SIAM J. SCI. COMPUT.
Vol. 43, No. 5, pp. A3055–A3081

UNDERSTANDING AND MITIGATING GRADIENT FLOW PATHOLOGIES IN PHYSICS-INFORMED NEURAL NETWORKS*

SIFAN WANG†, YUJUN TENG‡, AND PARIS PERDIKARIS‡

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 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 50–100× across a range of problems in computational physics. All code and data accompanying this manuscript are publicly available at https://github.com/PredictiveIntelligenceLab/GradientPathologiesPINNs.

Key words. deep learning, differential equations, optimization, stiff dynamics, computational physics

AMS subject classifications. 68T99, 65M99, 68U20

DOI. 10.1137/20M1318043

1. Introduction.

Thanks to breakthrough results across a diverse range of scientific disciplines [25, 12, 3, 4, 46], 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 overparameterized, 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 [25, 13, 38], 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 associ-

*Submitted to the journal's Methods and Algorithms for Scientific Computing section February 12, 2020; accepted for publication (in revised form) May 25, 2021; published electronically September 9, 2021.

https://doi.org/10.1137/20M1318043

Funding: This work was supported by the U.S. Department of Energy under the Advanced Scientific Computing Research program through grant DE-SC0019116 and by the Defense Advanced Research Projects Agency under the Physics of Artificial Intelligence program through grant HR00111890034.

†Graduate Group in Applied Mathematics and Computational Science, University of Pennsylvania, Philadelphia, PA 19104 USA (sifanw@sas.upenn.edu).

‡Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA 19104 USA (yjteng@seas.upenn.edu, pgp@seas.upenn.edu).

ated with a given predictive task. Without a doubt, the most celebrated example in this category is convolutional neural networks [27, 19], 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 [30]. Another example includes covariant neural networks [24], which 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) is 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

$$\mathcal{L}(\theta) := \frac{1}{N_u}\sum_{i=1}^{N_u}[\boldsymbol{u}_i - f_\theta(\boldsymbol{x}_i)]^2 + \frac{1}{\lambda}\mathcal{R}[f_\theta(\boldsymbol{x})], \tag{1.1}$$

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 constrain 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* [40, 55, 21, 51], in which the outputs of a neural network are constrained to approximately satisfy a system of partial differential equations (PDEs) 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 problems in computational physics by reviving the original ideas in [37, 26] using modern software developments in reverse-mode differentiation [6] 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 [41, 39, 56, 17, 47, 52, 23, 49], the effects of the regularization mechanism in (1.1) remain poorly understood, and in several cases can even lead to unstable and erroneous predictions (see, e.g., remarks in [39, 49]).

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 (1.1). 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 conventional 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 50–100× 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 following the original formulation of Raissi, Perdikaris, and Karniadakis [40]. 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) [40] aim at inferring a continuous latent function $\boldsymbol{u}(\boldsymbol{x}, t)$ that arises as the solution to a system of nonlinear PDEs of the general form

$$\begin{aligned} &\boldsymbol{u}_t + \mathcal{N}_{\boldsymbol{x}}[\boldsymbol{u}] = 0, \quad \boldsymbol{x} \in \Omega, t \in [0, T], \\ &\boldsymbol{u}(\boldsymbol{x}, 0) = h(\boldsymbol{x}), \quad x \in \Omega, \\ &\boldsymbol{u}(\boldsymbol{x}, t) = g(\boldsymbol{x}, t), \quad t \in [0, T], \quad \boldsymbol{x} \in \partial\Omega, \end{aligned} \tag{2.1}$$

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. We remark that this formulation can be easily generalized to higher order PDEs since they can be written as systems of first-order PDEs. Following the original work of [40], 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 (2.1) as

$$\boldsymbol{r}_\theta(\boldsymbol{x}, t) := \frac{\partial}{\partial t} f_\theta(\boldsymbol{x}, t) + \mathcal{N}_{\boldsymbol{x}}[f_\theta(\boldsymbol{x}, t)], \tag{2.2}$$

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 [6]. 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

$$\mathcal{L}(\theta) := \mathcal{L}_r(\theta) + \sum_{i=1}^{M} \lambda_i \mathcal{L}_i(\theta), \tag{2.3}$$

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

$$\mathcal{L}_r = \frac{1}{N_r} \sum_{i=1}^{N_r} [\boldsymbol{r}(\boldsymbol{x}_r^i, t_r^i)]^2, \tag{2.4}$$

$$\mathcal{L}_{u_b} = \frac{1}{N_b} \sum_{i=1}^{N_b} [\boldsymbol{u}(\boldsymbol{x}_b^i, t_b^i) - g_b^i]^2, \tag{2.5}$$

$$\mathcal{L}_{u_0} = \frac{1}{N_0} \sum_{i=1}^{N_0} [\boldsymbol{u}(\boldsymbol{x}_0^i, 0) - h_0^i)]^2, \tag{2.6}$$

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 $\mathcal{L}_{u_0}$ enforce the boundary conditions and the initial conditions, respectively. As $\mathcal{L}(\theta)$ is typically minimized used stochastic gradient descent, a 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 is to construct a neural network representation $f_\theta(\boldsymbol{x}, t)$ for which $\mathcal{L}(\theta)$ is as close to zero as possible.

2.2. Gradient pathologies in physics-informed neural networks. Despite a series of promising results [41, 39, 47, 52, 23, 49, 55], the original formulation of Raissi, Perdikaris, and Karniadakis [40] 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,

$$\Delta u(x, y) + k^2 u(x, y) = q(x, y), \quad (x, y) \in \Omega := (-1, 1) \times (-1, 1), \tag{2.7}$$

$$u(x, y) = h(x, y), \quad (x, y) \in \partial\Omega, \tag{2.8}$$

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

$$\begin{aligned} q(x, y) = &-(a_1\pi)^2 \sin(a_1\pi x) \sin(a_2\pi y) - (a_2\pi)^2 \sin(a_1\pi x) \sin(a_2\pi y) \\ &+ k^2 \sin(a_1\pi x) \sin(a_2\pi y), \end{aligned} \tag{2.9}$$

$$h(x, y) = 0. \tag{2.10}$$

A PINN approximation to solving (2.7) 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.,

$$\mathcal{L}(\theta) = \mathcal{L}_r(\theta) + \mathcal{L}_{u_b}(\theta). \tag{2.11}$$

Without loss of generality, let us consider an example prediction scenario in which we choose parameters $k = 1$, $a_1 = 1$, and $a_2 = 4$ and $f_\theta(x, y)$ 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 (2.11) using the Adam optimizer [22] 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 pointwise 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 [18] 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. A snapshot of the histogram of the gradients at the end of training is illustrated in Figure 2.

One can observe that the gradients of the boundary loss term $\mathcal{L}_{u_b}(\theta)$ in each layer are sharply concentrated at the origin and overall attain significantly smaller values than the gradients of the PDE residual loss $\mathcal{L}_r(\theta)$. This implies that minimizing the PDE residual dominates the total training process, and, consequently, our trained model is heavily biased towards returning a solution that leads to a small error in the PDE residual, while a large error is attained in fitting the boundary conditions. As we know, a PDE system may have infinitely many solutions if no proper boundary or initial conditions are specified [14]. Therefore, we may conclude that the gradients of the boundary loss $\nabla_\theta \mathcal{L}_{u_b}(\theta)$ and the residual loss $\nabla_\theta \mathcal{L}_r(\theta)$ should match each other in magnitude, otherwise the network ultimately tends to learn any solutions that satisfy the equation and thus is prone to returning erroneous predictions.

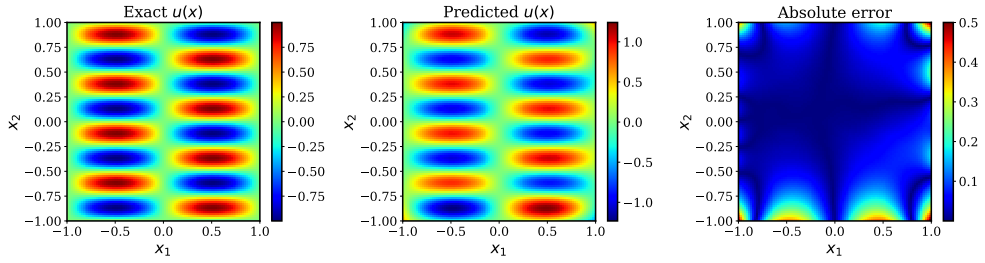


FIG. 1. Helmholtz equation: *Exact solution versus the prediction of a conventional physics-informed neural network model with* 4 *hidden layers and* 50 *neurons per layer after* 40,000 *iterations of training with gradient descent (relative* L^2 *error:* $1.81e-01$*).*

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 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 more likely 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

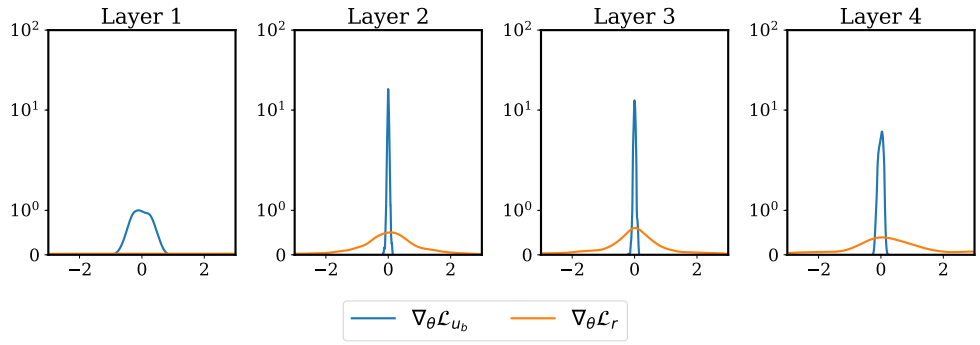


FIG. 2. Helmholtz equation: *Histograms of back-propagated gradients* $\nabla_\theta \mathcal{L}_r(\theta)$ *and* $\nabla_\theta \mathcal{L}_{u_b}(\theta)$ *at each layer at the* 40,000*th iteration of training a standard PINN model for solving the Helmholtz equation.*

Poisson equation

$$\begin{aligned} &\Delta u(x) = g(x), \quad x \in [0,1], \\ &u(x) = h(x), \quad x = 0 \text{ and } x = 1. \end{aligned} \tag{2.12}$$

To this end, let us consider exact solutions of the form $u(x) = \sin(Cx)$.

Then the corresponding source term $g(x)$ and the boundary condition $h(x)$ can be directly computed using the exact solution $u(x)$.

Under the PINN 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

$$\begin{aligned} \mathcal{L}(\theta) &= \mathcal{L}_r(\theta) + \mathcal{L}_{u_b}(\theta) \\ &= \frac{1}{N_b}\sum_{i=1}^{N_b}[f_\theta(x_b^i) - h(x_b^i)]^2 + \frac{1}{N_r}\sum_{i=1}^{N_r}\left[\frac{\partial^2}{\partial x^2} f_\theta(x_r^i) - g(x_r^i)\right]^2. \end{aligned} \tag{2.13}$$

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).

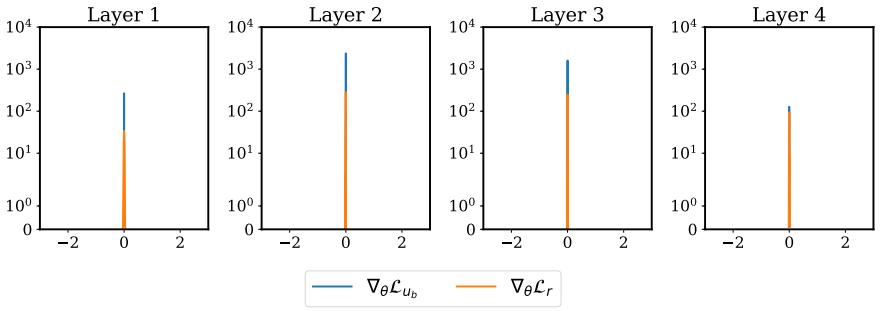
Then we may express our approximation as $f_\theta(x) = u(x)\epsilon_\theta(x)$, where $\epsilon_\theta(x)$ is a smooth function defined in $[0,1]$, and for which $|\epsilon_\theta(x) - 1| \le A$ for some $A > 0$, and $\|\frac{\partial^k \epsilon_\theta(x)}{\partial x^k}\|_{L^\infty} < A$ for all nonnegative 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 section SM1 of the supplementary material):

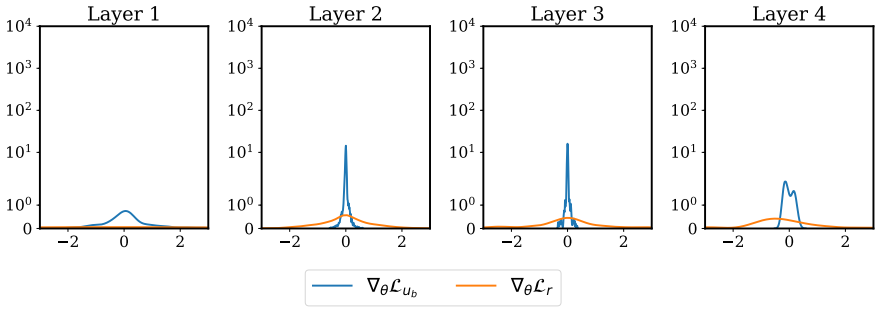
$$\|\nabla_\theta \mathcal{L}_{u_b}(\theta)\|_{L^\infty} \le 2A \cdot \|\nabla_\theta \epsilon_\theta(x)\|_{L^\infty}, \tag{2.14}$$

$$\|\nabla_\theta \mathcal{L}_r(\theta)\|_{L^\infty} \le O(C^4) \cdot A \cdot \|\nabla_\theta \epsilon_\theta(x)\|_{L^\infty}. \tag{2.15}$$

It is worth emphasizing that these two bounds only assume that the network is some approximation to the exact solution, and that it thus has no relation with the training process (sampling, optimizer, batch sizes, etc.).



(a) $C = 1$



(b) $C = 4$

FIG. 3. Poisson equation: *Histograms of back-propagated gradients* $\nabla_\theta \mathcal{L}_r(\theta)$ *and* $\nabla_\theta \mathcal{L}_{u_b}(\theta)$ *at each layer at 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* (2.12).

TABLE 1
Poisson equation: *Relative* L^2 *prediction error for different values of* C *obtained by training the same neural network (*5 *layers,* 50 *hidden units, tanh activations) after* 40,000 *iterations of Adam optimizer with default settings.*

	$C = 1$	$C = 2$	$C = 4$	$C = 8$	$C = 16$
Rel. L^2 error	$1.34e-04$	$4.65e-04$	$3.82e-03$	$1.30e-02$	$9.44e+00$

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 than 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 (2.12) 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 $\nabla_\theta \mathcal{L}_{u_b}(\theta)$, which ultimately leads to an inaccurate reconstruction of the PDE solution. This conclusion can be verified by Table 1, from which one can see that the resulting relative L^2 error increases as the value of C increases.

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

$$\frac{d\theta}{dt} = -\nabla_\theta \mathcal{L}_r(\theta) - \sum_{i=1}^{M} \nabla_\theta \mathcal{L}_i(\theta). \tag{2.16}$$

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 (2.29) correspond to a forward Euler discretization [20] of (2.16). 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 PINNs 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 [42, 35]. This immediately follows from performing a stability analysis for the linearized system

$$\frac{d}{dt}\tilde{\theta}(t) = -\nabla^2_\theta \mathcal{L}(\tilde{\theta}(t)) \cdot \tilde{\theta}(t). \tag{2.17}$$

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 the gradient descent update rule of (2.29). In fact, a classical result in numerical analysis on the conditional stability of the forward Euler method requires bounding the learning rate as $\eta < 2/\sigma_{\max}(\nabla^2_\theta \mathcal{L}(\theta))$ [20].

To illustrate the presence of stiffness in the gradient flow dynamics of PINNs, 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 (2.7). 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 PINN 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 4 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 PINN 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 PINNs with gradient descent. Suppose that at the nth step of gradient decent during the training, we have

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}(\theta_n) = \theta_n - \eta \left[\nabla_\theta \mathcal{L}_r(\theta_n) + \nabla_\theta \mathcal{L}_{u_b}(\theta_n)\right], \tag{2.18}$$

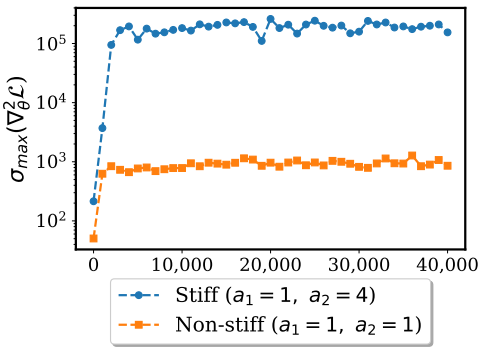


FIG. 4. Stiffness in the gradient flow dynamics: *Largest eigenvalue of the Hessian* $\nabla^2_\theta \mathcal{L}(\theta)$ *during the training of a PINN model for approximating the solution to the two-dimensional Helmholtz problem (see* (2.7)*) and for different values of the control parameters* a_1 *and* a_2.

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

(2.19)
$$\mathcal{L}(\theta_{n+1}) = \mathcal{L}(\theta_n) + (\theta_{n+1} - \theta_n) \cdot \nabla_\theta \mathcal{L}(\theta_n) + \frac{1}{2}(\theta_{n+1} - \theta_n)^T \nabla^2_\theta \mathcal{L}(\xi)(\theta_{n+1} - \theta_n),$$

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 (2.18) to (2.19), we obtain (see the proof in section SM2 of the supplementary materials)

$$\mathcal{L}(\theta_{n+1}) - \mathcal{L}(\theta_n) = \eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 \left(-1 + \frac{1}{2}\eta \sum_{i=1}^N \lambda_i y_i^2\right), \tag{2.20}$$

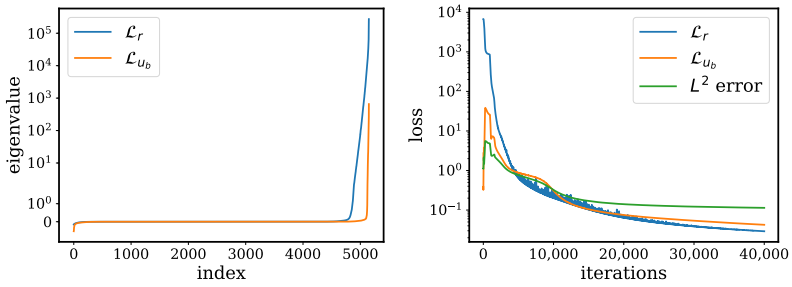
$$\mathcal{L}_r(\theta_{n+1}) - \mathcal{L}_r(\theta_n) = \eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 \left(-1 + \frac{1}{2}\eta \sum_{i=1}^N \lambda_i^r y_i^2\right), \tag{2.21}$$

$$\mathcal{L}_{u_b}(\theta_{n+1}) - \mathcal{L}_{u_b}(\theta_n) = \eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 \left(-1 + \frac{1}{2}\eta \sum_{i=1}^N \lambda_i^{u_b} y_i^2\right) \tag{2.22}$$

for some $\boldsymbol{y} = (y_1, \ldots, y_N)$ satisfying $\|\boldsymbol{y}\| = 1$, where $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_N$, $\lambda_1^r \le \lambda_2^r \le \cdots \le \lambda_N^r$, and $\lambda_1^{u_b} \le \lambda_2^{u_b} \le \cdots \le \lambda_N^{u_b}$ are eigenvalues of $\nabla^2_\theta \mathcal{L}(\xi)$, $\nabla^2_\theta \mathcal{L}_r(\xi)$, and $\nabla^2_\theta \mathcal{L}_{u_b}(\xi)$, respectively.

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, 5].

To verify our hypothesis, we still consider the example in section 2. Specifically, we choose a 5-layer neural network with 50 units in each hidden layer. To reduce stochastic effects introduced by minibatch gradient descent we use full batch gradient decent with an initial learning rate of 10^{-3} and an exponential decay with a decay rate of 0.9 and a decay step of 1,000 iterations. All collocation points are randomly



(a) (b)

FIG. 5. Helmholtz equation: (a) *All eigenvalues of* $\nabla^2_\theta \mathcal{L}_r(\theta)$ *and* $\nabla^2_\theta \mathcal{L}_{u_b}(\theta)$ *at the last step in the training, arranged in increasing order.* (b) *Loss curves of* $\mathcal{L}_r(\theta)$ *and* $\mathcal{L}_{u_b}$ *and relative* L^2 *error, respectively, after* $40,000$ *iterations of Adam optimizer* [2] *in full batch mode.*

sampled from both the boundary and the interior domain at each iteration during training. Figure 5(a) shows the eigenvalues of $\nabla^2_\theta \mathcal{L}_{u_b}(\theta)$ and $\nabla^2_\theta \mathcal{L}_r(\theta)$ at the last step of the training 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 5(b) 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. In addition, 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 the gradient flow system may cause severe difficulties in the training of PINNs with gradient descent. It also hints at the importance of choosing the learning rate such that a stable discretization of the gradient flow is achieved.

Moreover, applying the classical Sobolev inequality [2] to $\nabla_\theta \mathcal{L}_r$ and $\nabla_\theta \mathcal{L}_{u_b}$ gives

$$\|\nabla_\theta \mathcal{L}_r(\theta)\|_{L^\infty} \leq C \|\nabla^2_\theta \mathcal{L}_r(\theta)\|_{L^\infty}, \tag{2.23}$$

$$\|\nabla_\theta \mathcal{L}_{u_b}(\theta)\|_{L^\infty} \leq C \|\nabla^2_\theta \mathcal{L}_{u_b}(\theta)\|_{L^\infty}, \tag{2.24}$$

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 5(a), 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, (2.23) and (2.24) 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 [22]) may be a poor, or even unstable, choice for training PINNs. From a numerical analysis standpoint, it is then natural to investigate whether more stable discretizations, such as implicit or IMEX operator splitting schemes [20, 48], 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 [31], as well as effective algorithms for finding Nash equilibria in multiplayer games [5, 43]. Nevertheless, these intriguing directions go beyond the scope of the present work and

define areas for future investigation. As discussed in the next section, here our goal is to retain the simplicity of gradient descent while providing a nonintrusive and adaptive heuristic for enhancing its performance. Similar approaches can be found in the numerical analysis literature for integrating stiff and multiscale dynamical systems; see, for example, the multirate schemes of Gear and Wells [16] and the flow-averaging integration techniques put forth by Tao, Owhadi, and Marsden [50].

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

$$\mathcal{L}(\theta) := \mathcal{L}_r(\theta) + \sum_{i=1}^{M} \mathcal{L}_i(\theta), \tag{2.25}$$

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

$$\begin{aligned} \theta_{n+1} &= \theta_n - \eta \nabla_\theta \mathcal{L}(\theta_n) \\ &= \theta_n - \eta \left[\nabla_\theta \mathcal{L}_r(\theta_n) + \sum_{i=1}^{M} \nabla_\theta \mathcal{L}_i(\theta_n) \right], \end{aligned} \tag{2.26}$$

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 [7]:

$$\mathcal{L}(\theta) := \mathcal{L}_r(\theta) + \sum_{i=1}^{M} \lambda_i \mathcal{L}_i(\theta). \tag{2.27}$$

Consequently, the corresponding gradient descent updates now take the form

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}(\theta_n) \tag{2.28}$$

$$= \theta_n - \eta \nabla_\theta \mathcal{L}_r(\theta_n) - \eta \sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta_n), \tag{2.29}$$

where we see how the constants λ_i can effectively introduce a rescaling of the learning rate corresponding to each loss term. Obviously, the next question that needs to be answered is how those weights λ_i should 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 [22]—one of 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 2.1, 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 (2.27) is appropriately balanced.

Algorithm 2.1 Learning rate annealing for physics-informed neural networks.

Consider a PINN $f_\theta(\boldsymbol{x})$ 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 follows:

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

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

$$\hat{\lambda}_i = \frac{\max_{\theta_n}\{|\nabla_\theta \mathcal{L}_r(\theta_n)|\}}{\overline{|\nabla_\theta \lambda_i \mathcal{L}_i(\theta_n)|}}, \quad i = 1, \ldots, M, \tag{2.30}$$

where θ_n denotes the values of the parameters θ at the nth iteration, $|\cdot|$ denotes the elementwise absolute value, and $\overline{|\nabla_\theta \mathcal{L}_i(\theta_n)|}$ denotes the mean of $|\nabla_\theta \mathcal{L}_i(\cdot)|$ evaluated at θ_n.

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

$$\lambda_i = (1 - \alpha)\lambda_i + \alpha \hat{\lambda}_i, \quad i = 1, \ldots, M. \tag{2.31}$$

(c) Update the parameters θ via gradient descent

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}_r(\theta_n) - \eta \sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta_n). \tag{2.32}$$

end for

The recommended hyperparameter values are $\eta = 10^{-3}$ and $\alpha = 0.1$.

The intuition behind the proposed algorithm is fairly straightforward. Recall that our goal is to assign appropriate weight to each term in the loss function such that theirs gradients during back-propagation are similar in magnitude. According to our analysis in previous sections, we may conclude that the gradients of residual loss generally dominate the gradients of the others. Based on such a finding, our goal can be converted to the following: for any given loss term $\mathcal{L}_i$, find λ_i such that

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

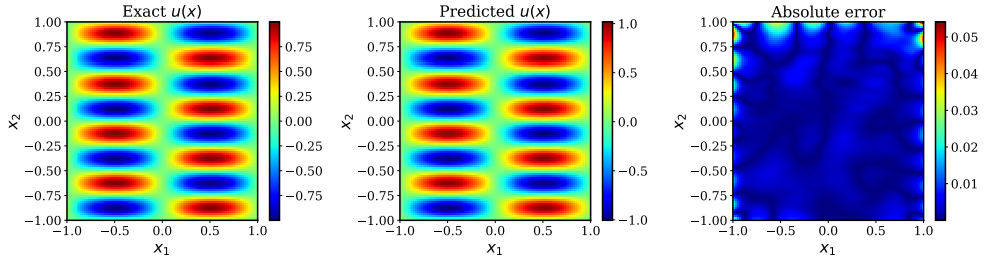


FIG. 6. Helmholtz equation: *Exact solution versus the prediction of a PINN model with* 4 *hidden layers and* 50 *neurons per layer after* 40,000 *iterations of training using the proposed learning rate annealing algorithm (relative* L^2 *error:* $1.27e-02$*). Here we remark that the relative* L^2 *error of the original PINNs (Figure* 1*) is* $1.81e-01$.

which follows

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

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 (2.30) and (2.31). Here we remark that the updates in (2.30) and (2.31) can take place either 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 is performed to update the neural network parameters θ using the current weight values stored in λ_i; see (2.32). 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., multivariate problems with multiple boundary conditions on different variables), while the extra computational overhead associated with computing the gradient statistics in (2.30) is small, especially in the case of infrequent updates. Moreover, our computational studies confirm very low sensitivity on the additional hyperparameter α, as the accuracy of our results does not exhibit any significant variance when this parameter takes values within a reasonable range (e.g., $\alpha \in [0.05, 0.2]$).

A first illustration of the effectiveness of the proposed learning rate annealing algorithm is provided through the lens of the Helmholtz benchmark presented in (2.7). Figures 6 and 7 summarize the predictions of the same 4-layer deep PINN model used in section 2.2 after training it using Algorithm 2.1 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 pointwise error at the domain boundaries has been significantly reduced. Finally, Figure 8(a) summarizes the convergent evolution of the constant λ_{u_b} used to scale the boundary condition loss $\mathcal{L}_{u_b}(\theta)$ in (2.11) during model training using Algorithm 2.1.

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 neu-

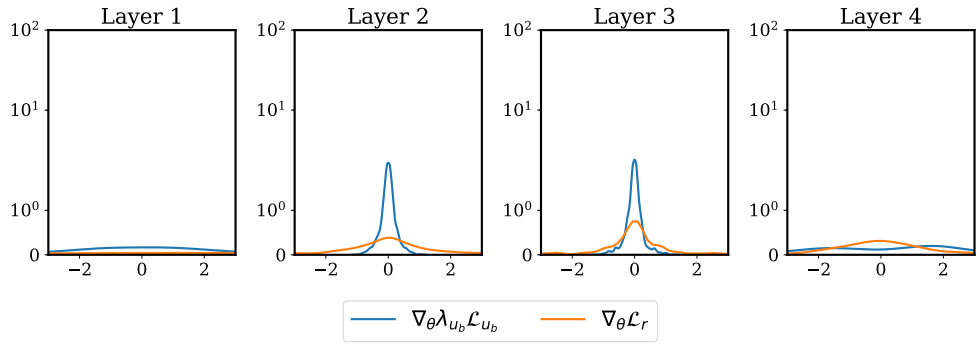
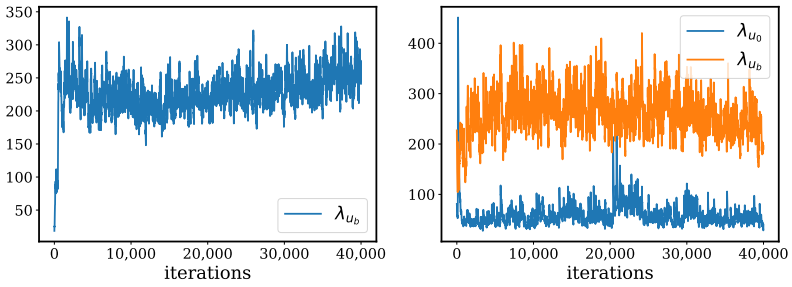


FIG. 7. Helmholtz equation: *Histograms of back-propagated gradients of* $\nabla_\theta \mathcal{L}_r$ *and* $\nabla_\theta \mathcal{L}_{u_b}$ *at each layer at the end of training a PINN with the proposed Algorithm* 2.1 *to solve the Helmholtz equation.*



(a) (b)

FIG. 8. (a) Helmholtz equation: *Evolution of the constant* λ_{u_b} *used to scale the boundary condition loss* $\mathcal{L}_{u_b}(\theta)$ *in* (2.11) *during model training using Algorithm* 2.1. (b) 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* (3.6)*), during the training of model M*2 *using Algorithm* 2.1.

ral networks [25, 19] 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 [28, 19] are well suited to modeling sequence data due to their ability to respect temporal invariance and capture long-term dependencies. Although there are some related works using convolutional neural networks [56, 15] to solve PDE-constrained optimization problems in scientific computing, 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 nontrivial 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 [40, 52, 47, 51, 49]. 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 [53], here we present a novel neural network architecture with the following characteristics: (i) explicitly accounts for multiplicative interactions between different input dimensions, and (ii) enhances the hidden states with residual connections. 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 pointwise multiplication operation to update the hidden layers according to the following forward propagation rule:

$$U = \phi(XW^1 + b^1), \quad V = \phi(XW^2 + b^2), \tag{2.33}$$

$$H^{(1)} = \phi(XW^{z,1} + b^{z,1}), \tag{2.34}$$

$$Z^{(k)} = \phi(H^{(k)}W^{z,k} + b^{z,k}), \quad k = 1, \dots, L, \tag{2.35}$$

$$H^{(k+1)} = (1 - Z^{(k)}) \odot U + Z^{(k)} \odot V, \quad k = 1, \dots, L, \tag{2.36}$$

$$f_\theta(x) = H^{(L+1)}W + b, \tag{2.37}$$

where X denotes the $(n \times d)$ design matrix of the input data points, and $\odot$ denotes elementwise multiplication. The parameters of this model are essentially the same as in a standard fully connected architecture, with the addition of the weights and biases used by the two transformer networks, i.e.,

$$\theta = \{W^1, b^1, W^2, b^2, (W^{z,l}, b^{z,l})_{l=1}^L, W, b\}. \tag{2.38}$$

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 note 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 9 summarizes the prediction of the proposed fully connected architecture with a depth of 4 layers for a PINN model trained using Algorithm 2.1 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, Perdikaris, and Karniadakis [40]. In particular, by comparing Figures 1 and 9, notice how the absolute pointwise error at the domain boundaries has been effectively diminished.

Perhaps a more interesting fact is that the choice of the neural architecture also has 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 10 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 3× 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 PINNs, 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

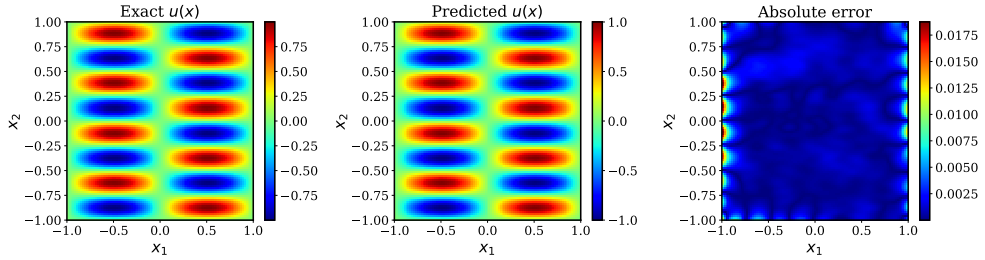


FIG. 9. Helmholtz equation: *Exact solution versus the prediction of a PINN model with the proposed improved architecture (4 hidden layers, 50 neurons each) after 40,000 iterations of training using the proposed learning rate annealing algorithm (relative L^2 error: $3.69e-03$).*

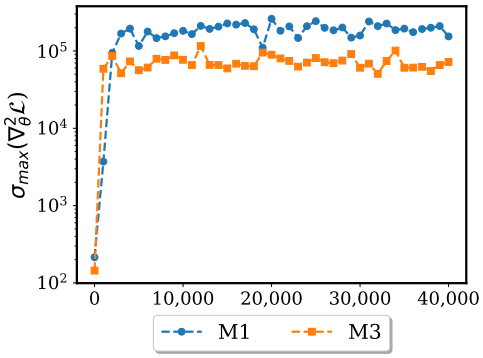


FIG. 10. An improved fully connected architecture for PINNs: *Largest eigenvalue of the Hessian $\nabla^2_\theta \mathcal{L}(\theta)$ during the training of two separate PINN models for approximating the solution to the two-dimensional Helmholtz problem (see (2.7)) with $a_1 = 1$, $a_2 = 4$. Model M1 is a conventional dense, 4-layer deep architecture, while model M3 corresponds to the proposed improved architecture also with a depth of 4 layers.*

accuracy of PINNs by a factor of 50–100× 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 inferring the solution of PDEs. Throughout all benchmarks we will employ and compare the performance of four different approaches as summarized in Table 2. In all cases we employ hyperbolic tangent activation functions, a minibatch size of 128 data points, and train the networks using single precision arithmetic via stochastic gradient descent using the Adam optimizer with default settings [22]. Moreover, all networks are initialized using the Glorot scheme [18], 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 to update their loss weights λ_i every 10 iterations of stochastic gradient descent (see (2.30) and (2.31)). All algorithms were implemented in TensorFlow [1], and the reported run-times were 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 are publicly available at https://github.com/PredictiveIntelligenceLab/GradientPathologiesPINNs.

TABLE 2
Models considered in the numerical studies presented in section 3.

Model	Description
M1	Original formulation of PINNs as put forth in Raissi, Perdikaris, and Karniadakis [40]
M2	PINNs with the proposed learning rate annealing algorithm (see section 2.5)
M3	PINNs with the proposed improved fully connected architecture (see section 2.6)
M4	PINNs combining the proposed learning rate annealing algorithm and fully connected architecture

3.1. Helmholtz equation. First, let us revisit the Helmholtz equation benchmark described in section 2.2 (see (2.7)). This equation is closely related to many problems in the natural and engineering sciences, such as wave propagation in acoustic, elastic, and electromagnetic media [34]. It also routinely arises in conventional numerical discretization methods for PDEs such as finite element and spectral methods [57, 10]. Our goal here is to use this canonical benchmark problem to systematically analyze the performance of the different models listed in Table 2 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 3 summarizes our results, averaged over 10 independent trials with randomized weight initializations using the Glorot scheme [18]. Evidently, the original formulation of Raissi, Perdikaris, and Karniadakis [40] (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, Perdikaris, and Karniadakis [40] (model M1). This indicates that the proposed architecture seems to have a better ability to represent complicated functions than conventional fully connected neural networks, and it may also lead to more stable gradient descent dynamics as discussed in section 2.6. In addition, it is worth noting that increasing the width of the network can enhance the accuracy of the predictions, while increasing the depth of the network does not usually lead to better results. 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 0.13–0.59%). Particularly, we are interested in the performance of the proposed algorithm and the architecture for different k in (2.7). To this end, we fix the architecture of the network to 5 fully connected layers with 50 neurons per hidden layer and train the network using $M1$–$M4$ as summarized in Table 2 for 40,000 iterations of Adam optimizer. Table 4 shows the relative L^2 errors for different $k = 2, 3, 4$. It can be observed that our proposed methods (M2–M4) remain robust for different values of k.

TABLE 3

Helmholtz equation: *Mean of the relative L^2 errors and their standard deviation between the predicted and the exact solution $u(x, y)$ for the different methods summarized in Table* 2*, and for different neural architectures obtained by varying the number of hidden layers and different number of neurons per layer over* 10 *independent trials.*

Architecture	M1	M2
30 units / 3 hidden layers	$4.36e-01 \pm 1.57e-01$	$4.31e-02 \pm 1.68e-02$
50 units / 3 hidden layers	$1.94e-01 \pm 3.91e-02$	$1.44e-02 \pm 6.37e-03$
100 units / 3 hidden layers	$8.51e-02 \pm 2.55e-02$	$4.41e-03 \pm 1.54e-03$
30 units / 5 hidden layers	$4.19e-01 \pm 2.82e-01$	$1.47e-02 \pm 3.31e-03$
50 units / 5 hidden layers	$2.02e-01 \pm 5.69e-02$	$6.33e-03 \pm 1.03e-03$
100 units / 5 hidden layers	$1.31e-01 \pm 6.17e-02$	$\mathbf{4.00e-03 \pm 1.22e-03}$
30 units / 7 hidden layers	$3.41e-01 \pm 1.17e-01$	$1.59e-02 \pm 3.54e-03$
50 units / 7 hidden layers	$1.97e-01 \pm 4.81e-02$	$6.44e-03 \pm 1.12e-03$
100 units / 7 hidden layers	$\mathbf{6.98e-02 \pm 3.55e-02}$	$4.40e-03 \pm 9.39e-04$
Architecture	M3	M4
30 units / 3 hidden layers	$4.03e-02 \pm 1.06e-02$	$5.85e-03 \pm 1.10e-03$
50 units / 3 hidden layers	$2.12e-02 \pm 4.32e-03$	$3.30e-03 \pm 4.29e-04$
100 units / 3 hidden layers	$1.34e-02 \pm 1.13e-03$	$1.67e-03 \pm 3.50e-04$
30 units / 5 hidden layers	$3.22e-02 \pm 9.74e-03$	$4.05e-03 \pm 1.11e-03$
50 units / 5 hidden layers	$1.70e-02 \pm 4.20e-03$	$1.95e-03 \pm 3.00e-04$
100 units / 5 hidden layers	$\mathbf{1.01e-02 \pm 1.31e-03}$	$\mathbf{1.29e-03 \pm 1.79e-04}$
30 units / 7 hidden layers	$2.57e-02 \pm 3.73e-03$	$3.62e-03 \pm 4.31e-04$
50 units / 7 hidden layers	$1.69e-02 \pm 3.37e-03$	$2.03e-03 \pm 3.14e-04$
100 units / 7 hidden layers	$1.12e-02 \pm 1.25e-03$	$2.10e-03 \pm 3.23e-04$

TABLE 4

Helmholtz equation: *Relative L^2 error of the prediction solutions for different k, obtained by training* 5 *fully connected layers with* 50 *neurons per layer for* 40,000 *iterations of the Adam optimizer using M1–M4 summarized in Table* 2.

Method / k	M1	M2	M3	M4
$k = 2$	5.20e-01	3.03e-02	1.77e-02	5.68e-03
$k = 3$	7.63e-01	5.12e-02	2.28e-02	3.57e-03
$k = 4$	1.78e+00	3.61e-01	5.15e-02	3.19e-03

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 nonlinear equation which plays a significant role in many scientific applications such as solid-state physics, nonlinear optics, and quantum physics [8]. The initial-boundary value problem in one spatial dimension takes the form

$$u_{tt} + \alpha u_{xx} + \beta u + \gamma u^k = f(x,t), \qquad (x,t) \in \Omega \times [0,T], \tag{3.1}$$

$$u(x,0) = g_1(x), \qquad x \in \Omega, \tag{3.2}$$

$$u_t(x,0) = g_2(x), \qquad x \in \Omega, \tag{3.3}$$

$$u(x,t) = h(x,t), \qquad (x,t) \in \partial\Omega \times [0,T], \tag{3.4}$$

where α, β, γ, and k are known constants, $k = 2$ when we have a quadratic nonlinearity, and $k = 3$ when we have a cubic nonlinearity. 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,

$$u(x,t) = x\cos(5\pi t) + (xt)^3, \tag{3.5}$$

and correspondingly derive a consistent forcing term $f(x,t)$ using (3.1), 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 (3.5).

A PINN model $f_\theta(x,t)$ can now be trained to approximate the latent solution $u(x,t)$ by formulating the composite loss

$$\mathcal{L}(\theta) = \mathcal{L}_r(\theta) + \lambda_{u_b}\mathcal{L}_{u_b}(\theta) + \lambda_{u_0}\mathcal{L}_{u_0}(\theta), \tag{3.6}$$

where $\mathcal{L}_r(\theta), \mathcal{L}_{u_b}(\theta), \mathcal{L}_{u_0}(\theta)$ are defined in (2.4), (2.5), and (2.6). Notice that if $\lambda_{u_b} = \lambda_{u_0} = 1$, then the loss function above gives rise to the original PINN formulation of Raissi, Perdikaris, and Karniadakis [40] (model M1).

Given this 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, Perdikaris, and Karniadakis [40] (model M1). The results summarized in Figure 11 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 $\nabla_\theta\mathcal{L}_r$ stays 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 PINN 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 PINN models [40] for solving PDE systems that have sufficiently complex dynamics leading to solutions with nontrivial behavior (e.g., directional anisotropy, multiscale 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 12 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 5. Notice that here our goal is not to perform an exhaustive hyperparameter search to find the best performing model, but rather 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,

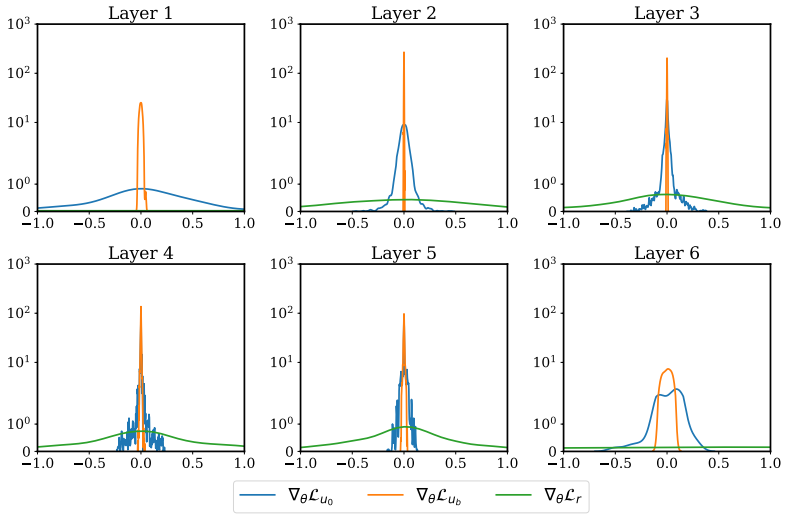


FIG. 11. Klein–Gordon equation: *Histograms of back-propagated gradients for model M*1 *after* 40,000 *iterations of stochastic gradient descent using the Adam optimizer.*

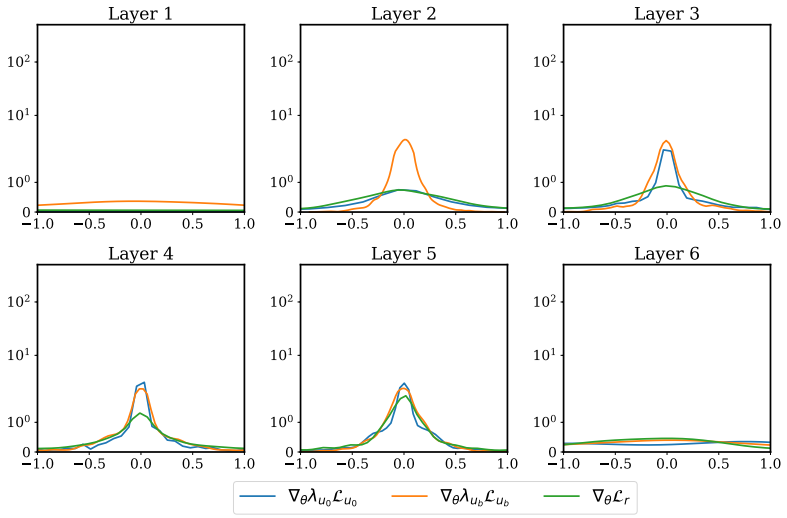


FIG. 12. Klein–Gordon equation: *Back-propagated gradients for model M*2 *after* 40,000 *iterations of stochastic gradient descent using the Adam optimizer.*

Perdikaris, and Karniadakis [40], 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.

Figure 13 provides a more detailed visual assessment of the predictive accuracy for models M1 and M4. As we expected, in the top panel of Figure 13, model M1 fails to obtain accurate predictive solutions and has increased absolute error up to about 0.2 as time t increases. In the bottom panel of Figure 13, 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

TABLE 5

Klein–Gordon equation: *Relative L^2 error between the predicted and the exact solution $u(t,x)$ for the different methods summarized in Table 2. 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.*

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

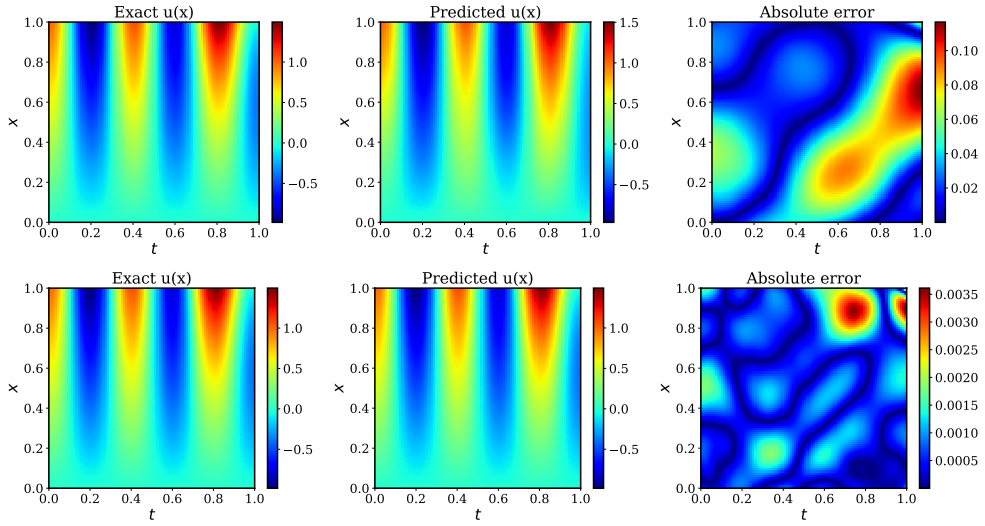


FIG. 13. Klein–Gordon equation: Top: *Predicted solution obtained using M1 and M4 (from top to bottom) with 5 hidden layers and 50 neurons in each layer. After 40,000 iterations of stochastic gradient descent using the Adam optimizer, the relative L^2 error of the prediction solution is $1.79e-01$ (M1) and $2.22e-03$ (M4).*

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 12 depicts the 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 (3.6)), during the training of model M2 using Algorithm 2.1.

3.3. Flow in a lid-driven cavity. In our last example, we 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 nondimensional form as

$$\boldsymbol{u}\cdot\nabla\boldsymbol{u}+\nabla p-\frac{1}{Re}\Delta\boldsymbol{u}=0 \qquad \text{in } \Omega, \tag{3.7}$$

$$\nabla\cdot\boldsymbol{u}=0 \qquad \text{in } \Omega, \tag{3.8}$$

$$\boldsymbol{u}(\boldsymbol{x})=(1,0) \qquad \text{on } \Gamma_1, \tag{3.9}$$

$$\boldsymbol{u}(\boldsymbol{x})=(0,0) \qquad \text{on } \Gamma_0, \tag{3.10}$$

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 have 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 [9]. Our goal here is to train a PINN 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 reference solution using conventional finite difference methods, thereby creating a high-resolution validation data set $\{\boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{u}^i\}_{i=1}^N$ (see details in section SM4 of the supplementary material).

Following the stream-function formulation of the two-dimensional incompressible Naiver–Stokes equations, one can use a neural network to 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.,

$$[x, y] \xmapsto{f_\theta} [\psi(x, y), p(x, y)].$$

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

$$r_\theta^u(x, y) := u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} - \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \tag{3.11}$$

$$r_\theta^v(x, y) := u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} - \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right), \tag{3.12}$$

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 [6]. Although we are now able to exactly satisfy the incompressibility constraint, notice that this approach incurs a larger computational cost since third-order derivatives of the neural network need to be computed. Despite this additional cost, our experience indicates that this modification is crucial for obtaining accurate and unique solutions for (3.7) by minimizing a physics-informed loss of the form

$$\mathcal{L}(\theta) = \mathcal{L}_{r_u}(\theta) + \mathcal{L}_{r_v}(\theta) + \mathcal{L}_{u_b}(\theta) + \mathcal{L}_{v_b}(\theta), \tag{3.13}$$

with

$$\mathcal{L}_{r_u}(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r}[r_\theta^u(x_r^i, y_r^i)]^2, \tag{3.14}$$

$$\mathcal{L}_{r_v}(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r}[r_\theta^v(x_r^i, y_r^i)]^2, \tag{3.15}$$

$$\mathcal{L}_{u_b}(\theta) = \frac{1}{N_b}\sum_{i=1}^{N_b}[u(x_b^i, y_b^i) - u_b^i]^2, \tag{3.16}$$

$$\mathcal{L}_{v_b}(\theta) = \frac{1}{N_b}\sum_{i=1}^{N_b}[v(x_b^i, y_b^i) - v_b^i]^2, \tag{3.17}$$

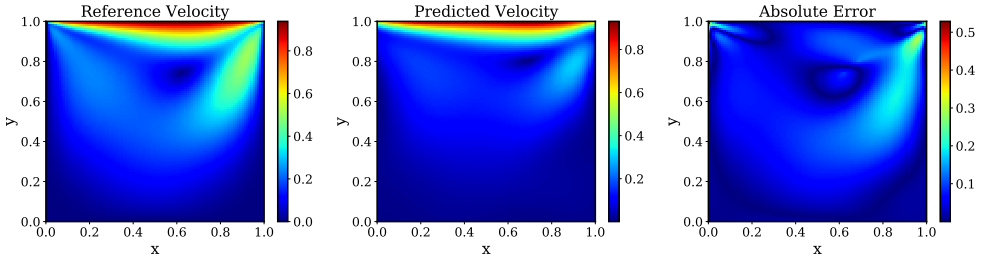
where $\{(x_r^i, y_r^i)\}_{i=1}^{N_r}$ is a set of collocation points in which we aim to minimize the residual of (3.11) and (3.12), while $\{(x_b^i, y_b^i), u_b^i\}_{i=1}^{N_b}$ and $\{(x_b^i, y_b^i), v_b^i\}_{i=1}^{N_b}$ denote the boundary data for the two velocity components at the domain boundaries Γ_0 and Γ_1, respectively.

Recall that we 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 [6]. 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 (3.13). The results of this experiment are summarized in Figure 14. 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.

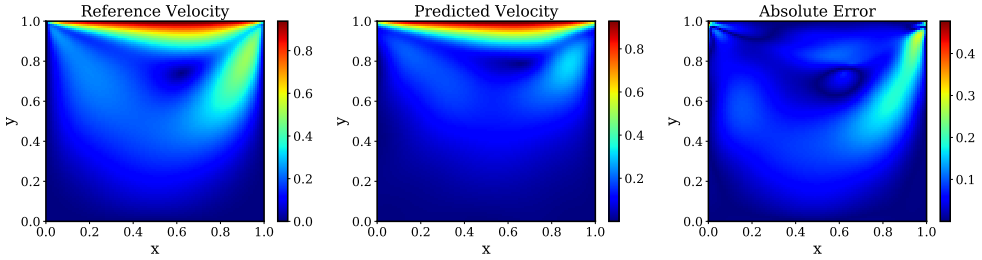
4. Summary and discussion. Despite recent success across a range of applications [40, 51, 23, 52, 49, 41], physics-informed neural networks (PINNs) often struggle to accurately approximate the solution of PDEs. In this work we identify and analyze a fundamental mode of failure of PINNs related to stiff 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 adaptively 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 issues with unbalanced gradients, e.g., multitask learning [11]. Finally, we propose a novel neural network architecture that can enhance the accuracy and robustness of PINNs by decreasing the stiffness of gradient flows, indicating that specialized architectures can play a prominent role in the future success of PINN models. Taken together, our developments provide new insights into the training of PINNs and consistently improve their predictive accuracy by a factor of 50–100$\times$ across a range of problems in computational physics.

Although there are some works related to the theoretical foundation of PINNs [44, 54, 29, 45, 33], one of the main limitations of the present work is a lack of sufficient convergence guarantees. Specifically, the proposed algorithms are based on empirical observations without any rigorous mathematical proof. Although we have verified that such an approach can be applied to many cases and yield practical improvements, the robustness of the proposed algorithm is worth further investigation.

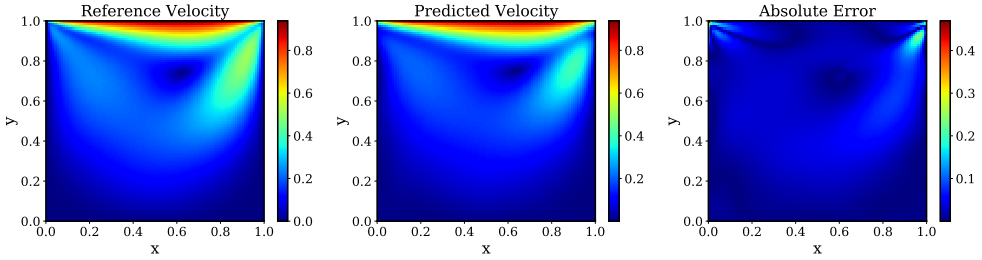
More generally, here we have to admit that we are still at the very early stages of rigorously understanding the capabilities and limitations of PINNs. To help advance this knowledge, we need to answer a series of open questions: What is the relation between stiffness in a given PDE and stiffness in the gradient flow dynamics of the associated PINN model? Can stiffness in the gradient flow dynamics be reduced



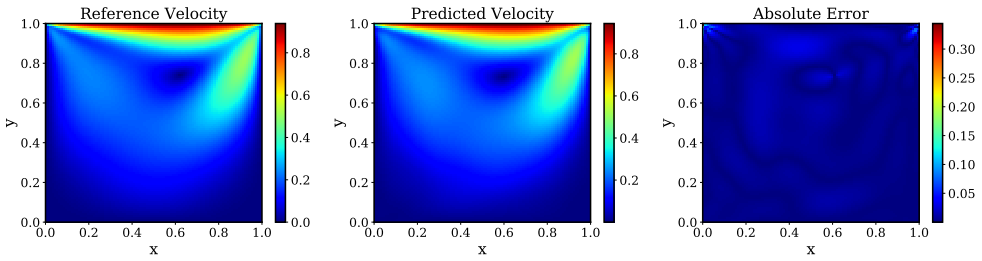
(a) Model M1 (relative L^2 prediction error: 2.71e-01).



(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).

FIG. 14. Flow in a lid-driven cavity, stream-function pressure representation: *Reference solution using a conventional finite difference solver, prediction of a* 5*-layer deep PINN (models* $M1$*–*$M4$*, top to bottom), and absolute pointwise error.*

(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.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al., *TensorFlow: A system for large-scale machine learning*, in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.

[2] R. A. Adams and J. J. Fournier, *Sobolev Spaces*, Pure Appl. Math. (Amst.) 140, Elsevier, 2003.

[3] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, *Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning*, Nature Biotechnol., 33 (2015), p. 831.

[4] P. Baldi, P. Sadowski, and D. Whiteson, *Searching for exotic particles in high-energy physics with deep learning*, Nature Commun., 5 (2014), 4308.

[5] D. Balduzzi, S. Racaniere, J. Martens, J. Foerster, K. Tuyls, and T. Graepel, *The Mechanics of n-Player Differentiable Games*, preprint, https://arxiv.org/abs/1802.05642, 2018.

[6] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, *Automatic differentiation in machine learning: A survey*, J. Mach. Learn. Res., 18 (2018), 153.

[7] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*, Academic Press, 2014.

[8] J. D. Bjorken and S. D. Drell, *Relativistic Quantum Mechanics*, McGraw-Hill, 1965.

[9] C.-H. Bruneau and M. Saad, *The 2D lid-driven cavity problem revisited*, Comput. & Fluids, 35 (2006), pp. 326–348.

[10] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang, *Spectral Methods*, Springer, 2006.

[11] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, *GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks*, preprint, https://arxiv.org/abs/1711.02257, 2017.

[12] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation*, preprint, https://arxiv.org/abs/1406.1078, 2014.

[13] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, *Long-term recurrent convolutional networks for visual recognition and description*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 2625–2634.

[14] L. C. Evans, *Partial Differential Equations*, AMS, Providence, RI, 1998.

[15] H. Gao, L. Sun, and J.-X. Wang, *PhyGeoNet: Physics-Informed Geometry-Adaptive Convolutional Neural Networks for Solving Parametric PDEs on Irregular Domain*, preprint, https://arxiv.org/abs/2004.13145, 2020.

[16] C. W. Gear and D. Wells, *Multirate linear multistep methods*, BIT, 24 (1984), pp. 484–502.

[17] N. Geneva and N. Zabaras, *Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks*, J. Comput. Phys., 403 (2020), 109056.

[18] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[20] A. Iserles, *A First Course in the Numerical Analysis of Differential Equations*, Cambridge University Press, 2009.

[21] E. Kharazmi, Z. Zhang, and G. Karniadakis, *Variational Physics-Informed Neural Networks*

for Solving Partial Differential Equations, preprint, https://arxiv.org/abs/1912.00873, 2019.

[22] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, preprint, https://arxiv.org/abs/1412.6980, 2014.

[23] G. Kissas, Y. Yang, E. Hwuang, W. R. Witschey, J. A. Detre, and P. Perdikaris, *Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks*, Comput. Methods Appl. Mech. Engrg., 358 (2020), 112623.

[24] R. Kondor, H. T. Son, H. Pan, B. Anderson, and S. Trivedi, *Covariant Compositional Networks for Learning Graphs*, preprint, https://arxiv.org/abs/1801.02144, 2018.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet classification with deep convolutional neural networks*, in Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.

[26] I. E. Lagaris, A. Likas, and D. I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Trans. Neural Networks, 9 (1998), pp. 987–1000.

[27] Y. LeCun and Y. Bengio, *Convolutional networks for images, speech, and time series*, in The Handbook of Brain Theory and Neural Networks, MIT Press, 1995, pp. 255–258.

[28] Z. C. Lipton, J. Berkowitz, and C. Elkan, *A Critical Review of Recurrent Neural Networks for Sequence Learning*, preprint, https://arxiv.org/abs/1506.00019, 2015.

[29] T. Luo and H. Yang, *Two-Layer Neural Networks for Partial Differential Equations: Optimization and Generalization Theory*, preprint, https://arxiv.org/abs/2006.15733, 2020.

[30] S. Mallat, *Understanding deep convolutional networks*, Philos. Trans. Roy. Soc. A Math. Phys. Engrg. Sci., 374 (2016), 20150203.

[31] M. Mardani, Q. Sun, D. Donoho, V. Papyan, H. Monajemi, S. Vasanawala, and J. Pauly, *Neural proximal gradient descent for compressive imaging*, in Advances in Neural Information Processing Systems, 2018, pp. 9573–9583.

[32] P. Mertikopoulos, C. Papadimitriou, and G. Piliouras, *Cycles in adversarial regularized learning*, in Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2018, pp. 2703–2717, https://doi.org/10.1137/1.9781611975031.172.

[33] S. Mishra and R. Molinaro, *Estimates on the Generalization Error of Physics Informed Neural Networks (PINNs) for Approximating PDEs* II*: A Class of Inverse Problems*, preprint, https://arxiv.org/abs/2007.01138, 2020.

[34] J.-C. Nédélec, *Acoustic and Electromagnetic Equations: Integral Representations for Harmonic Problems*, Springer Science & Business Media, 2001.

[35] G. K. Nilsen, A. Z. Munthe-Kaas, H. J. Skaug, and M. Brun, *Efficient Computation of Hessian Matrices in TensorFlow*, preprint, https://arxiv.org/abs/1905.05559, 2019.

[36] N. Parikh and S. Boyd, *Proximal algorithms*, Found. Trends Optim., 1 (2014), pp. 127–239.

[37] D. C. Psichogios and L. H. Ungar, *A hybrid neural network-first principles approach to process modeling*, AIChE J., 38 (1992), pp. 1499–1511.

[38] A. Radford, L. Metz, and S. Chintala, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, preprint, https://arxiv.org/abs/1511.06434, 2015.

[39] M. Raissi, *Deep hidden physics models: Deep learning of nonlinear partial differential equations*, J. Mach. Learn. Res., 19 (2018), pp. 932–955.

[40] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, J. Comput. Phys., 378 (2019), pp. 686–707.

[41] M. Raissi, A. Yazdani, and G. E. Karniadakis, *Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations*, Science, 367 (2020), pp. 1026–1030.

[42] L. Sagun, L. Bottou, and Y. LeCun, *Eigenvalues of the Hessian in Deep Learning: Singularity and Beyond*, preprint, https://arxiv.org/abs/1611.07476, 2016.

[43] F. Schäfer and A. Anandkumar, *Competitive Gradient Descent*, preprint, https://arxiv.org/abs/1905.12103, 2019.

[44] Y. Shin, J. Darbon, and G. E. Karniadakis, *On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs*, Commun. Comput. Phys., 28 (2020), pp. 2042–2074.

[45] Y. Shin, Z. Zhang, and G. E. Karniadakis, *Error Estimates of Residual Minimization Using Neural Networks for Linear PDEs*, preprint, https://arxiv.org/abs/2010.08019, 2020.

[46] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton et al., *Mastering the game of go without human knowledge*, Nature, 550 (2017), pp. 354–359.

[47] J. Sirignano and K. Spiliopoulos, *DGM: A deep learning algorithm for solving partial dif-*

ferential equations, J. Comput. Phys., 375 (2018), pp. 1339–1364.

[48] G. Strang, *On the construction and comparison of difference schemes*, SIAM J. Numer. Anal., 5 (1968), pp. 506–517, https://doi.org/10.1137/0705041.

[49] L. Sun, H. Gao, S. Pan, and J.-X. Wang, *Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data*, Comput. Methods Appl. Mech. Engrg., 361 (2020), 112732.

[50] M. Tao, H. Owhadi, and J. E. Marsden, *Nonintrusive and structure preserving multiscale integration of stiff ODEs, SDEs, and Hamiltonian systems with hidden slow dynamics via flow averaging*, Multiscale Model. Simul., 8 (2010), pp. 1269–1324, https://doi.org/10.1137/090771648.

[51] A. M. Tartakovsky, C. O. Marrero, P. Perdikaris, G. D. Tartakovsky, and D. Barajas-Solano, *Physics-informed deep neural networks for learning parameters and constitutive relationships in subsurface flow problems*, Water Resources Res., 56 (2020), e2019WR026731.

[52] R. K. Tripathy and I. Bilionis, *Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification*, J. Comput. Phys., 375 (2018), pp. 565–588.

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, in Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.

[54] S. Wang, X. Yu, and P. Perdikaris, *When and Why PINNs Fail to Train: A Neural Tangent Kernel Perspective*, preprint, https://arxiv.org/abs/2007.14527, 2020.

[55] Y. Yang and P. Perdikaris, *Adversarial uncertainty quantification in physics-informed neural networks*, J. Comput. Phys., 394 (2019), pp. 136–152.

[56] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris, *Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data*, J. Comput. Phys., 394 (2019), pp. 56–81.

[57] O. C. Zienkiewicz, R. L. Taylor, P. Nithiarasu, and J. Zhu, *The Finite Element Method*, Vol. 3, McGraw-Hill, London, 1977.

SUPPLEMENTARY MATERIALS: UNDERSTANDING AND MITIGATING GRADIENT FLOW PATHOLOGIES IN PHYSICS-INFORMED NEURAL NETWORKS*

SIFAN WANG†, YUJUN TENG‡, AND PARIS PERDIKARIS‡

SM1. A bound for the gradients of PINNs boundary and residual loss functions for a one-dimensional Poisson problem. Recall that the loss function is given by

$$\begin{aligned}\mathcal{L}(\theta) &= \mathcal{L}_r(\theta) + \mathcal{L}_{u_b}(\theta) \\ &= \frac{1}{N_b}\sum_{i=1}^{N_b}[f_\theta(x_b^i) - h(x_b^i)]^2 + \frac{1}{N_r}\sum_{i=1}^{N_r}[\frac{\partial^2}{\partial x^2}f_\theta(x_r^i) - g(x_r^i)]^2.\end{aligned} \tag{SM1.1}$$

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

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

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

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

*Supplementary material for SISC MS#M131804.
https://doi.org/10.1137/20M1318043

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

‡Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA, 19104 (yjteng@seas.upenn.edu, pgp@seas.upenn.edu).

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

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

And

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

So we have

(SM1.2)
$$\left|\frac{\partial^2 u_\theta(x)}{\partial x \partial \theta}\left(\frac{\partial^2 u_\theta(x)}{\partial x^2} - \frac{\partial^2 u(x)}{\partial x^2}\right)\Big|_0^1\right| \le 2\left(C\left\|\frac{\partial \epsilon_\theta(x)}{\partial \theta}\right\|_{L^\infty} + \left\|\frac{\partial \epsilon'_\theta(x)}{\partial \theta}\right\|_{L^\infty}\right)\left(C^2 A + 2A\right)$$

(SM1.3)
$$= O\left(C^3\right)\cdot A \cdot \left\|\frac{\partial \epsilon_\theta(x)}{\partial \theta}\right\|_{L^\infty}$$

Similarly,

(SM1.4)
$$\left|\frac{\partial u_\theta(x)}{\partial \theta}\left(\frac{\partial^3 u_\theta(x)}{\partial x^3} - \frac{\partial^3 u(x)}{\partial x^3}\right)\Big|_0^1\right| = \left|\frac{\partial u_\theta(x)}{\partial \theta}\left(\frac{\partial^3 u(x)\epsilon_\theta(x)}{\partial x^3} - \frac{\partial^3 u(x)}{\partial x^3}\right)\Big|_0^1\right|$$

(SM1.5)
$$\le O\left(C^3\right)\cdot A \cdot \left\|\frac{\partial \epsilon_\theta(x)}{\partial \theta}\right\|_{L^\infty}$$

Finally,

$$\left|\int_0^1 \frac{\partial u_\theta(x)}{\partial \theta}\left(\frac{\partial^4 u_\theta(x)}{\partial x^4} - \frac{\partial^4 u(x)}{\partial x^4}\right) dx\right| \leq O\left(C^4\right) \cdot A \cdot \left\|\frac{\partial \epsilon_\theta(x)}{\partial \theta}\right\|_{L^\infty} \tag{SM1.6}$$

Therefore, plugging all these together we obtain

$$\left|\frac{\partial \mathcal{L}_r}{\partial \theta}\right| \leq O\left(C^4\right) \cdot A \cdot \left\|\frac{\partial \epsilon_\theta(x)}{\partial \theta}\right\|_{L^\infty} \tag{SM1.7}$$

SM2. A proof of analyzing the difficulty in training PINNs with stiff gradient flow dynamics. By direct computation,

(SM2.1)

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

$$= -\eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 + \frac{1}{2}\eta^2 \nabla_\theta \mathcal{L}(\theta_n)^T \nabla_\theta^2 \mathcal{L}(\xi) \nabla_\theta \mathcal{L}(\theta_n) \tag{SM2.2}$$

$$= -\eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 + \frac{1}{2}\eta^2 \nabla_\theta \mathcal{L}(\theta_n)^T \left(\nabla_\theta^2 \mathcal{L}_r(\xi) + \nabla_\theta^2 \mathcal{L}_{u_b}(\xi)\right) \nabla_\theta \mathcal{L}(\theta_n) \tag{SM2.3}$$

Here, note that

$$\nabla_\theta \mathcal{L}(\theta_n)^T \nabla_\theta^2 \mathcal{L}(\xi) \nabla_\theta \mathcal{L}(\theta_n) = \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 \frac{\nabla_\theta \mathcal{L}(\theta_n)^T}{\|\nabla_\theta \mathcal{L}(\theta_n)\|} \nabla_\theta^2 \mathcal{L}(\xi) \frac{\nabla_\theta \mathcal{L}(\theta_n)}{\|\nabla_\theta \mathcal{L}(\theta_n)\|} \tag{SM2.4}$$

$$= \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 x^T Q^T \mathrm{diag}(\lambda_1, \lambda_2 \cdots \lambda_n) Q x \tag{SM2.5}$$

$$= \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 y^T \mathrm{diag}(\lambda_1, \lambda_2 \ldots \lambda_M) y \tag{SM2.6}$$

$$= \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 \sum_{i=1}^M \lambda_i y_i^2 \tag{SM2.7}$$

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

$$\nabla_\theta \mathcal{L}(\theta_n)^T \nabla_\theta^2 \mathcal{L}_r(\xi) \nabla_\theta \mathcal{L}(\theta_n) = \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 \sum_{i=1}^M \lambda_i^r y_i^2 \tag{SM2.8}$$

$$\nabla_\theta \mathcal{L}(\theta_n)^T \nabla_\theta^2 \mathcal{L}_{u_b}(\xi) \nabla_\theta \mathcal{L}(\theta_n) = \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 \sum_{i=1}^M \lambda_i^{u_b} y_i^2 \tag{SM2.9}$$

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 $\nabla_\theta^2 \mathcal{L}_r$ and $\nabla_\theta^2 \mathcal{L}_{u_b}$ respectively. Thus, combining these together we get

$$\mathcal{L}(\theta_{n+1}) - \mathcal{L}(\theta_n) = \eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 (-1 + \frac{1}{2}\eta \sum_{i=1}^N \lambda_i y_i^2) \tag{SM2.10}$$

$$\mathcal{L}_r(\theta_{n+1}) - \mathcal{L}_r(\theta_n) = \eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 (-1 + \frac{1}{2}\eta \sum_{i=1}^N \lambda_i^r y_i^2) \tag{SM2.11}$$

$$\mathcal{L}_{u_b}(\theta_{n+1}) - \mathcal{L}_{u_b}(\theta_n) = \eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 (-1 + \frac{1}{2}\eta \sum_{i=1}^N \lambda_i^{u_b} y_i^2) \tag{SM2.12}$$

SM3. Definitions of loss functions in Lid-Driven Cavity flow.

SM3.1. Velocity-pressure representation .

$$\mathcal{L}(\theta) = \mathcal{L}_{r_u}(\theta) + \mathcal{L}_{r_v}(\theta) + \mathcal{L}_{r_c}(\theta) + \mathcal{L}_{u_b}(\theta) + \mathcal{L}_{v_b}(\theta), \tag{SM3.1}$$

$$\mathcal{L}_{r_u}(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r}[r_\theta^u(x_r^i, y_r^i)]^2 \tag{SM3.2}$$

$$\mathcal{L}_{r_v}(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r}[r_\theta^v(x_r^i, y_r^i)]^2 \tag{SM3.3}$$

$$\mathcal{L}_{r_c}(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r}[r_\theta^c(x_r^i, y_r^i)]^2 \tag{SM3.4}$$

$$\mathcal{L}_{u_b}(\theta) = \frac{1}{N_b}\sum_{i=1}^{N_b}[u(x_b^i, y_b^i) - u_b^i]^2, \tag{SM3.5}$$

$$\mathcal{L}_{v_b}(\theta) = \frac{1}{N_b}\sum_{i=1}^{N_b}[v(x_b^i, y_b^i) - v_b^i]^2 \tag{SM3.6}$$

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.

SM4. 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 [SM1]:

$$\begin{cases} \dfrac{\partial \omega}{\partial t} + \dfrac{\partial \psi}{\partial y}\dfrac{\partial \omega}{\partial x} - \dfrac{\partial \psi}{\partial x}\dfrac{\partial \omega}{\partial y} = \nu \Delta \omega, \\ \Delta \psi = -\omega, \end{cases} \tag{SM4.1}$$

where

$$u = \frac{\partial \psi}{\partial y},\ v = -\frac{\partial \psi}{\partial x},\ \omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}. \tag{SM4.2}$$

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 SM4.3.

$$\begin{aligned} \frac{\partial A}{\partial x}|_{i,j} &\approx \frac{A_{i+1,j} - A_{i-1,j}}{2h}, \\ \frac{\partial A}{\partial y}|_{i,j} &\approx \frac{A_{i,j+1} - A_{i,j-1}}{2h}, \\ \Delta A|_{i,j} &\approx \frac{A_{i+1,j} + A_{i-1,j} + A_{i,j+1} + A_{i,j-1} - 4A_{i,j}}{h^2}. \end{aligned} \tag{SM4.3}$$

According to the boundary condition of velocity

$$(u, v) = \begin{cases} (1, 0), & y = 1, \\ (0, 0), & \text{otherwise}, \end{cases} \tag{SM4.4}$$

we can derive the boundary condition of stream function

$$\psi = \begin{cases} h/2, & y = 1, \\ 0, & \text{otherwise}, \end{cases} \tag{SM4.5}$$

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 [SM2]

$$\omega_0 = -\frac{1}{2}\omega_1 - \frac{3}{h^2}(\psi_1 - \psi_0) - \frac{3}{h}v_\tau - \frac{3}{2}\frac{\partial v_n}{\partial \tau} + \frac{h}{2}\frac{\partial^2 v_\tau}{\partial \tau^2}, \tag{SM4.6}$$

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 SM4.5 and SM4.6. Set time $t = 0$.

Step.2 Calculate the vorticity ω at inner points at the time $t + \Delta t$ with equation SM4.1(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 SM4.1(2) and boundary condition equation SM4.5.

Step.4 Calculate the vorticity ω on the boundary with equation SM4.6.

Step.5 Update the velocity (u, v) at the time $t+\Delta$ with equation SM4.2, and calculate the error between the velocity at the time t and $t + \Delta t$

$$\begin{aligned} \text{error}_u &= \frac{\max_{i,j}\{u_{i,j}(t + \Delta t) - u_{i,j}(t)\}}{\Delta t}, \\ \text{error}_v &= \frac{\max_{i,j}\{v_{i,j}(t + \Delta t) - v_{i,j}(t)\}}{\Delta t}. \end{aligned} \tag{SM4.7}$$

If $\max\{\text{error}_u, \text{error}_v\} < \varepsilon$, 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 points to $N = 128$, the time step to $\Delta t = 1 \times 10^3$, and the criterion for convergence to $\varepsilon = 1 \times 10^{-4}$.

REFERENCES

[SM1] U. Ghia, K. N. Ghia, and C. T. Shin, *High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method*, J. Comput. Phys., 48 (1982), pp. 387–411.

[SM2] L. C. Woods, *A note on the numerical solution of fourth order differential equations*, Aeronaut. Quart., 5 (1954), pp. 176–184.