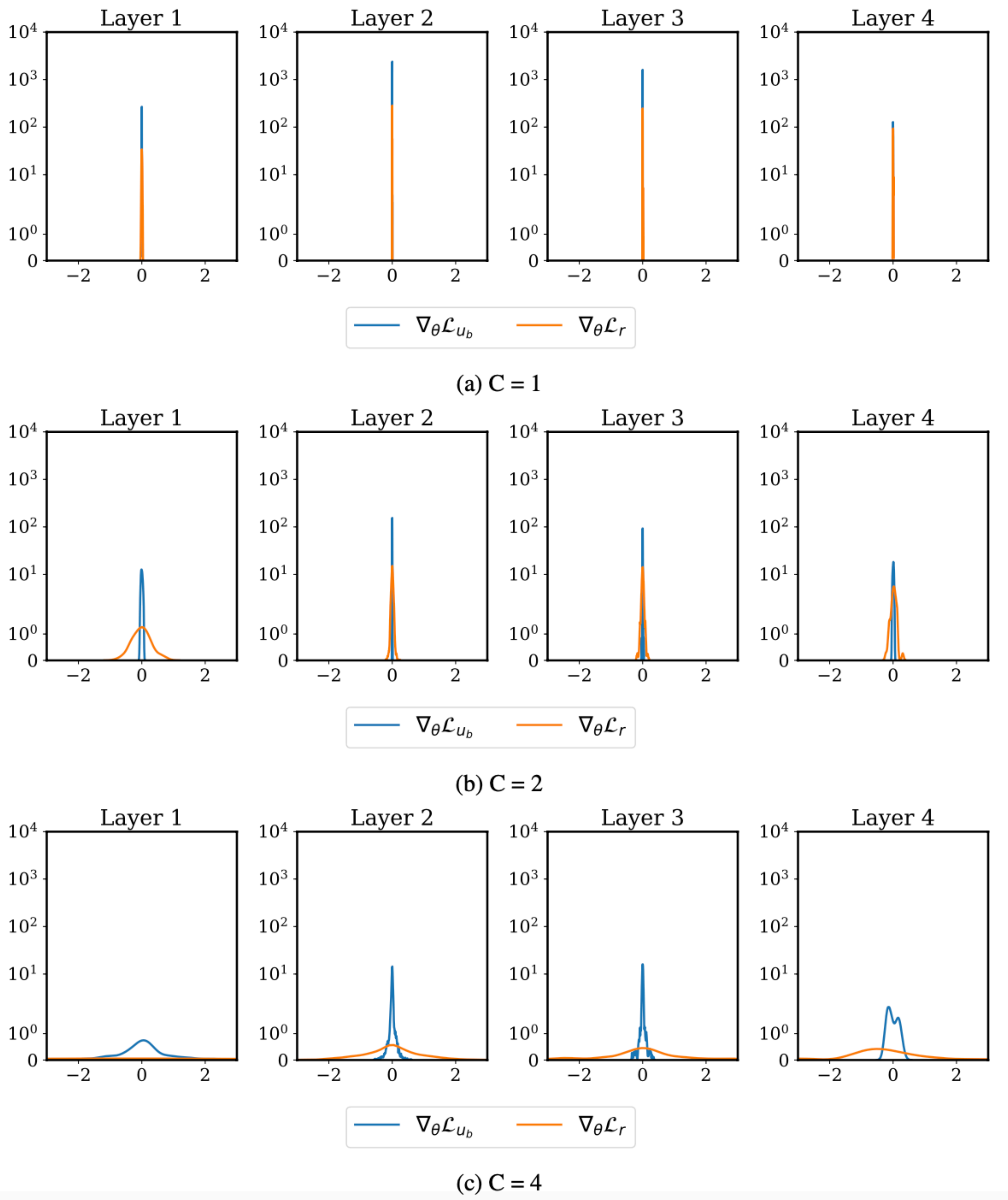
Proof (1) in Appendix

- Large C ➔ $\|\nabla_\theta \mathcal{L}_r(\theta)\|_{L^\infty} > \|\nabla_\theta \mathcal{L}_{u_b}(\theta)\|_{L^\infty}$
  - Results in presented pathologies

Gradient Analysis

What is causing the gradient imbalance ?

- Increasing imbalance with increasing C



Histograms of back-propagated gradients per layer, during 40,000th iteration

Stiffness in gradient flow dynamics

Root cause of gradient imbalance

- Gradient imbalance for large C values
- Why is this happening, what is the root cause ?
- Hypotheses:
    1. Stiffness exist in gradient flow dynamics of PINNs
    2. Stiffness comes along with imbalanced gradients

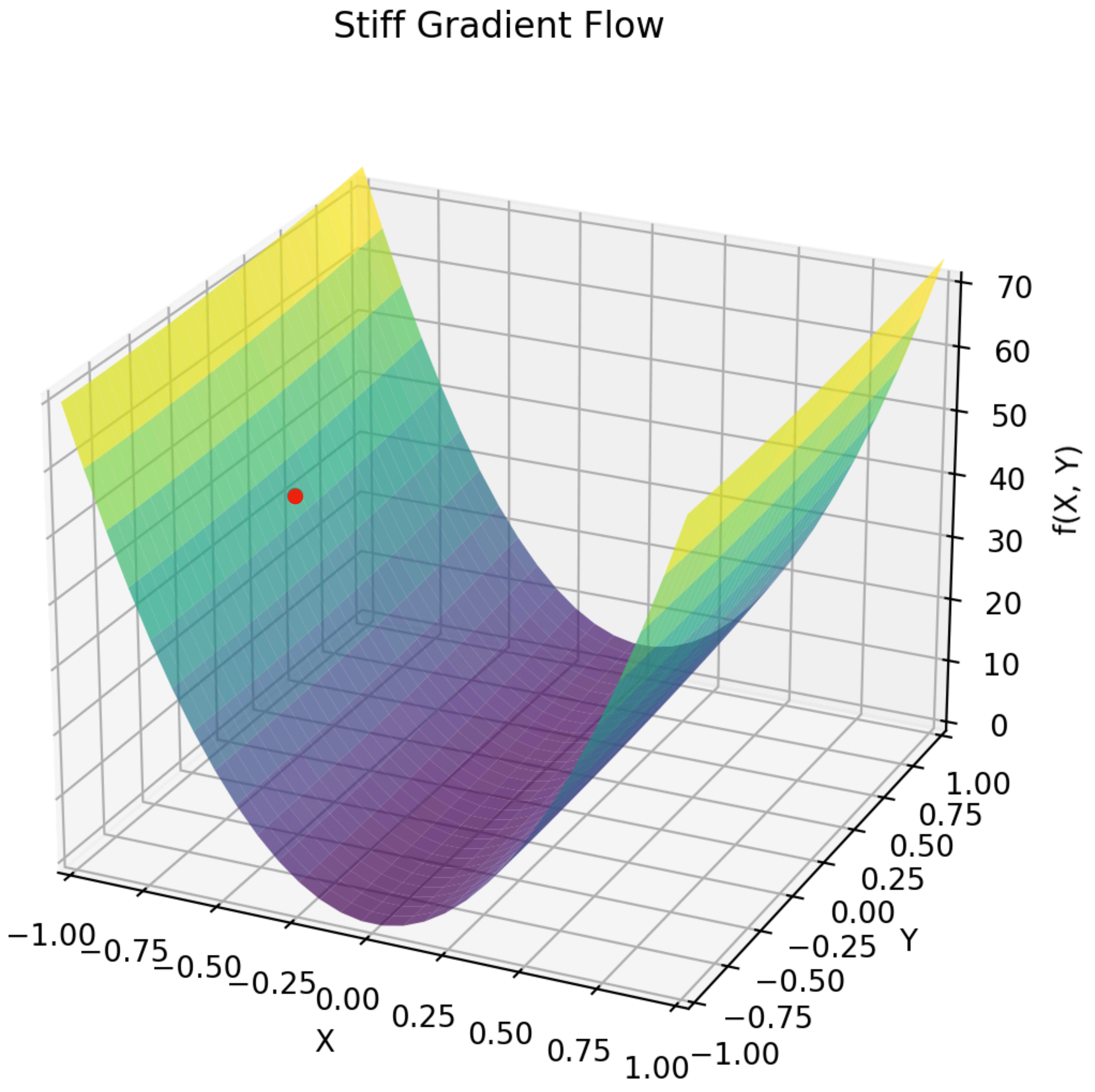
Stiffness in gradient flow dynamics

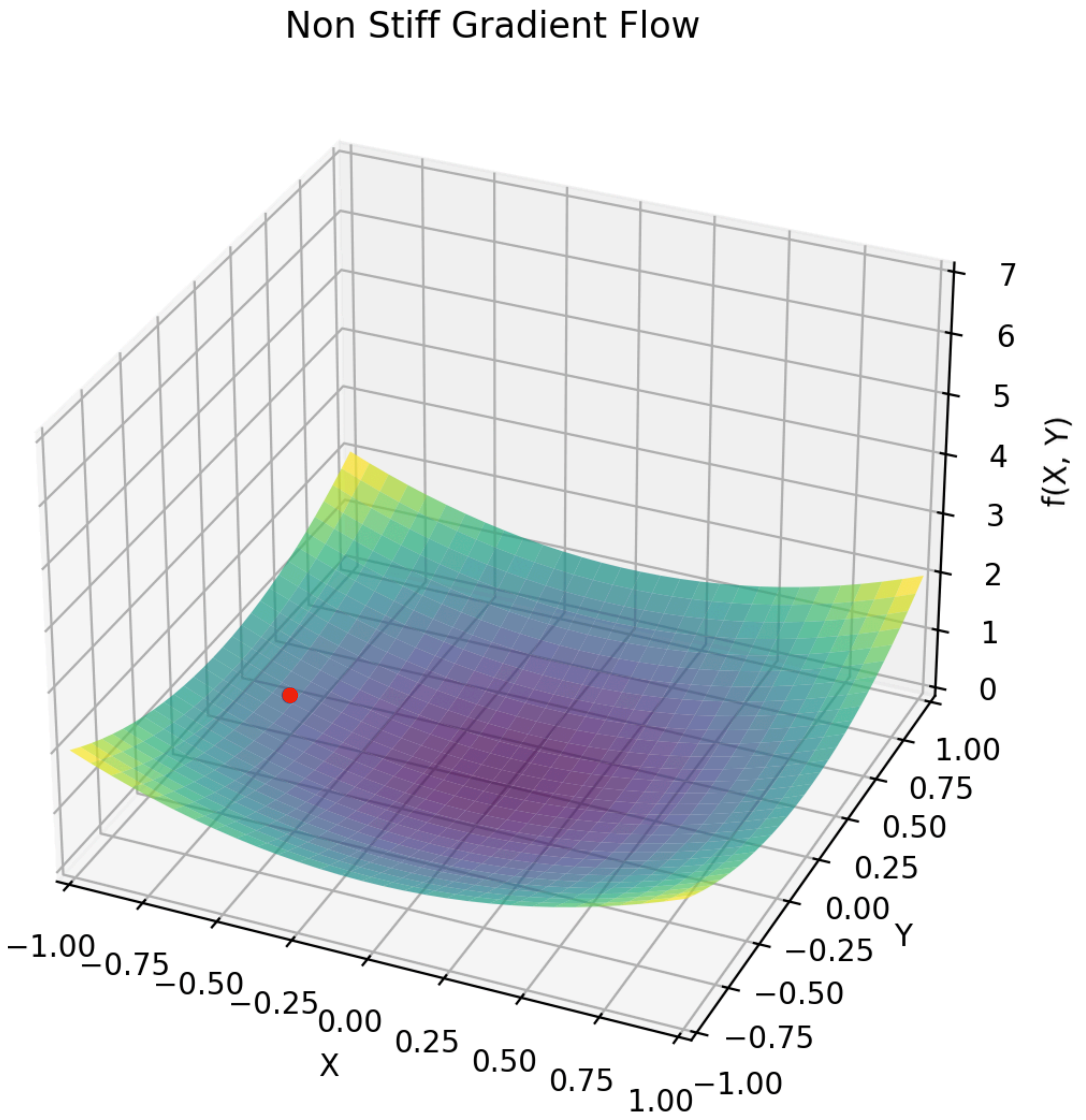
What is stiffness (in gradient flow)?

- Large disparity between eigenvalues, characterized by largest eigenvalue $\sigma_{\max}(\nabla_\theta^2 \mathcal{L}(\theta))$
- Intuitively corresponds to the curvature of the loss function along specific direction
  - High stiffness ➔ highly curved loss function with steep slopes

Stiffness in gradient flow dynamics

Stiffness Example

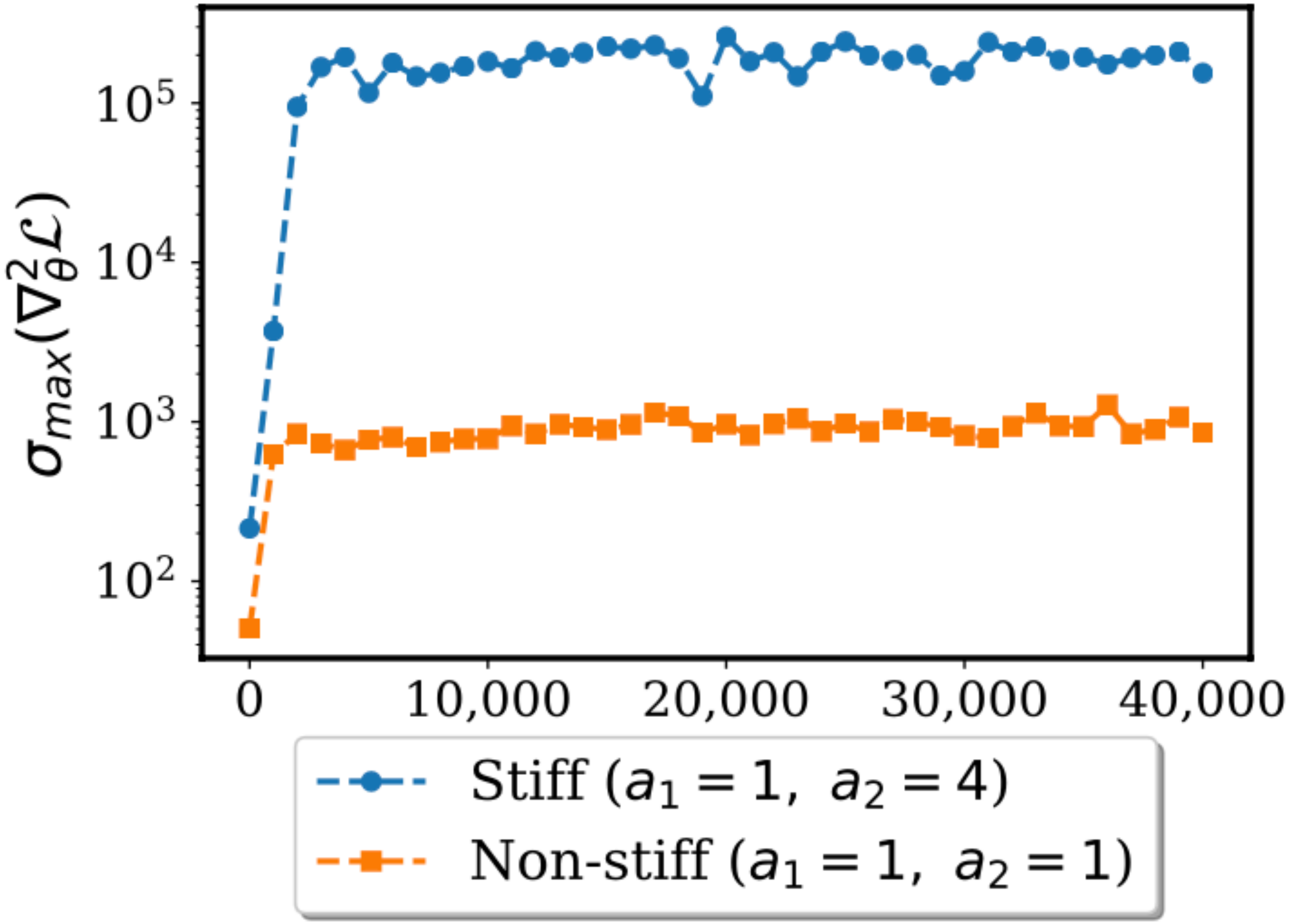




Stiffness in gradient flow dynamics

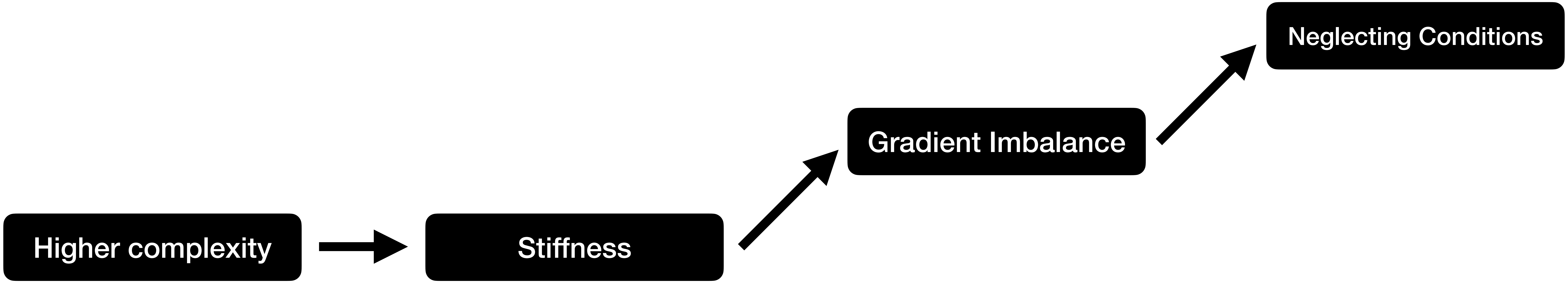
Example Helmholtz Equation

- Simple exact solution:
  $u(x, y) = \sin(a_1 \pi x) \sin(a_2 \pi y)$
- Increasing target complexity increases stiffness



Largest Eigenvalues for the Hessian $\nabla^2_\theta L(\theta)$ during training for different parameters.

The big picture



Stiffness in gradient flow dynamics

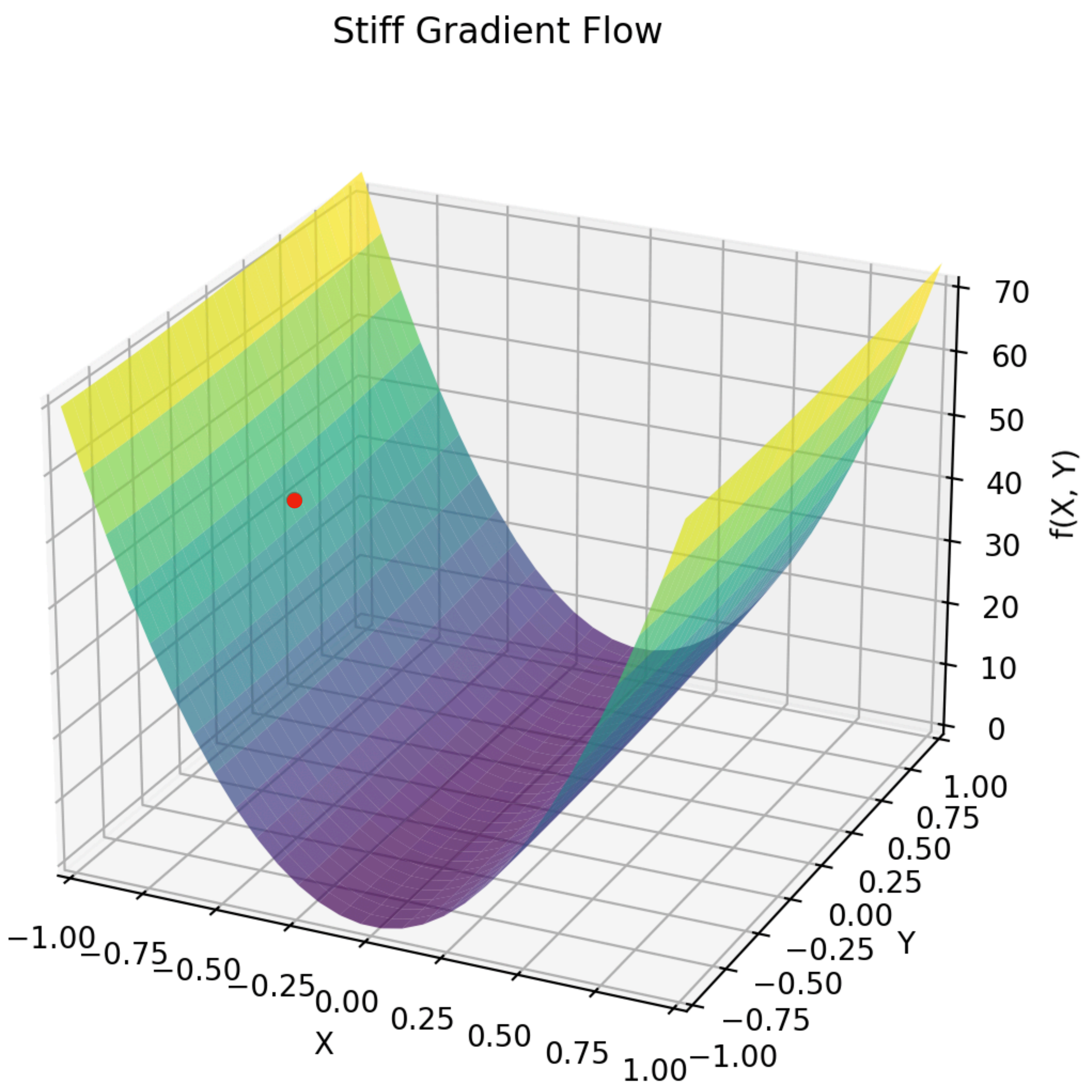
Stiffness consequences

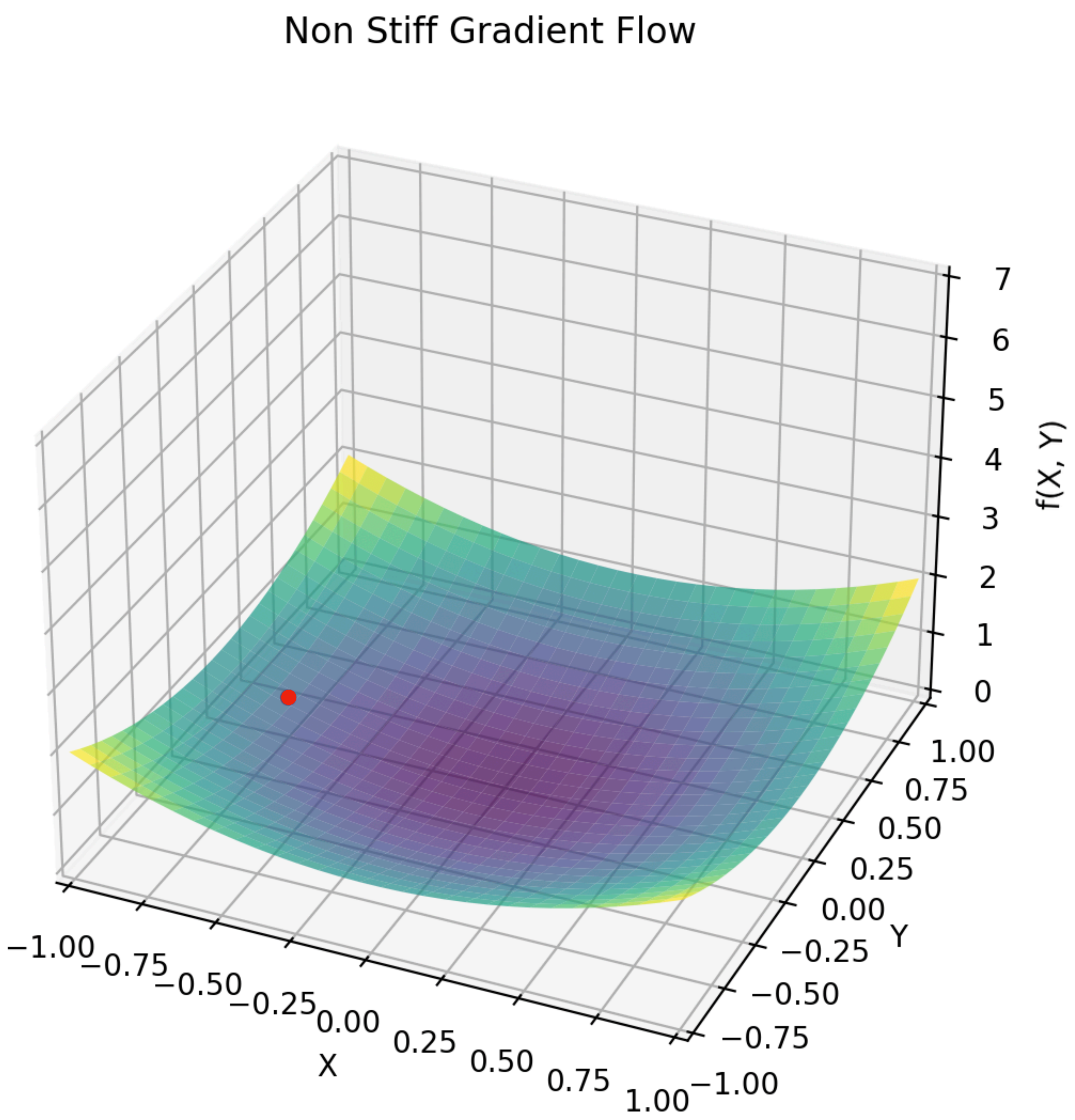
- The consequences of high stiffness
- 1. Small learning rate and slow convergence
    - Conditional stability requires $\eta < 2/\sigma_{\max}(\nabla_\theta^2 \mathcal{L}(\theta))$
- 2. Otherwise gradient descent might fail to decrease loss
    - even if decent direction is correct

Proof(2) in Appendix

Stiffness in gradient flow dynamics

Stiffness Example





The big picture

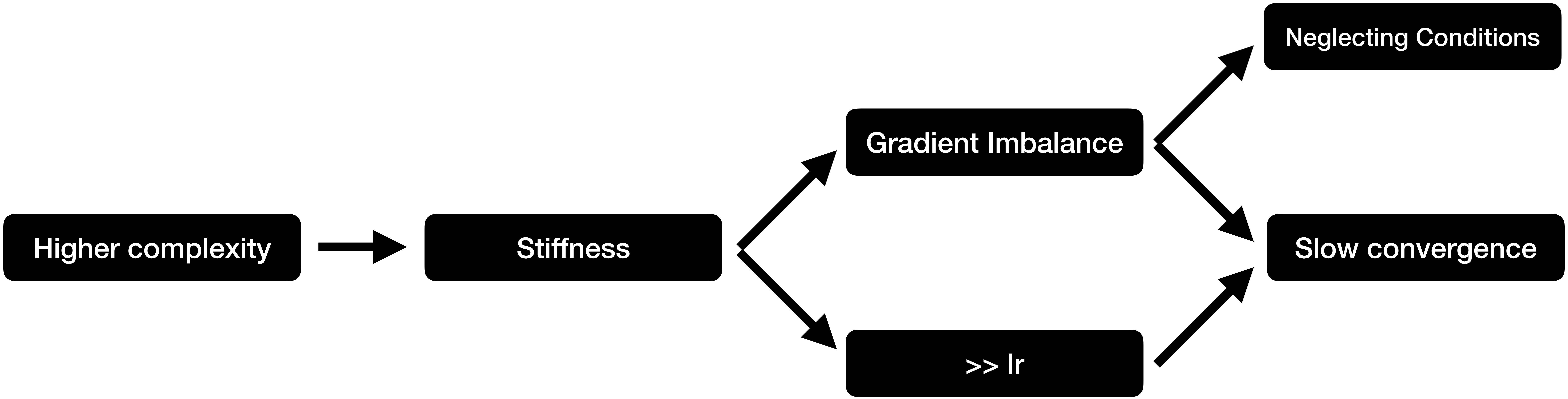


Table of Content

Contributions

- **M1:** Basic Architecture
- **M2:** Basic Architecture + Algorithm
- **M3:** New Architecture
- **M4:** New Architecture + Algorithm

Balance Gradients

Inspiration - Adam Optimizer

- **Momentum:** Use gradient from steps before

  -> smooth gradient

  -> overcome saddle-points

- **RMSProp:** Scale LR based on magnitude of previous gradients

  -> fast in flat areas / slow in steep areas

(RMSProp = Root Mean Square Propagation)

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Balance Losses

Without λ

Loss function:

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

GD update:

$$\begin{aligned}\theta_{n+1} &= \theta_n - \eta \nabla_\theta \mathcal{L}(\theta_n) \\ &= \theta_n - \eta[\nabla_\theta \mathcal{L}_r(\theta_n) + \sum_{i=1}^{M} \nabla_\theta \mathcal{L}_i(\theta_n)]\end{aligned}$$

With λ

Loss function:

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

GD update:

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

Balance Losses

Idea

$$\frac{1}{d}\sum_{j=1}^{d}\left|\frac{\partial \mathcal{L}_i}{\partial \theta_j}\right| = \lambda_i \overline{|\nabla_\theta \mathcal{L}_i(\theta)|} = \max_{\theta_n}\{|\nabla_\theta \mathcal{L}_r(\theta_n)|\} = \max\left\{\left|\frac{\partial \mathcal{L}_r}{\partial \theta_1}\right|, \left|\frac{\partial \mathcal{L}_r}{\partial \theta_2}\right|, \ldots, \left|\frac{\partial \mathcal{L}_r}{\partial \theta_d}\right|\right\}$$

Avg component
of gradient of $\mathcal{L}_i$

Max component
of gradient of $\mathcal{L}_r$

Balance Losses

Idea

$$= \max\left\{ \left|\frac{\partial \mathcal{L}_r}{\partial \theta_1}\right|, \left|\frac{\partial \mathcal{L}_r}{\partial \theta_2}\right|, \ldots, \left|\frac{\partial \mathcal{L}_r}{\partial \theta_d}\right| \right\}$$

Max component of gradient of $\mathcal{L}_r$

$$\lambda_i = \frac{\max_{\theta_n}\{|\nabla_\theta \mathcal{L}_r(\theta_n)|\}}{\overline{|\nabla_\theta \mathcal{L}_i(\theta)|}}$$

$$= \frac{1}{d}\sum_{j=1}^{d} \left|\frac{\partial \mathcal{L}_i}{\partial \theta_j}\right|$$

Avg component of gradient of $\mathcal{L}_i$

Algorithm 1: Learning rate annealing for physics-informed neural networks

Consider a physics-informed neural network $f_\theta(\boldsymbol{x})$ with parameters θ and a loss function

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

where $\mathcal{L}_r(\theta)$ denotes the PDE residual loss, the $\mathcal{L}_i(\theta)$ correspond to data-fit terms (e.g., measurements, initial or boundary conditions, etc.), and $\lambda_i = 1$, $i = 1, \dots, M$ are free parameters used to balance the interplay between the different loss terms. Then use S steps of a gradient descent algorithm to update the parameters θ as:

for $n = 1, \dots, S$ **do**

(a) Compute $\hat{\lambda}_i$ by

$$\hat{\lambda}_i = \frac{\max_\theta\{|\nabla_\theta \mathcal{L}_r(\theta_n)|\}}{\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}}, \quad i = 1, \dots, M, \tag{40}$$

where $\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}$ denotes the mean of $|\nabla_\theta \mathcal{L}_i(\theta_n)|$ with respect to parameters θ.

(b) Update the weights λ_i using a moving average of the form

$$\lambda_i = (1 - \alpha)\lambda_i + \alpha \hat{\lambda}_i, \quad i = 1, \dots, M. \tag{41}$$

(c) Update the parameters θ via gradient descent

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}_r(\theta_n) - \eta \sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta_n) \tag{42}$$

end

The recommended hyper-parameter values are: $\eta = 10^{-3}$ and $\alpha = 0.9$.

Algorithm 1: Learning rate annealing for physics-informed neural networks

Consider a physics-informed neural network $f_\theta(\boldsymbol{x})$ with parameters θ and a loss function

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

where $\mathcal{L}_r(\theta)$ denotes the PDE residual loss, the $\mathcal{L}_i(\theta)$ correspond to data-fit terms (e.g., measurements, initial or boundary conditions, etc.), and $\lambda_i = 1,\ i = 1, \ldots, M$ are free parameters used to balance the interplay between the different loss terms. Then use S steps of a gradient descent algorithm to update the parameters θ as:

for $n = 1, \ldots, S$ **do**

(a) Compute $\hat{\lambda}_i$ by

$$\hat{\lambda}_i = \frac{\max_\theta\{|\nabla_\theta \mathcal{L}_r(\theta_n)|\}}{\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}}, \quad i = 1, \ldots, M, \tag{40}$$

where $\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}$ denotes the mean of $|\nabla_\theta \mathcal{L}_i(\theta_n)|$ with respect to parameters θ.

(b) Update the weights λ_i using a moving average of the form

$$\lambda_i = (1-\alpha)\lambda_i + \alpha\hat{\lambda}_i, \quad i = 1, \ldots, M. \tag{41}$$

(c) Update the parameters θ via gradient descent

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}_r(\theta_n) - \eta \sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta_n) \tag{42}$$

end

The recommended hyper-parameter values are: $\eta = 10^{-3}$ and $\alpha = 0.9$.

Algorithm 1: Learning rate annealing for physics-informed neural networks

Consider a physics-informed neural network $f_\theta(x)$ with parameters θ and a loss function

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

where $\mathcal{L}_r(\theta)$ denotes the PDE residual loss, the $\mathcal{L}_i(\theta)$ correspond to data-fit terms (e.g., measurements, initial or boundary conditions, etc.), and $\lambda_i = 1,\ i = 1, \dots, M$ are free parameters used to balance the interplay between the different loss terms. Then use S steps of a gradient descent algorithm to update the parameters θ as:

for $n = 1, \dots, S$ **do**

(a) Compute $\hat{\lambda}_i$ by

$$\hat{\lambda}_i = \frac{\max_\theta\{|\nabla_\theta \mathcal{L}_r(\theta_n)|\}}{\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}},\quad i = 1, \dots, M, \tag{40}$$

where $\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}$ denotes the mean of $|\nabla_\theta \mathcal{L}_i(\theta_n)|$ with respect to parameters θ.

(b) Update the weights λ_i using a moving average of the form

$$\lambda_i = (1-\alpha)\lambda_i + \alpha\hat{\lambda}_i,\quad i = 1, \dots, M. \tag{41}$$

(c) Update the parameters θ via gradient descent

$$\theta_{n+1} = \theta_n - \eta\nabla_\theta \mathcal{L}_r(\theta_n) - \eta\sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta_n) \tag{42}$$

end

The recommended hyper-parameter values are: $\eta = 10^{-3}$ and $\alpha = 0.9$.

Algorithm 1: Learning rate annealing for physics-informed neural networks

Consider a physics-informed neural network $f_\theta(x)$ with parameters θ and a loss function

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

where $\mathcal{L}_r(\theta)$ denotes the PDE residual loss, the $\mathcal{L}_i(\theta)$ correspond to data-fit terms (e.g., measurements, initial or boundary conditions, etc.), and $\lambda_i = 1$, $i = 1, \ldots, M$ are free parameters used to balance the interplay between the different loss terms. Then use S steps of a gradient descent algorithm to update the parameters θ as:

for $n = 1, \ldots, S$ **do**

(a) Compute $\hat{\lambda}_i$ by

$$\hat{\lambda}_i = \frac{\max_\theta\{|\nabla_\theta \mathcal{L}_r(\theta_n)|\}}{\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}}, \quad i = 1, \ldots, M, \tag{40}$$

where $\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}$ denotes the mean of $|\nabla_\theta \mathcal{L}_i(\theta_n)|$ with respect to parameters θ.

(b) Update the weights λ_i using a moving average of the form

$$\lambda_i = (1-\alpha)\lambda_i + \alpha\hat{\lambda}_i, \quad i = 1, \ldots, M. \tag{41}$$

(c) Update the parameters θ via gradient descent

$$\theta_{n+1} = \theta_n - \eta\nabla_\theta \mathcal{L}_r(\theta_n) - \eta\sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta_n) \tag{42}$$

end

The recommended hyper-parameter values are: $\eta = 10^{-3}$ and $\alpha = 0.9$.

Algorithm 1: Learning rate annealing for physics-informed neural networks

Consider a physics-informed neural network $f_\theta(x)$ with parameters θ and a loss function

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

where $\mathcal{L}_r(\theta)$ denotes the PDE residual loss, the $\mathcal{L}_i(\theta)$ correspond to data-fit terms (e.g., measurements, initial or boundary conditions, etc.), and $\lambda_i = 1$, $i = 1, \ldots, M$ are free parameters used to balance the interplay between the different loss terms. Then use S steps of a gradient descent algorithm to update the parameters θ as:

for $n = 1, \ldots, S$ **do**

(a) Compute $\hat{\lambda}_i$ by

$$\hat{\lambda}_i = \frac{\max_\theta\{|\nabla_\theta \mathcal{L}_r(\theta_n)|\}}{\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}}, \quad i = 1, \ldots, M, \tag{40}$$

where $\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}$ denotes the mean of $|\nabla_\theta \mathcal{L}_i(\theta_n)|$ with respect to parameters θ.

(b) Update the weights λ_i using a moving average of the form

$$\lambda_i = (1-\alpha)\lambda_i + \alpha\hat{\lambda}_i, \quad i = 1, \ldots, M. \tag{41}$$

(c) Update the parameters θ via gradient descent

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}_r(\theta_n) - \eta \sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta_n) \tag{42}$$

end

The recommended hyper-parameter values are: $\eta = 10^{-3}$ and $\alpha = 0.9$.

Algorithm 1: Learning rate annealing for physics-informed neural networks

Consider a physics-informed neural network $f_\theta(x)$ with parameters θ and a loss function

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

where $\mathcal{L}_r(\theta)$ denotes the PDE residual loss, the $\mathcal{L}_i(\theta)$ correspond to data-fit terms (e.g., measurements, initial or boundary conditions, etc.), and $\lambda_i = 1$, $i = 1, \dots, M$ are free parameters used to balance the interplay between the different loss terms. Then use S steps of a gradient descent algorithm to update the parameters θ as:

for $n = 1, \dots, S$ **do**

(a) Compute $\hat{\lambda}_i$ by

$$\hat{\lambda}_i = \frac{\max_\theta\{|\nabla_\theta \mathcal{L}_r(\theta_n)|\}}{\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}}, \quad i = 1, \dots, M, \tag{40}$$

where $\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}$ denotes the mean of $|\nabla_\theta \mathcal{L}_i(\theta_n)|$ with respect to parameters θ.

(b) Update the weights λ_i using a moving average of the form

$$\lambda_i = (1-\alpha)\lambda_i + \alpha\hat{\lambda}_i, \quad i = 1, \dots, M. \tag{41}$$

(c) Update the parameters θ via gradient descent

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}_r(\theta_n) - \eta \sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta_n) \tag{42}$$

end

The recommended hyper-parameter values are: $\eta = 10^{-3}$ and $\alpha = 0.9$.

Algorithm 1: Learning rate annealing for physics-informed neural networks

Consider a physics-informed neural network $f_\theta(x)$ with parameters θ and a loss function

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

where $\mathcal{L}_r(\theta)$ denotes the PDE residual loss, the $\mathcal{L}_i(\theta)$ correspond to data-fit terms (e.g., measurements, initial or boundary conditions, etc.), and $\lambda_i = 1, i = 1, \ldots, M$ are free parameters used to balance the interplay between the different loss terms. Then use S steps of a gradient descent algorithm to update the parameters θ as:

for $n = 1, \ldots, S$ **do**

(a) Compute $\hat{\lambda}_i$ by

$$\hat{\lambda}_i = \frac{\max_\theta\{|\nabla_\theta \mathcal{L}_r(\theta_n)|\}}{\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}}, \quad i = 1, \ldots, M, \tag{40}$$

where $\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}$ denotes the mean of $|\nabla_\theta \mathcal{L}_i(\theta_n)|$ with respect to parameters θ.

(b) Update the weights λ_i using a moving average of the form

$$\lambda_i = (1 - \alpha)\lambda_i + \alpha\hat{\lambda}_i, \quad i = 1, \ldots, M. \tag{41}$$

(c) Update the parameters θ via gradient descent

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}_r(\theta_n) - \eta \sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta_n) \tag{42}$$

end

The recommended hyper-parameter values are: $\eta = 10^{-3}$ and $\alpha = 0.9$.

Algorithm 1: Learning rate annealing for physics-informed neural networks

Consider a physics-informed neural network $f_\theta(x)$ with parameters θ and a loss function

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

where $\mathcal{L}_r(\theta)$ denotes the PDE residual loss, the $\mathcal{L}_i(\theta)$ correspond to data-fit terms (e.g., measurements, initial or boundary conditions, etc.), and $\lambda_i = 1$, $i = 1, \ldots, M$ are free parameters used to balance the interplay between the different loss terms. Then use S steps of a gradient descent algorithm to update the parameters θ as:

for $n = 1, \ldots, S$ **do**

(a) Compute $\hat{\lambda}_i$ by

$$\hat{\lambda}_i = \frac{\max_\theta\{|\nabla_\theta \mathcal{L}_r(\theta_n)|\}}{\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}}, \quad i = 1, \ldots, M, \tag{40}$$

where $\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}$ denotes the mean of $|\nabla_\theta \mathcal{L}_i(\theta_n)|$ with respect to parameters θ.

(b) Update the weights λ_i using a moving average of the form

$$\lambda_i = (1 - \alpha)\lambda_i + \alpha\hat{\lambda}_i, \quad i = 1, \ldots, M. \tag{41}$$

(c) Update the parameters θ via gradient descent

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}_r(\theta_n) - \eta \sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta_n) \tag{42}$$

end

The recommended hyper-parameter values are: $\eta = 10^{-3}$ and $\alpha = 0.9$.

Algorithm 1: Learning rate annealing for physics-informed neural networks

Consider a physics-informed neural network $f_\theta(\boldsymbol{x})$ with parameters θ and a loss function

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

where $\mathcal{L}_r(\theta)$ denotes the PDE residual loss, the $\mathcal{L}_i(\theta)$ correspond to data-fit terms (e.g., measurements, initial or boundary conditions, etc.), and $\lambda_i = 1$, $i = 1, \dots, M$ are free parameters used to balance the interplay between the different loss terms. Then use S steps of a gradient descent algorithm to update the parameters θ as:

for $n = 1, \dots, S$ **do**

(a) Compute $\hat{\lambda}_i$ by

$$\hat{\lambda}_i = \frac{\max_\theta\{|\nabla_\theta \mathcal{L}_r(\theta_n)|\}}{\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}}, \quad i = 1, \dots, M, \tag{40}$$

where $\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}$ denotes the mean of $|\nabla_\theta \mathcal{L}_i(\theta_n)|$ with respect to parameters θ.

(b) Update the weights λ_i using a moving average of the form

$$\lambda_i = (1-\alpha)\lambda_i + \alpha\hat{\lambda}_i, \quad i = 1, \dots, M. \tag{41}$$

(c) Update the parameters θ via gradient descent

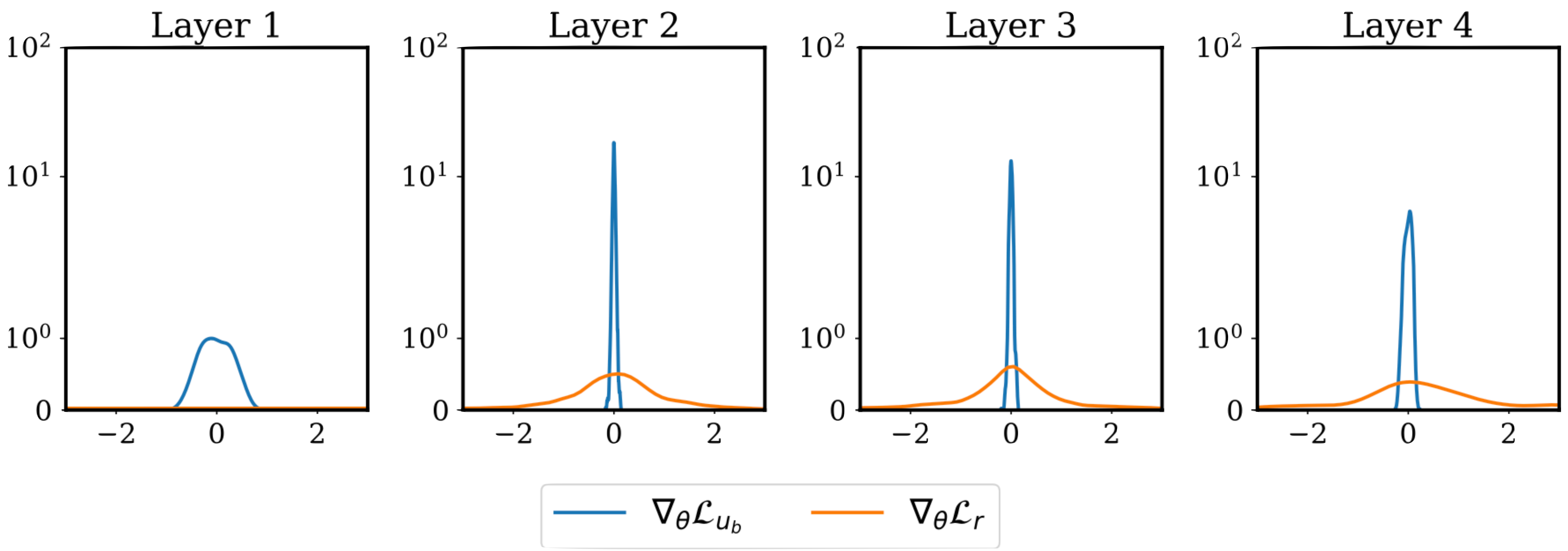
$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}_r(\theta_n) - \eta \sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta_n) \tag{42}$$

end

The recommended hyper-parameter values are: $\eta = 10^{-3}$ and $\alpha = 0.9$.

Results: M1 vs M2

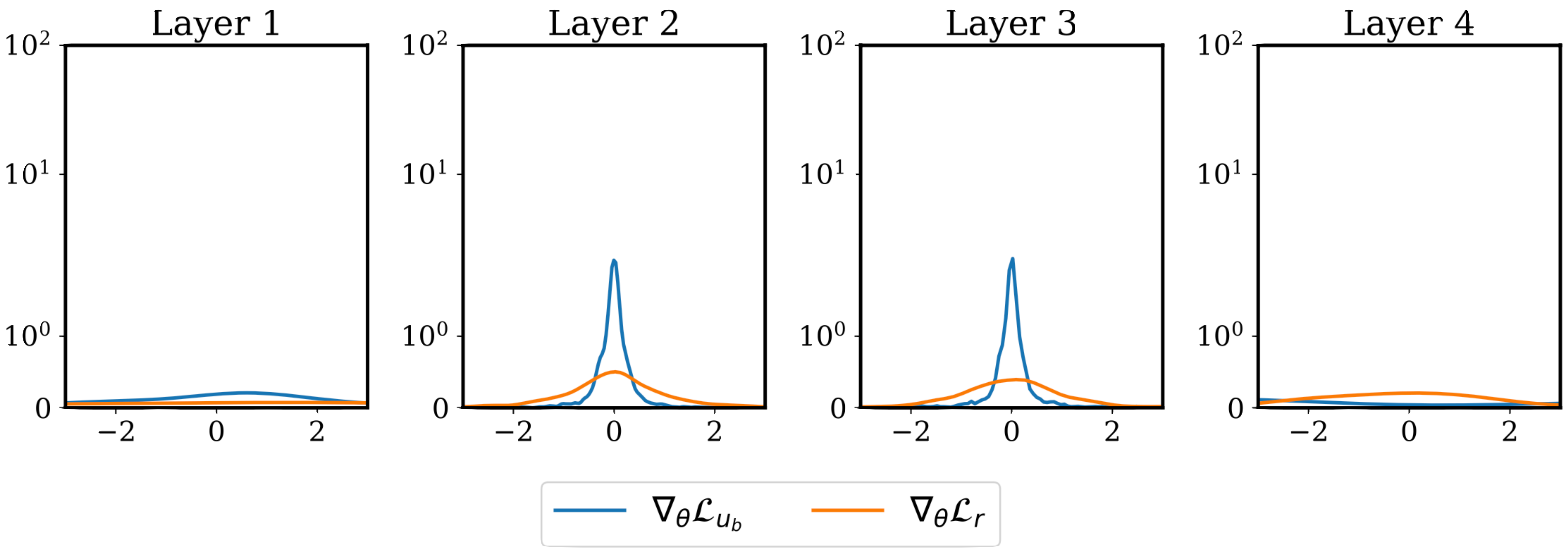
Helmholtz



M1:

$\nabla_\theta \mathcal{L}_{u_b}(\theta)$ spikes at 0

➡ Imbalanced



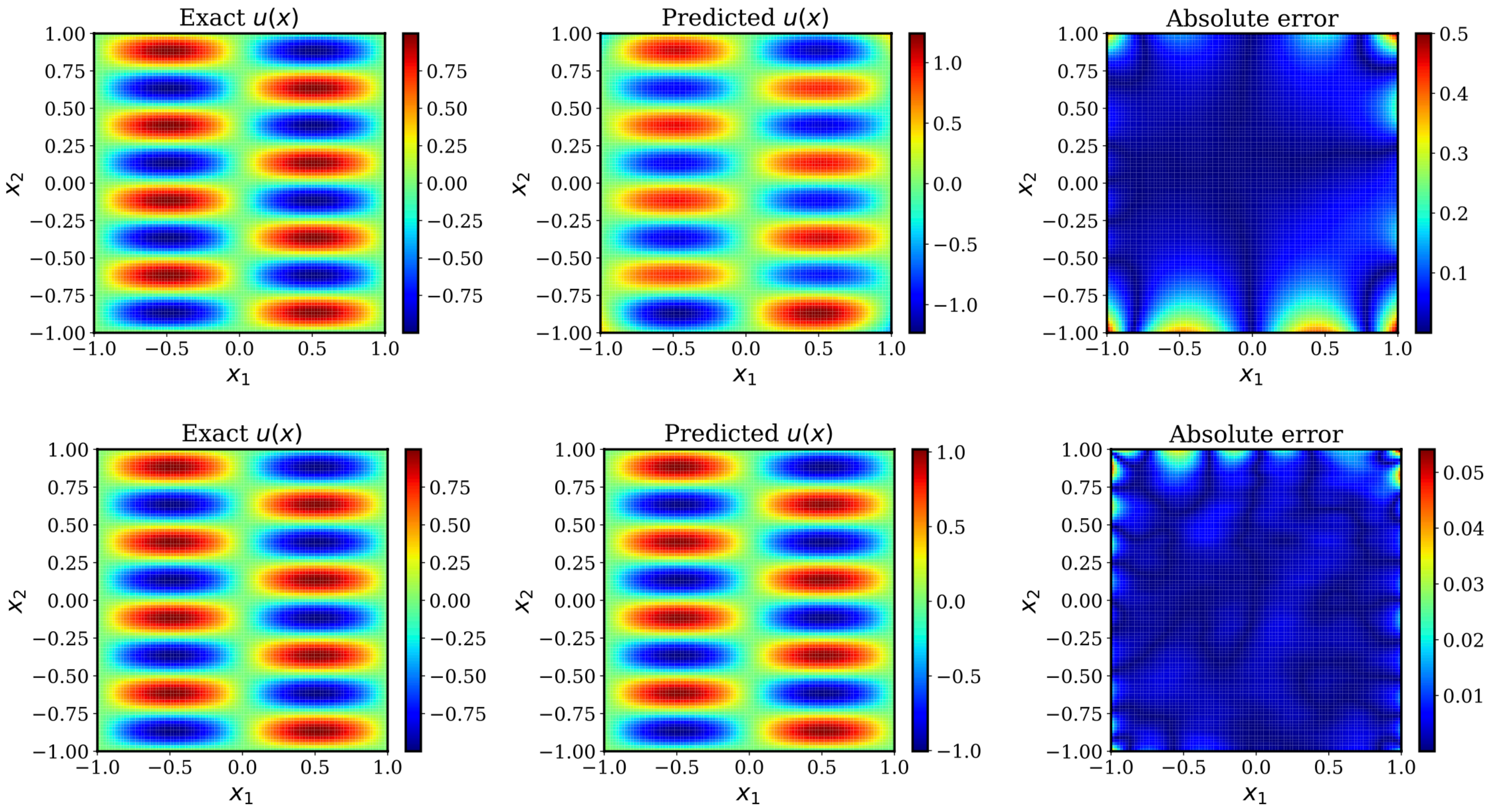
M2:

$\nabla_\theta \mathcal{L}_{u_b}(\theta)$ more spread out

➡ More balanced

Results: M1 vs M2

Helmholtz



- 10x less error than M1

Novel Architecture

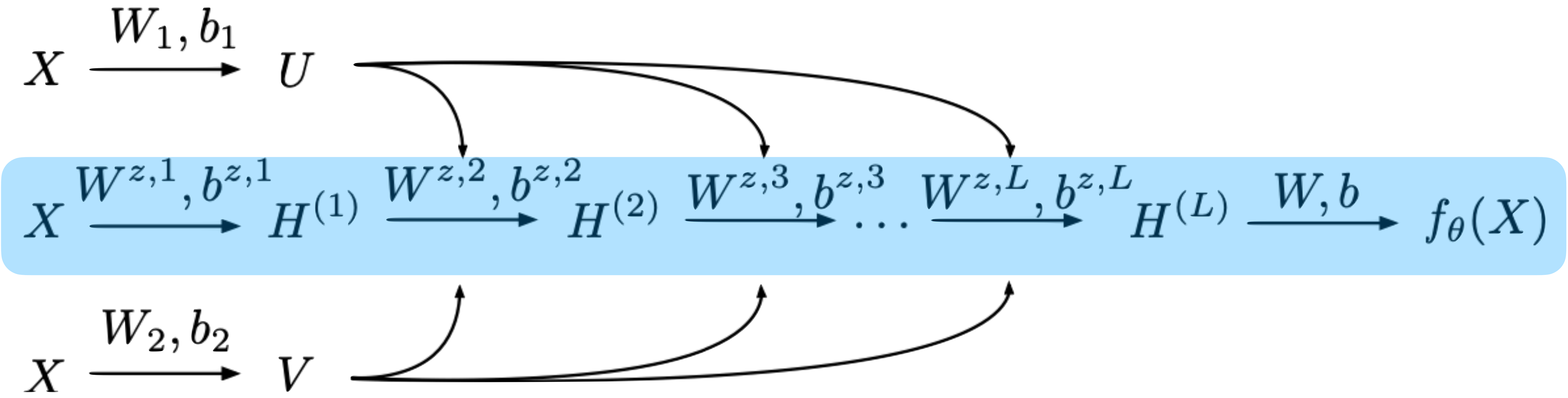
$$X \xrightarrow{W_1, b_1} U$$

$$X \xrightarrow{W^{z,1}, b^{z,1}} H^{(1)} \xrightarrow{W^{z,2}, b^{z,2}} H^{(2)} \xrightarrow{W^{z,3}, b^{z,3}} \cdots \xrightarrow{W^{z,L}, b^{z,L}} H^{(L)} \xrightarrow{W, b} f_\theta(X)$$

$$X \xrightarrow{W_2, b_2} V$$

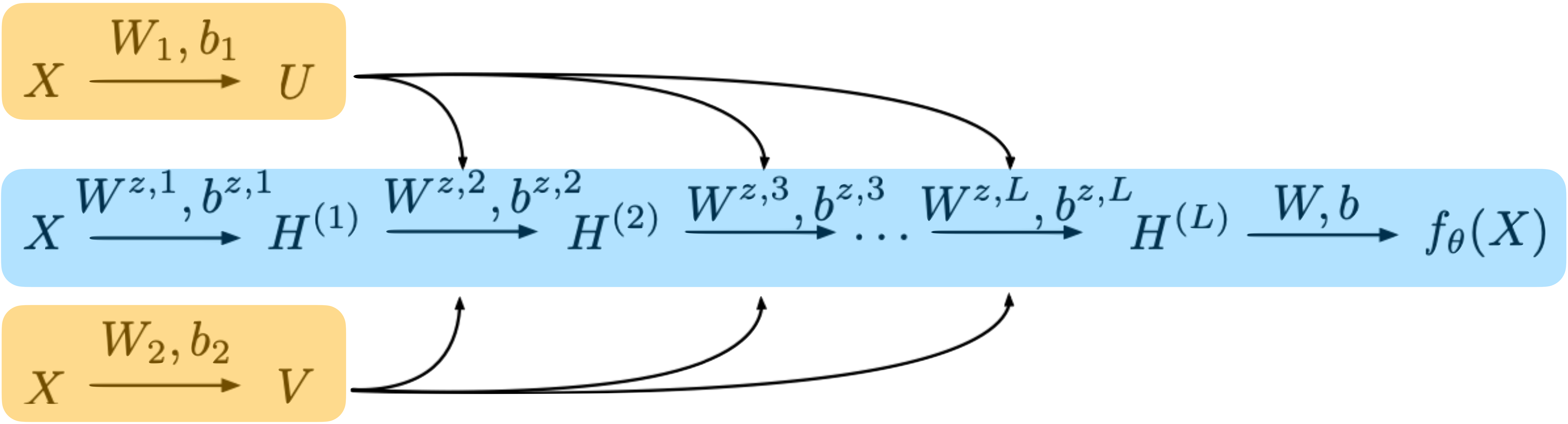
Novel Architecture

- **Fully Connected NN:** as in **M1**

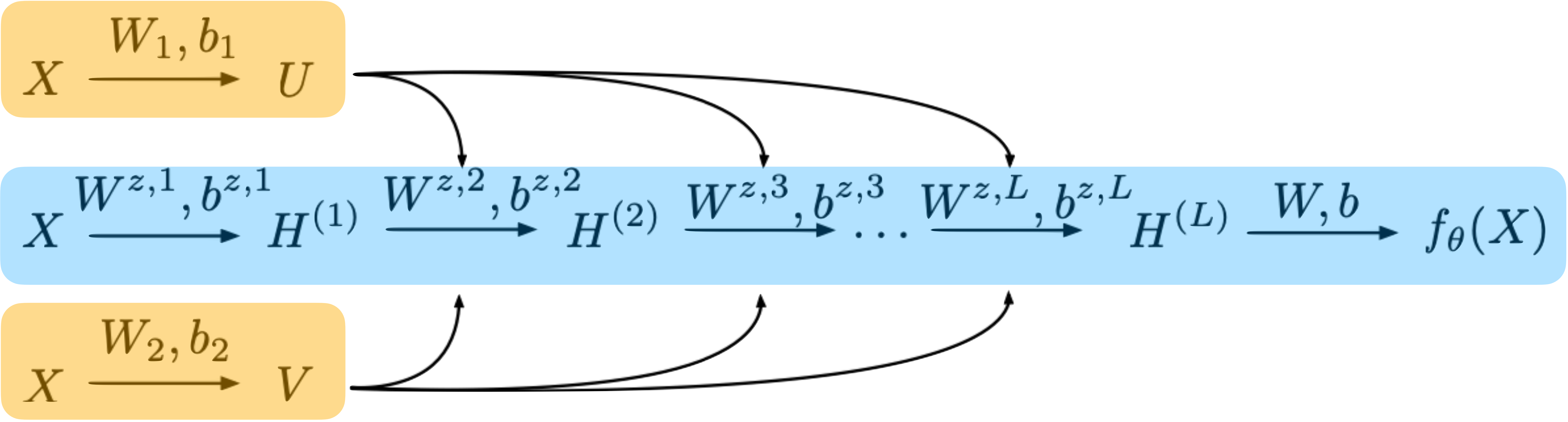


Novel Architecture

- **Fully Connected NN:** as in **M1**
- **Encoders:** Transforms input into high-dimensional embedding



Novel Architecture



$$U = \phi(XW^1 + b^1), \quad V = \phi(XW^2 + b^2),$$

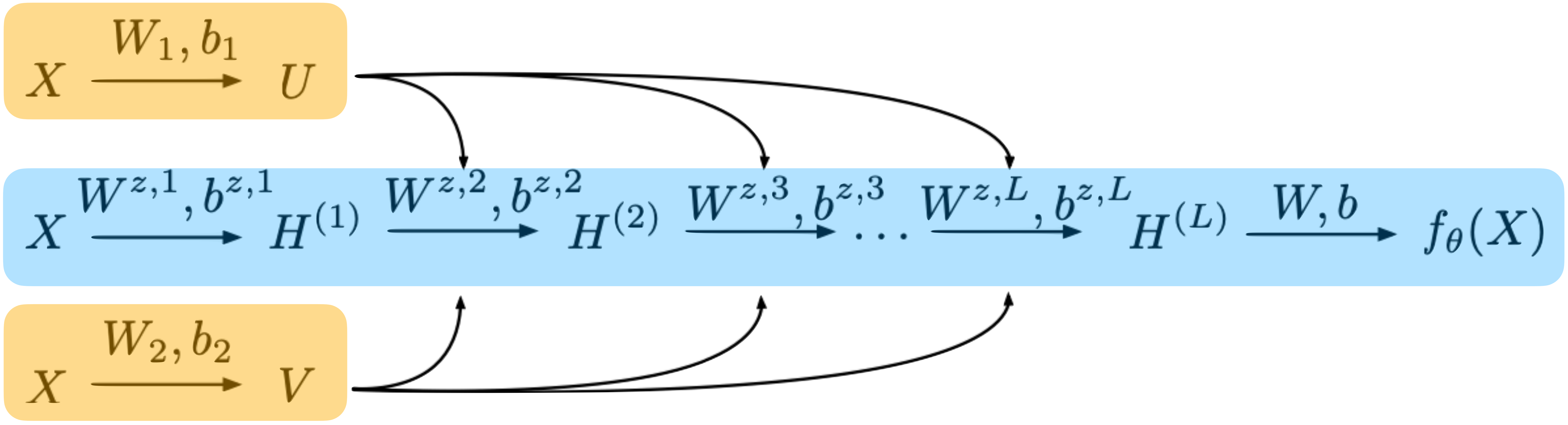
$$H^{(1)} = \phi(XW^{z,1} + b^{z,1}),$$

$$Z^{(k)} = \phi(H^{(k)}W^{z,k} + b^{z,k}), \quad k = 1, \ldots, L,$$

$$H^{(k+1)} = (1 - Z^{(k)}) \odot U + Z^{(k)} \odot V, \quad k = 1, \ldots, L,$$

$$f_\theta(x) = H^{(L+1)}W + b,$$

Novel Architecture



$$U = \phi(XW^1 + b^1), \quad V = \phi(XW^2 + b^2),$$

$$H^{(1)} = \phi(XW^{z,1} + b^{z,1}),$$

$$Z^{(k)} = \phi(H^{(k)}W^{z,k} + b^{z,k}), \quad k = 1, \ldots, L,$$

$$H^{(k+1)} = (1 - Z^{(k)}) \odot U + Z^{(k)} \odot V, \quad k = 1, \ldots, L,$$

$$f_\theta(x) = H^{(L+1)}W + b,$$

Novel Architecture

$$
\begin{tikzcd}
X \arrow[r, "{W_1, b_1}"] & U \\
X \arrow[r, "{W^{z,1}, b^{z,1}}"] & H^{(1)} \arrow[r, "{W^{z,2}, b^{z,2}}"] & H^{(2)} \arrow[r, "{W^{z,3}, b^{z,3}}"] & \cdots \arrow[r, "{W^{z,L}, b^{z,L}}"] & H^{(L)} \arrow[r, "{W, b}"] & f_\theta(X) \\
X \arrow[r, "{W_2, b_2}"] & V
\end{tikzcd}
$$

$$U = \phi(XW^1 + b^1), \quad V = \phi(XW^2 + b^2),$$

$$H^{(1)} = \phi(XW^{z,1} + b^{z,1}),$$

$$Z^{(k)} = \phi(H^{(k)}W^{z,k} + b^{z,k}), \quad k = 1, \dots, L,$$

$$H^{(k+1)} = (1 - Z^{(k)}) \odot U + Z^{(k)} \odot V, \quad k = 1, \dots, L,$$

$$f_\theta(x) = H^{(L+1)}W + b,$$

Novel Architecture

- **Multiplicative Interactions:** accounts for multiplicative relations among different input dimensions

$$U = \phi(XW^1 + b^1), \quad V = \phi(XW^2 + b^2),$$
$$H^{(1)} = \phi(XW^{z,1} + b^{z,1}),$$
$$Z^{(k)} = \phi(H^{(k)}W^{z,k} + b^{z,k}), \quad k = 1, \dots, L,$$
$$H^{(k+1)} = (1 - Z^{(k)}) \odot U + Z^{(k)} \odot V, \quad k = 1, \dots, L,$$
$$f_\theta(x) = H^{(L+1)}W + b,$$

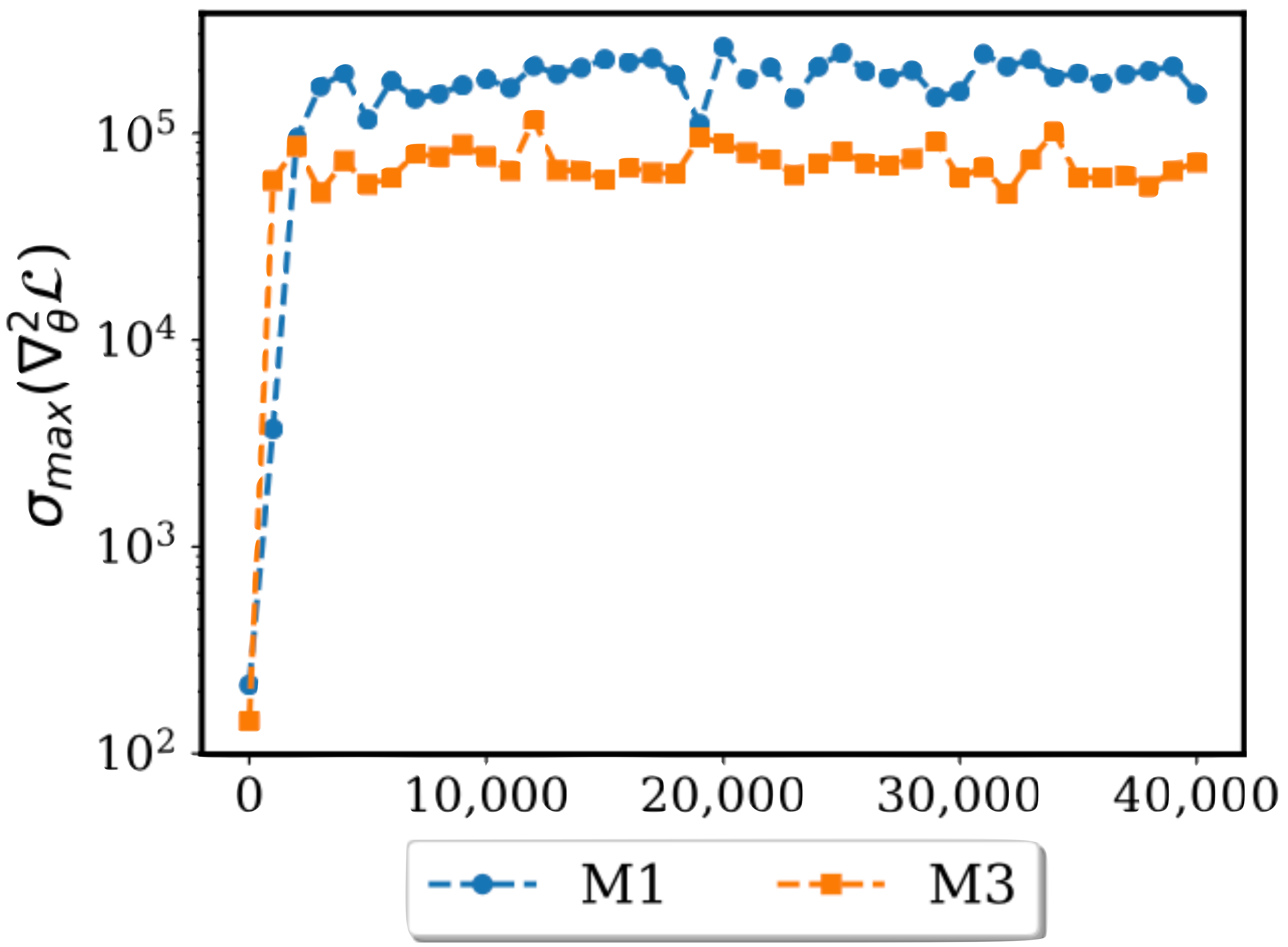
Novel Architecture

- **Multiplicative Interactions**: accounts for multiplicative relations among different input dimensions
- **Residual Connections**: enhances hidden states -> less vanishing gradient

$$
\begin{aligned}
&U = \phi(XW^1 + b^1), \quad V = \phi(XW^2 + b^2), \\
&H^{(1)} = \phi(XW^{z,1} + b^{z,1}), \\
&Z^{(k)} = \phi(H^{(k)} W^{z,k} + b^{z,k}), \quad k = 1, \dots, L, \\
&H^{(k+1)} = (1 - Z^{(k)}) \odot U + Z^{(k)} \odot V, \quad k = 1, \dots, L, \\
&f_\theta(x) = H^{(L+1)} W + b,
\end{aligned}
$$

Results: M1 vs M3

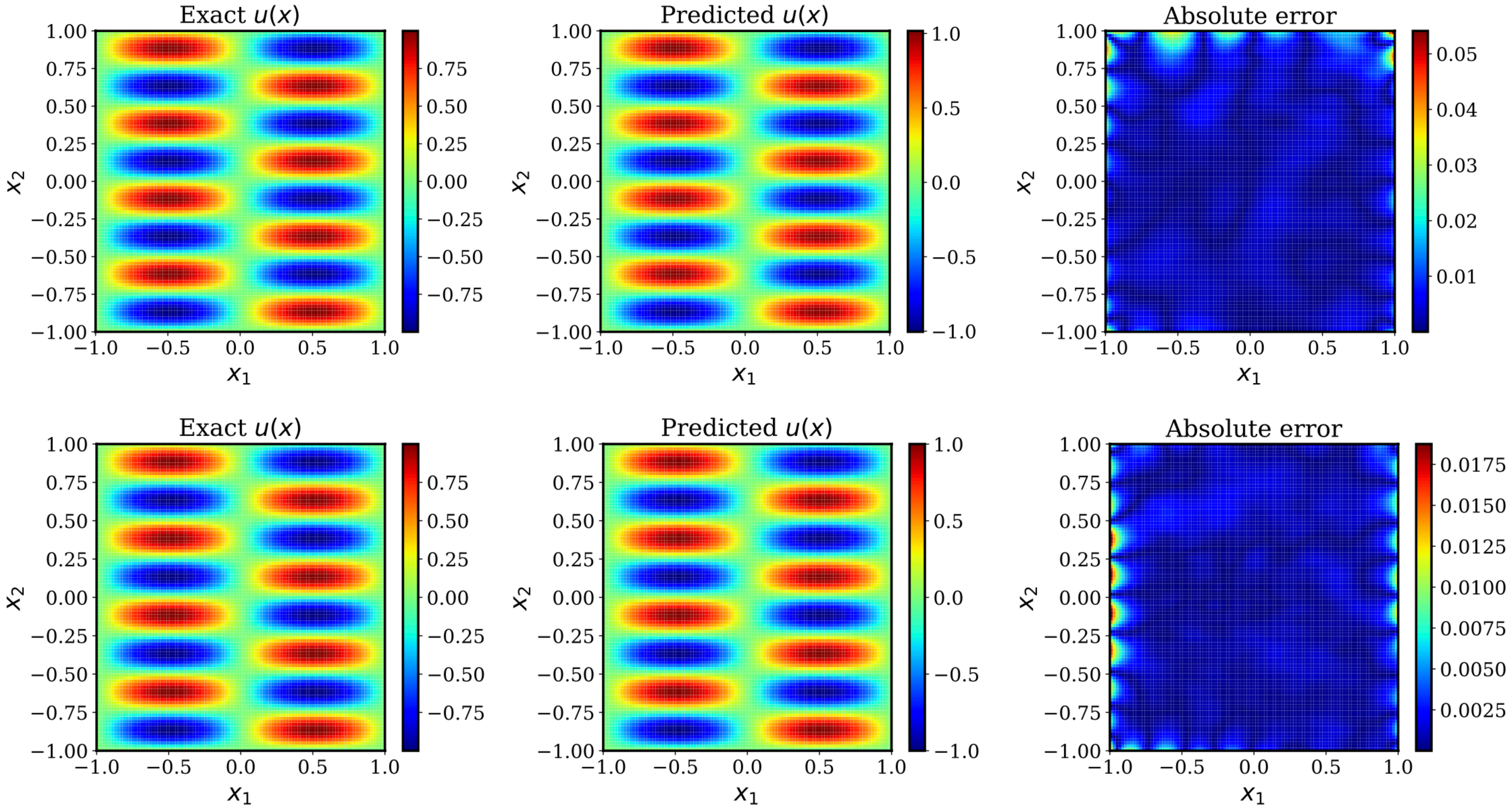
Helmholtz



➡ Less stiffness

Results: M2 vs M4

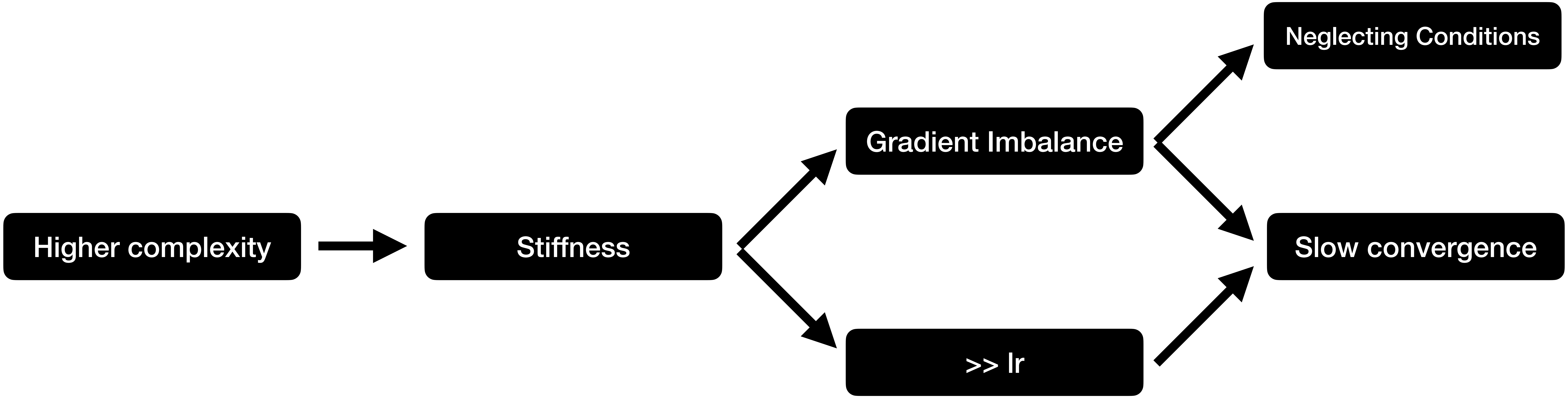
Helmholtz



- 3x less error than M2
- 30x less error than M1

Table of Content

The big picture



Summary

- Loss terms of different nature cause imbalanced gradients
- Adaptive learning rates balance different terms in loss function
- Novel architectures can prevent gradient-related pathologies
- Loss is reduced by a factor of 50-100x across many problems.
- Developments generalizable to other tasks with multiple objective functions
- Still at very early stages of understanding the capabilities and limitations

Outlook

Open questions, Further Research needed

- Exact relation unknown: PDE stiffness <-> Gradient Flow stiffness
- Can gradient flow stiffness be reduced?
  (e.g. using domain decomposition techniques, different choices of loss functions, more effective neural architectures, etc.)
- If stiffness turns out to be an inherent property of PINNs, what else can we do to enhance the robustness of their training and the accuracy of their predictions?
- Can we devise more stable and effective optimization algorithms to train PINN models with stiff gradient flow dynamics?
- How does stiffness affect the approximation error and generalization error of PINNs?

Discussion

Our opinion:

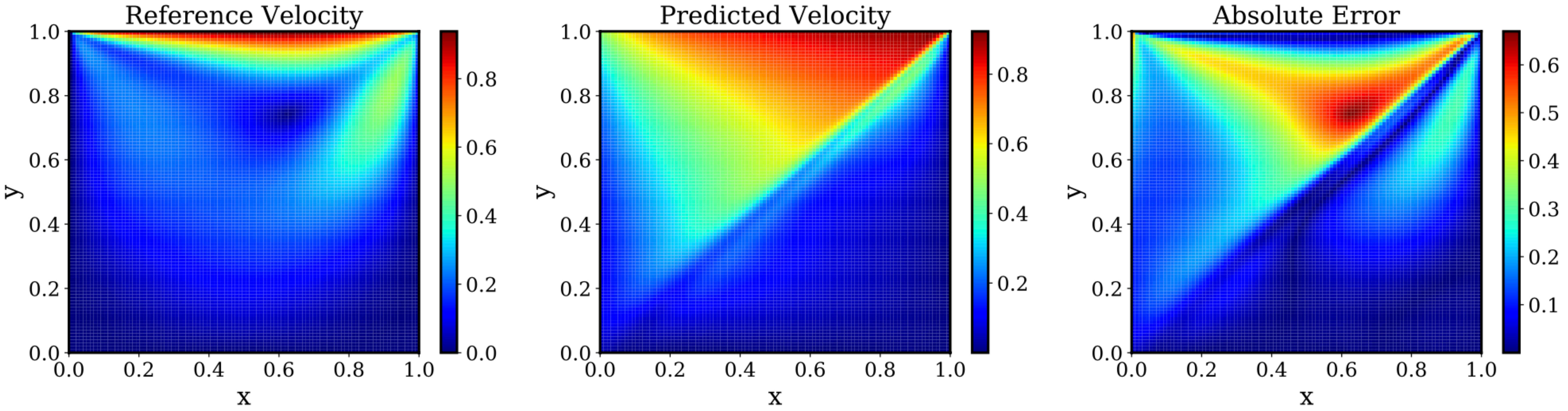
	--	-	o	+	++
• Understandability:			▯		
• Novelty:					▯
• Replicability:					▯
• Relevance:					▯

Thank you!

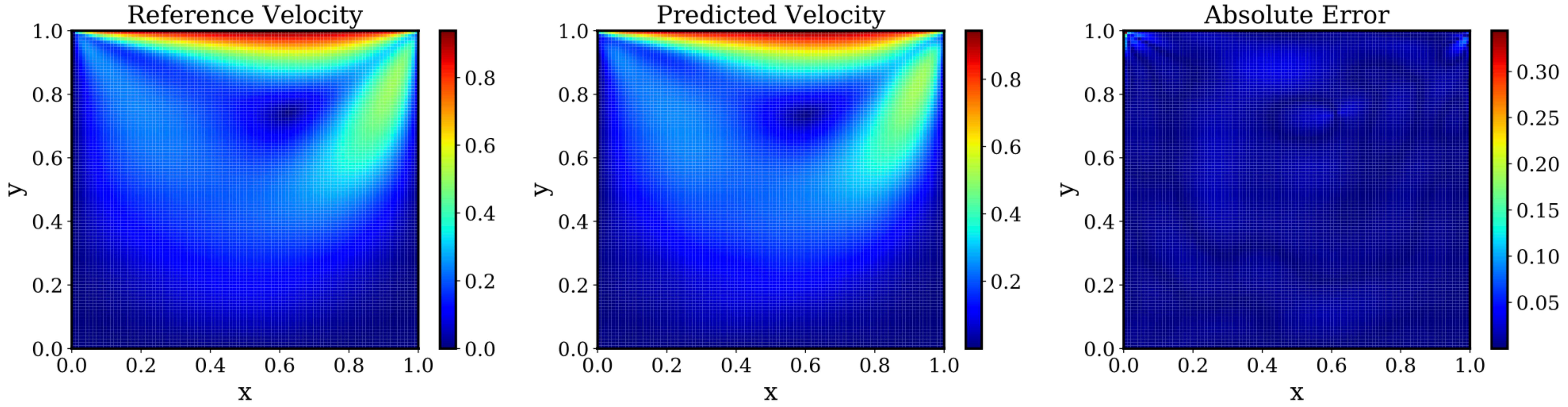
Questions?

Bonus Slide: How to set up a PINN

Flow in a lid driven cavity



(d) Model M4 (relative L^2 prediction error: 7.53e-01).



(d) Model M4 (relative L^2 prediction error: 3.42e-02).

Appendix for further detail

Proof(1) Gradient Analysis

Proof. Recall that the loss function is given by

$$\begin{aligned}\mathcal{L}(\theta) &= \mathcal{L}_r(\theta) + \mathcal{L}_{u_b}(\theta) \\ &= \frac{1}{N_b}\sum_{i=1}^{N_b}[f_\theta(x_b^i) - h(x_b^i)]^2 + \frac{1}{N_r}\sum_{i=1}^{N_r}[\frac{\partial^2}{\partial x^2} f_\theta(x_r^i) - g(x_r^i)]^2.\end{aligned} \tag{75}$$

Here we fix $\theta \in \Theta$, where Θ denote all weights in a neural network. Then by assumptions, $\frac{\partial \mathcal{L}_{u_b}(\theta)}{\partial \theta}$ can be computed by

$$\begin{aligned}\left|\frac{\partial \mathcal{L}_{u_b}(\theta)}{\partial \theta}\right| &= \left|\frac{\partial}{\partial \theta}\left(\frac{1}{2}\sum_{i=1}^{2}\left(u_\theta\left(x_b^i\right) - h\left(x_b^i\right)\right)^2\right)\right| \\ &= \left|\sum_{i=1}^{2}\left(u_\theta\left(x_b^i\right) - h\left(x_b^i\right)\right)\frac{\partial u_\theta\left(x_b^i\right)}{\partial \theta}\right| \\ &= \left|\sum_{i=1}^{2}\left(u\left(x_b^i\right)\cdot \epsilon_\theta\left(x_b^i\right) - u\left(x_b^i\right)\right) u\left(x_b^i\right)\frac{\partial \epsilon_\theta\left(x_b^i\right)}{\partial \theta}\right| \\ &= \left|\sum_{i=1}^{2} u\left(x_b^i\right)\left(1 - \epsilon_\theta\left(x_b^i\right)\right) u\left(x_b^i\right)\frac{\partial \epsilon_\theta\left(x_b^i\right)}{\partial \theta}\right| \\ &\leqslant \left\|\frac{\partial \epsilon_\theta(x)}{\partial \theta}\right\|_{L^\infty}\cdot 2\epsilon\end{aligned}$$

Appendix for further detail

Proof(1) Gradient Analysis

Next, we may rewrite the $\mathcal{L}_r$ as

$$\mathcal{L}_r = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial^2 u_\theta}{\partial x^2}\left(x_f^i\right) - \frac{\partial^2 u}{\partial x^2}\left(x_f^i\right)\right|^2 \approx \int_0^1 \left(\frac{\partial^2 u_\theta(x)}{\partial x^2} - \frac{\partial^2 u(x)}{\partial x^2}\right)^2 dx$$

Then by integration by parts we have,

$$\begin{aligned}
\frac{\partial \mathcal{L}_r}{\partial \theta} &= \frac{\partial}{\partial \theta}\int_0^1 \left(\frac{\partial^2 u_\theta(x)}{\partial x^2} - \frac{\partial^2 u(x)}{\partial x^2}\right)^2 dx \\
&= \int_0^1 \frac{\partial}{\partial \theta}\left(\frac{\partial^2 u_\theta(x)}{\partial x^2} - \frac{\partial^2 u(x)}{\partial x^2}\right)^2 dx \\
&= \int_0^1 2\left(\frac{\partial^2 u_\theta(x)}{\partial x^2} - \frac{\partial^2 u(x)}{\partial x^2}\right)\frac{\partial}{\partial \theta}\left(\frac{\partial^2 u_\theta(x)}{\partial x^2}\right) dx \\
&= 2\int_0^1 \left(\frac{\partial^2 u_\theta(x)}{\partial x^2} - \frac{\partial^2 u(x)}{\partial x^2}\right)\frac{\partial^2}{\partial x^2}\frac{\partial\left(u_\theta(x)\right)}{\partial \theta}\Bigg) dx \\
&= 2\left[\frac{\partial^2 u_\theta(x)}{\partial x \partial \theta}\left(\frac{\partial^2 u_\theta(x)}{\partial x^2} - \frac{\partial^2 u(x)}{\partial x^2}\right)\Bigg|_0^1 - \int_0^1 \frac{\partial^2 u_\theta(x)}{\partial x \partial \theta}\left(\frac{\partial^3 u_\theta(x)}{\partial x^3} - \frac{\partial^3 u(x)}{\partial x^3}\right) dx\right] \\
&= 2\Bigg[\frac{\partial^2 u_\theta(x)}{\partial x \partial \theta}\left(\frac{\partial^2 u_\theta(x)}{\partial x^2} - \frac{\partial^2 u(x)}{\partial x^2}\right)\Bigg|_0^1 - \frac{\partial u_\theta(x)}{\partial \theta}\left(\frac{\partial^3 u_\theta(x)}{\partial x^3} - \frac{\partial^3 u(x)}{\partial x^3}\right)\Bigg|_0^1 \\
&+ \int_0^1 \frac{\partial u_\theta(x)}{\partial \theta}\left(\frac{\partial^4 u_\theta(x)}{\partial x^4} - \frac{\partial^4 u(x)}{\partial x^4}\right) dx\Bigg]
\end{aligned}$$

Appendix for further detail

Proof(1) Gradient Analysis

Note that

$$\begin{aligned}\left|\frac{\partial^2 u_\theta(x)}{\partial x \partial\theta}\right| &= \left|\frac{\partial^2 u(x)\epsilon_\theta(x)}{\partial x\partial\theta}\right| = \left|\frac{\partial}{\partial\theta}\left(u'(x)\epsilon_\theta(x) + u(x)\epsilon'_\theta(x)\right)\right| \\ &= \left|u'(x)\frac{\partial\epsilon_\theta(x)}{\partial\theta} + u(x)\frac{\partial\epsilon'_\theta(x)}{\partial\theta}\right| \leq C\left\|\frac{\partial\epsilon_\theta(x)}{\partial\theta}\right\|_{L^\infty} + \left\|\frac{\partial\epsilon'_\theta(x)}{\partial\theta}\right\|_{L^\infty}\end{aligned}$$

Appendix for further detail

Proof(1) Gradient Analysis

And

$$\begin{aligned}\left|\frac{\partial^2 u_\theta(x)}{\partial x^2}-\frac{\partial^2 u(x)}{\partial x^2}\right| &= \left|\frac{\partial^2 u(x)\epsilon_\theta(x)}{\partial x^2}-\frac{\partial^2 u(x)}{\partial x^2}\right| \\ &= \left|u''(x)\epsilon_\theta(x)+2u(x)\epsilon_\theta''(x)-u''(x)\right| \\ &= \left|u''(x)\left(\epsilon_\theta(x)-1\right)+2u(x)\epsilon_\theta''(x)\right| \\ &\leq C^2\epsilon+2\epsilon\end{aligned}$$

So we have

$$\left|\frac{\partial^2 u_\theta(x)}{\partial x\partial\theta}\left(\frac{\partial^2 u_\theta(x)}{\partial x^2}-\frac{\partial^2 u(x)}{\partial x^2}\right)\Big|_0^1\right| \leq 2\left(C\left\|\frac{\partial\epsilon_\theta(x)}{\partial\theta}\right\|_{L\infty}+\left\|\frac{\partial\epsilon_\theta'(x)}{\partial\theta}\right\|_{L\infty}\right)\left(C^2\epsilon+2\epsilon\right) \tag{76}$$

$$= O\left(C^3\right)\cdot\epsilon\cdot\left\|\frac{\partial\epsilon_\theta(x)}{\partial\theta}\right\|_{L\infty} \tag{77}$$

Similarly,

$$\left|\frac{\partial u_\theta(x)}{\partial\theta}\left(\frac{\partial^3 u_\theta(x)}{\partial x^3}-\frac{\partial^3 u(x)}{\partial x^3}\right)\Big|_0^1\right| = \left|\frac{\partial u_\theta(x)}{\partial\theta}\left(\frac{\partial^3 u(x)\epsilon_\theta(x)}{\partial x^3}-\frac{\partial^3 u(x)}{\partial x^3}\right)\Big|_0^1\right| \tag{78}$$

$$\leq O\left(C^3\right)\cdot\epsilon\cdot\left\|\frac{\partial\epsilon_\theta(x)}{\partial\theta}\right\|_{L\infty} \tag{79}$$

Appendix for further detail

Proof(1) Gradient Analysis

Finally,

$$\left|\int_0^1 \frac{\partial u_\theta(x)}{\partial \theta}\left(\frac{\partial^4 u_\theta(x)}{\partial x^4}-\frac{\partial^4 u(x)}{\partial x^4}\right) dx\right| \leq O\left(C^4\right) \cdot \epsilon \cdot\left\|\frac{\partial \epsilon_\theta(x)}{\partial \theta}\right\|_{L^\infty} \tag{80}$$

Therefore, plugging all these together we obtain

$$\left|\frac{\partial \mathcal{L}_r}{\partial \theta}\right| \leq O\left(C^4\right) \cdot \epsilon \cdot\left\|\frac{\partial \epsilon_\theta(x)}{\partial \theta}\right\|_{L^\infty} \tag{81}$$

□

Appendix for further detail

Proof(2) Gradient Descent Failure

The following simple analysis may reveal the connection between stiffness in the gradient flow dynamics and the evident difficulty in training physics-informed neural networks with gradient descent. Suppose that at n-th step of gradient decent during the training, we have

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}(\theta_n) = \theta_n - \eta \left[\nabla_\theta \mathcal{L}_r(\theta_n) + \nabla_\theta \mathcal{L}_{u_b}(\theta_n) \right] \tag{18}$$

where η is the learning rate. Then applying second order Taylor expansion to the loss function $\mathcal{L}(\theta)$ at θ_n gives

$$\mathcal{L}(\theta_{n+1}) = \mathcal{L}(\theta_n) + (\theta_{n+1} - \theta_n) \cdot \nabla_\theta \mathcal{L}(\theta_n) + \frac{1}{2}(\theta_{n+1} - \theta_n)^T \nabla_\theta^2 \mathcal{L}(\xi)(\theta_{n+1} - \theta_n) \tag{19}$$

where $\xi = t\theta_n + (1-t)\theta_{n+1}$ for some $t \in [0, 1]$ and $\nabla_\theta^2 \mathcal{L}(\xi)$ is the Hessian matrix of the loss function $\mathcal{L}(\theta)$ evaluated at ξ. Now applying 18 to 19, we obtain

$$\begin{aligned}
\mathcal{L}(\theta_{n+1}) - \mathcal{L}(\theta_n) &= -\eta \nabla_\theta \mathcal{L}(\theta_n) \cdot \nabla_\theta \mathcal{L}(\theta_n) + \frac{1}{2}\eta^2 \nabla_\theta \mathcal{L}(\theta_n)^T \nabla_\theta^2 \mathcal{L}(\xi) \eta \nabla_\theta \mathcal{L}(\theta_n) && (20) \\
&= -\eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 + \frac{1}{2}\eta^2 \nabla_\theta \mathcal{L}(\theta_n)^T \nabla_\theta^2 \mathcal{L}(\xi) \nabla_\theta \mathcal{L}(\theta_n) && (21) \\
&= -\eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 + \frac{1}{2}\eta^2 \nabla_\theta \mathcal{L}(\theta_n)^T \left(\nabla_\theta^2 \mathcal{L}_r(\xi) + \nabla_\theta^2 \mathcal{L}_{u_b}(\xi) \right) \nabla_\theta \mathcal{L}(\theta_n) && (22) \\
&= -\eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 + \frac{1}{2}\eta^2 \nabla_\theta \mathcal{L}(\theta_n)^T \nabla_\theta^2 \mathcal{L}_r(\xi) \nabla_\theta \mathcal{L}(\theta_n) + \frac{1}{2}\eta^2 \nabla_\theta \mathcal{L}(\theta_n)^T \nabla_\theta^2 \mathcal{L}_{u_b}(\xi) \nabla_\theta \mathcal{L}(\theta_n) && (23)
\end{aligned}$$

Appendix for further detail

Proof(2) Gradient Descent Failure

Here, note that

$$\nabla_\theta \mathcal{L}(\theta_n)^T \nabla^2_\theta \mathcal{L}(\xi) \nabla_\theta \mathcal{L}(\theta_n) = \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 \frac{\nabla_\theta \mathcal{L}(\theta_n)^T}{\|\nabla_\theta \mathcal{L}(\theta_n)\|} \nabla^2_\theta \mathcal{L}(\xi) \frac{\nabla_\theta \mathcal{L}(\theta_n)}{\|\nabla_\theta \mathcal{L}(\theta_n)\|} \tag{24}$$

$$= \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 x^T Q^T \mathbf{diag}(\lambda_1, \lambda_2 \cdots \lambda_n) Q x \tag{25}$$

$$= \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 y^T \mathbf{diag}(\lambda_1, \lambda_2 \ldots \lambda_M) y \tag{26}$$

$$= \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 \sum_{i=1}^{M} \lambda_i y_i^2 \tag{27}$$

Appendix for further detail

Proof(2) Gradient Descent Failure

where $x = \frac{\nabla_\theta \mathcal{L}(\theta_n)}{\|\nabla_\theta \mathcal{L}(\theta_n)\|}$, Q is an orthogonal matrix diagonalizing $\nabla_\theta^2 \mathcal{L}(\xi)$ and $y = Qx$. And $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_N$ are eigenvalues of $\nabla_\theta^2 \mathcal{L}(\xi)$. Similarly, we have

$$\nabla_\theta \mathcal{L}(\theta_n)^T \nabla_\theta^2 \mathcal{L}_r(\xi) \nabla_\theta \mathcal{L}(\theta_n) = \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 \sum_{i=1}^{M} \lambda_i^r y_i^2 \tag{28}$$

$$\nabla_\theta \mathcal{L}(\theta_n)^T \nabla_\theta^2 \mathcal{L}_{u_b}(\xi) \nabla_\theta \mathcal{L}(\theta_n) = \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 \sum_{i=1}^{M} \lambda_i^{u_b} y_i^2 \tag{29}$$

where $\lambda_1^r \le \lambda_2^r \le \cdots \le \lambda_N^r$ and $\lambda_1^{u_b} \le \lambda_2^{u_b} \le \cdots \le \lambda_N^{u_b}$ are eigenvalues of $\nabla_\theta^2 \mathcal{L}_r$ and $\nabla_\theta^2 \mathcal{L}_{u_b}$ respectively. Thus, combining these together we get

$$\mathcal{L}(\theta_{n+1}) - \mathcal{L}(\theta_n) = \eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 (-1 + \frac{1}{2}\eta \sum_{i=1}^{N} \lambda_i y_i^2) \tag{30}$$

$$\mathcal{L}_r(\theta_{n+1}) - \mathcal{L}_r(\theta_n) = \eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 (-1 + \frac{1}{2}\eta \sum_{i=1}^{N} \lambda_i^r y_i^2) \tag{31}$$

$$\mathcal{L}_{u_b}(\theta_{n+1}) - \mathcal{L}_{u_b}(\theta_n) = \eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 (-1 + \frac{1}{2}\eta \sum_{i=1}^{N} \lambda_i^{u_b} y_i^2) \tag{32}$$