UNDERSTANDING AND MITIGATING GRADIENT PATHOLOGIES IN PHYSICS-INFORMED NEURAL NETWORKS

A PREPRINT

Sifan Wang
Graduate Group in Applied Mathematics
and Computational Science
University of Pennsylvania
Philadelphia, PA 19104
sifanw@sas.upenn.edu

Yujun Teng
Department of Mechanichal Engineering
and Applied Mechanics
University of Pennsylvania
Philadelphia, PA 19104
yjteng@seas.upenn.edu

Paris Perdikaris
Department of Mechanichal Engineering
and Applied Mechanics
University of Pennsylvania
Philadelphia, PA 19104
pgp@seas.upenn.edu

January 15, 2020

ABSTRACT

The widespread use of neural networks across different scientific domains often involves constraining them to satisfy certain symmetries, conservation laws, or other domain knowledge. Such constraints are often imposed as soft penalties during model training and effectively act as domain-specific regularizers of the empirical risk loss. Physics-informed neural networks is an example of this philosophy in which the outputs of deep neural networks are constrained to approximately satisfy a given set of partial differential equations. In this work we review recent advances in scientific machine learning with a specific focus on the effectiveness of physics-informed neural networks in predicting outcomes of physical systems and discovering hidden physics from noisy data. We will also identify and analyze a fundamental mode of failure of such approaches that is related to numerical stiffness leading to unbalanced back-propagated gradients during model training. To address this limitation we present a learning rate annealing algorithm that utilizes gradient statistics during model training to balance the interplay between different terms in composite loss functions. We also propose a novel neural network architecture that is more resilient to such gradient pathologies. Taken together, our developments provide new insights into the training of constrained neural networks and consistently improve the predictive accuracy of physics-informed neural networks by a factor of 50-100x across a range of problems in computational physics. All code and data accompanying this manuscript are publicly available at https://github.com/PredictiveIntelligenceLab/GradientPathologiesPINNs.

***K*eywords** Deep learning · Differential equations · Optimization · Stiff dynamics · Computational physics

1 Introduction

Thanks to breakthrough results across a diverse range of scientific disciplines [1, 2, 3, 4, 5], deep learning is currently influencing the way we process data, recognize patterns, and build predictive models of complex systems. Many of these predictive tasks are currently being tackled using over-parameterized, black-box discriminative models such as deep neural networks, in which interpretability and robustness is often sacrificed in favor of flexibility in representation

and scalability in computation. Such models have yielded remarkable results in data-rich domains [1, 6, 7], yet their effectiveness in the small data regime still remains questionable, motivating the question of how can one endow these powerful black-box function approximators with prior knowledge and appropriate inductive biases.

Attempts to address this question are currently defining two distinct schools of thought. The first pertains to efforts focused on designing specialized neural network architectures that implicitly embed any prior knowledge and inductive biases associated with a given predictive task. Without a doubt, the most celebrated example in this category is convolutional neural networks [8, 9] which have revolutionized the field of computer vision by craftily respecting invariance along the groups of symmetries and distributed pattern representations found in natural images [10]. Another example includes covariant neural networks [11], that are tailored to conform with the rotation and translation invariance present in many-body molecular systems. Despite their remarkable effectiveness, such approaches are currently limited to tasks that are characterized by relatively simple and well-defined symmetry groups, and often require craftsmanship and elaborate implementations. Moreover, their extension to more complex tasks is challenging as the underlying invariance that characterize many physical systems (e.g., fluid flows) are often poorly understood or hard to implicitly encode in a neural architecture.

The second school of thought approaches the problem of endowing a neural net with prior knowledge from a different angle. Instead of designing specialized architectures that implicitly bake in this knowledge, current efforts aim to impose such constraints in a soft manner by appropriately penalizing the loss function of conventional neural network approximations denoted by $f_\theta(\cdot)$ and typically parametrized by a set of weights and biases θ. These penalty constraints lead to loss functions taking the general form

$$\mathcal{L}(\theta) := \frac{1}{N_u}\sum_{i=1}^{N_u}[\boldsymbol{u}_i - f_\theta(\boldsymbol{x}_i)]^2 + \frac{1}{\lambda}\mathcal{R}[f_\theta(\boldsymbol{x})], \tag{1}$$

where the empirical risk loss over a given set of input-output pairs $\{\boldsymbol{x}_i, \boldsymbol{u}_i\}$, $i = 1, \ldots, N_u$ is penalized by some appropriate functional $\mathcal{R}[f_\theta(\boldsymbol{x})]$ that is designed to constraint the outputs of the neural network to satisfy a set of specified conditions, as controlled by the regularization parameter λ. A representative example of this approach is *physics-informed neural networks* [12, 13, 14, 15], in which the outputs of a neural network are constrained to approximately satisfy a system of partial differential equations (PDE) by using a regularization functional $\mathcal{R}[f_\theta(\boldsymbol{x})]$ that typically corresponds to the residual or the variational energy of the PDE system under the neural network representation. This framework has enabled the solution of forward and inverse problem in computational physics by reviving the original ideas in [16, 17] using modern software developments in reverse-mode differentiation [18] in order to automatically compute any derivatives present in the PDE operator. Although such approaches appear seemingly straightforward and have yielded remarkable results across a range of problems in computational science and engineering [19, 20, 21, 22, 23, 24, 25, 26] , the effects of the regularization mechanism in equation 1 remain poorly understood, and in several cases can even lead to unstable and erroneous predictions (for e.g. see remarks in [20, 26]).

In this work, we use physics-informed neural networks as a test-bed for analyzing the performance of constrained neural networks trained using regularized loss functions in the form of equation 1. Our specific contributions can be summarized in the following points:

- Our analysis reveals a fundamental mode of failure in physics-informed neural networks related to stiffness in the gradient flow dynamics.
- This leads to an unstable imbalance in the magnitude of the back-propagated gradients during model training using gradient descent.
- We propose a simple solution based on an adaptive learning rate annealing algorithm that aims to balance the interplay between data-fit and regularization.
- We also propose a novel neural network architecture that has less stiffness than the convention fully-connected neural network.
- We systematically test the proposed ideas and demonstrate consistent improvements in the predictive accuracy of physics-informed neural networks by a factor of 50-100x across a range of problems in computational physics.

Taken all together, our developments provide new insight into the training of constrained neural networks that can help us endow deep learning tools with prior knowledge and reduce the energy barrier for adapting them to new scientific domains.

The paper is structured as follows. In section 2, we first present a brief overview of the physics-informed neural networks (PINNs) following the original formulation of Raissi *et. al.* [12]. Next, we introduce a simple benchmark

problem that can guide our analysis and highlight the difficulties introduced by stiffness in the gradient flow dynamics of physics-informed neural networks, see sections 2.2 - 2.4. To address these difficulties we proposed an adaptive learning rate algorithm along with a novel fully-connected neural architecture, as described in sections 2.5 and 2.6. In section 3 we present the detailed evaluation of our proposed algorithm and neural network architecture across a range of representative benchmark examples. Finally, in section 4, we summarize our findings and provide a discussion on potential pitfalls and promising future directions. All code and data accompanying this manuscript are publicly available at `https://github.com/PredictiveIntelligenceLab/GradientPathologiesPINNs`.

2 Methods

2.1 A primer in physics-informed neural networks

Physics-informed neural networks (PINNs) [12] aim at inferring a continuous latent function $\boldsymbol{u}(\boldsymbol{x}, t)$ that arises as the solution to a system of nonlinear partial differential equations (PDE) of the general form

$$\begin{aligned} &\boldsymbol{u}_t + \mathcal{N}_{\boldsymbol{x}}[\boldsymbol{u}] = 0, \;\; \boldsymbol{x} \in \Omega, t \in [0, T] \\ &\boldsymbol{u}(\boldsymbol{x}, 0) = h(\boldsymbol{x}), \;\; x \in \Omega \\ &\boldsymbol{u}(\boldsymbol{x}, t) = g(\boldsymbol{x}, t), \;\; t \in [0, T], \;\; \boldsymbol{x} \in \partial\Omega \end{aligned} \tag{2}$$

where $\boldsymbol{x}$ and t denote space and time coordinates, subscripts denote partial differentiation, $\mathcal{N}_{\boldsymbol{x}}[\cdot]$ is a nonlinear differential operator, Ω is a subset of $\mathbb{R}^D$, and $\partial\Omega$ is the boundary of Ω. Following the original work of [12], we then proceed by approximating $\boldsymbol{u}(\boldsymbol{x}, t)$ by a deep neural network $f_\theta(\boldsymbol{x}, t)$, and define the residual of equation 2 as

$$\boldsymbol{r}_\theta(\boldsymbol{x}, t) := \frac{\partial}{\partial t} f_\theta(\boldsymbol{x}, t) + \mathcal{N}_{\boldsymbol{x}}[f_\theta(\boldsymbol{x}, t)], \tag{3}$$

where the partial derivatives of the neural network representation with respect to the space and time coordinates can be readily computed to machine precision using reverse mode differentiation [18]. Notice how the neural network parameters θ (i.e. the weights and biases of the neural network) are shared between the representation of the latent solution $\boldsymbol{u}(\boldsymbol{x}, t)$, and the PDE residual $\boldsymbol{r}(\boldsymbol{x}, t)$. A good set of candidate parameters can be identified via gradient descent using a composite loss function of the general form

$$\mathcal{L}(\theta) := \mathcal{L}_r(\theta) + \sum_{i=1}^{M} \lambda_i \mathcal{L}_i(\theta), \tag{4}$$

where $\mathcal{L}_r(\theta)$ is a loss term that penalizes the PDE residual, and $\mathcal{L}_i(\theta)$, $i = 1, \ldots, M$ correspond to data-fit terms (e.g., measurements, initial or boundary conditions, etc.). For a typical initial and boundary value problem, these loss functions would take the specific form

$$\mathcal{L}_r = \frac{1}{N_r} \sum_{i=1}^{N_r} [\boldsymbol{r}(\boldsymbol{x}_r^i, t_r^i)]^2 \tag{5}$$

$$\mathcal{L}_{u_b} = \frac{1}{N_b} \sum_{i=1}^{N_b} [\boldsymbol{u}(\boldsymbol{x}_b^i, t_b^i) - g_b^i]^2, \tag{6}$$

$$\mathcal{L}_{u_0} = \frac{1}{N_0} \sum_{i=1}^{N_0} [\boldsymbol{u}(\boldsymbol{x}_0^i, 0) - h_0^i)]^2, \tag{7}$$

where $\{\boldsymbol{x}_0^i, h_0^i)\}_{i=1}^{N_0}$ denotes the initial data, $\{(\boldsymbol{x}_b^i, t_b^i), g_b^i\}_{i=1}^{N_b}$ denotes the boundary data, and $\{(\boldsymbol{x}_r^i, t_r^i), \boldsymbol{0}\}_{i=1}^{N_r}$ a set of collocation points that are randomly placed inside the domain Ω in order to minimize the PDE residual. Consequently, $\mathcal{L}_r$ penalizes the equation not being satisfied on the collocation points. Moreover, $\mathcal{L}_{u_b}$ and $\mathcal{L}_{u_0}$ enforces the boundary conditions and the initial conditions respectively. As $\mathcal{L}(\theta)$ is typically minimized used stochastic gradient descent, an very large number of training points ($\mathcal{O}(10^5 - 10^8)$) can be sampled as the locations of $\boldsymbol{x}_0^i, (\boldsymbol{x}_b^i, t_b^i), (\boldsymbol{x}_r^i, t_r^i)$ can be randomized within each gradient descent iteration. The ultimate goal of this procedure it to construct a neural network representation $f_\theta(\boldsymbol{x}, t)$ for which $\mathcal{L}(\theta)$ is as close to zero as possible.

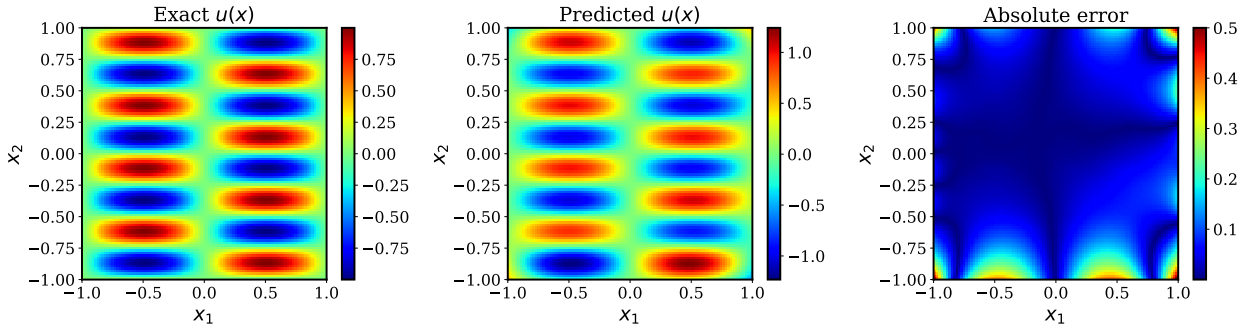


Figure 1: *Helmholtz equation:* Exact solution versus the prediction of a conventional physics-informed neural network model with 4 hidden layers and 50 neurons each layer after 40,000 iterations of training with gradient descent (relative L^2-error: 1.81e-01).

2.2 Gradient pathologies in physics-informed neural networks

Despite a series of promising results [19, 20, 23, 24, 25, 26, 13], the original formulation of Raissi *et. al.* [12] often has difficulties in constructing an accurate approximation to the exact latent solution $\boldsymbol{u}(\boldsymbol{x}, t)$ for reasons that yet remain poorly understood. These pathologies can arise even in the simplest possible setting corresponding to solving classical linear elliptic equations. As an example, let us consider the Helmholtz equation in two space dimensions

$$\Delta u(x,y) + k^2 u(x,y) = q(x,y), \quad (x,y) \in \Omega := (-1,1) \tag{8}$$

$$u(x,y) = h(x,y), \quad (x,y) \in \partial\Omega \tag{9}$$

where Δ is the Laplace operator. One can easily fabricate an exact solution to this problem taking the form $u(x,y) = \sin(a_1\pi x)\sin(a_2\pi y)$, corresponding to a source term of the form

$$q(x,y) = -(a_1\pi)^2 \sin(a_1\pi x)\sin(a_2\pi y) - (a_2\pi)^2 \sin(a_1\pi x)\sin(a_2\pi y) + k^2 \sin(a_1\pi x)\sin(a_2\pi y) \tag{10}$$

where we take $a_1 = 1$ and $a_2 = 4$. A PINN approximation to solving equation 8 can be constructed by parametrizing its solution with a deep neural network $f_\theta(x,y)$, whose parameters θ can be identified by minimizing a composite loss function that aims to fit the known boundary conditions, while also penalizing the Helmholtz equation residual inside the domain Ω, i.e.,

$$\mathcal{L}(\theta) = \mathcal{L}_r(\theta) + \mathcal{L}_{u_b}(\theta) \tag{11}$$

Without loss of generality, let us consider an example prediction scenario in which $f_\theta(x,y)$ is a 4-layer deep fully connected neural network with 50 neurons per layer and a hyperbolic tangent activation function. We train this network for $40,000$ stochastic gradient descent steps by minimizing the loss of equation 11 using the Adam optimizer [27] with an initial learning rate of 10^{-3} and a decreasing annealing schedule. In figure 1 we compare the predictions of this trained model against the exact solution for this problem, and report the point-wise absolute discrepancy between the two. It is evident that the PINN approximation does a poor job at fitting the boundary conditions, leading to a 15.7% prediction error measured in the relative L^2 norm.

To explore the reason why this model fails to return accurate predictions, we draw motivation from the seminal work of Glorot and Bengio [28] and monitor the distribution of the back-propagated gradients of our loss with respect to the neural network parameters during training. Rather than tracking the gradients of the aggregate loss $\mathcal{L}(\theta)$, we track the gradients of each individual terms $\mathcal{L}_{u_b}(\theta)$ and $\mathcal{L}_r(\theta)$ with respect to the weights in each hidden layer of the neural network. As illustrated in figure 2, the gradients corresponding to the boundary loss term $\mathcal{L}_{u_b}(\theta)$ in each layer are sharply concentrated around zero and overall attain significantly smaller values than the gradients corresponding to the PDE residual loss $\mathcal{L}_r(\theta)$. As we know, in lack of proper restrictions such as boundary conditions or initial conditions, a PDE system may have infinitely many solutions [29]. Therefore, if the gradients $\nabla_\theta \mathcal{L}_{u_b}(\theta)$ are very small during training, then our PINN model should experience difficulties in fitting the boundary conditions. One the other hand, while the gradients $\nabla_\theta \mathcal{L}_r(\theta)$ are large, the neural network can easily learn any solutions that satisfy the equation. As a result, our trained model is heavily biased towards returning a solution that leads to a small PDE residual, but without however respecting the given boundary conditions, it is prone to return erroneous predictions.

2.3 Gradient analysis for physics-informed neural networks

It is now natural to ask what is the mechanism that gives rise to this gradient imbalance between the two loss terms, $\mathcal{L}_{u_b}(\theta)$ and $\mathcal{L}_r(\theta)$, respectively. For starters, one may be quick to notice that the chain rule for computing the gradients

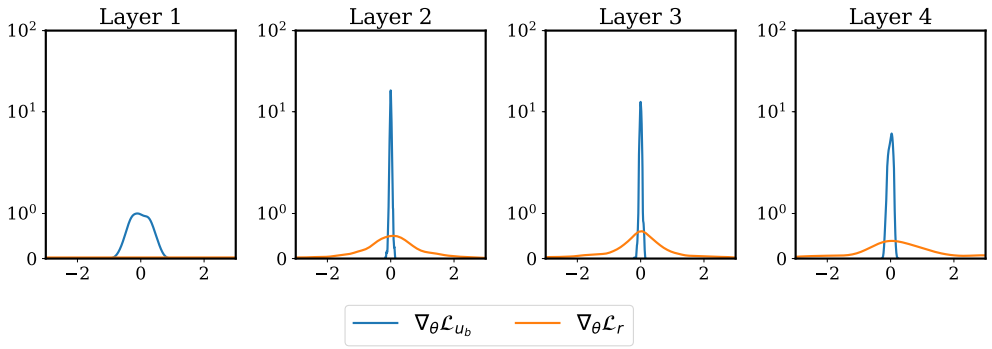


Figure 2: *Helmholtz equation:* Histograms of back-propagated gradients $\nabla_\theta \mathcal{L}_r(\theta)$ and $\nabla_\theta \mathcal{L}_{u_b}(\theta)$ at each layer during the $40,000$th iteration of training a standard PINN model for solving the Helmholtz equation.

of $\mathcal{L}_r(\theta)$ is deeper than the chain rule for computing the gradients of $\mathcal{L}_b(\theta)$, and thus the gradients of $\mathcal{L}_r$ are easier to suffer from vanishing gradient pathologies. Paradoxically, here we observe the opposite behavior as it is the $\nabla_\theta \mathcal{L}_{u_b}(\theta)$ gradients that appears to vanish during model training (see figure 2). To understand what gives rise to this behavior, let us take a step back and consider an even simpler benchmark, the one-dimensional Poisson equation

$$\begin{aligned} &\Delta u(x) = g(x), \quad x \in [0,1] \\ &u(x) = h(x), \quad x = 0 \text{ and } x = 1. \end{aligned} \tag{12}$$

To this end, let us consider exact solutions of the form $u(x) = \sin(Cx)$, corresponding to a source term given by $g(x) = -C^2 \sin(Cx) = -C^2 u(x)$. Under the PINNs framework, we will use a fully-connected deep neural network $f_\theta(x)$ parametrized by θ to approximate the latent solution $u(x)$. Then the corresponding loss function over a collection of boundary and residual data-points is given by

$$\begin{aligned} \mathcal{L}(\theta) &= \mathcal{L}_r(\theta) + \mathcal{L}_{u_b}(\theta) \\ &= \frac{1}{N_b} \sum_{i=1}^{N_b} [f_\theta(x_b^i) - h(x_b^i)]^2 + \frac{1}{N_r} \sum_{i=1}^{N_r} [\frac{\partial^2}{\partial x^2} f_\theta(x_r^i) - g(x_r^i)]^2. \end{aligned} \tag{13}$$

To provide some analysis, let us assume that there exists a trained neural network $f_\theta(x)$ that can provide a good approximation to the latent solution $u(x)$ (i.e. the target solution is in the Hilbert space spanned by the neural network degrees of freedom). Then we may express our approximation as $f_\theta(x) = u(x)\epsilon_\theta(x)$, where $\epsilon_\theta(x)$ is a smooth function defined in $[0,1]$, and for which $|\epsilon_\theta(x) - 1| \leq \epsilon$ for some $\epsilon > 0$, and $\| \frac{\partial^k \epsilon_\theta(x)}{\partial x^k} \|_{L^\infty} < \epsilon$, for all non-negative integer k. This construction allows us to derive the following bound for the gradients of the PDE boundary loss and residual loss (see proof in Appendix 5.1)

$$\|\nabla_\theta \mathcal{L}_{u_b}(\theta)\|_{L^\infty} \leq 2\epsilon \cdot \|\nabla_\theta \epsilon_\theta(x)\|_{L^\infty} \tag{14}$$

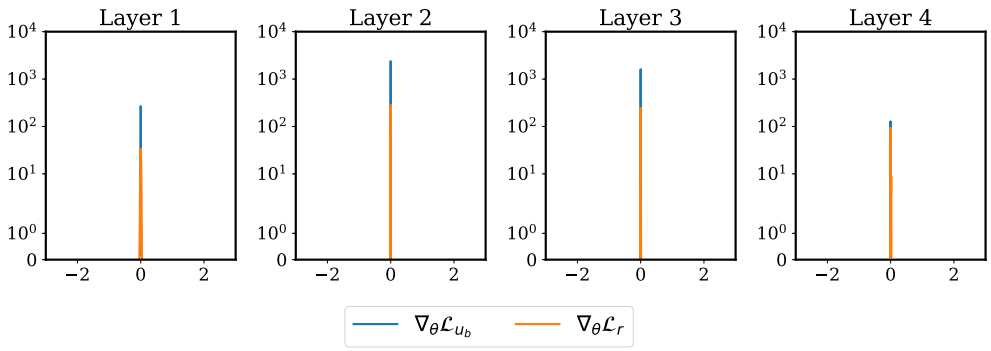
$$\|\nabla_\theta \mathcal{L}_r(\theta)\|_{L^\infty} \leq O(C^4) \cdot \epsilon \cdot \|\nabla_\theta \epsilon_\theta(x)\|_{L^\infty} \tag{15}$$

Based on this simple analysis, we can conclude that if the constant C is large, then the norm of gradients of $\mathcal{L}_r(\theta)$ may be much greater the gradients of $\mathcal{L}_{u_b}(\theta)$, thus biasing the neural network training towards neglecting the contribution of the boundary data-fit term. To confirm this result we have performed a series of simulations in which standard PINN models are trained to approximate the solution of equation 12 for different values of the constant C. Our results are summarized in figure 3, indicating that larger values of the constant C lead to a pronounced imbalance in the back-propagated gradients $\nabla_\theta \mathcal{L}_r(\theta)$ and $\nabla_\theta \mathcal{L}_{u_b}(\theta)$, that ultimately leads to an inaccurate reconstruction of the PDE solution.

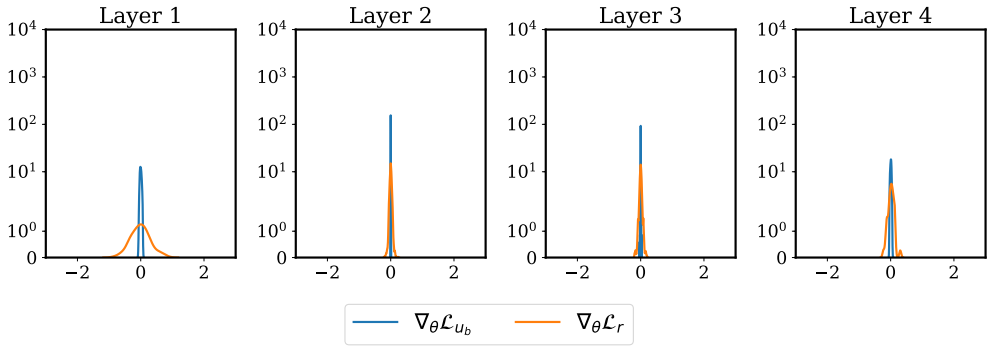
2.4 Stiffness in the gradient flow dynamics

To provide further insight into the pathology of imbalanced gradients, let us consider the continuous limit of the learning dynamics for calibrating the neural network parameters θ, as governed by the gradient flow system

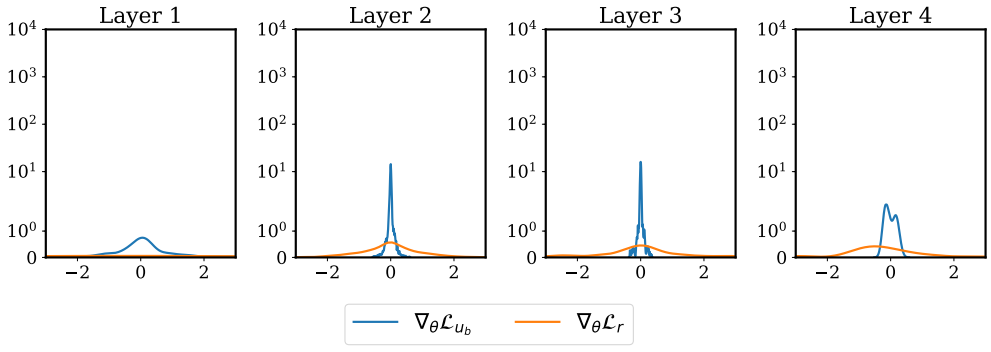
$$\frac{d\theta}{dt} = -\nabla_\theta \mathcal{L}_r(\theta) - \sum_{i=1}^{M} \nabla_\theta \mathcal{L}_i(\theta). \tag{16}$$



(a) C = 1



(b) C = 2



(c) C = 4

Figure 3: *Possion equation:* Histograms of back-propagated gradients $\nabla_\theta \mathcal{L}_r(\theta)$ and $\nabla_\theta \mathcal{L}_{u_b}(\theta)$ at each layer during the $40,000$th iteration of training a standard PINN model for solving the one-dimensional Poisson equation for different values of the constant C, see equation 12.

Starting from an initial guess, one can integrate over the gradient flow to obtain a local minimum of the total loss $\mathcal{L}(\theta)$. It is straightforward to see that the gradient descent updates of equation 39 correspond to a forward Euler discretization [30] of equation 16. It is also well understood that the stability of this explicit, first-order discretization strategy is severely limited by the choice of the step-size/learning rate, especially for cases in which the governing gradient flow dynamics are stiff. We believe that such cases arise routinely in the context of physics-informed neural networks in which the different terms in their loss have inherently different nature and often correspond to competing objectives (e.g., fitting a set of noisy data versus enforcing a zero PDE residual).

To obtain a quantitative assessment of stiffness in the gradient flow dynamics of a neural network one can compute and monitor the largest eigenvalue $\sigma_{\max}(\nabla^2_\theta \mathcal{L}(\theta))$ of the Hessian matrix $\nabla^2_\theta \mathcal{L}(\theta)$ during model training [31]. This immediately follows from performing a stability analysis for the linearized system

$$\frac{d}{dt}\tilde{\theta}(t) = -\nabla^2_\theta \mathcal{L}(\tilde{\theta}(t)) \cdot \tilde{\theta}(t). \tag{17}$$

It is well known that the largest eigenvalue of the Hessian dictates the fastest time-scale of the system and directly imposes a restriction on the learning rate that one needs to employ to ensure stability in the forward Euler discretization, as reflected by gradient descent update rule of equation 39. In fact, a classical result in numerical analysis on the conditional stability of the forward Euler method requires bounding the learning rate as $\eta < 2/\sigma_{\max}(\nabla^2_\theta \mathcal{L}(\theta))$ [30].

To illustrate the presence of stiffness in the gradient flow dynamics of physics-informed neural networks, let us revisit the simple simulation study presented in section 2.2 for the two-dimensional Helmholtz benchmark. Specifically, we consider two cases by modulating the control parameters a_1 and a_2 in equation 8. In the first case we set $a_1 = 1$, $a_2 = 1$ to fabricate a relatively simple solution $u(x, y)$ that has an isotropic length-scale across both spatial dimensions. In the second case we choose $a_1 = 1$, $a_2 = 4$ to fabricate a solution $u(x, y)$ that exhibits directional anisotropy and has more oscillations along the y-coordinate. For each case we then employ the same simulation setup presented in section 2.2 to train a 4-layer deep physics-informed neural network for approximating the latent solution $u(x, y)$. During training we probe the stiffness of the gradient flow dynamics by computing and monitoring the largest eigenvalue of the Hessian matrix $\sigma_{\max}(\nabla^2_\theta \mathcal{L}(\theta))$. In figure 5 we present the recorded values of $\sigma_{\max}(\nabla^2_\theta \mathcal{L}(\theta))$ at different iterations of the gradient descent algorithm. Our results reveal that, as the complexity of the target solution is increased, the training dynamics of the physics-informed neural network become increasingly stiffer, ultimately leading to a severe restriction in the required learning rate needed for stable gradient descent updates.

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

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

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

$$\mathcal{L}(\theta_{n+1}) - \mathcal{L}(\theta_n) = -\eta \nabla_\theta \mathcal{L}(\theta_n) \cdot \nabla_\theta \mathcal{L}(\theta_n) + \frac{1}{2}\eta^2 \nabla_\theta \mathcal{L}(\theta_n)^T \nabla^2_\theta \mathcal{L}(\xi) \eta \nabla_\theta \mathcal{L}(\theta_n) \tag{20}$$

$$= -\eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 + \frac{1}{2}\eta^2 \nabla_\theta \mathcal{L}(\theta_n)^T \nabla^2_\theta \mathcal{L}(\xi) \nabla_\theta \mathcal{L}(\theta_n) \tag{21}$$

$$= -\eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 + \frac{1}{2}\eta^2 \nabla_\theta \mathcal{L}(\theta_n)^T \left(\nabla^2_\theta \mathcal{L}_r(\xi) + \nabla^2_\theta \mathcal{L}_{u_b}(\xi)\right) \nabla_\theta \mathcal{L}(\theta_n) \tag{22}$$

$$= -\eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 + \frac{1}{2}\eta^2 \nabla_\theta \mathcal{L}(\theta_n)^T \nabla^2_\theta \mathcal{L}_r(\xi) \nabla_\theta \mathcal{L}(\theta_n) + \frac{1}{2}\eta^2 \nabla_\theta \mathcal{L}(\theta_n)^T \nabla^2_\theta \mathcal{L}_{u_b}(\xi) \nabla_\theta \mathcal{L}(\theta_n) \tag{23}$$

Here, note that

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

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

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

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

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

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

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

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

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

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

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

Thus, if the gradient flow is stiff, then many eigenvalues of $\nabla^2_\theta \mathcal{L}(\xi)$ may be large. As a result, it is very possible that $\mathcal{L}(\theta_{n+1}) - \mathcal{L}(\theta_n) > 0$, which implies that the gradient decent method fails to decrease the loss even if $\nabla_\theta \mathcal{L}(\theta_n)$ is the right decent direction. This result comes to no surprise, as it is well understood that gradient descent may get stuck in limit cycles or even diverge in the presence of multiple competing objectives [32, 33].

To verify our hypothesis, we still consider the example in section 2. Specifically, we choose a five layer neural network with 50 units in each hidden layer. To reduce stochastic effects introduced by mini-batch gradient descent we use full batch gradient decent with an initial learning rate of $10e-3$ and an exponential decay with a decay-rate of 0.9 and a decay-step of 1,000 iterations. All collocation points are uniformly sampled from both the boundary and the interior domain. Figure 4a shows the eigenvalues of $\nabla^2_\theta \mathcal{L}_{u_b}(\theta)$ and $\nabla^2_\theta \mathcal{L}_r(\theta)$ in ascending order. Clearly, we can conclude that many eigenvalues of $\nabla^2_\theta \mathcal{L}_r(\theta)$ are extremely large up to 10^5 and the stiffness of the gradient flow is dominated by $\mathcal{L}_r(\theta)$. As a consequence, in figure 4b we observe that the loss of $\mathcal{L}_r(\theta)$ oscillates wildly even when full batch gradient descent is employed, while the loss of $\mathcal{L}_{u_b}(\theta)$ exhibits a decreasing tendency since the eigenvalues of $\nabla^2_\theta \mathcal{L}_{u_b}(\theta)$ remain relatively small. Besides, note that the oscillation of the loss of $\mathcal{L}_r(\theta)$ becomes small as the iteration increases since the learning rate decay. The above analysis may give another angle to illustrate how the stiff nature of gradient flow system may cause severe difficulties in the training of physics-informed neural networks with gradient descent. It also hints to the importance of choosing the learning rate such us a stable discretization of the gradient flow is achieved..

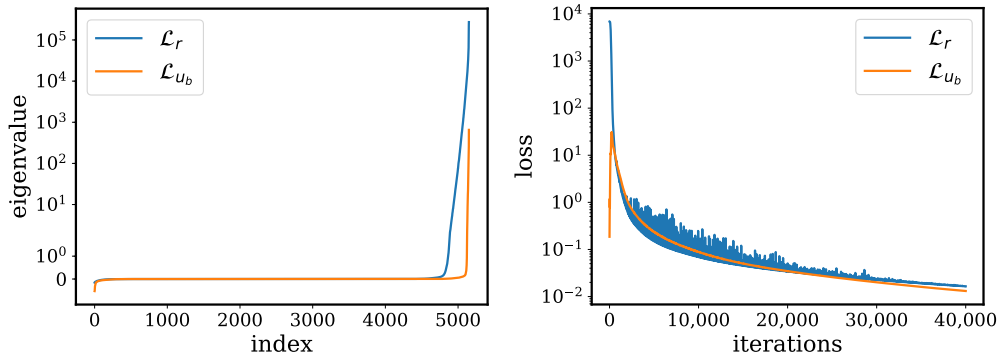
Moreover, applying the classical Sobolev inequality [34] to $\nabla_\theta \mathcal{L}_r$ and $\nabla_\theta \mathcal{L}_{u_b}$ gives

$$\|\nabla_\theta \mathcal{L}_r(\theta)\|_{L^\infty} \leq C \|\nabla^2_\theta \mathcal{L}_r(\theta)\|_{L^\infty} \tag{33}$$

$$\|\nabla_\theta \mathcal{L}_{u_b}(\theta)\|_{L^\infty} \leq C \|\nabla^2_\theta \mathcal{L}_{u_b}(\theta)\|_{L^\infty} \tag{34}$$

where C is a constant that does not depend on $\nabla \mathcal{L}_r(\theta)$ and $\nabla \mathcal{L}_{u_b}(\theta)$. By diagonalizing $\nabla^2 \mathcal{L}_r(\theta)$ and $\nabla^2 \mathcal{L}_{u_b}(\theta)$ with orthonormal matrices, we know that $\|\nabla^2_\theta \mathcal{L}_r(\theta)\|_{L^\infty}$ and $\|\nabla^2_\theta \mathcal{L}_{u_b}(\theta)\|_{L^\infty}$ are bounded by the largest eigenvalues of $\nabla^2 \mathcal{L}_r(\theta)$ and $\nabla^2 \mathcal{L}_{u_b}(\theta)$ respectively. Then again from figure 4a, we can conclude that $\|\nabla^2_\theta \mathcal{L}_r(\theta)\|_{L^\infty}$ is much greater than $\|\nabla^2_\theta \mathcal{L}_{u_b}(\theta)\|_{L^\infty}$. Therefore, equations 33, 34 give another loose upper bound for $\nabla \mathcal{L}_r(\theta)$ and $\nabla \mathcal{L}_{u_b}(\theta)$ and reveal the reason why stiffness of the gradient flow may result in unbalanced gradients pathologies.

Based on these findings we may conclude that the *de facto* use of gradient descent and its modern variants (e.g., Adam [27]) may be a poor, or even unstable choice for training physics-informed neural networks. From a numerical analysis standpoint, it is then natural to investigate whether more stable discretizations, such as implicit or IMEX operator splitting schemes [30, 35], can yield more effective optimization algorithms for constrained neural networks. From an optimization standpoint, this setting has motivated the development of proximal gradient algorithms [36] that have been successfully applied to ill-posed inverse problems involving stiff regularization terms [37], as well as effective algorithms for finding Nash equilibria in multi-player games [33, 38]. Nevertheless, these intriguing directions go beyond the scope of the present work and define an areas for future investigation. As discussed in the next section, here our goal is to retain the simplicity of gradient descent while providing an non-intrusive and adaptive heuristic for



(a) (b)

Figure 4: *Helmholtz equation:* (a) All eigenvalues of $\nabla^2_\theta \mathcal{L}_r(\theta)$ and $\nabla^2_\theta \mathcal{L}_{u_b}(\theta)$, respectively, arranged in increasing order. (b) Loss curves of $\mathcal{L}_r(\theta)$ and $\mathcal{L}_{u_b}$, respectively, after 40,000 iterations of gradient decent using the Adam optimizer in full batch mode.

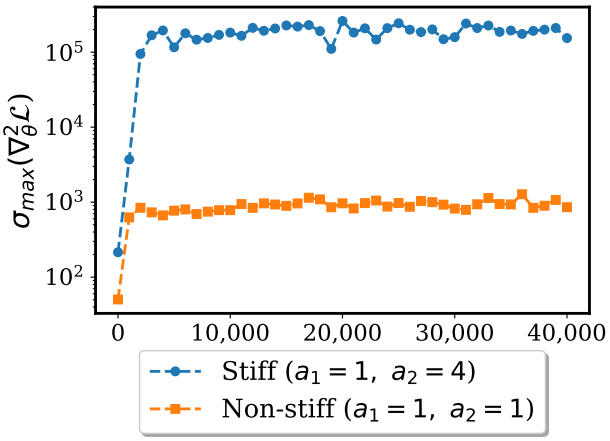


Figure 5: *Stiffness in the gradient flow dynamics:* Largest eigenvalue of the Hessian $\nabla^2_\theta \mathcal{L}(\theta)$ during the training of a physics-informed neural network model for approximating the solution to the two-dimensional Helmholtz problem (see equation 8) and for different values of the control parameters a_1 and a_2.

enhancing its performance. Similar approaches can be found in the numerical analysis literature for integrating stiff and multi-scale dynamical systems, see for example the multi-rate schemes of Gear *et. al.* [39] and the flow-averaging integration techniques put forth by Tao *et. al.* [40].

2.5 Learning rate annealing for physics-informed neural networks

Having identified a common mode of failure for physics-informed neural networks related to unbalanced gradients during back-propagation, we can investigate potential remedies for overcoming this pathology. To do so, let us re-examine the general form of a PINNs loss function

$$\mathcal{L}(\theta) := \mathcal{L}_r(\theta) + \sum_{i=1}^{M} \mathcal{L}_i(\theta), \tag{35}$$

the minimization of which is typically performed according to the following gradient descent update

$$\begin{aligned} \theta_{n+1} &= \theta_n - \eta \nabla_\theta \mathcal{L}(\theta_n) \\ &= \theta_n - \eta [\nabla_\theta \mathcal{L}_r(\theta_n) + \sum_{i=1}^{M} \nabla_\theta \mathcal{L}_i(\theta_n)] \end{aligned} \tag{36}$$

where η is a learning rate parameter. To balance the interplay between the different terms in this loss, a straightforward way is to multiply a constant λ_i to each $\mathcal{L}_i(\theta)$ term. More specifically, we consider minimizing the following loss in which the weights λ resemble the role of penalty coefficients in constrained optimization [41]

$$\mathcal{L}(\theta) := \mathcal{L}_r(\theta) + \sum_{i=1}^{M} \lambda_i \mathcal{L}_i(\theta). \tag{37}$$

Consequently, the corresponding gradient descent updates now take the form

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}(\theta_n) \tag{38}$$

$$= \theta_n - \eta \nabla_\theta \mathcal{L}_r(\theta_n) - \eta \sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta_n), \tag{39}$$

where we see how the constants λ_i can effectively introduce a re-scaling of the learning rate corresponding to each loss term. Obviously, the next question that needs to be answered is should those weights how λ_i be chosen? It is straightforward to see that choosing λ_i arbitrarily following a trial and error procedure is extremely tedious and may not produce satisfying results. Moreover, the optimal constants may vary greatly for different problems, which means we cannot find a fixed empirical recipe that is transferable across different PDEs. Most importantly, the loss function always consists of various parts that serve to provide restrictions on the equation. It is impractical to give different weights to different parts of the loss function manually.

Here we draw motivation from Adam [27] – one the most widely used adaptive learning rate optimizers in the deep learning literature – to derive an adaptive rule for choosing the λ_i weights online during model training. The basic idea behind Adam is to keep track of the first- and second-order moments of the back-propagated gradients during training, and utilize this information to adaptively scale the learning rate associated with each parameter in the θ vector. In a similar spirit, our proposed learning rate annealing procedure, as summarized in algorithm 1, is designed to automatically tune the λ_i weights by utilizing the back-propagated gradient statistics during model training, such that the interplay between all terms in equation 37 is appropriately balanced.

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

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

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

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

(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 $\alpha = 0.9$.

The proposed algorithm is characterized by two steps. First, we compute instantaneous values for the constants $\hat{\lambda}_i$ by computing the ratio between the maximum gradient value attained by $\nabla_\theta \mathcal{L}_r(\theta)$ and the mean of the gradient magnitudes

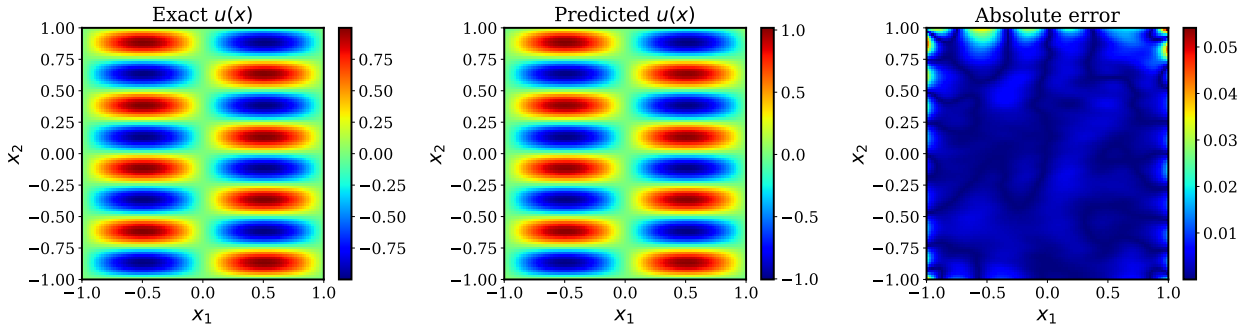


Figure 6: *Helmholtz equation:* Exact solution versus the prediction of a physics-informed neural network model with 4 hidden layers and 50 neurons each layer after 40,000 iterations of training using the proposed learning rate annealing algorithm (relative L^2-error: 1.27e-02).

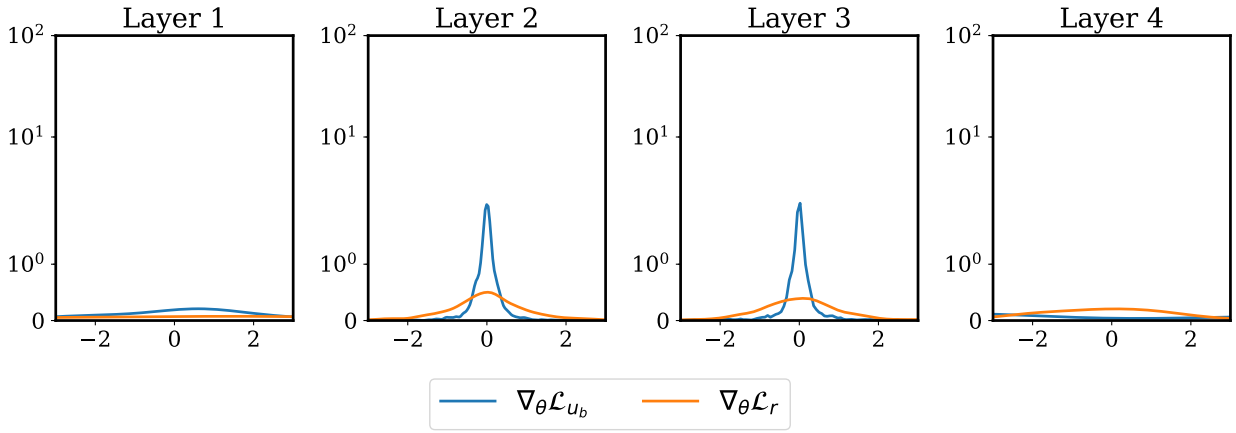


Figure 7: *Helmholtz equation:* Histograms of back-propagated gradients of $\nabla_\theta \mathcal{L}_r$ and $\nabla_\theta \mathcal{L}_{u_b}$ at each layer during training a PINN with the proposed algorithm 1 to solve Helmholtz equation.

computed for each of the $\mathcal{L}_i(\theta)$ loss terms, namely, $\overline{|\nabla_\theta \mathcal{L}_i(\theta)|}$, see equation 40. As these instantaneous values are expected to exhibit high variance due to the stochastic nature of the gradient descent updates, the actual weights λ_i are computed as a running average of their previous values, as shown in equation 41. Notice that the updates in equations 40 and 41 can either take place at every iteration of the gradient descent loop, or at a frequency specified by the user (e.g., every 10 gradient descent steps). Finally, a gradient descent update takes is performed to update the neural network parameters θ using the current weight values stored in λ_i, see equation 42.

The key benefits of the proposed automated procedure is that this adaptive method can be easily generalized to loss functions consisting of multiple terms (e.g., multi-variate problems with multiple boundary conditions on different variables), while the extra computational overhead associated with computing the gradient statistics in equation 40 is small, especially in the case of infrequent updates. Moreover, our computational studies confirm very low sensitivity on the additional hyper-parameter α, as the accuracy of our results does not exhibit any significant variance when this parameter take values within a reasonable range (e.g., $\alpha \in [0.5, 0.9]$).

A first illustration of the effectiveness of the proposed learning-rate annealing algorithm is provided through the lens of the Helmholtz benchmark presented in equation 8. Figures 6 and 7 summarize the predictions of the same 4-layer deep physics-informed neural network model used in section 2.2 after training it using algorithm 1 for $40,000$ gradient descent iterations. Evidently, the proposed training scheme is able to properly balance the interplay between the boundary and the residual loss, and improve the relative prediction error by more than one order of magnitude. In particular, by comparing figures 1 and 6 notice how the absolute point-wise error at the domain boundaries has been significantly reduced. Finally, figure 8 summarizes the convergent evolution of the constant λ_{u_b} used to scale the boundary condition loss $\mathcal{L}_{u_b}(\theta)$ in equation 11 during model training using algorithm 1.

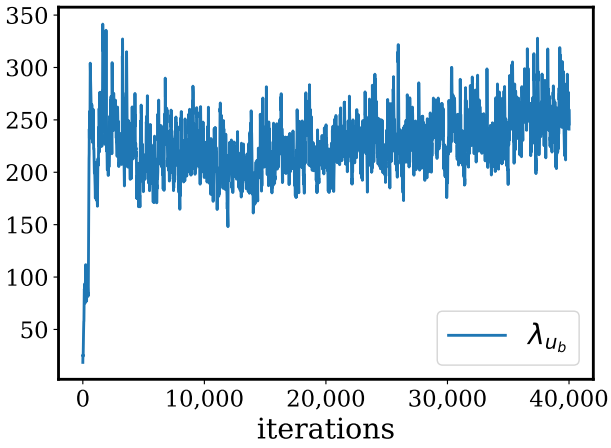


Figure 8: *Helmholtz equation:* Evolution of the constant λ_{u_b} used to scale the boundary condition loss $\mathcal{L}_{u_b}(\theta)$ in equation 11 during model training using algorithm 1.

2.6 An improved fully-connected neural architecture

Besides carefully formulating a loss function (and effective algorithms to minimize it), another ingredient that is key to the success of deep neural networks is the design of their architecture. The most successful neural architectures in the literature are in principle designed to exploit prior knowledge associated with a given task. For instance, convolutional neural networks [1, 9] are widely used for object recognition and image classification tasks. This is because the convolution operation naturally adheres to the symmetries induced by the translation group, which is a crucial feature in image recognition. Similarly, recurrent neural networks [42, 9] are well suited to modeling sequence data due to their ability to respect temporal invariance and capture long-term dependencies. Although respecting the invariances that characterize a given task is crucial in designing an effective neural architecture, in many scenarios related to physical systems the underlying symmetry groups are often non-trivial or even unknown. This is the main reason why the majority of recent works focused on applying deep learning tools to modeling and simulating physical systems governed by PDEs, still relies on generic fully connected architectures [12, 24, 23, 43, 26]. Examples of cases include recent studies on physics-informed neural networks [12, 44], which primarily use fully-connected architectures with weak inductive biases and simply rely on the fact that the representational capacity of these over-parametrized networks can be sufficient to correctly capture the target solution of a given PDE. However, in absence of a universal approximation theorem for physics-informed neural networks (i.e., fully-connected networks trained subject to PDE constraints), it is currently unclear whether standard fully-connected architectures offer sufficiently flexible representations for inferring the solution of complex PDEs. Although this question remains hard to answer, in this section we show how a simple extension to standard fully-connected networks can lead to a novel architecture that appears to perform uniformly and significantly better across all the benchmark studies considered in this work.

Inspired by neural attention mechanisms recently employed for computer vision and natural language processing tasks [45], here we present a novel neural network architecture, which has the following characteristics: (i) explicitly accounts for multiplicative interactions between different input dimensions, and (ii) enhances the hidden states with residual connections. As shown in figure 9 the key extension to conventional fully-connected architectures is the introduction of two transformer networks that project the inputs variables to a high-dimensional feature space, and then use a point-wise multiplication operation to update the hidden layers according to the following forward propagation rule

$$U = \phi(XW^1 + b^1), \;\; V = \phi(XW^2 + b^2) \tag{43}$$

$$H^{(1)} = \phi(XW^{z,1} + b^{z,1}) \tag{44}$$

$$Z^{(k)} = \phi(H^{(k)}W^{z,k} + b^{z,k}), \;\; k = 1, \ldots, L \tag{45}$$

$$H^{(k+1)} = (1 - Z^{(k)}) \odot U + Z^{(k)} \odot V, \;\; k = 1, \ldots, L \tag{46}$$

$$f_\theta(x) = H^{(L+1)}W + b \tag{47}$$

where X denotes the $(n \times d)$ design matrix of the input data-points, and $\odot$ denotes element-wise multiplication. The parameters of this model are essentially the same as in a standard fully-connected architecture, with the addition of the

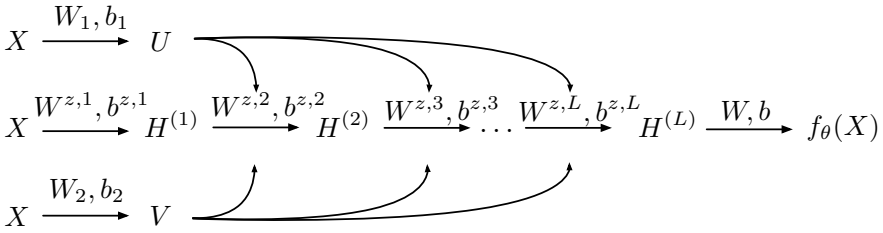


Figure 9: *An improved fully-connected architecture for physics-informed neural networks:* Introducing residual connections and accounting for multiplicative interactions between the inputs can lead to improved predictive performance.

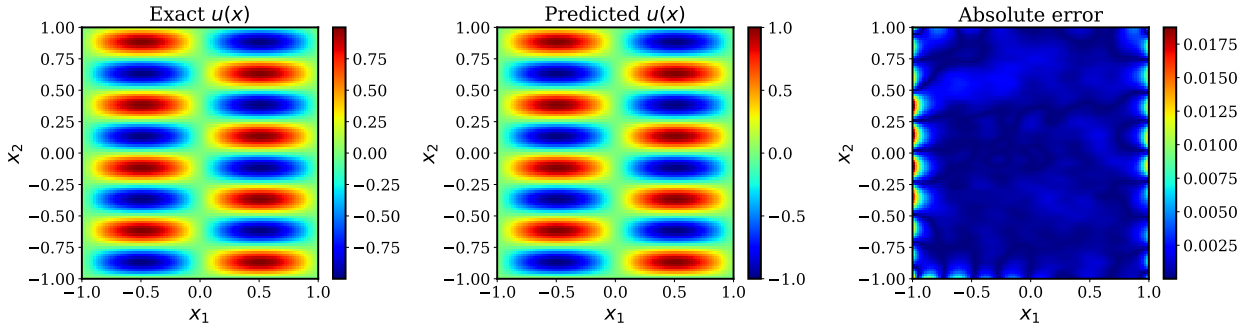


Figure 10: *Helmholtz equation:* Exact solution versus the prediction of a physics-informed neural network model with the proposed improved architecture (4 hidden layers, 50 neurons each) after 40,000 iterations of training using the proposed learning rate annealing algorithm (relative L^2-error: 3.69e-03).

weights and biases used by the two transformer networks, i.e.,

$$\theta = \{W^1, b^1, W^2, b^2, (W^{z,l}, b^{z,l})_{l=1}^{L}, W, b\} \tag{48}$$

Finally, the number of units in each layer is M and $\phi : \mathbb{R}^M \to \mathbb{R}^M$ is a nonlinear activation function. Here we should also notice that the proposed architecture, and its associated forward pass, induce a relatively small computational and memory overhead while leading to significant improvements in predictive accuracy.

A first indication of the improved performance brought by the proposed neural architecture can be seen by revisiting the Helmholtz benchmark discussed in section 2.2. Figure 10 summarizes the prediction of the proposed fully connected architecture with a depth of 4 layers for a physics-informed neural network model trained it using algorithm 1 for $40,000$ gradient descent iterations. Evidently, the proposed training scheme is able to properly balance the interplay between the boundary and the residual loss, and improve the relative prediction error by almost two orders of magnitude compared to the original formulation of Raissi *et. al.* [12]. In particular, by comparing figures 1 and 10 notice how the absolute point-wise error at the domain boundaries has been effectively diminished.

Perhaps a more interesting fact, is that the choice of the neural architecture has also an influence on the stiffness of the gradient flow dynamics. To illustrate this point, we report the largest eigenvalue of the Hessian matrix $\sigma_{\max}(\nabla^2_\theta \mathcal{L}(\theta))$ for the Helmholtz benchmark discussed in section 2.2 using a stiff parameter setting corresponding to $a_1 = 1$, $a_2 = 4$. Specifically, in figure 11 we compare the evolution of $\sigma_{\max}(\nabla^2_\theta \mathcal{L}(\theta))$ during model training for the conventional dense, 4-layer deep architecture employed in section 2.2 (denoted as model M1), versus the proposed neural architecture also with a depth of 4 layers (denoted as model M3). Evidently, the proposed architecture yields a roughly 3x decrease in the magnitude of the leading Hessian eigenvalue, suggesting that tweaks in the neural network architecture could potentially stabilize and accelerate the training of physics-informed neural networks; a direction that will be explored in future work. Moreover, as reported in section 3, the combination of the architecture and the learning rate annealing scheme put forth in section 2.5 consistently improves the accuracy of physics-informed neural networks by a factor of 50-100x across a range of benchmark problems in computational physics.

3 Results

In this section we provide a collection of comprehensive numerical studies that aim to assess the performance or the proposed methodologies against the current state-of-the-art in using fully-connected deep neural network models for

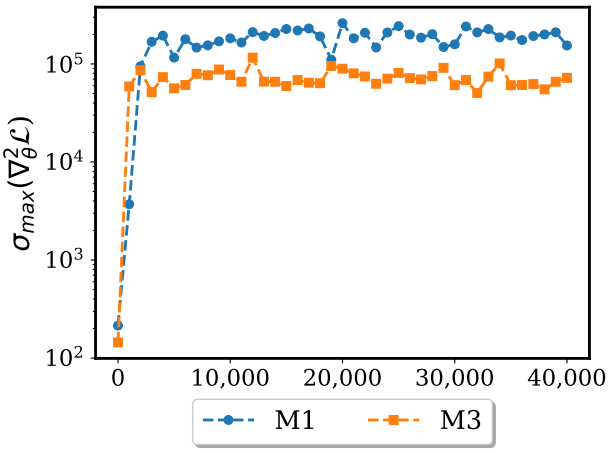


Figure 11: *An improved fully-connected architecture for physics-informed neural networks:* Largest eigenvalue of the Hessian $\nabla^2_\theta \mathcal{L}(\theta)$ during the training of two different physics-informed neural network models for approximating the solution to the two-dimensional Helmholtz problem (see equation 8) with $a_1 = 1$, $a_2 = 4$. Model M1 is a conventional dense, 4-layer deep architecture, while model M3 corresponds to the proposed improved architecture also with a depth of 4 layers.

Model	Description
M1	Original formulation of physics-informed neural networks as put forth in Raissi *et. al.* [12]
M2	Physics-informed neural networks with the proposed learning rate annealing algorithm (see section 2.5)
M3	Physics-informed neural networks with the proposed improved fully-connected architecture (see section 2.6)
M4	Physics-informed neural networks combining the proposed learning rate annealing algorithm and fully-connected architecture

Table 1: Models considered in the numerical studies presented in section 3.

inferring the solution of PDEs. Throughout all benchmarks we will employ and compare the performance of four different approaches as summarized in table 1. In all cases we employ hyperbolic tangent activation functions, a mini-batch size of 128 data-points, and train the networks using single precision arithmetic via stochastic gradient descent using the Adam optimizer with default settings [27]. Moreover, all networks are initialized using the Glorot scheme [28], and no additional regularization techniques are employed (e.g., dropout, L^1/L^2 penalties, etc.). Models M2 and M4 use the proposed learning rate annealing algorithm update their loss weights λ_i every 10 iterations of stochastic gradient descent (see equations 40 and 41). All algorithms are implemented in Tensorflow [46] and the reported run-times are obtained on a Lenovo X1 Carbon ThinkPad laptop with an Intel Core i5 2.3GHz processor and 8Gb of RAM memory. The code and data accompanying this manuscript is publicly available at `https://github.com/PredictiveIntelligenceLab/GradientPathologiesPINNs`.

3.1 Helmholtz Equation

First, let us revisit the Helmholtz equation benchmark described in section 2.2 (see equation 8). This equation is closely related to many problems in natural and engineering sciences, such as wave propagation in acoustic, elastic and electromagnetic media [47]. It also routinely arises in conventional numerical discretization methods for PDEs such as finite element and spectral methods [48, 49]. Our goal here is to use this canonical benchmark problem to systematically analyze the performance of the different models listed in table 1, and quantify their predictive accuracy for different choices of the underlying neural network architecture. Specifically, we have varied the number of hidden layers and the number of neurons per layer that define each model's architecture and report the resulting relative L^2 error for the predicted solution after $40,000$ Adam iterations.

Table 2 summarizes our results, averaged over 10 independent trials with randomized weight initializations using the Glorot scheme [28]. Evidently, the original formulation of Raissi *et. al.* [12] (model M1) is very sensitive to the choice

of neural architecture and fails to attain a stable prediction accuracy, yielding errors ranging between 8-31% in the relative L^2 norm. In contrast, the proposed models (models M2-M4) appear to be very robust with respect to network architectures and show a consistent trend in improving the prediction accuracy as the number of hidden layers and neural units is increased. We also observe that, by introducing a small number of additional weights and biases, the improved fully-connected architecture corresponding to model M3 can consistently yield better prediction accuracy than the conventional fully-connected architectures used in the original work of Raissi *et. al.* [12] (model M1). This indicates that the proposed architecture seems to have better ability to represent complicated functions than conventional fully-connected neural networks, as well as it may lead to more stable gradient descent dynamics as discussed in section 2.6. Last but not least, the combination of the proposed improved architecture with the adaptive learning rate algorithm of section 2.5 (model M4) uniformly leads to the most accurate results we have obtained for this problem (relative L^2 errors ranging between 0.12-0.25%).

Architecture	M1	M2	M3	M4
30 units / 3 hidden layers	2.44e-01	5.39e-02	5.31e-02	1.07e-02
50 units / 3 hidden layers	1.06e-01	1.12e-02	2.46e-02	3.87e-03
100 units / 3 hidden layers	9.07e-02	4.84e-03	1.17e-02	2.71e-03
30 units / 5 hidden layers	2.47e-01	2.74e-02	4.12e-02	3.49e-03
50 units / 5 hidden layers	1.40e-01	7.43e-03	1.97e-02	2.54e-03
100 units / 5 hidden layers	1.15e-01	5.37e-03	1.08e-02	1.63e-03
30 units / 7 hidden layers	3.10e-01	2.67e-02	3.17e-02	3.59e-03
50 units / 7 hidden layers	1.98e-01	7.33e-03	2.37e-02	2.04e-03
100 units / 7 hidden layers	8.14e-02	4.52e-03	9.36e-03	1.49e-03

Table 2: *Helmholtz equation:* Relative L^2 error between the predicted and the exact solution $u(x, y)$ for the different methods summarized in table 1, and for different neural architectures obtained by varying the number of hidden layers and different number of neurons per layer.

3.2 Klein-Gordon equation

To emphasize the ability of the proposed methods to handle nonlinear and time-dependent problems, leading to composite loss functions with multiple competing objectives, let us consider the time-dependent Klein-Gordon equation. The Klein–Gordon equation is a non-linear equation, which plays a significant role in many scientific applications such as solid-state physics, nonlinear optics, and quantum physics [50]. The initial-boundary value problem in one spatial dimension takes the form

$$u_{tt} + \alpha u_{xx} + \beta u + \gamma u^k = f(x,t), \qquad (x,t) \in \Omega \times [0,T] \tag{49}$$

$$u(x,0) = g_1(x), \qquad x \in \Omega \tag{50}$$

$$u_t(x,0) = g_2(x), \qquad x \in \Omega \tag{51}$$

$$u(x,t) = h(x,t), \qquad (x,t) \in \partial\Omega \times [0,T] \tag{52}$$

where α, β, γ and k are known constants, $k = 2$ when we have a quadratic non-linearity and $k = 3$ when we have a cubic non-linearity. The functions $f(x,t), g_1(x), g_2(x)$ and $h(x)$ are considered to be known, and the function $u(x,t)$ represents the latent solution to this problem. Here we choose $\Omega = [0,1] \times [0,1], T = 1, \alpha = -1, \beta = 0, \gamma = 1, k = 3$, and the initial conditions satisfying $g_1(x) = g_2(x) = 0$ for all $x \in \Omega$. To assess the accuracy of our models we will use a fabricated solution

$$u(x,t) = x\cos(5\pi t) + (xt)^3, \tag{53}$$

and correspondingly derive a consistent forcing term $f(x,t)$ using equation 49, while the Dirichlet boundary condition term $h(x,y)$ imposed on the spatial domain boundaries $\partial\Omega$ is directly extracted from the fabricated solution of equation 53.

A physics-informed neural network model $f_\theta(x,t)$ can now be trained to approximate the latent solution $u(x,t)$ by formulating the following composite loss

$$\mathcal{L}(\theta) = \mathcal{L}_r(\theta) + \lambda_{u_b}\mathcal{L}_{u_b}(\theta) + \lambda_{u_0}\mathcal{L}_{u_0}(\theta) \tag{54}$$

where $\mathcal{L}_r(\theta), \mathcal{L}_{u_b}(\theta), \mathcal{L}_{u_0}(\theta)$ are defined in equations 5, 6, and 7. Notice that if $\lambda_{u_b} = \lambda_{u_0} = 1$, then the loss function above gives rise to the original physics-informed neural network formulation of Raissi *et. al.* [12] (model M1).

Given this problem setup, let us now investigate the distribution of back-propagated gradients during the training of a 5-layer deep fully-connected network with 50 neurons per layer using the the original formulation of Raissi *et. al.* [12]

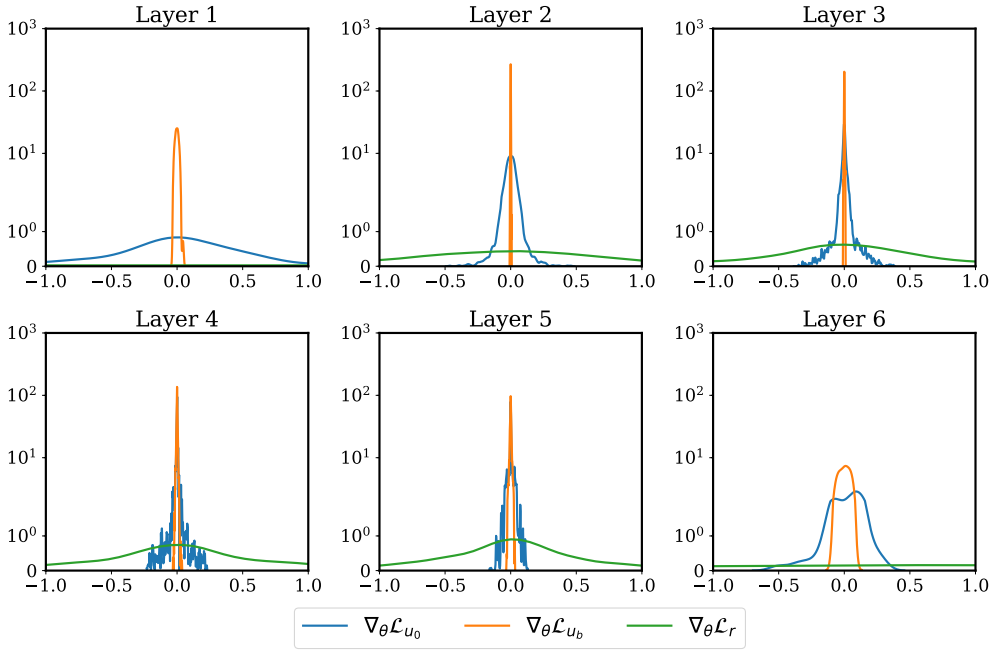


Figure 12: *Klein-Gordon equation:* Histograms of back-propagated gradients for model M1 after 40,000 iterations of stochastic gradient descent using the Adam optimizer.

(model M1). The results summarized in figure 12 indicate that $\nabla_\theta \mathcal{L}_{u_b}$ and $\nabla_\theta \mathcal{L}_{u_0}$ accumulate at the origin around which they form two sharp peaks, while $\nabla_\theta \mathcal{L}_r$ keeps flat. This implies that the gradients of $\mathcal{L}_{u_b}$ and $\mathcal{L}_{u_r}$ nearly diminish in comparison to the gradients of $\mathcal{L}_r$. This is a clear manifestation of the imbalanced gradient pathology described in section 2.2, which we believe is the leading mode of failure for conventional physics-informed neural network models. Consequently, model M1 fails to accurately fit the initial and boundary conditions data, and therefore, as expected, yields a poor prediction for the target solution $u(x,t)$ with a relative L^2 error of 17.9%. From our experience, this behavior is extremely common in using conventional physics-informed neural network models [12] for solving PDE systems that have sufficiently complex dynamics leading to solutions with non-trivial behavior (e.g., directional anisotropy, multi-scale features, etc.).

It is now natural to ask whether the proposed methods can effectively mitigate this gradient imbalance pathology and lead to accurate and robust predictions. To this end, in figure 13 we present the distribution of back-propagated gradients for $\nabla_\theta \mathcal{L}_r$, $\nabla_\theta \mathcal{L}_{u_b}$, $\nabla_\theta \mathcal{L}_{u_0}$ at each hidden layer of the same neural architecture used above, albeit using the proposed learning rate annealing algorithm (model M2) for adaptively choosing the constants λ_{u_b} and λ_{u_0} during model training. We observe that all gradient distributions do not sharply peak around zero values indicating the fact that a healthy gradient signal is back-propagated through the network from all the contributing terms in the loss function.

A comparative study on the performance of all models (M1-M4) is summarized in table 3. Notice that here our goal is not to perform an exhaustive hyper-parameter search to find the best performing model, rather our goal is to present indicative results for the performance of each method (models M1-M4). To do so, we consider a fixed neural architecture with 5 hidden layers and 50 neurons and report the relative L^2 prediction error of each method after $40,000$ iterations of training, along with the total wall clock time required to complete each simulation. It is clear that the proposed methodologies consistently outperform the original implementation of Raissi *et. al.* [12], while the combined use of the proposed learning rate annealing algorithm with the improved fully-connected architecture (model M4) lead to the best results, albeit at a higher computational cost.

Figures 14 and 15 provide a more detailed visual assessment of the predictive accuracy for models M1 and M4. As we expected, in figure 14, model M1 fails to obtain accurate predictive solutions and has increased absolute error up to about 0.2 as time t increases. In figure 15, we present the solution obtained model M4 that combines the proposed learning rate annealing algorithm (see section 2.5) and improved neural architecture put forth in section 2.6. It can be observed that the absolute error takes a maximum value of 0.004 and the relative L^2 prediction error is 0.28%, which is about two orders of magnitude lower than the one obtained using model M1, and one order of magnitude lower that the predictive error of models M2 and M3. Finally, figure 13 depicts the evolution of the constants λ_{u_b} and λ_{u_0} used to

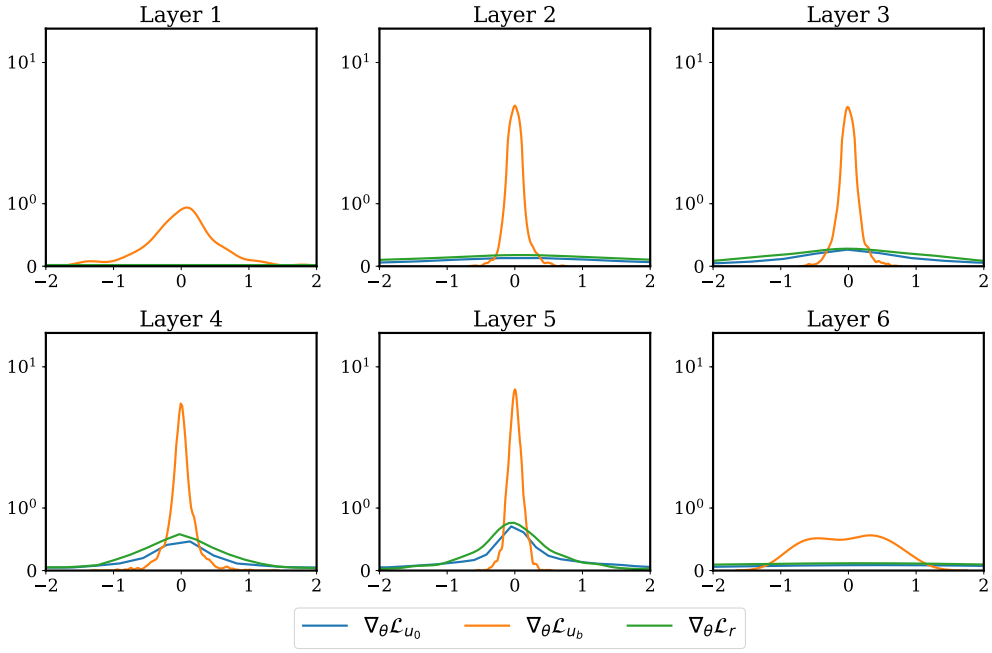


Figure 13: *Klein-Gordon equation:* Back-propagated gradients for model M2 after 40,000 iterations of stochastic gradient descent using the Adam optimizer.

Method	M1	M2	M3	M4
Relative L^2 error	1.79e-01	1.06e-02	1.97e-02	2.81e-03
Training time (sec)	200	200	760	1040

Table 3: *Klein-Gordon equation:* Relative L^2 error between the predicted and the exact solution $u(t, x)$ for the different methods summarized in table 1. Here the network architecture is fixed to 5 fully-connected layers with 50 neurons per layer. The reported CPU time corresponds to the total wall-clock time to train each network for 40,000 iterations of stochastic gradient descent on a Lenovo X1 Carbon ThinkPad laptop with an Intel Core i5 2.3GHz processor and 8Gb of RAM memory.

scale the boundary and initial conditions loss, $\mathcal{L}_{u_b}(\theta)$ and $\mathcal{L}_{u_0}(\theta)$, respectively (see equation 54), during the training of model M2 using algorithm 1.

3.3 Flow in a lid-driven cavity

In our last example, we present a study that highlights a few important remarks on how a problem's formulation can be as crucial as the algorithm used to solve it. To this end, let us consider a classical benchmark problem in computational fluid dynamics, the steady-state flow in a two-dimensional lid-driven cavity. The system is governed by the incompressible Navier-Stokes equations which can be written in non-dimensional form as

$$\boldsymbol{u} \cdot \nabla \boldsymbol{u} + \nabla p - \frac{1}{Re} \Delta \boldsymbol{u} = 0 \qquad \text{in } \Omega \tag{55}$$

$$\nabla \cdot \boldsymbol{u} = 0 \qquad \text{in } \Omega \tag{56}$$

$$\boldsymbol{u}(\boldsymbol{x}) = (1, 0) \qquad \text{on } \Gamma_1 \tag{57}$$

$$\boldsymbol{u}(\boldsymbol{x}) = (0, 0) \qquad \text{on } \Gamma_0 \tag{58}$$

where $\boldsymbol{u}(\boldsymbol{x}) = (u(\boldsymbol{x}), v(\boldsymbol{x}))$ is a velocity vector field, p is a scalar pressure field, and $\boldsymbol{x} = (x, y) \in \Omega = (0, 1) \times (0, 1)$ with Ω being a two-dimensional square cavity. Moreover, Γ_1 is the top boundary of the cavity, Γ_0 denotes the other three sides, and Re is the Reynolds number of the flow. In this example, we chosen a relatively simple case corresponding to a Reynolds number of $Re = 100$, for which it is well understood that the flow quickly converges to an incompressible steady-state solution [51]. Our goal here is to train a physics-informed neural network to predict the latent velocity and pressure fields. In order to assess the accuracy of our predictions, we have also simulated this system to obtain a

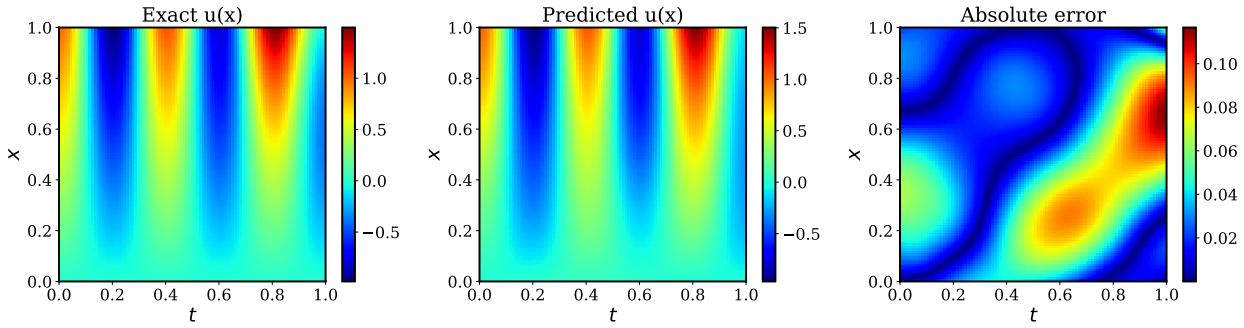


Figure 14: *Klein Gordon equation:* Predicted solution of model M1, a conventional physics-informed neural network [12] with 5 hidden layers and 50 neurons in each layer.The relative L^2 error of the prediction solution is 1.79e-01 after 40,000 iterations of stochastic gradient descent using the Adam optimizer.

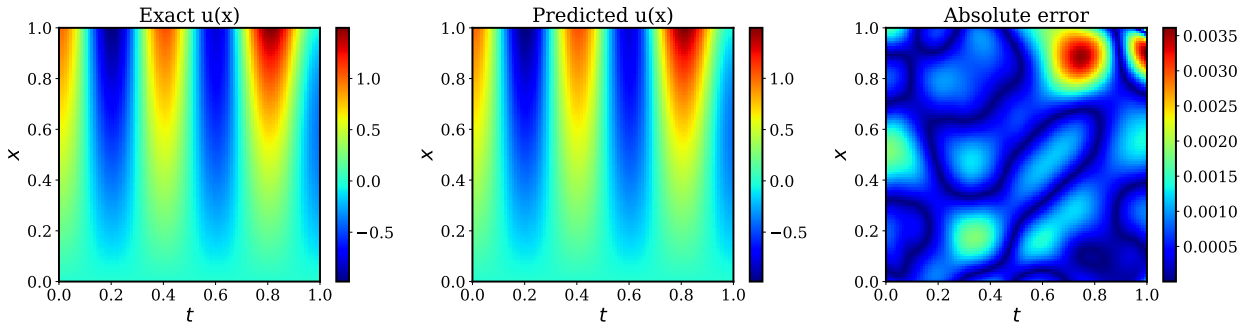


Figure 15: *Klein Gordon equation:* Predicted solution of model M4, a physics-informed neural network [12] with 5 hidden layers and 50 neurons in each layer trained using the proposed learning rate annealing algorithm (see section 2.5), and the improved architecture described in 2.6. The relative L^2 error of the prediction solution is 2.22e-03 after 40,000 iterations of stochastic gradient descent using the Adam optimizer.

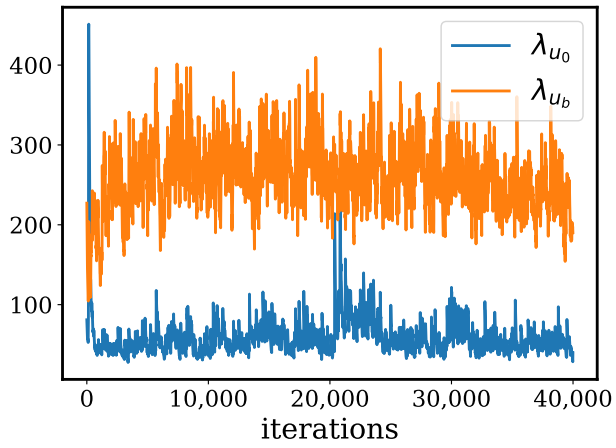


Figure 16: *Klein Gordon equation:* Evolution of the constants λ_{u_b} and λ_{u_0} used to scale the boundary and initial conditions loss, $\mathcal{L}_{u_b}(\theta)$ and $\mathcal{L}_{u_0}(\theta)$, respectively (see equation 54), during the training of model M2 using algorithm 1.

reference solution using conventional finite difference methods, there-by creating a high-resolution validation data set $\{\boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{u}^{i}\}_{i=1}^{N}$ (see details in Appendix 5.2). In the following subsections we will consider two different neural network representations to highlight a crucial aspect of simulating incompressible fluid flows with physics-informed neural networks.

3.4 Velocity-pressure representation

A straightforward application of physics-informed neural networks for approximating the solution of equation 55 would introduce a neural network representation with parameters θ that aims to learn how to map spatial coordinates (x, y) to the latent solution functions $u(x, y)$, $v(x, y)$ and $p(x, y)$, i.e.,

$$[x, y] \overset{f_\theta}{\longmapsto} [u(x, y), v(x, y), p(x, y)]$$

Using this representation one can then introduce the residuals of the momentum and continuity equations as

$$r_\theta^u(x, y) := u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} - \frac{1}{Re}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \tag{59}$$

$$r_\theta^v(x, y) := u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} - \frac{1}{Re}(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) \tag{60}$$

$$r_\theta^c(x, y) = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}, \tag{61}$$

where $(u, v, p) = f_\theta(x, y)$ are the outputs of the chosen neural network representation, and all required gradients can be readily computed using automatic differentiation [18]. Given these residuals, along with a set of appropriate boundary conditions for equation 55 we can now formulate a loss function for training a physics-informed neural network as

$$\mathcal{L}(\theta) = \mathcal{L}_{r_u}(\theta) + \mathcal{L}_{r_v}(\theta) + \mathcal{L}_{r_c}(\theta) + \mathcal{L}_{u_b}(\theta) + \mathcal{L}_{v_b}(\theta), \tag{62}$$

with

$$\mathcal{L}_{r_u}(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r}[r_\theta^u(x_r^i, y_r^i)]^2 \tag{63}$$

$$\mathcal{L}_{r_v}(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r}[r_\theta^v(x_r^i, y_r^i)]^2 \tag{64}$$

$$\mathcal{L}_{r_c}(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r}[r_\theta^c(x_r^i, y_r^i)]^2 \tag{65}$$

$$\mathcal{L}_{u_b}(\theta) = \frac{1}{N_b}\sum_{i=1}^{N_b}[u(x_b^i, y_b^i) - u_b^i]^2, \tag{66}$$

$$\mathcal{L}_{v_b}(\theta) = \frac{1}{N_b}\sum_{i=1}^{N_b}[v(x_b^i, y_b^i) - v_b^i]^2 \tag{67}$$

where $\{(x_r^i, y_r^i)\}_{i=1}^{N_r}$ is a set of collocation points in which we aim to minimize the PDE residual, while $\{(x_b^i, y_b^i), u_b^i\}_{i=1}^{N_b}$ and $\{(x_b^i, y_b^i), v_b^i\}_{i=1}^{N_b}$ denote the boundary data for the two velocity components at the domain boundaries Γ_0 and Γ_1, respectively. Taken together, these terms aim to constrain the neural network approximation to satisfy the Navier-Stokes system and boundary conditions prescribed in equation 55. In practice, this composite loss is minimized using stochastic gradient descent, where a different set of collocation and boundary points are randomly sampled at each gradient descent iteration.

Here, our goal is to learn the two velocity components $u(x, y)$ and $v(x, y)$, as well as the latent pressure field $p(x, y)$ by training a 5-layer deep neural network using the mean squared error loss of 62. Each hidden layer contains 50 neurons and a hyperbolic tangent activation function. Figure 17 summarizes the results of our experiment, which shows the magnitude of the predicted solution $|\boldsymbol{u}(x)| = \sqrt{u^2(\boldsymbol{x}) + v^2(\boldsymbol{x})}$ predicted by each model M1-M4, after $40,000$ stochastic gradient descent updates using the Adam optimizer [27]. It is evident that none of the models is capable of producing a reasonable approximation. We postulate that this failure is related to the residual loss used to train the models. Specifically, it is inevitable that solutions inferred by minimizing equation 62 will not exactly satisfy the

prescribed boundary conditions, nor the incompressibility constraint, potentially giving rise to issues of non-uniqueness. This fundamental limitation highlights the importance of adopting a consistent problem formulation, leading to effective objectives for training the physics-informed neural networks. As we will see in the next section, representing the latent velocity field as the gradient of a scalar potential can guarantee incompressible solutions and lead to stable training dynamics, as well as accurate predictions.

3.5 Streamfunction-pressure representation

Instead of directly representing the two velocity components $u(x, y)$ and $v(x, y)$, one can use a neural network represent a scalar potential function $\psi(x, y)$ such that an incompressible velocity field can be directly computed as $(u, v) = (\partial\psi/\partial y, -\partial\psi/\partial x)$. Hence, we can introduce a neural network representation with parameters θ that aims to learn how to map spatial coordinates (x, y) to the latent scalar functions $\psi(x, y)$, $p(x, y)$, i.e.,

$$[x, y] \overset{f_\theta}{\longmapsto} [\psi(x, y), p(x, y)]$$

Then, we only need to consider the residuals for the momentum equations, as the incompressibility constraint is now exactly satisfied, i.e.

$$r^u_\theta(x, y) := u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} - \frac{1}{Re}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \tag{68}$$

$$r^v_\theta(x, y) := u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} - \frac{1}{Re}(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) \tag{69}$$

where $(u, v) = (\partial\psi/\partial y, -\partial\psi/\partial x)$, $(\psi, p) = f_\theta(x, y)$ are the outputs of the chosen neural network representation f_θ, and all required gradients are computed using automatic differentiation [18]. Although we are now able to exactly satisfy the incompressibilty constraint, notice that this approach incurs a larger computational cost since third-order derivatives of the neural network are need to be computed. Despite this additional cost, our experience indicates that this modification is crucial for obtaining accurate and unique solutions for equation 55 by minimizing a physics-informed loss of the form

$$\mathcal{L}(\theta) = \mathcal{L}_{r_u}(\theta) + \mathcal{L}_{r_v}(\theta) + \mathcal{L}_{u_b}(\theta) + \mathcal{L}_{v_b}(\theta), \tag{70}$$

where

$$\mathcal{L}_{r_u}(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r}[r^u_\theta(x^i_r, y^i_r)]^2 \tag{71}$$

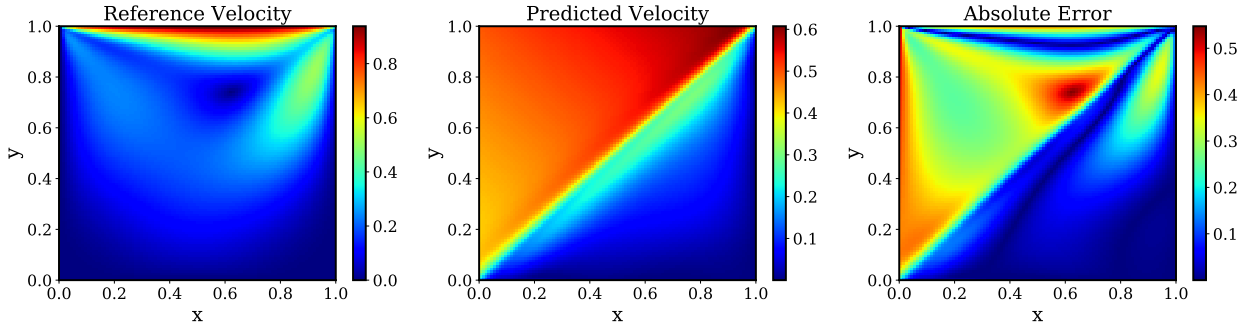
$$\mathcal{L}_{r_v}(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r}[r^v_\theta(x^i_r, y^i_r)]^2 \tag{72}$$

$$\mathcal{L}_{u_b}(\theta) = \frac{1}{N_b}\sum_{i=1}^{N_b}[u(x^i_b, y^i_b) - u^i_b]^2, \tag{73}$$

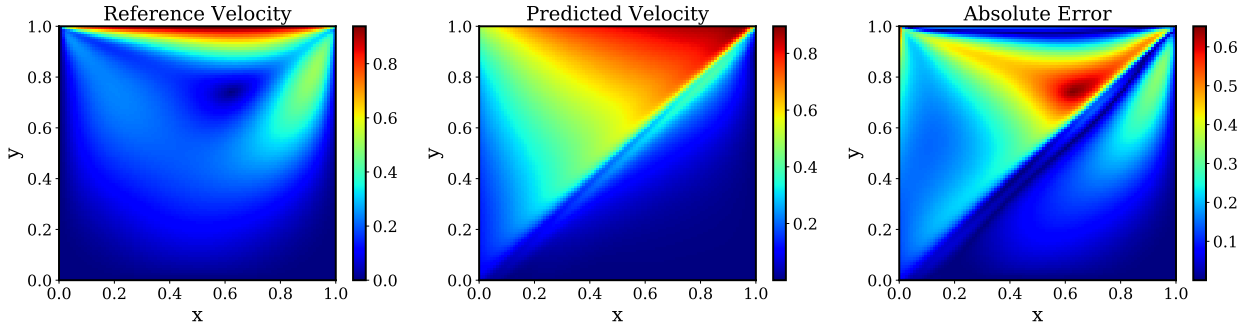
$$\mathcal{L}_{v_b}(\theta) = \frac{1}{N_b}\sum_{i=1}^{N_b}[v(x^i_b, y^i_b) - v^i_b]^2 \tag{74}$$

where $\{(x^i_r, y^i_r)\}_{i=1}^{N_r}$ is a set of collocation points in which we aim to minimize the residual of equations 68 and 69, while $\{(x^i_b, y^i_b), u^i_b\}_{i=1}^{N_b}$ and $\{(x^i_b, y^i_b), v^i_b\}_{i=1}^{N_b}$ denote the boundary data for the two velocity components at the domain boundaries Γ_0 and Γ_1, respectively.

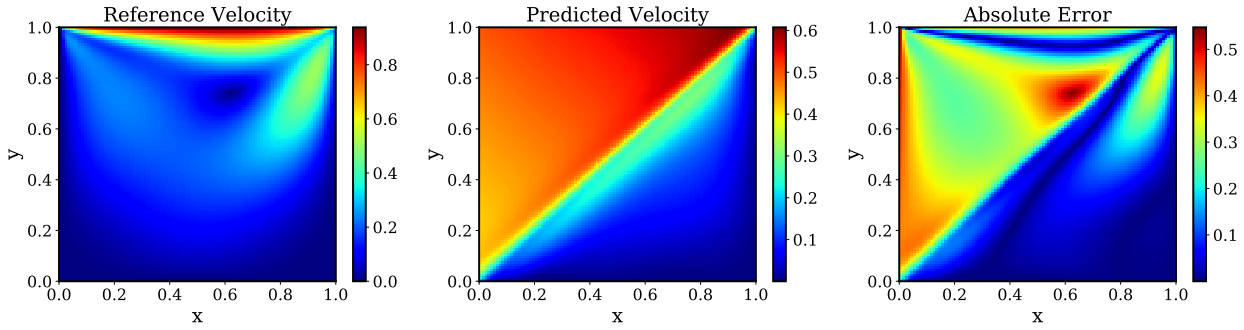
Similarly, we still aim to learn the velocity components $u(x, y)$ and $v(x, y)$, as well as the latent pressure field $p(x, y)$. Now the velocity components can be obtained by taking derivatives of the stream function $\psi(x, y)$ with respect to the x and y coordinates using automatic differentiation [18]. We chose to jointly represent the latent function $[\psi(x, y), p(x, y)]$ using a 5-layer deep neural network with 50 neurons per hidden layer and a hyperbolic tangent activation function, trained using the mean squared error loss of equation 71. The results of this experiment are summarized in figure 18. In particular, we present a comparison between the reference and the predicted solutions obtained using the different models M1 - M4. It can be observed that the velocity fields obtained using M2-M4 are in good agreement with the reference solution while M1 is not able to yield satisfactory prediction solution. As expected, model M4 yields the best predicted solution with an error of 3% measured in the relative L^2-norm.



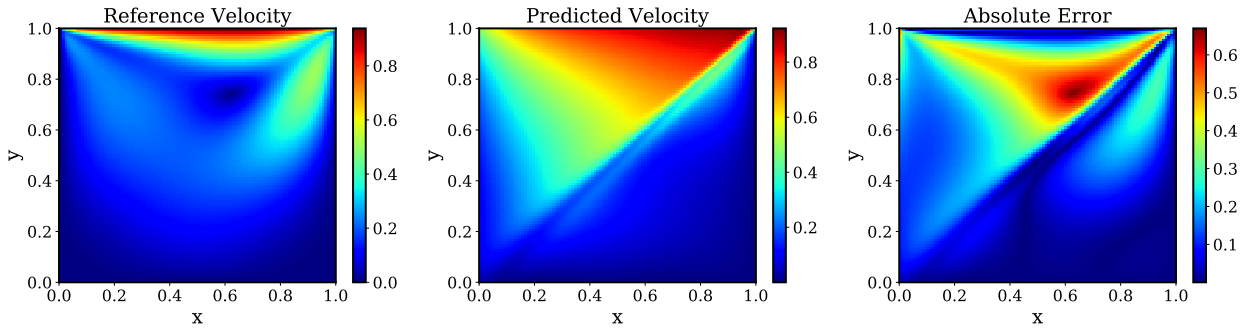
(a) Model M1 (relative L^2 prediction error: 7.84e-01).



(b) Model M2 (relative L^2 prediction error: 7.48e-01).

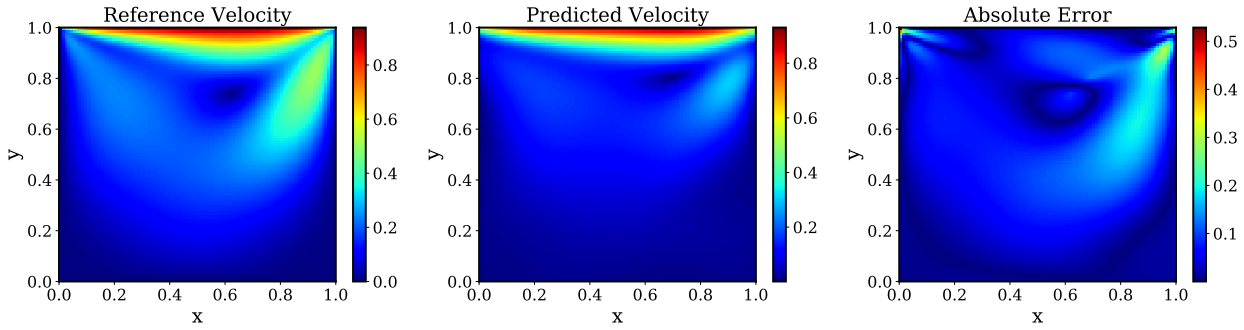


(c) Model M3 (relative L^2 prediction error: 7.48e-01).

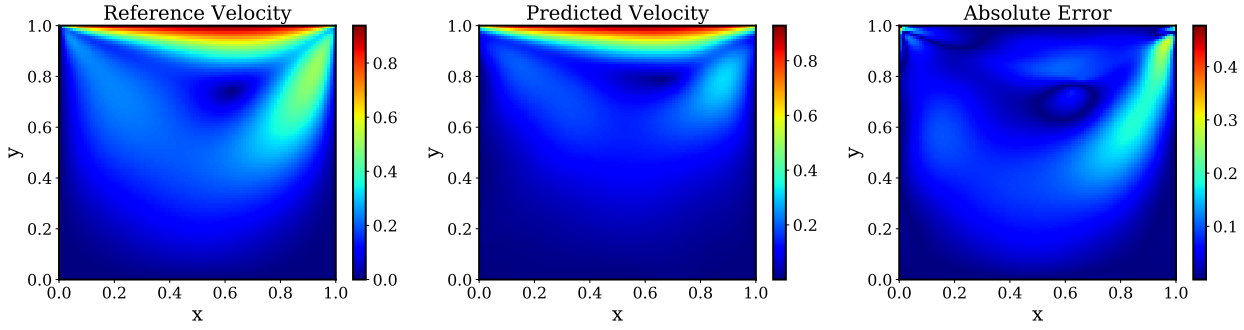


(d) Model M4 (relative L^2 prediction error: 7.53e-01).

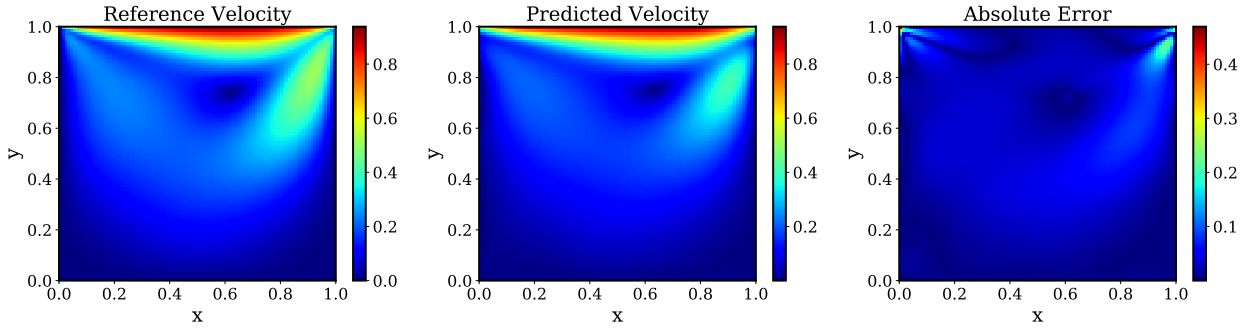
Figure 17: *Flow in a lid-driven cavity, velocity-pressure representation:* Reference solution using a conventional finite difference solver, prediction of a 5-layer deep physics-informed neural network (models M1-M4, top to bottom), and absolute point-wise error.



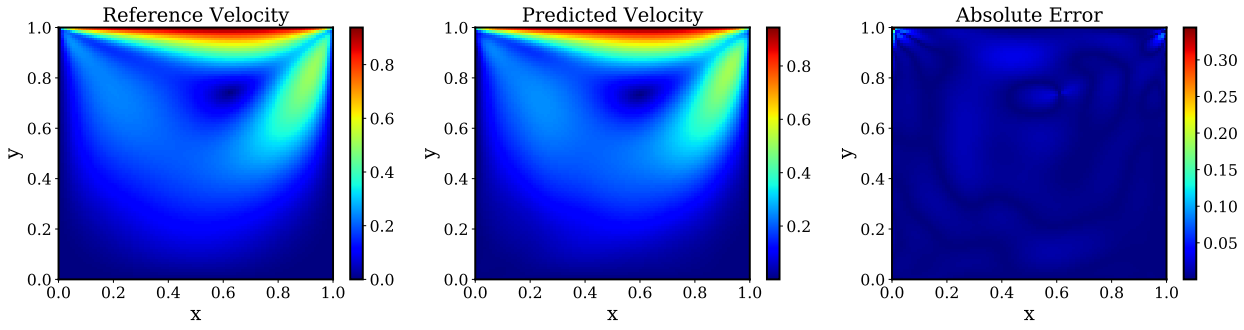
(a) Model M1 (relative L^2 prediction error: 2.71e-01).



(b) Model M2 (relative L^2 prediction error: 2.49e-01).



(c) Model M3 (relative L^2 prediction error: 1.21e-01).



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

Figure 18: *Flow in a lid-driven cavity, streamfunction-pressure representation:* Reference solution using a conventional finite difference solver, prediction of a 5-layer deep physics-informed neural network (models M1-M4, top to bottom), and absolute point-wise error.

4 Summary and Discussion

Despite recent success across a range of applications [12, 43, 25, 24, 26, 19], physics-informed neural networks (PINNs) often struggle to accurately approximate the solution of partial differential equations. In this work we identify and analyze a fundamental mode of failure of physics-informed neural networks related to stiff gradient flow dynamics that lead to unbalanced gradients during model training via back-propagation. To provide further insight, we quantify and analyze the stiffness of the gradient flow dynamics and elucidate the difficulties of training PINNs via gradient descent. This stiffness analysis not only plays a key role in understanding the gradient flow dynamics, but also motivates future research directions to consider more stable discretizations of the gradient flow system. We mitigate this shortcoming by proposing a learning rate annealing algorithm that utilizes gradient statistics during model training to adaptive assign appropriate weights to different terms in a PINNs loss function. The new algorithm aims to address the unbalanced gradients pathology and effectively lead to noticeable improvements in predictive accuracy. This development is not limited to PINNs, but can be straightforwardly generalized to other tasks involving the interplay of multiple objective functions that may lead to unbalanced gradients issue, e.g. multi-task learning [52]. Finally, we propose a novel neural network architecture that can enhance the accuracy and robustness of PINNs by decreasing the stiffness of gradient flow, indicating that specialized architectures can play a prominent role in the future success of PINNs models. Taken together, our developments provide new insights into the training of physics-informed neural networks and consistently improve their predictive accuracy by a factor of 50-100x across a range of problems in computational physics.

Despite recent progress, we have to admit that we are still at the very early stages of rigorously understanding the capabilities and limitations of physics-informed neural networks. To bridge this gap we need to answer a series of open questions: What is the relation between stiffness in a given PDE and stiffness in the gradient flow dynamics of the associated PINN model? Can stiffness in the gradient flow dynamics be reduced (for 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? Addressing these challenges calls for a fruitful synergy between deep learning, optimization, numerical analysis and dynamical systems theory that has the potential to crystallize new exciting developments in computational science and engineering.

Acknowledgements

This work received support from the US Department of Energy under the Advanced Scientific Computing Research program (grant de-sc0019116) and the Defense Advanced Research Projects Agency under the Physics of Artificial Intelligence program (grant HR00111890034).

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[3] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 33(8):831, 2015.

[4] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.

[5] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[6] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[7] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[8] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[10] Stéphane Mallat. Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150203, 2016.

[11] Risi Kondor, Hy Truong Son, Horace Pan, Brandon Anderson, and Shubhendu Trivedi. Covariant compositional networks for learning graphs. *arXiv preprint arXiv:1801.02144*, 2018.

[12] 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.

[13] Yibo Yang and Paris Perdikaris. Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics*, 394:136–152, 2019.

[14] Ehsan Kharazmi, Zhongqiang Zhang, and GE Karniadakis. Variational physics-informed neural networks for solving partial differential equations. *arXiv preprint arXiv:1912.00873*, 2019.

[15] Alexandre M Tartakovsky, Carlos Ortiz Marrero, D Tartakovsky, and David Barajas-Solano. Learning parameters and constitutive relationships with physics informed deep neural networks. *arXiv preprint arXiv:1808.03398*, 2018.

[16] Dimitris C Psichogios and Lyle H Ungar. A hybrid neural network-first principles approach to process modeling. *AIChE Journal*, 38(10):1499–1511, 1992.

[17] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[18] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153), 2018.

[19] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: A navier-stokes informed deep learning framework for assimilating flow visualization data. *arXiv preprint arXiv:1808.04327*, 2018.

[20] Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, 19(1):932–955, 2018.

[21] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.

[22] Nicholas Geneva and Nicholas Zabaras. Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. *Journal of Computational Physics*, 403:109056, 2020.

[23] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

[24] Rohit K Tripathy and Ilias Bilionis. Deep uq: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *Journal of Computational Physics*, 375:565–588, 2018.

[25] Georgios Kissas, Yibo Yang, Eileen Hwuang, Walter R Witschey, John A Detre, and Paris Perdikaris. Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow mri data using physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 358:112623, 2020.

[26] 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.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[29] Lawrence C Evans. Partial differential equations. *Providence, RI*, 1998.

[30] Arieh Iserles. *A first course in the numerical analysis of differential equations*. Number 44. Cambridge university press, 2009.

[31] Levent Sagun, Leon Bottou, and Yann LeCun. Eigenvalues of the hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

[32] Panayotis Mertikopoulos, Christos Papadimitriou, and Georgios Piliouras. Cycles in adversarial regularized learning. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2703–2717. SIAM, 2018.

[33] David Balduzzi, Sebastien Racaniere, James Martens, Jakob Foerster, Karl Tuyls, and Thore Graepel. The mechanics of n-player differentiable games. *arXiv preprint arXiv:1802.05642*, 2018.

[34] Robert A Adams and John JF Fournier. *Sobolev spaces*, volume 140. Elsevier, 2003.

[35] Gilbert Strang. On the construction and comparison of difference schemes. *SIAM journal on numerical analysis*, 5(3):506–517, 1968.

[36] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[37] Morteza Mardani, Qingyun Sun, David Donoho, Vardan Papyan, Hatef Monajemi, Shreyas Vasanawala, and John Pauly. Neural proximal gradient descent for compressive imaging. In *Advances in Neural Information Processing Systems*, pages 9573–9583, 2018.

[38] Florian Schäfer and Anima Anandkumar. Competitive gradient descent. *arXiv preprint arXiv:1905.12103*, 2019.

[39] Charles William Gear and DR Wells. Multirate linear multistep methods. *BIT Numerical Mathematics*, 24(4):484–502, 1984.

[40] Molei Tao, Houman Owhadi, and Jerrold E Marsden. Nonintrusive and structure preserving multiscale integration of stiff odes, sdes, and hamiltonian systems with hidden slow dynamics via flow averaging. *Multiscale Modeling & Simulation*, 8(4):1269–1324, 2010.

[41] Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[42] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[43] AM Tartakovsky, CO Marrero, P Perdikaris, GD Tartakovsky, and D Barajas-Solano. Learning parameters and constitutive relationships with physics informed deep neural networks, arxiv preprint (2018). *arXiv preprint arXiv:1808.03398*.

[44] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Multistep neural networks for data-driven discovery of nonlinear dynamical systems. *arXiv preprint arXiv:1801.01236*, 2018.

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[46] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[47] Jean-Claude Nédélec. *Acoustic and electromagnetic equations: integral representations for harmonic problems*. Springer Science & Business Media, 2001.

[48] Olgierd Cecil Zienkiewicz, Robert L Taylor, Perumal Nithiarasu, and JZ Zhu. *The finite element method*, volume 3. McGraw-hill London, 1977.

[49] Claudio Canuto, M Youssuff Hussaini, Alfio Quarteroni, and Thomas A Zang. *Spectral methods*. Springer, 2006.

[50] James D Bjorken and Sidney D Drell. *Relativistic quantum mechanics*. McGraw-Hill, 1965.

[51] Charles-Henri Bruneau and Mazen Saad. The 2d lid-driven cavity problem revisited. *Computers & Fluids*, 35(3):326–348, 2006.

[52] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *arXiv preprint arXiv:1711.02257*, 2017.

[53] Ukng Ghia, Kirti N Ghia, and CT Shin. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of computational physics*, 48(3):387–411, 1982.

[54] Lc Ce Woods. A note on the numerical solution of fourth order differential equations. *The Aeronautical Quarterly*, 5(4):176–184, 1954.

5 Appendix

5.1 A bound for the gradients of PINNs boundary and residual loss functions for a one-dimensional Poisson problem

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

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

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,

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

Note that

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

And

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

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,

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

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

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

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

□

5.2 Reference solution for a flow in a two-dimensional lid-driven cavity via a finite difference approximation

If we introduce the stream function ψ and vorticity ω, the Navier-Stokes equation can be written in the following form [53]:

$$\begin{cases}\dfrac{\partial\omega}{\partial t}+\dfrac{\partial\psi}{\partial y}\dfrac{\partial\omega}{\partial x}-\dfrac{\partial\psi}{\partial x}\dfrac{\partial\omega}{\partial y}=\nu\Delta\omega \\ \Delta\psi=-\omega\end{cases} \tag{82}$$

where

$$u=\frac{\partial\psi}{\partial y},\ v=-\frac{\partial\psi}{\partial x},\ \omega=\frac{\partial v}{\partial x}-\frac{\partial u}{\partial y} \tag{83}$$

The setup of the simulation is as follows. A set of points (x_i, y_j) is uniformly distributed in the domain $[0,1]\times[0,1]$, with $x_i = i/N$, $y_i = j/N$, $i,j = 0,1,\cdots,N$. The grid resolution h equals $1/N$. We denote $A_{i,j}$ as the value of physical variable A (velocity, pressure, etc.) at the point (x_i, y_j). All the spacial derivatives are treated with 2nd-order discretization scheme shown in Eq.84.

$$\begin{aligned}\frac{\partial A}{\partial x}|_{i,j} &\approx \frac{A_{i+1,j}-A_{i-1,j}}{2h} \\ \frac{\partial A}{\partial y}|_{i,j} &\approx \frac{A_{i,j+1}-A_{i,j-1}}{2h} \\ \Delta A|_{i,j} &\approx \frac{A_{i+1,j}+A_{i-1,j}+A_{i,j+1}+A_{i,j-1}-4A_{i,j}}{h^2}\end{aligned} \tag{84}$$

According to the boundary condition of velocity

$$(u,v)=\begin{cases}(1,0),\ y=1 \\ (0,0),\ otherwise\end{cases} \tag{85}$$

we can derive the boundary condition of stream function

$$\psi = \begin{cases} h/2, & y = 1 \\ 0, & \text{otherwise} \end{cases} \tag{86}$$

under the assumption that the x-component of velocity u grows linearly from 0 to 1 between $(0, 1 - 1/N)$ and $(0, 1)$, and between $(1, 1 - 1/N)$ and $(1, 1)$.

The boundary condition of vorticity is derived from the Wood's formula [54]

$$\omega_0 = -\frac{1}{2}\omega_1 - \frac{3}{h^2}(\psi_1 - \psi_0) - \frac{3}{h}v_\tau - \frac{3}{2}\frac{\partial v_n}{\partial \tau} + \frac{h}{2}\frac{\partial^2 v_\tau}{\partial \tau^2} \tag{87}$$

where (ψ_0, ω_0) is the local stream function and vorticity at a boundary point, (ψ_1, ω_1) is the stream function and vorticity at the adjacent point along the normal direction, (v_n, v_τ) is the normal and tangential component of velocity, and τ is the tangential direction.

The algorithm is composed of the following steps:

Step.1 Set the stream function ψ and vorticity ω at inner points to zero, and calculate the ψ and ω at the boundary using equation 86 and Eq.87. Set time $t = 0$;

Step.2 Calculate the vorticity ω at inner points at the time $t + \Delta t$ with equation 82(1), substituting $\partial\omega/\partial t$ with $(\omega(t + \Delta t) - \omega(t))/\Delta t$;

Step.3 Calculate the stream function ψ at inner points at the time $t + \Delta t$ with equation 82(2) and boundary condition equation 86;

Step.4 Calculate the vorticity ω on the boundary with equation 87;

Step.5 Update the velocity (u, v) at the time $t + \Delta$ with equation 83, and calculate the error between the velocity at the time t and $t + \Delta t$

$$\begin{aligned} error_u &= \frac{\max\limits_{i,j}\{u_{i,j}(t + \Delta t) - u_{i,j}(t)\}}{\Delta t} \\ error_v &= \frac{\max\limits_{i,j}\{v_{i,j}(t + \Delta t) - v_{i,j}(t)\}}{\Delta t} \end{aligned} \tag{88}$$

If $\max\{error_u, error_v\} < \varepsilon$, the flow has reached the steady state, and the computation terminates. Otherwise, set $t \leftarrow t + \Delta t$, and return to Step.2.

In our simulation, we set the number of grid $N = 128$, time step $\delta t = 1 \times 10^3$, and criterion for convergence $\varepsilon = 1 \times 10^{-4}$.