Inverse Dirichlet weighting enables reliable training of physics informed neural networks

MACHINE LEARNING Science and Technology

CrossMark

OPEN ACCESS

RECEIVED
3 August 2021

REVISED
18 October 2021

ACCEPTED FOR PUBLICATION
5 November 2021

PUBLISHED
15 February 2022

PAPER

Inverse Dirichlet weighting enables reliable training of physics informed neural networks

Suryanarayana Maddu[1,2,3,7], Dominik Sturm[9,10], Christian L Müller[4,5,6] and Ivo F Sbalzarini[1,2,3,7,8,*]

1 Technische Universität Dresden, Faculty of Computer Science, 01069 Dresden, Germany
2 Max Planck Institute of Molecular Cell Biology and Genetics, 01307 Dresden, Germany
3 Center for Systems Biology Dresden, 01307 Dresden, Germany
4 Center for Computational Mathematics, Flatiron Institute, New York, NY, United States of America
5 Department of Statistics, LMU München, Munich, Germany
6 Institute of Computational Biology, Helmholtz Zentrum München, D-85764 Neuherberg, Germany
7 Center for Scalable Data Analytics and Artificial Intelligence ScaDS.AI, Dresden/Leipzig, Germany
8 Cluster of Excellence Physics of Life, TU Dresden, Germany
9 Helmholtz-Zentrum Dresden-Rossendorf, D-01328 Dresden, Germany
10 Center for Advanced Systems Understanding (CASUS), D-02826 Görlitz, Germany
* Author to whom any correspondence should be addressed.

E-mail: ivos@mpi-cbg.de

Keywords: physics-informed neural networks, multi-scale modeling, active turbulence, catastrophic forgetting, multi-objective training, gradient flow regularization

Abstract

We characterize and remedy a failure mode that may arise from multi-scale dynamics with scale imbalances during training of deep neural networks, such as physics informed neural networks (PINNs). PINNs are popular machine-learning templates that allow for seamless integration of physical equation models with data. Their training amounts to solving an optimization problem over a weighted sum of data-fidelity and equation-fidelity objectives. Conflicts between objectives can arise from scale imbalances, heteroscedasticity in the data, stiffness of the physical equation, or from catastrophic interference during sequential training. We explain the training pathology arising from this and propose a simple yet effective inverse Dirichlet weighting strategy to alleviate the issue. We compare with Sobolev training of neural networks, providing the baseline of analytically ε-optimal training. We demonstrate the effectiveness of inverse Dirichlet weighting in various applications, including a multi-scale model of active turbulence, where we show orders of magnitude improvement in accuracy and convergence over conventional PINN training. For inverse modeling using sequential training, we find that inverse Dirichlet weighting protects a PINN against catastrophic forgetting.

1. Introduction

Data-driven modeling has emerged as a powerful and complementary approach to first-principles modeling. It was made possible by advances in imaging and measurement technology, computing power, and innovations in machine learning, in particular neural networks. The success of neural networks in data-driven modeling can be attributed to their powerful function approximation properties for diverse functional forms, from image recognition [1] and natural language processing [2] to learning policies in complex environments [3].

Recently, there has been a surge in exploiting the versatile approximation properties of neural networks for surrogate modeling in scientific computing and for data-driven modeling of physical systems [4, 5]. These applications hinge on the approximation properties of neural networks for Sobolev-regular functions [6, 7]. The popular physics informed neural networks (PINNs) rely on knowing a differential equation model of the system in order to solve a soft-constrained optimization problem [5, 8]. Thanks to their mesh-free character, PINNs can be used for both forward and inverse modeling in domains including material science [9–11], fluid dynamics [8, 12–14] and turbulence [15, 16], biology [17], medicine [18, 19], earth science [20], mechanics [21] and uncertainty quantification [22], as well as for solving stochastic [23], high-dimensional [24] and fractional differential equations [25].

The wide application scope of PINNs rests on their parameterization as deep neural networks, which can be tuned by minimizing a weighted sum of data-fitting and equation-fitting objectives. However, finding network parameters that impartially optimize for several objectives is challenging when objectives impose conflicting requirements. In addition, PINNs suffer from *spectral bias* [26–29], potentially resulting in large discrepancies between convergence rates of different objectives during training [30]. For multi-task learning (MTL) problems, this issue has been addressed by various strategies for relative weighting of conflicting objectives, e.g. based on uncertainty [31], gradient normalization [32] or Pareto optimality [33]. The PINN community in parallel has developed heuristics for loss weighting with demonstrated gains in accuracy and convergence [34, 35]. However, there is neither consensus on when to use such strategies in PINNs, nor are there optimal benchmark solutions for objectives based on differential equations.

Here, we fill this gap by characterizing training pathologies of PINNs and providing criteria for their occurrence. We mathematically explain these pathologies by showing a connection between PINNs and Sobolev training, and we propose a strategy for loss weighting in PINNs based on the Dirichlet energy of the task-specific gradients. We show that this strategy reduces optimization bias and protects against catastrophic forgetting. We evaluate the proposed inverse Dirichlet weighting by comparing with Sobolev training in a case where provably optimal weights can be derived, and by empirical comparison with two conceptually different state-of-the-art PINN weighting approaches.

2. Training a physics informed neural network is a multi-objective optimization problem

An optimization problem involving multiple tasks or objectives can be written as:

$$\text{minimize } \mathcal{L}_k(\boldsymbol{\theta}^{\text{sh}}, \boldsymbol{\theta}^k), \qquad k = 1, \ldots, K. \tag{1}$$

The total number of tasks is given by K. $\boldsymbol{\theta}^{\text{sh}}$ are shared parameters between the tasks, and $\boldsymbol{\theta}^k$ are the task-specific parameters. For conflicting objectives, there is no single optimal solution, but Pareto-dominated solutions can be found using heuristics like evolutionary algorithms [36]. Alternatively, the multi-objective optimization problem can be converted to a single-objective optimization problem using scalarization, e.g. as a weighted sum with real-valued relative weights λ_k associated with each loss objective $\mathcal{L}_k$:

$$\min_{\boldsymbol{\theta}^{\text{sh}}, \boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^K} \sum_{k=1}^{K} \lambda_k \mathcal{L}_k(\boldsymbol{\theta}^{\text{sh}}, \boldsymbol{\theta}^k). \tag{2}$$

Often, the λ_k are dynamically tuned along training epochs [33]. The weights λ_k intuitively quantify the extent to which we aim to minimize the objective $\mathcal{L}_k$. Large λ_k favor small $\mathcal{L}_k$, and vice-versa [37]. For $K > 2$, manual tuning of the λ_k and grid search become prohibitively expensive for large parameterizations as common in deep neural networks.

The loss function of a PINN is a scalarized multi-objective loss, where each objective corresponds to either a data-fidelity term or a constraint derived from the physics of the underlying problem [4, 5]. A typical PINN loss contains five objectives:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{u}_{\boldsymbol{\theta}}, \boldsymbol{u}) = {} & \frac{\lambda_1}{N_{\text{int}}} \sum_{i=1}^{N_{\text{int}}} \left| \partial_t \boldsymbol{u}_{\boldsymbol{\theta}} - \mathcal{F}(\boldsymbol{u}_{\boldsymbol{\theta}}(t_i, \boldsymbol{x}_i^{\Omega}), \Sigma) \right|^2 \\
& + \frac{\lambda_2}{N_{\mathcal{B}}} \sum_{i=1}^{N_{\mathcal{B}}} \left| \mathcal{B}(t_i, \boldsymbol{x}_i^{\partial\Omega}) \right|^2 + \frac{\lambda_3}{N_{\text{I}}} \sum_{i=1}^{N_{\text{I}}} \left| \mathcal{I}(t=0, \boldsymbol{x}_i^{\Omega}) \right|^2 \\
& + \frac{\lambda_4}{N_{\text{int}}} \sum_{i=1}^{N_{\text{int}}} \left| \mathcal{H}(\boldsymbol{u}_{\boldsymbol{\theta}}(t_i, \boldsymbol{x}_i^{\Omega})) \right|^2 + \frac{\lambda_5}{N_{\text{int}}} \sum_{i=1}^{N_{\text{int}}} \left| \boldsymbol{u}_{\boldsymbol{\theta}}(t_i, \boldsymbol{x}_i^{\Omega}) - \boldsymbol{u}_i \right|^2.
\end{aligned} \tag{3}$$

We distinguish the neural network approximation $\boldsymbol{u}_{\boldsymbol{\theta}}(t_i, \boldsymbol{x}_i^{\Omega})$ from the training data $\boldsymbol{u}_i = \boldsymbol{u}(t_i, \boldsymbol{x}_i^{\Omega})$ sampled at the N_{int} space-time points $(t_i, \boldsymbol{x}_i^{\Omega})$ in the interior of the solution domain Ω, $N_{\mathcal{B}}$ points $(t_i, \boldsymbol{x}_i^{\partial\Omega})$ on the boundary $\partial\Omega$, and N_{I} samples of the initial state at time $t = 0$. The λ_1, λ_2, and λ_3 weight the residuals of the physical equation $\partial_t \boldsymbol{u} = \mathcal{F}(\boldsymbol{u}, \Sigma)$ with right-hand side $\mathcal{F}$ and coefficients Σ, the boundary condition $\mathcal{B}$, and the initial condition $\mathcal{I}$, respectively. The objective with weight λ_4 imposes an additional constraint $\mathcal{H}$ on the state variable $\boldsymbol{u}$, such as, e.g. divergence-freeness ($\nabla \cdot \boldsymbol{u} = 0$) of the velocity field of an incompressible flow. The weight λ_5 penalizes deviations of the neural network approximation from the available measurement data.

PINNs can be used to solve both forward and inverse modeling problems of ordinary differential equations (ODEs) and partial differential equations (PDEs). In the forward problem, a numerical approximation to the solution of an ODE or PDE is to be learned. For this, $\boldsymbol{\theta}^{\mathrm{sh}}$ are the networks parameters and $\boldsymbol{\theta}^{k}$ is an empty set. In the inverse problem, the coefficients of an ODE or PDE are to be learned from data such that the equation describes the dynamics in the data. Then, the task-specific parameters $\boldsymbol{\theta}^{k}$ correspond to the coefficients ($\boldsymbol{\xi} \subseteq \Sigma$) of the physical equation that shall be inferred. During training of a PINN, the parameters $(\boldsymbol{\theta}^{\mathrm{sh}}, \boldsymbol{\theta}^{k})$ are determined by minimizing the total loss $\mathcal{L}$ for given data.

2.1. Sobolev training is a special case of PINN training

Like most deep neural networks, PINNs are usually trained by first-order parameter update based on (approximate) loss gradients. The update rule can be interpreted as a discretization of the gradient flow, i.e.

$$\boldsymbol{\theta}(\tau+1) = \boldsymbol{\theta}(\tau) - \eta(\tau) \sum_{k=1}^{K} \lambda_k \nabla_{\boldsymbol{\theta}} \mathcal{L}_k. \tag{4}$$

The learning rate $\eta(\tau)$ depends on the training epoch τ, reflecting the adaptive step size used in optimizers like Adam [38]. The gradients are batch gradients $\nabla_{\boldsymbol{\theta}} \mathcal{L}_k = \frac{1}{|\mathrm{B}|} \sum_{i \in \mathrm{B}} \nabla \ell_k(\boldsymbol{u}^i_{\boldsymbol{\theta}}, \boldsymbol{u}_i)$, where B is a mini-batch, created by randomly splitting the N training data points into $N/|\mathrm{B}|$ distinct partitions, and ℓ_k is one summand (for one data point i) of the loss $\mathcal{L}_k$ corresponding to the kth objective. For batch size $|\mathrm{B}| = 1$, mini-batch gradient descent reduces to stochastic gradient descent algorithm.

For approximating a function $\boldsymbol{u}(\boldsymbol{x})$ from data $\boldsymbol{u}_i$, adding derivatives $\mathcal{D}^m_{\boldsymbol{x}}$, $m \geqslant 1$, of the state variable into the training loss has been shown to improve data efficiency and generalization power of neural networks [7]. Minimizing the resulting loss

$$\sum_{i=1}^{N} \left[\lambda_0 \, |\boldsymbol{u}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \boldsymbol{u}(\boldsymbol{x}_i)|^2 + \sum_{k=1}^{K} \lambda_k |\mathcal{D}^k_{\boldsymbol{x}} \boldsymbol{u}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \mathcal{D}^k_{\boldsymbol{x}} \boldsymbol{u}(\boldsymbol{x}_i)|^2 \right] \tag{5}$$

using the update rule in equation (4) can be interpreted as a finite approximation to the Sobolev gradient flow of the loss functional. Without loss of generality, we drop the dependence of the state variable on time, writing $\boldsymbol{u}_i = \boldsymbol{u}(\boldsymbol{x}_i)$. The neural network is then trained with access to the $(K+2)$-tuples $\{(\boldsymbol{x}_i, \boldsymbol{u}(\boldsymbol{x}_i), \mathcal{D}^1_{\boldsymbol{x}} \, \boldsymbol{u}(\boldsymbol{x}_i), \ldots, \mathcal{D}^K_{\boldsymbol{x}} \boldsymbol{u}(\boldsymbol{x}_i))\}_{i=1}^N$ instead of the usual training set $\{(\boldsymbol{x}_i, \boldsymbol{u}(\boldsymbol{x}_i))\}_{i=1}^N$. Such Sobolev training is an instance of scalarized multi-objective optimization with each objective based on the approximation of a specific derivative. The weights are usually chosen as $\lambda_0 = \ldots = \lambda_K = 1$ [7].

The additional information from the derivatives introduces an inductive bias, which has been found to improve data efficiency and network generalization [7]. This is akin to PINNs, which use derivatives from the physical equation model to achieve inductive bias. Indeed, the Sobolev loss in equation (5) is a special case of the PINN loss from equation (3) with each objective based on a PDE with right-hand side $\mathcal{F} = \mathcal{D}^k_{\boldsymbol{x}} \boldsymbol{u}$, $k = 1, \ldots, K$.

2.2. Vanishing task-specific gradients are a failure mode of PINNs

The update rule in equation (4) is known to lead to stiff learning dynamics for loss functionals whose Hessian matrix has large positive eigenvalues [34]. This is the case for Sobolev training, even with $K = 1$, when the function $\boldsymbol{u}(\boldsymbol{x})$ is highly oscillatory, leading us to the following proposition, proven in appendix B.1:

Proposition 2.2.1 (Stiff learning dynamics in first-order parameter update). *For Sobolev training on the tuples $\{\boldsymbol{x}_i, \boldsymbol{u}(\boldsymbol{x}_i), \partial^m_{\boldsymbol{x}} \boldsymbol{u}(\boldsymbol{x}_i)\}_{i=1}^N, m \geqslant 1$, with total loss $\mathcal{L} = \sum_{i=1}^N \left(\lambda_0 |\boldsymbol{u}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \boldsymbol{u}(\boldsymbol{x}_i)|^2 + \lambda_1 |\mathcal{D}^m_{\boldsymbol{x}} \boldsymbol{u}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \mathcal{D}^m_{\boldsymbol{x}} \boldsymbol{u}(\boldsymbol{x}_i)|^2\right)$, the rate of change of the residual of the dominant Fourier mode $\widetilde{\boldsymbol{r}}(\boldsymbol{k}_0) = \widetilde{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{k}_0) - \widetilde{\boldsymbol{u}}(\boldsymbol{k}_0)$ using first-order parameter update is:*

$$\left| \frac{\mathrm{d}\widetilde{\boldsymbol{r}}(\boldsymbol{k}_0)}{\mathrm{d}\tau} \right| = \frac{\eta(\tau)}{O(\boldsymbol{k}_0)} \left[\lambda_0 O(1) + \lambda_1 O(\boldsymbol{k}_0^{2m}) \right] \left| \frac{\partial \widetilde{\boldsymbol{r}}(\boldsymbol{k}_0)}{\partial \boldsymbol{\theta}} \right|,$$

where $O(\cdot)$ is the Bachmann-Landau big-O symbol. For $\lambda_0 = \lambda_1 = 1$, this leads to training dynamics with slow time scale $\mathrm{d}/\mathrm{d}t \in O(\boldsymbol{k}_0^{-1})$ and fast time scale $\mathrm{d}/\mathrm{d}t \in O(\boldsymbol{k}_0^{2m-1})$ for $\widetilde{\boldsymbol{r}}(\boldsymbol{k}_0) \in O(1)$.

This proposition suggests that losses containing high orders of derivatives m, dominant high frequencies $\boldsymbol{k}_0$, or a combination of both exhibit dominant gradient statistics. This can lead to biased optimization of the dominant objectives with large gradients at the expense of the other objectives with relatively smaller gradients. This is akin to the well-known vanishing gradients phenomenon across the layers of a neural network [39], albeit now across different objectives of a multi-objective optimization problem. We therefore call this phenomenon *vanishing task-specific gradients*.

Proposition 2.2.1 also admits a qualitative comparison with the phenomenon of numerical stiffness in PDEs of the form

$$\frac{\partial \widetilde{\boldsymbol{u}}}{\partial t} = \mathbf{L}\widetilde{\boldsymbol{u}} + \mathbf{F}(\widetilde{\boldsymbol{u}}, t). \tag{6}$$

If the real parts of the eigenvalues of $\mathbf{L}$, $c = \mathrm{Re}(\lambda(\mathbf{L})) \gg 1$, then the solution of the above PDE has fast modes $\widetilde{\boldsymbol{u}} \in O(1)$ with $\mathrm{d}/\mathrm{d}t \in O(c)$ and slow modes $\widetilde{\boldsymbol{u}} \in O(1/c)$ with $\mathrm{d}/\mathrm{d}t \in O(1)$. Therefore, high frequency modes evolve on shorter time scales than low frequency modes. The rate amplitude for spatial derivatives of order m is $\mathrm{d}/\mathrm{d}t \in O(\boldsymbol{k}^{-m})$ for the wavenumber $\boldsymbol{k}$ [40]. Due to this analogy, we call the learning dynamics of a neural network with vanishing task-specific gradients *stiff*. Stiff learning dynamics leads to discrepancy in the convergence rates of different objectives in a loss function.

Taken together, vanishing task-specific gradients constitute a likely failure mode of PINNs, since their training amounts to a generalized version of Sobolev training.

3. Strategies for balanced PINN training

Since PINN training minimizes a weighted sum of multiple objectives (equation (3)), the result depends on appropriately chosen weights λ_k. The ratio between any two weights λ_i/λ_j, $i \neq j$, defines the relative importance of the two objectives during training. Suitable weighting is therefore quintessential to finding a Pareto-optimal solution that balances all objectives.

3.1. Weighting based on mean gradient statistics

While ad hoc manual tuning is still commonplace, it has recently been proposed to determine the weights as moving averages over the inverse gradient magnitude statistics of a given objective as [34]:

$$\begin{aligned} \hat{\lambda}_k(\tau) &= \frac{\max\{|\nabla_{\boldsymbol{\theta}^{\mathrm{sh}}}\mathcal{L}_1(\tau)|\}}{\lambda_k(\tau)\overline{|\nabla_{\boldsymbol{\theta}^{\mathrm{sh}}}\mathcal{L}_k(\tau)|}}, \\ \lambda_k(\tau+1) &= \alpha\lambda_k(\tau) + (1-\alpha)\hat{\lambda}_k(\tau), \end{aligned} \tag{7}$$

where $\nabla_{\boldsymbol{\theta}^{\mathrm{sh}}}\mathcal{L}_1$ is the gradient of the residual, i.e. of the first objective in equation (3), and α is a user-defined learning rate (here always $\alpha = 0.5$). All gradients are taken with respect to the shared parameters across tasks, and $|\cdot|$ is the component-wise absolute value. The overbar signifies the algebraic mean over the vector components. The maximum in the numerator is over all components of the vector. While this *max/avg* weighting has been shown to improve PINN performance [34], it does not completely alleviate the problem of *vanishing task-specific gradients*.

3.2. Inverse Dirichlet weighting

Instead of determining the loss weights proportional to the inverse average gradient magnitude, we here propose to use weights based on the gradient variance. In particular, we propose to use weights for which the variances over the components of the back-propagated weighted gradients $\lambda_k\nabla_{\boldsymbol{\theta}}\mathcal{L}_k$ become equal across all objectives, thus directly preventing *vanishing task-specific gradients*. This can be achieved in any (stochastic) gradient-descent optimizer by using the update rule in equation (4) with weights

$$\begin{aligned} \hat{\lambda}_k(\tau) &= \frac{\gamma(\tau)}{\mathrm{std}\{\nabla_{\boldsymbol{\theta}^{\mathrm{sh}}}\mathcal{L}_k(\tau)\}}, \\ \lambda_k(\tau+1) &= \alpha\lambda_k(\tau) + (1-\alpha)\hat{\lambda}_k(\tau), \end{aligned} \tag{8}$$

where $\gamma(\tau) = \max_{t=1,\ldots,K}(\mathrm{std}\{\nabla_{\boldsymbol{\theta}^{\mathrm{sh}}}\mathcal{L}_t(\tau)\})$, and $\mathrm{std}\{\cdot\}$ is the empirical (sample) standard deviation over the vector components. Under the assumption that the back-propagated gradients are normally distributed, the weighting strategy in equation (8) leads to balanced gradient distributions with variance $\gamma(\tau)^2$, i.e.

$$\frac{\gamma(\tau)}{\mathrm{std}\{\nabla_{\boldsymbol{\theta}^{\mathrm{sh}}}\mathcal{L}_k(\tau)\}}\mathcal{N}\left(\mu_k, \mathrm{Var}\{\nabla_{\boldsymbol{\theta}^{\mathrm{sh}}}\mathcal{L}_k(\tau)\}\right) = \mathcal{N}\left(\mu_k, \gamma(\tau)^2\right),$$

$\mathcal{N}(\cdot)$ is the normal distribution with mean $\mu_k, k = 1, \ldots, K$, and given variance.

For optimizers that are invariant to diagonal scaling of the gradients, such as the popular Adam optimizer [38], the weights in equation (8) can be efficiently computed as the inverse of the square root of the Dirichlet energy of each objective, i.e.

$$\mathrm{std}\{\nabla_{\boldsymbol{\theta}^{\mathrm{sh}}}\mathcal{L}_k(\tau)\} \propto \sqrt{\int_{\Omega_{\boldsymbol{\theta}}} |\nabla_{\boldsymbol{\theta}^{\mathrm{sh}}}\mathcal{L}_k(\tau)|^2 \,\mathrm{d}\boldsymbol{\theta}}\,.$$

We therefore refer to this weighting strategy as *inverse Dirichlet weighting*.

The weighting strategies in equations (7) and (8) are fundamentally different. Equation (7) weights objectives in inverse proportion to *certainty* as quantified by the average gradient magnitude. Equation (8) uses weights that are inversely proportional to *uncertainty* as quantified by the variance of the loss gradients. Equation (8) is also different from previous uncertainty-weighted approaches [31] because it measures the *training uncertainty* stemming from variance in the loss gradients rather than the uncertainty from the model's observational noise.

3.3. Gradient-based multi-objective optimization

We compare weighting-based scalarization approaches with gradient-based multi-objective approaches using Karush-Kuhn-Tucker (KKT) local optimality conditions to find a descent direction that decreases all objectives. Specifically, we adapt the multiple gradient descent algorithm (MGDA) [33] to PINNs. This algorithm leverages the KKT conditions

$$\exists \boldsymbol{\lambda} = \{\lambda_k\} \in \mathbb{R}_+^K$$
$$\text{s.t.} \sum_{k=1}^{K} \lambda_k \nabla_{\boldsymbol{\theta}^{\text{sh}}} \mathcal{L}_k(\boldsymbol{\theta}^{\text{sh}}, \boldsymbol{\theta}^k) = 0 \quad \text{and} \quad \sum_{k=1}^{K} \lambda_k = 1 \tag{9}$$

to solve an MTL problem. The MGDA is guaranteed to converge to a Pareto-stationary point [41]. Given the gradients $\nabla_{\boldsymbol{\theta}^{\text{sh}}}\mathcal{L}_k$ of all objectives $k = 1, \ldots, K$ with respect to the shared parameters, we find weights λ_k that satisfy the above KKT conditions. The resulting $\boldsymbol{\lambda}$ leads to a descent direction that improves all objectives [33], see also figure A.1. We adapt the MGDA to PINNs as described in appendix A.2. We refer to this adapted algorithm as *pinn-MGDA*.

4. Results

As we posit above, PINNs are expected to fail for problems with dominant high frequencies, i.e. with high $\boldsymbol{k}_0$ or high derivatives m, as is typical for multi-scale PDE models. We therefore present results of numerical experiments for such cases and compare the approximation properties of PINNs with different weighting schemes. This showcases training failure modes originating from vanishing task-specific gradients.

We first consider Sobolev training of neural networks, which is a special case of PINN training, as discussed in section 2.1. Due to the linear nature of the Sobolev loss (equation (5)), this test case allows us to compute ε-optimal analytical weights (see appendix A.1), providing a baseline for our comparison. Second, we consider a real-world example of a nonlinear PDE, using PINNs to model active turbulence.

In the first example, we specifically consider the Sobolev training problem with target function derivatives up to fourth order, i.e. $K = 4$, and total loss

$$\sum_{i=1}^{N} \left[\left| \boldsymbol{u}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \boldsymbol{u}(\boldsymbol{x}_i) \right|^2 + \sum_{k=1}^{4} \lambda_k \left| \hat{\xi}_k \mathcal{D}_{\boldsymbol{x}}^k \boldsymbol{u}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \xi_k \mathcal{D}_{\boldsymbol{x}}^k \boldsymbol{u}(\boldsymbol{x}_i) \right|^2 \right]. \tag{10}$$

The neural network is trained on the data $\boldsymbol{u}(\boldsymbol{x}_i)$ and their derivatives. To mimic inverse-modeling scenarios, in addition to producing an accurate estimate $\boldsymbol{u}_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$ of the function, the neural network is also tasked to infer unknown scalar pre-factors $\hat{\boldsymbol{\xi}} = (\hat{\xi}_1, \hat{\xi}_2, \hat{\xi}_3, \hat{\xi}_4)$ with true values chosen as $\xi_k = 1$. This mimics scenarios in which unknown coefficients of the PDE model also need to be inferred from data. For this loss, ε-optimal weights can be analytically determined (see appendix A.1) such that all objectives are minimized in an unbiased fashion [42]. This provides the baseline against which we benchmark the different weighting schemes.

4.1. Inverse Dirichlet weighting avoids vanishing task-specific gradients

We characterize the phenomenon of vanishing task-specific gradients, demonstrate its impact on the accuracy of a learned function approximation, and empirically validate Proposition 2.2.1. For this, we consider a generic neural network with five layers and 64 neurons per layer, tasked with learning a 2D function that contains a wide spectrum of frequencies and unknown coefficients $\hat{\boldsymbol{\xi}} = (\hat{\xi}_1, \hat{\xi}_2, \hat{\xi}_3, \hat{\xi}_4)$ using Sobolev training with the loss from equation (10). The details of the test problem and the training setup are given in appendix B.

We first confirm that in this test case vanishing task-specific gradients occur when using uniform weights, i.e. when setting $\lambda_k = 1$ for all $k = 0, \ldots, 4$. For this, we plot the gradient histogram of the five loss terms in figure 1(A). The histograms clearly show that training is biased in this case to mainly optimize the objective containing the highest derivative $k = 4$. All other objectives are neglected, as predicted by

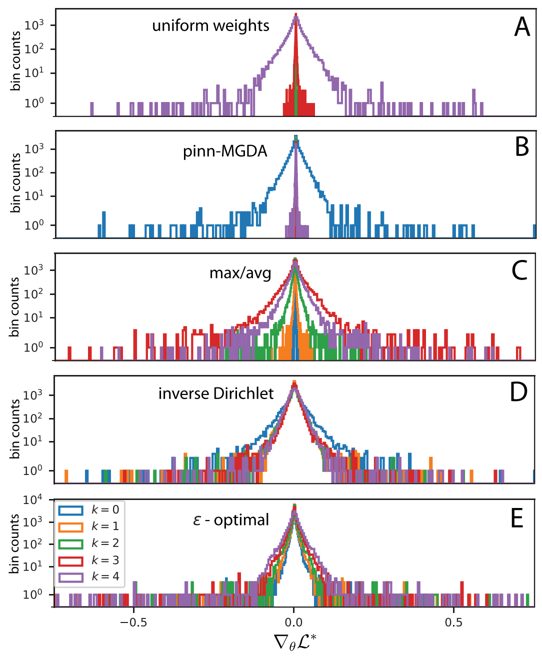

Figure 1. Loss gradient histograms illustrating vanishing task-specific gradients. The gradient distributions of the different objectives ($k = 0,\ldots,4$, see main text) are shown in different colors (inset legend) at training epoch 5000. The panels correspond to different λ weighting schemes as named. The histograms show normalized back-propagated weighted gradients for each objective, computed as $\nabla_\theta \mathcal{L}^*_k = \lambda_k \nabla_\theta \mathcal{L}_k / \max\{|\lambda_k \nabla_\theta \mathcal{L}_k|\}$ for $k = 0,\ldots,4$ and $|\cdot|$ is the component-wise absolute value.

proposition 2.2.1. This leads to a uniformly high approximation error on the test data (training vs. test data split 50:50, see appendix B), as shown in figure 2(B), as well as a uniformly high error in the estimated coefficients $\boldsymbol{\xi}$ (figure 2(C)). When determining the weights using the Pareto-seeking pinn-MGDA, the accuracy considerably improves (figures 2(B) and (C)). However, the phenomenon of vanishing task-specific gradients is still present, as seen in figure 1(B), this time favoring the objective for $k = 0$ and neglecting the derivatives. Max/avg weighting as proposed in reference [34] and given in equation (7) improves the accuracy for higher derivatives (figure 2(B)), but still suffers from unbalanced gradient distributions (figure 1(C)), which leads to a uniformly high error in the estimated coefficients $\boldsymbol{\xi}$ (figure 2(C)). The inverse Dirichlet weighting proposed here and detailed in equation (8) shows balanced gradient histograms (figure 1(D)) closest to those observed when using ε-optimal analytical weights (figure 1(E)). This leads to orders of magnitude better accuracy in both the function approximation and the coefficient estimates, as shown in figures 2(B) and (C), reaching the performance achieved when using ε-optimal weights.

This is confirmed when looking at the power spectrum of the function approximation $\boldsymbol{u_\theta}$ learned by the neural network in figure 2(D). Inverse Dirichlet weighting leads to results that are as good as those achieved by optimal weights in this test case, in particular when approximating high frequencies, as predicted by proposition 2.2.1. A closer look at the power spectra learned by the different weighting strategies also reveals that the max/avg strategy fails to capture high frequencies, whereas the pinn-MDGA performs surprisingly well (figure 2(D)) despite its unbalanced gradients (figure 1(B)). This may be problem-specific, as the weight trajectories taken by pinn-MDGA during training (figure B.1) are fundamentally different from the behavior of the other methods. This could be because the pinn-MGDA uses a fundamentally different strategy, purely based on satisfying Pareto-stationarity conditions (equation (9)). As shown in figure B.2, this leads to a rapid attenuation of the Fourier spectrum of the residual $|\tilde{r}(\boldsymbol{k})|$ over the first few learning epochs in this test case, allowing pinn-MGDA to escape the stiffness defined in proposition 2.2.1.

In summary, we observe stark differences between different weighting strategies. Of the strategies compared, only the inverse Dirichlet weighting proposed here is able to avoid vanishing task-specific gradients, performing on par with the ε-optimal solution. The pinn-MGDA seems to be able to escape, rather than avoid, the stiffness caused by vanishing task-specific gradients. We also observe a clear correlation between having balanced task-specific gradient distributions and achieving good approximation and estimation accuracy.

4.2. Inverse Dirichlet weighting enables multi-scale modeling using PINNs

We investigate the use of PINNs in modeling multi-scale dynamics in space and time where vanishing task-specific gradients are a common problem. As a real-world test problem, we consider meso-scale active

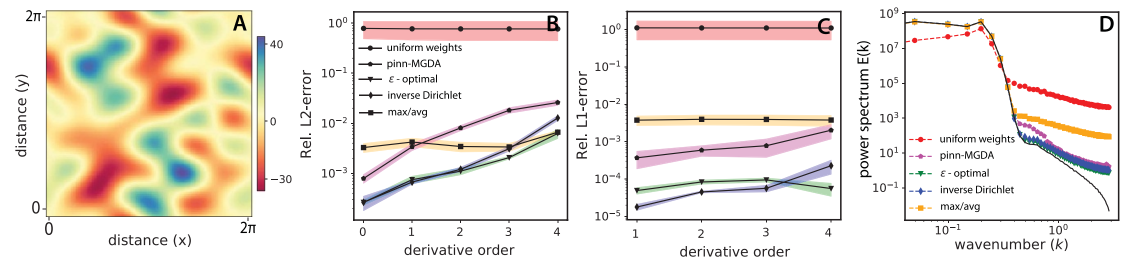

Figure 2. Approximation accuracy and convergence for Sobolev training using different weighting strategies. (A) One instance of the function $\boldsymbol{u}$ given by equation (B4), visualized on a 128 × 128 regular Cartesian grid. (B) The relative L2 error of the learned function approximation, i.e. $\|\boldsymbol{u_\theta}(\boldsymbol{x}_i) - \boldsymbol{u}(\boldsymbol{x}_i)\|_2 / \|\boldsymbol{u}(\boldsymbol{x}_i)\|_2$, across all grid nodes $\boldsymbol{x}_i$ of the test data $\boldsymbol{u}(\boldsymbol{x}_i)$ for different objective weighting methods (symbols, see inset legend). (C) The relative L1 error in the estimated model coefficients, i.e. $\|\boldsymbol{\xi} - \hat{\boldsymbol{\xi}}\|_1 / \|\boldsymbol{\xi}\|_1$, for the different objective weighting methods. (D) The power spectrum $E(k)$ of the true function $\boldsymbol{u}$ (solid black line) compared with those of the learned estimators $\boldsymbol{u_\theta}$ for the different weighting methods (colored lines with symbols). All results are shown after training for 20 000 epochs using the Adam optimizer. The colored bands show to the standard deviation over 5 repetitions for different realizations (i.e. RNG seeds) of the random function $\boldsymbol{u}$ (see equation (B4)).

turbulence as it occurs in living fluids, such as bacterial suspensions [43, 44] and multi-cellular tissues [45]. Unlike inertial turbulence, active turbulence has energy injected at small length scales comparable to the size of the actively moving entities, like bacteria or individual motor proteins. The spatio-temporal dynamics of active turbulence can be modeled using the generalized incompressible Navier–Stokes equation [43, 44]:

$$\begin{aligned}\frac{\partial \boldsymbol{u}}{\partial t} + \xi_0(\boldsymbol{u}\cdot\nabla\boldsymbol{u}) &= -\nabla p + \xi_1\,\nabla|\boldsymbol{u}|^2 - \alpha\boldsymbol{u} - \beta|\boldsymbol{u}|^2\,\boldsymbol{u}\\ &\quad + \Gamma_0\,\Delta\boldsymbol{u} - \Gamma_2\,\Delta^2\,\boldsymbol{u} \qquad (11)\\ \nabla\cdot\boldsymbol{u} &= 0\,,\end{aligned}$$

where $\boldsymbol{u}(t,\boldsymbol{x})$ is the mean-field flow velocity and $p(t,\boldsymbol{x})$ the pressure field. The coefficient ξ_0 scales the contribution from advection and ξ_1 the active pressure. The α and β terms correspond to a quartic Landau-type velocity potential. The higher-order terms with coefficients Γ_0 and Γ_2 model passive and active stresses arising from hydrodynamic and steric interactions, respectively [44]. Figure C.2 illustrates the rich multi-scale dynamics observed when numerically solving this equation on a square for the parameter regime $\Gamma_0 < 0$ and $\Gamma_2 > 0$ [43] (see appendix C for details on the numerical methods used). An important characteristic of equation (11) is the presence of disparate length and time scales in addition to fourth-order derivatives and nonlinearities in $\boldsymbol{u}$, which jointly lead to considerable numerical stiffness [40]. Given such large dominant wavenumbers $\boldsymbol{k}_0$, any data-driven approach, forward or inverse, is expected to encounter significant learning pathologies as per proposition 2.2.1.

We demonstrate these pathologies by applying PINNs to approximate the forward solution of equation (11) in 2D for the parameter regime explored in reference [43] in both square (figure C.2) and annular (figure 3(E)) domains. The details of the simulation and training setups are given in appendix C. The loss function includes the terms as given in equation (3), including the PDE residual, initial and boundary conditions, and the divergence-freeness constraint.

Figures 3(A) and (B) show the average (over different initializations of the neural network weights) relative L2 errors of the vorticity field over training epochs when using different PINN weighting strategies. As predicted by proposition 2.2.1, the uniformly weighted PINN fails completely both in the square (figure 3(A)) and in the annular geometry (figure 3(B)). Optimization is entirely biased towards minimizing the PDE residual, neglecting the initial, boundary, and incompressibility conditions. This imbalance is also clearly visible in the loss gradient distributions shown in figure C.5(A). Comparing the approximation accuracy after 7000 training epochs confirms the findings from the previous section with inverse Dirichlet weighting outperforming the other methods, followed by pinn-MGDA and max/avg weighting. Inverse Dirichlet weighting is found to assign adequate weights for the initial and boundary conditions relative to the equation residual (see figures C.3(A) and (D)).

In figure 3(C), we compare the convergence of the learned approximation $\boldsymbol{u_\theta}$ to a ground-truth numerical solution for different spatial resolutions h of the discretization grid. The errors scale as $O(h^r)$ with convergence orders r as identified by the dashed lines. While the pinn-MGDA and max/avg weighting achieve linear convergence ($r = 1$), the inverse Dirichlet weighted PINN converges with order $r = 4$. The uniformly weighted PINN does not converge at all and is therefore not shown in the plot. The evolution of the Fourier spectra $|\widetilde{r}(\boldsymbol{k})|$ of the residuals of both velocity components $(u, v) = \boldsymbol{u}$ shown in figure C.4 confirm the rapid solution convergence achieved by inverse Dirichlet weighted PINNs.

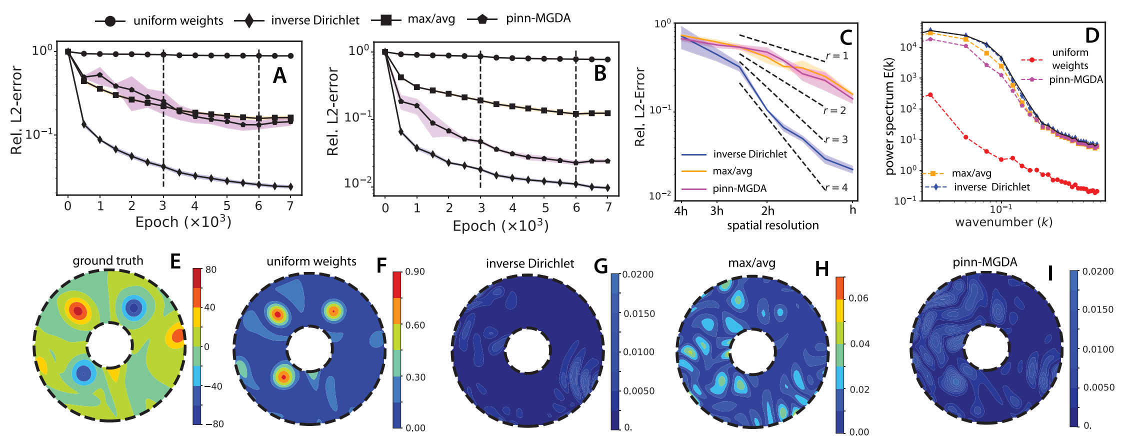

Figure 3. Forward modeling of active turbulence using PINNs. (A) and (B) Average relative L2 prediction errors for the vorticity field $\omega = \nabla \times \boldsymbol{u}$ of the active turbulence PDE in equation (11) in square (A) and annular (B) domains using different weighting strategies for PINN training (symbols, see legend). The dashed vertical lines mark epochs at which the learning rate $\eta(\tau)$ is adjusted. (C) Convergence of the different PINN predictions of the flow vorticity to the ground-truth numerical solution in a square domain for different grid resolutions. Dashed lines visualize integer convergence rates. The PINN solution is extracted at epoch 7000. The colored bands show the standard deviations over three different initializations of the neural network weights. (D) Power spectrum $E(k)$ of the turbulent flow velocity field (solid black line) compared with the power spectra of the approximations learned by PINNs with different weighting (see inset legend) at epoch 500. (E) Visualization of the ground-truth vorticity field in an annular domain obtained by numerically solving equation (11) as detailed in appendix C. (F)–(I) Spatial distribution of the relative prediction errors in the vorticity field when using PINNs with the weighting strategies named in the panel titles.

Inverse Dirichlet weighting also perfectly recovers the power spectrum $E(k)$ of the active turbulence velocity field, as shown in figure 3(D). This confirms that inverse Dirichlet weighting enables PINNs to capture high frequencies in the PDE solution, corresponding to fine structures in the fields, by avoiding the stiffness problem from proposition 2.2.1. Indeed, the PINNs trained with the other weighting methods fail to predict small structures in the flow field, as shown by the spatial distributions of the prediction errors in figures 3(F)–(I) compared to the ground-truth numerical solution in figure 3(E).

In Appendices D and E, we provide results for two additional examples: approximating the solution of the 2D Poisson equation (see figure D.1) on a square with a large spectrum of frequencies, and learning the spatio-temporal dynamics of curvature-driven level-set flow (see figures E.1 and E.2) developing small structures. In both of these additional multi-scale examples we notice similar behavior, with inverse Dirichlet weighting outperforming the other approaches either in terms of accuracy or convergence rate.

In summary, these results suggest that avoiding vanishing task-specific gradients can enable the use of PINNs in multi-scale problems previously not amenable to neural-network modeling. Moreover, we observe that seemingly small changes in the weighting strategy can lead to significant differences in the convergence order of the learned approximation.

4.3. Inverse Dirichlet weighting protects against catastrophic forgetting

Artificial neural networks tend to 'forget' their parameterization when sequentially training for multiple tasks [46], a phenomenon referred to as *catastrophic forgetting*. Sequential training is often used, e.g. when learning to de-noise data and estimate parameters at the same time, like combined image segmentation and denoising [47], or when learning solutions of fluid mechanics models with additional constraints like divergence-freeness of the flow velocity field [16].

In PINNs, catastrophic forgetting can occur, e.g. in the frequently practiced procedure of first training on measurement data alone and adding the PDE model only at later epochs [48]. Such sequential training can be motivated by the computational cost of evaluating the derivatives in the PDE using automatic differentiation over the network, so it seems prudent to pre-condition the network for data fitting and introduce the residual later in order to regress for the missing network parameters.

Using the active turbulence test case described above, we investigate how the different weighting schemes compared here influence the phenomenon of catastrophic forgetting. For this, we consider the inverse problem of inferring unknown model parameters and latent fields like the effective pressure p^* from flow velocity data $\boldsymbol{u}_i$. We train the PINNs in three sequential steps: (1) first fitting flow velocity data, (2) later including the additional objective for the incompressibility constraint $\nabla \cdot \boldsymbol{u} = 0$, (3) finally including also the objective for the PDE model residual. This creates a setup in which catastrophic forgetting occurs when using uniform PINN weights, as confirmed in figure 4(A) (arrows). As soon as additional objectives

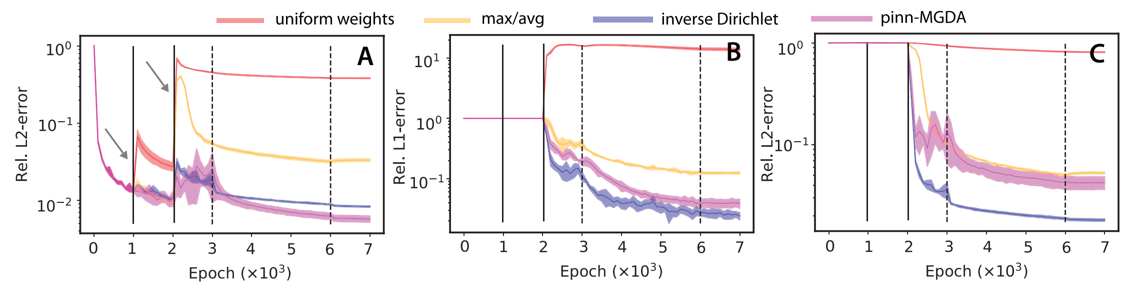

Figure 4. Catastrophic forgetting in sequential PINN training of active turbulence. (A) Test error in the velocity field $\boldsymbol{u}$ during the three-step sequential training described in the main text with different weighting strategies (colors, see legend). The solid vertical lines at epochs 1000 and 2000 mark the introduction of the incompressibility constraint and PDE residual, respectively. The dashed vertical lines mark epochs at which the learning rate $\eta(\tau)$ is adjusted. (B) Average relative L1 errors in the inferred values for the coefficients ξ_0 (ξ_1 is absorbed into the effective pressure p^*, see appendix C), α, β, Γ_0, and Γ_2 of equation (11) over training epochs. (C) Average relative L2 error in the inferred latent effective pressure gradient ∇p^* across training epochs.

(divergence-freeness at epoch 1000 and equation residual at epoch 2000) are introduced, the PINN loses the parameterization learned on the previous objectives, leading to large and increasing test error for the field $\boldsymbol{u}$.

Dynamic weighting strategies can protect PINNs against catastrophic forgetting by adequately adjusting the weights whenever an additional objective is introduced. This is confirmed in figure 4(A) with inverse Dirichlet and pinn-MGDA only minimally affected by catastrophic forgetting. The max/avg weighting strategy [34], however, does not protect the network quite as well. When looking at the test errors for the learned model coefficients (figure 4(B)) and the accuracy of the estimated latent pressure field $p(t,\boldsymbol{x})$ (figure 4(C)), for which no training data is given, the inverse Dirichlet weighting proposed here outperforms the other approaches. Snapshots of the reconstructed pressure fields using different strategies are shown in figure C.7. Results for inference from noisy data are shown in figures C.6 and C.8.

In summary, inverse Dirichlet weighting can protect a PINN against catastrophic forgetting, making sequential training a viable option. In our tests, it offered the best protection of all compared weighting schemes.

5. Conclusion and discussion

PINNs have rapidly been adopted by the scientific community for diverse applications in forward and inverse modeling. It is straightforward to show that the training of a PINN amounts to a MTL problem, which is sensitive to the choice of regularization weights. Trivial uniform weights are known to lead to failure of PINNs, empirically found to exacerbate for physical models with disparate scales or high derivatives.

As we have shown here, this failure can be explained by analogy to numerical stiffness of PDEs when understanding Sobolev training of neural networks [7] as a special case of PINN training. This connection enabled us to prove a proposition stating that PINN learning pathologies are caused by dominant high-frequency components or high derivatives in the physics prior. The asymptotic arguments in our proposition inspired a novel dynamic adaptation strategy for the regularization weights of a PINN during training, which we termed *inverse Dirichlet weighting*.

We empirically compared inverse Dirichlet weighting with the recently proposed max/avg weighting for PINNs [34]. Moreover, we looked at PINN training from the perspective of multi-objective optimization, which avoids *a-priori* choices of regularization weights. We therefore adapted to PINNs the state-of-the-art MGDA for training multi-objective artificial neural networks, which has previously been proven to converge to a Pareto-stationary solution [33]. This provided another baseline to compare with. Finally, we compared against analytically derived ε-optimal static weights in a simple linear test problem where those can be derived, providing an absolute baseline. In all comparisons, and across a wide range of problems from 2D Poisson to multi-scale active turbulence and curvature-driven level-set flow, inverse Dirichlet weighting empirically performed best. We also found that all weighting strategies are in general agnostic to the choice of the activation function, with inverse Dirichlet weighting outperforming the other strategies irrespective of the activation function chosen.

The inverse Dirichlet weighting proposed here can be interpreted as an uncertainty weighting with training uncertainty quantified by the batch variance of the loss gradients. This is different from previous approaches [31] that used weights quantifying the model uncertainty, rather than the training uncertainty. The proposition proven here provides an explanation for why the loss gradient variance may be a good choice, and our results confirmed that weighting based on the Dirichlet energy of the loss objectives outperforms other weighting heuristics, as well as Pareto-front seeking algorithms like pinn-MGDA. The proposed inverse Dirichlet weighting also performed best at protecting a PINN against catastrophic forgetting in sequential training, where loss objectives are introduced one-by-one, making sequential training of PINNs a viable option in practice.

While we have focused on PINNs here, our analysis equally applies to Sobolev training of other neural networks, and to other multi-objective losses, as long as training is done using an explicit first-order (stochastic) gradient descent optimizer, of which Adam [38] is the most commonly used. In all our examples, we found that the Adam optimizer leads to better convergence than other first-order update rules like RMSprop or Adagrad (see figure D.2). However, hybrid strategies can also be envisioned. For instance when the optimizer has converged close to a local optimum, the uncertainty in the gradients is small relative to the mean gradient component [49]. In such scenarios, inverse Dirichlet weighting can lead to large optimization step sizes that plateau the learning (see figure E.1). A hybrid strategy could then be beneficial, combining initial inverse Dirichlet weighting with, e.g. LBFGS-based fine-tuning close to a local optimum [16].

Taken together, we have presented a connection between PINNs and Sobolev training in order to explain a likely failure mode of PINNs, and we have proposed a simple yet effective strategy to avoid training failure. The proposed inverse Dirichlet weighting strategy is easily included into any first-order optimizer at almost no additional computational cost (see figure C.9). It has the potential to improve the accuracy and convergence of PINNs by orders of magnitude, to enable sequential training with less risk of catastrophic forgetting, and to extend the application realm of PINNs to multi-scale problems that were previously not amenable to data-driven modeling.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://github.com/mosaic-group/inverse-dirichlet-pinn.

Acknowledgments

This work was supported by the German Research Foundation (DFG) – EXC-2068, Cluster of Excellence 'Physics of Life', and by the Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI) Dresden/Leipzig, funded by the Federal Ministry of Education and Research (BMBF). The present work was also partly funded by the Center for Advanced Systems Understanding (CASUS), which is financed by Germany's Federal Ministry of Education and Research (BMBF) and by the Saxon Ministry for Science, Culture and Tourism (SMWK) with tax funds on the basis of the budget approved by the Saxon State Parliament.

Appendix A

A.1. Analytically optimal weights

For Sobolev loss functions based on elliptic PDEs, optimal weights λ_k that minimize all objectives in an unbiased way can be determined by ε-closeness analysis [42].

Definition 1 (ε-closeness). A candidate solution $\boldsymbol{u}$ is called ε-close to the true solution $\hat{\boldsymbol{u}}$ if it satisfies

$$\frac{|\mathcal{D}^{\boldsymbol{\beta}}\boldsymbol{u}(\boldsymbol{x}) - \mathcal{D}^{\boldsymbol{\beta}}\hat{\boldsymbol{u}}(\boldsymbol{x})|}{|\mathcal{D}^{\boldsymbol{\beta}}\hat{\boldsymbol{u}}(\boldsymbol{x})|} \leqslant \varepsilon \tag{A1}$$

for any multi-index $\boldsymbol{\beta} \in \mathbb{N}^d$ and $\boldsymbol{x} = (x_1, \ldots, x_d) \in \Omega \subset \mathbb{R}^d$, thus $\mathcal{D}^{\boldsymbol{\beta}}\boldsymbol{u}(\boldsymbol{x}) = \frac{\partial^{|\boldsymbol{\beta}|}\boldsymbol{u}(\boldsymbol{x})}{\partial x_1^{\beta_1}\partial x_2^{\beta_2}\ldots\partial x_d^{\beta_d}}$ with $|\boldsymbol{\beta}| = \sum_{i=1}^{d}\beta_i$.

For the loss function typically used in Sobolev training of neural networks, i.e.

$$\mathcal{L}_k(\cdot) = \int_\Omega \left(\mathcal{D}_{\boldsymbol{x}}^k\hat{\boldsymbol{u}} - \mathcal{D}_{\boldsymbol{x}}^k\boldsymbol{u}\right)^2 \,\mathrm{d}\boldsymbol{x}.$$

We can use ε-closeness analysis to show that

$$\mathcal{L}_k \leqslant \varepsilon^2 \int_\Omega \left(|\xi_k\mathcal{D}_{\boldsymbol{x}}^k(\hat{\boldsymbol{u}})|\right)^2 \,\mathrm{d}\boldsymbol{x}. \tag{A2}$$

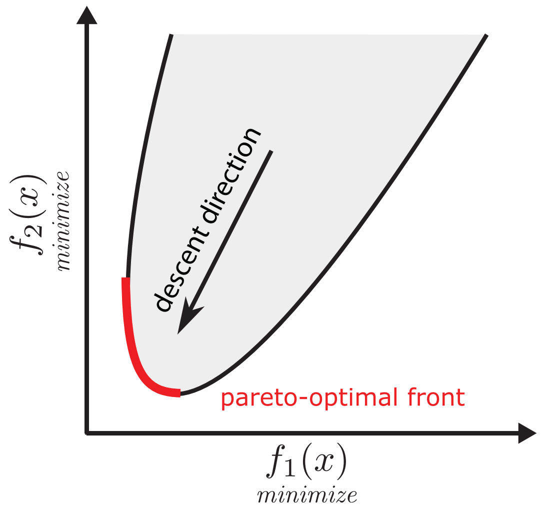

Figure A.1. Multiple gradient descent algorithm. Illustration of multi-objective optimization for minimizing two objectives $f_1(x)$ and $f_2(x)$. The red curve corresponds to the Pareto-optimal front. The MGDA algorithm searches for a descent direction (arrow) that minimizes all objectives and is close to the Pareto-optimal front. The gray region corresponds to the feasible objective space.

Using this inequality, we can bound the total loss as

$$\mathcal{L}(\boldsymbol{u}) \leqslant \varepsilon^2 \sum_{k=1}^{K} \lambda_k \int_\Omega \left(|\xi_k \mathcal{D}^k_{\boldsymbol{x}}(\hat{\boldsymbol{u}})|\right)^2 \, \mathrm{d}\boldsymbol{x} = \varepsilon^2 \sum_{k=1}^{K} \lambda_k \mathcal{I}_k, \tag{A3}$$

with $\mathcal{I}_k = \int_\Omega \left(|\xi_k \mathcal{D}^k_{\boldsymbol{x}}(\hat{\boldsymbol{u}})|\right)^2 \mathrm{d}\boldsymbol{x}$. This bound can be used to determine the weights $\{\lambda_k\}_{k=1}^K$ that result in the smallest ε:

$$\begin{aligned} \mathcal{L}_k(\boldsymbol{u}) &\leqslant \frac{1}{\lambda_k}\mathcal{L}(\boldsymbol{u}) \leqslant \varepsilon^2\ \lambda_k \mathcal{I}_k\ \forall k = 1,\ldots,K, \\ \mathcal{L}(\boldsymbol{u}) &\leqslant \varepsilon^2\ \min\{\lambda_1\,\mathcal{I}_1,\ldots,\lambda_K \mathcal{I}_K\}. \end{aligned} \tag{A4}$$

The weights $\lambda_k, \forall k \in 1,\ldots,K$, that lead to the tightest upper bound on the total loss in equation (A4) can be found by solving the maximization-minimization problem described in following corollary:

Corollary 1 (A.1.1.). *A maximization-minimization problem can be transformed into a piecewise linear convex optimization problem by introducing an auxiliary variable z, such that*

$$\begin{aligned} &\textit{maximize } z \\ &\textit{subject to } 1^\top \boldsymbol{\lambda}^* = 1; \\ &z \leqslant \lambda_k \mathcal{I}_k, \quad k = 1,\ldots,K. \end{aligned}$$

The analytical solution to this problem exists and is given by $\lambda_k^* = \frac{\Pi_{j\neq k}\mathcal{I}_j}{\sum_{k=1}^K \Pi_{j\neq k}\mathcal{I}_j}, \quad k = 1,\ldots,K.$

For a Sobolev training problem with loss given by equation (5), weights that lead to ε-optimal solutions are computed using

$$\mathcal{I}_k = \int_\Omega \left(|\xi_k \mathcal{D}^k_{\boldsymbol{x}}(\hat{\boldsymbol{u}})|\right)^2 \mathrm{d}\boldsymbol{x}.$$

A.2. PINN multiple gradient descent algorithm (pinn-MGDA)

We adapt the MGDA to PINNs as illustrated in figure A.1 by formulating the conditions for Pareto stationary as stated in equation (9) as a quadratic program:

$$\underset{\boldsymbol{\lambda}\in\mathbb{R}^K}{\text{minimize}}\ \frac{1}{2}\boldsymbol{\lambda}^\top \boldsymbol{Q}\boldsymbol{\lambda} \ \text{ s.t. }\ \lambda_k \geqslant 0\, \forall k \in [K] \ \text{ and } \ \sum_{k=1}^{K} \lambda_k = 1, \tag{A5}$$

where $\boldsymbol{Q} = \boldsymbol{U}^\top \boldsymbol{U}$. The matrix $\boldsymbol{U} \in \mathbb{R}^{|\mathrm{B}|\times K}$ is given by

$$\boldsymbol{U} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ \nabla_{\boldsymbol{\theta}^{\mathrm{sh}}}\mathcal{L}_1 & \nabla_{\boldsymbol{\theta}^{\mathrm{sh}}}\mathcal{L}_2 & \dots & \nabla_{\boldsymbol{\theta}^{\mathrm{sh}}}\mathcal{L}_K \\ \vdots & \vdots & & \vdots \end{bmatrix}.$$

We solve the quadratic program in equation (A5) using the Frank-Wolfe algorithm.

Appendix B. Sobolev training

B.1. Proof of proposition 2.2.1

Let us assume the Fourier spectrum of the signal is a delta function, i.e. $\widetilde{u}(k) = \Gamma_0\,\delta(k-k_0)$, with k_0 being the dominant wavenumber. The residual at training time (t) is given as $r(\cdot,t) = u_\theta - u$, and its corresponding Fourier coefficients for the wavenumber k is given by $\widetilde{r}(k,t) = \widetilde{u}_\theta(k,t) - \widetilde{u}(k)$. The learning rate of the residual for the dominant frequency is given as,

$$\begin{aligned} \left|\frac{\partial\widetilde{r}(k_0,\tau)}{\partial\tau}\right| &= \left|\frac{\partial\widetilde{u}_\theta(k_0,\tau)}{\partial\theta}\right|\left|\frac{\partial\theta}{\partial\tau}\right| \\ &= \left|\eta\frac{\partial\widetilde{u}_\theta(k_0)}{\partial\theta}\right|\left|\frac{\partial\mathcal{L}}{\partial\theta}\right| \\ &= \left|\eta\frac{\partial\widetilde{u}_\theta(k_0)}{\partial\theta}\right|\left|\lambda_0\frac{\partial\mathcal{L}_0}{\partial\theta} + \lambda_1\frac{\partial\mathcal{L}_1}{\partial\theta}\right|, \end{aligned} \tag{B1}$$

where $\mathcal{L}_0 = \sum_{i=1}^{N} |\boldsymbol{u}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \boldsymbol{u}(\boldsymbol{x}_i)|^2$ and $\mathcal{L}_1 = \sum_{i=1}^{N} |\mathcal{D}^m_{\boldsymbol{x}}\boldsymbol{u}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \mathcal{D}^m_{\boldsymbol{x}}\boldsymbol{u}(\boldsymbol{x}_i)|^2$. Equation (B1) follows from the gradient flow of the parameters $\partial_\tau\theta \approx \eta\partial_\theta\mathcal{L}$ as described in equation (4). From Parseval's Theorem [50], we can compute the magnitude of the gradients for both objectives $\mathcal{L}_0$ and $\mathcal{L}_1$ with respect to the network parameters θ.

$$\begin{aligned} \frac{\partial\mathcal{L}_0}{\partial\theta} &= 2\sum_{k=-N/2}^{N/2-1}\left|\mathbf{Re}\left[\widetilde{u}_\theta(k) - \widetilde{u}(k)\right]\frac{\partial\widetilde{u}_\theta(k)}{\partial\theta}\right| \\ &\leqslant 2\left|\Gamma_0\,\frac{\partial\widetilde{u}_\theta(k_0)}{\partial\theta}\right| + 2\sum_{k=-N/2}^{N/2-1}\left|\widetilde{u}_\theta(k)\frac{\partial\widetilde{u}_\theta(k)}{\partial\theta}\right| \\ &\approx O(1)\left|\frac{\partial\widetilde{r}(k_0)}{\partial\theta}\right|. \end{aligned} \tag{B2}$$

The second step in the above calculation involves triangular inequality and the assumption that the Fourier spectrum of the signal is a delta function. In a similar fashion, we can repeat the same analysis for the Sobolev part of the objective function as follows,

$$\begin{aligned} \frac{\partial\mathcal{L}_1}{\partial\theta} &= 2\sum_{k=-N/2}^{N/2-1}\left|\mathbf{Re}\left[(ik)^m\left(\widetilde{u}_\theta(k) - \widetilde{u}(k)\right)\right]\frac{\partial(ik)^m\widetilde{u}_\theta(k)}{\partial\theta}\right| \\ &= 2\sum_{k=-N/2}^{N/2-1}\left|\mathbf{Re}\left[(ik)^{2m}\left(\widetilde{u}_\theta(k) - \widetilde{u}(k)\right)\right]\frac{\partial\widetilde{u}_\theta(k)}{\partial\theta}\right| \\ &\leqslant 2\left|\Gamma_0\,\mathbf{Re}\left[(ik_0)^{2m}\right]\frac{\partial\widetilde{u}_\theta(k_0)}{\partial\theta}\right| + 2\sum_{k=-N/2}^{N/2-1}\left|\mathbf{Re}\left[(ik)^{2m}\right]\widetilde{u}_\theta(k)\frac{\partial\widetilde{u}_\theta(k)}{\partial\theta}\right| \\ &\approx O(k_0^{2m})\left|\frac{\partial\widetilde{r}(k_0)}{\partial\theta}\right|. \end{aligned} \tag{B3}$$

By substituting equations (B2) and (B3) into equation (B1), we get,

$$\begin{aligned} \left|\frac{\partial\widetilde{r}(k_0,\tau)}{\partial\tau}\right| &= \left|\eta\frac{\partial\widetilde{u}_\theta(k_0)}{\partial\theta}\right|\left(\lambda_0 O(1)\left|\frac{\partial\widetilde{r}(k_0)}{\partial\theta}\right| + \lambda_1 O(k_0^{2m})\left|\frac{\partial\widetilde{r}(k_0)}{\partial\theta}\right|\right) \\ &= \frac{\eta}{O(k_0)}\left[\lambda_0 O(1) + \lambda_1 O(k_0^{2m})\right]\left|\frac{\partial\widetilde{r}(k_0)}{\partial\theta}\right|. \end{aligned}$$

We use the fact that the spectral rate of the network function residual is inversely proportional to the wavenumber, i.e. $\partial\widetilde{u}_\theta(k)/\partial\theta = O\left(k^{-1}\right)$ [26].

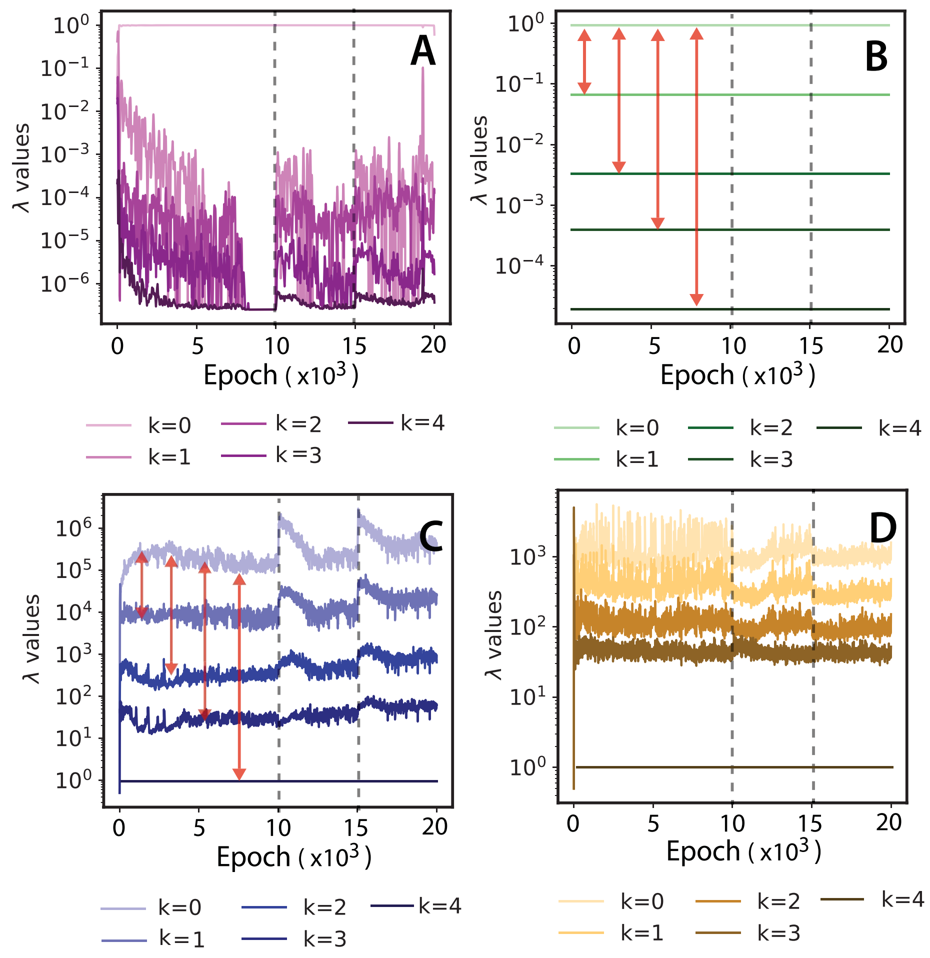

Figure B.1. $\boldsymbol{\lambda}$ trajectories for different weighing strategies. The dynamic weights $\lambda(\tau)$ over the training epochs τ for a single instance of Sobolev training of the neural networks for different methods: (A) pinn-MGDA, (B) ε-optimal, (C) inverse Dirichlet, (D) max/avg weighting. Different trajectories in the same plots correspond to the weights $\lambda_k, k = 1, \ldots, K$, associated with different objectives. We observe that the relative weights $(\lambda_i/\lambda_j)\forall(i,j) = 1, \ldots, K$ between objectives for the inverse Dirichlet and ε-optimal weights are of the same order of magnitude: $\lambda_0/\lambda_1 \approx 10$, $\lambda_0/\lambda_2 \approx 10^2 \ldots 10^3$, $\lambda_0/\lambda_3 \approx 10^3 \ldots 10^4$, $\lambda_0/\lambda_4 \approx 10^4 \ldots 10^5$ (shown by red double-headed arrows).

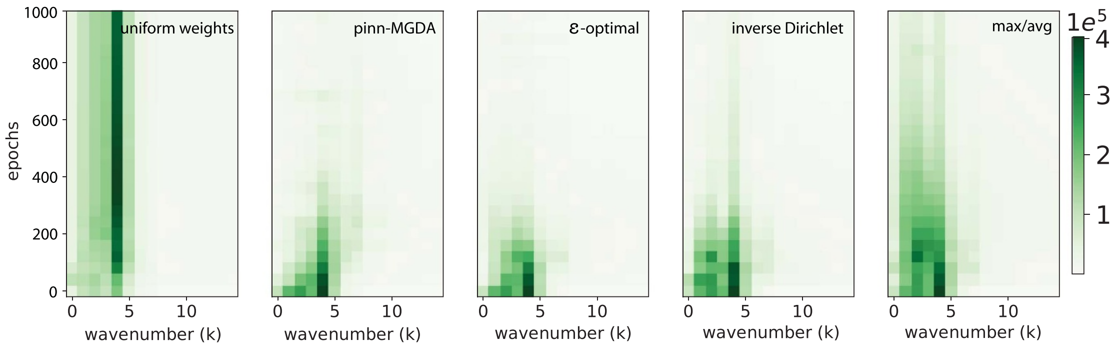

Figure B.2. Evolution of the Fourier spectrum of the function residual $\tilde{r}(\boldsymbol{k})$. The Fourier spectrum of the residual tracked over the training epochs for different methods (inset text). The color intensity corresponds to the magnitude of the residual, $|\tilde{r}(\boldsymbol{k})|$.

B.2. Sobolev training setup

We use a neural network $f_\theta : (x, y) \rightarrow (u)$ with five layers and 64 neurons per layer and $\sin(\cdot)$ activation function. The target function is a sinusoidal forcing term of the form,

$$u(x,y) = \sum_{i=1}^{M} A_x^i \cos(2\pi l_x^i x/L + \phi_x^i) A_y^i \sin(2\pi l_y^i y/L + \phi_y^i), \tag{B4}$$

where $M = 20$ and $L = 2\pi$ is the size of the domain. The parameters A_x, A_y are drawn independently and uniformly from the interval $[-5, 5]$, and similarly ϕ_x, ϕ_y are drawn from the interval $[0, 2\pi]$. The parameters l_x, l_y which set the local length scales are sampled uniformly from the set $\{1, 2, 3, 4, 5\}$. The target function $u(x, y)$ is given on a 2D spatial grid of resolution 128×128 covering the domain $[0, 2\pi] \times [0, 2\pi]$. The grid data is then divided into a 50/50 train and test split which corresponds to 8192 training points used. We train the neural network for 20 000 epochs using the Adam optimizer [38]. Each epoch is composed of two mini-batch updates of batch size $|\mathrm{B}| = 4096$. The initial leaning rate is chosen as $\eta = 10^{-3}$ and decreased by a factor of 10 after 10 000 and 15 000 epochs.

Appendix C. Forward and inverse modeling in active turbulence

The ground truth solution of the 2D incompressible active fluid is computed in the vorticity formulation which is given as follows,

$$\frac{\partial \omega}{\partial t} + \xi_0\, \boldsymbol{u} \cdot \nabla \omega = -(\alpha + \beta |\boldsymbol{u}|^2)\omega + \beta \nabla |\boldsymbol{u}|^2 \;\times \boldsymbol{u} + \Gamma_0\, \Delta \omega - \Gamma_2\, \Delta^2\, \omega. \tag{C1}$$

The simulation is conducted with a pseudo-spectral solver, where the derivatives are computed in the Fourier space and the time integration is done using the Integration factor method [40, 51]. The equations are solved in the domain $[0, 2\pi] \times [0, 2\pi]$ with a spatial resolution of 256×256 and a time-step $dt = 0.0001$. The incompressibility is enforced by projecting the intermediate velocities on to the divergence-free manifold. The simulations in the square domain are performed with periodic boundary conditions, where the values of the parameters are chosen as follows: $\xi_0 = 3.5, \alpha = -1, \beta = 0.5, \Gamma_0 = -0.045, \Gamma_2 = |\Gamma|^3$ [43].

C.1. Forward problem setup

We study the forward solution of the active turbulence problem using PINNs in the $(\omega - \psi)$ formulation. We employ a neural network $f_\theta : (x, y, t) \rightarrow (u, v)$ with seven layers and 100 neurons per layer and sin-activation functions. For choosing the activation function we conducted extensive numerical experiments using different activation functions for a simple function reconstruction problem with the objective

$$\mathcal{L}(\boldsymbol{u}, \boldsymbol{u}_\theta, \theta) = \frac{1}{N_{\mathrm{int}}} \sum_{i=1}^{N_{\mathrm{int}}} \left| \boldsymbol{u}_{\boldsymbol{\theta}}(t_i, \boldsymbol{x}_i^{\Omega}) - \boldsymbol{u}_i) \right|^2,$$

where $\boldsymbol{u}_i = \boldsymbol{u}(t_i, \boldsymbol{x}_i^{\Omega})$. In figure C.1 we clearly see that the sin activation function leads to the best approximation accuracy for the velocities and their associated derivatives. The vorticity ω is directly computed from the network outputs using automatic differentiation. For the solution in the square domain, we select a sub-domain $[\frac{\pi}{4}, \frac{3\pi}{4}] \times [\frac{\pi}{4}, \frac{3\pi}{4}]$ with 50 time steps $(dt = 0.01)$ accounting to a total time of 0.5 units in simulation time. We choose $N_{\mathrm{I}} = 4096$ points for the initial conditions, $N_{\mathrm{B}} = 12\,800$ boundary points and a total of $N_{\mathrm{int}} = 192200$ residual points in the interior of the square domain. The network is trained for 7000 epochs using the Adam optimizer with each epoch consisting of 49 mini-batch updates with a batch size $|\mathrm{B}| = 4000$. The velocities are provided at the boundaries of the training domain.

For the forward simulation in the annular geometry, the domain is chosen with the inner radius of $r_{\mathrm{inner}} \approx 0.245$ units and the outer radius $r_{\mathrm{outer}} \approx 0.85$ units. We choose $N_{\mathrm{I}} = 3536$ points for the initial conditions, $N_{\mathrm{B}} = 14\,400$ boundary points and a total of $N_{\mathrm{int}} = 162\,400$ residual points in the interior of the domain. The network training is again performed over 7000 epochs using Adam with each epoch consisting of 41 mini-batch updates with a batch size $|\mathrm{B}| = 4000$. Dirichlet boundary conditions are enforced at the inner and outer rims of the annular geometry by providing the velocities. The network is optimized subject to the loss function

$$\mathcal{L}(\boldsymbol{u}, \theta) = \lambda_1\, \mathcal{L}_{\mathrm{boundary}} + \lambda_2\, \mathcal{L}_{\mathrm{res}} + \lambda_3\, \mathcal{L}_{\mathrm{div}} + \lambda_4\, \mathcal{L}_{\mathrm{init}}.$$

Here, $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ correspond to the weights of the objectives for boundary condition, equation residual, divergence, and initial condition, respectively.

C.2. Inverse problem setup

For studying the inverse problem we resort to the primitive formulation (u, v, p^*) of the active turbulence model, with $p^* = -\nabla p + \xi_1 \nabla |\boldsymbol{u}|^2$ as the effective pressure. We used a neural network $f_\theta : (x, y, t) \rightarrow (u, v, p^*)$ with five layers and 100 neurons per layer and sin-activation functions. Given the measurement data of flow velocities $\{u_i, v_i\}_{i=1}^{N}$ at discrete points, we infer the effective pressure p^* along with the parameters $\Sigma = \{\xi_0, \alpha, \beta, \Gamma_0, \Gamma_2\}$. The total of 204 800 points, sampled from the same domain as for the forward

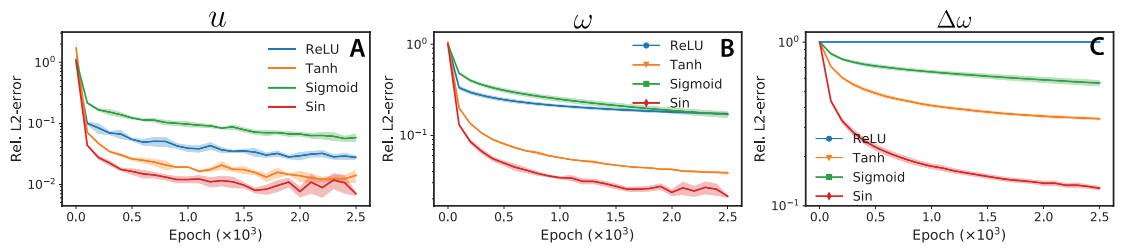

Figure C.1. Loss trajectories for the reconstruction problem using different activation functions. (A) Relative L2 error trajectories along the training epochs for reconstructing the horizontal component u. (B) Relative L2 error trajectories for reconstructing the vorticity field ω. (C) Relative L2 error trajectories for reconstructing the Laplacian of the vorticity field, $\Delta\omega$. The problem was set up in a square domain with $N_{\text{int}} = 192\,200$.

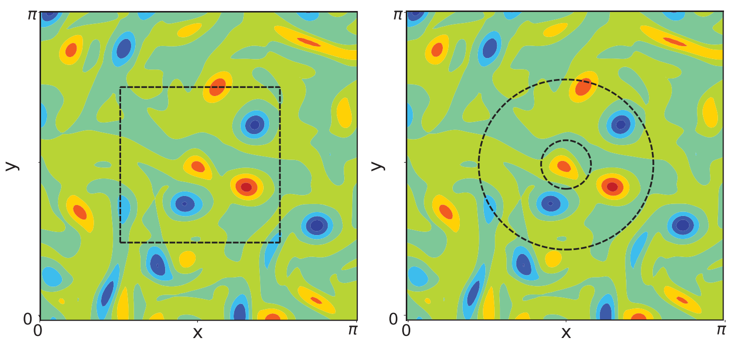

Figure C.2. Training domain for the active turbulence problem. Left: training domain inside the dotted square used for forward solution in the square domain. Right: training domain interior of the dotted annular outline used for forward solution in the annular geometry.

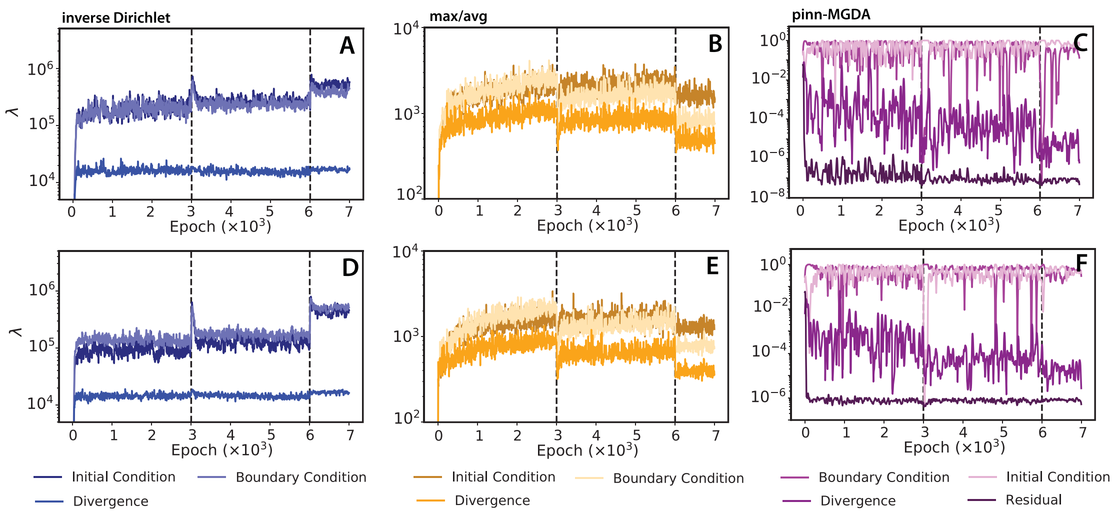

Figure C.3. λ trajectories for different weighting strategies for the active turbulence problem. (A)–(C) correspond to the λ trajectories for the forward solution of the active turbulence problem in the square domain (see left figure C.2). (D)–(F) correspond to the λ trajectories for the forward solution in the annular geometry (see right figure C.2). The columns correspond to the different weighting strategies as named on the top left corner of each column.

solution in the square domain, are utilized with a 70/30 train and test split, which corresponds to 143 360 training points used. The network is trained for a total of 7000 epochs using the Adam optimizer with each epoch consisting of 35 mini-batch updates with a batch-size $|B| = 4096$. We optimize the model subject to the loss function

$$\mathcal{L}(\boldsymbol{u},\theta) = \lambda_1\,\mathcal{L}_{\text{fit}} + \lambda_2\,\mathcal{L}_{\text{res}} + \lambda_3\,\mathcal{L}_{\text{div}},$$

with $\lambda_1, \lambda_2, \lambda_3$ corresponding to the data-fidelity, equation residual and the objective that penalises non-zero divergence in velocities, respectively. The training is done in a three-step approach, with the first 1000 epochs

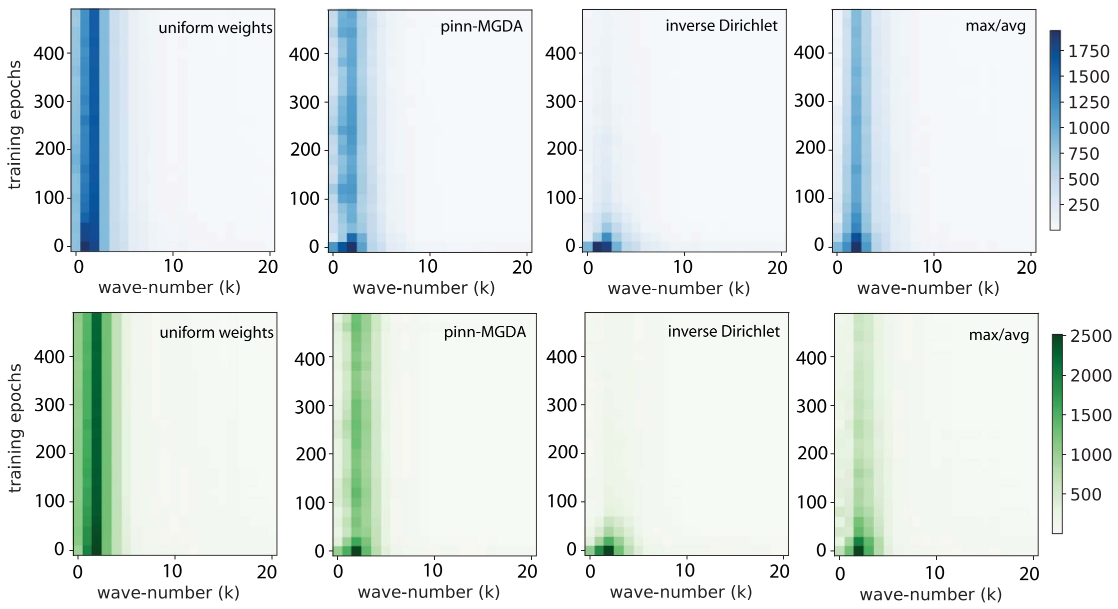

Figure C.4. Evolution of the Fourier spectrum of the function residual $\widetilde{r}(\boldsymbol{k})$. Top row: the Fourier spectrum of the residual for the horizontal velocity component u. Bottom row: the Fourier spectrum of the residual for the vertical velocity component v. The columns correspond to different methods (inset text). The color intensity corresponds to the magnitude of the residual, $|\widetilde{r}(\boldsymbol{k})|$.

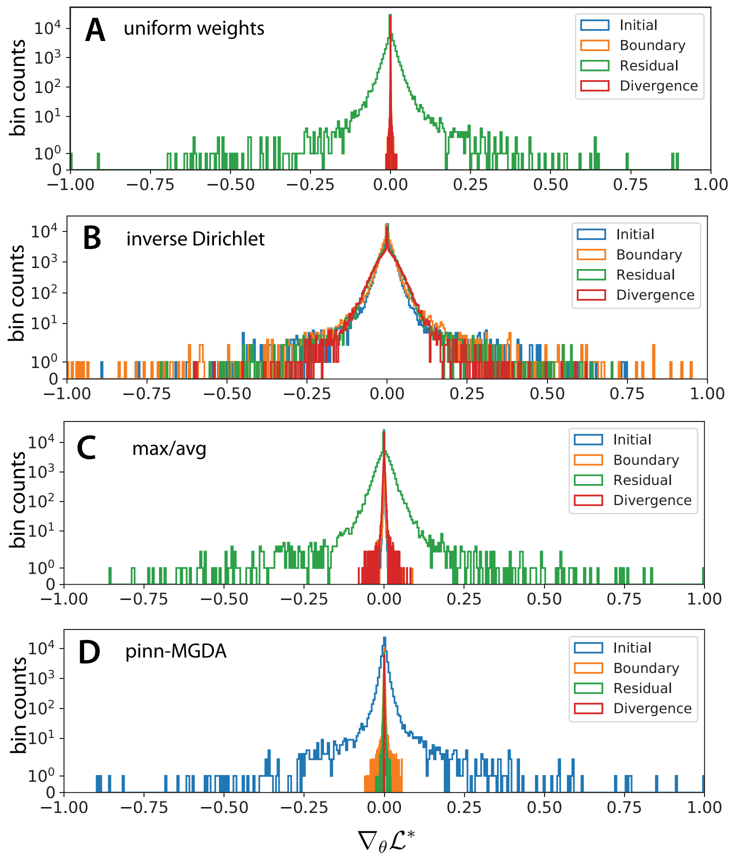

Figure C.5. Loss gradient histograms illustrating vanishing task-specific gradients for the active turbulence problem. The weighted gradient distributions of the different objectives (insets on the right) at training epoch 5000 for the forward problem in a squared domain. The panels correspond to different λ weighting schemes as named.

used for fitting the measurement data (pre-training, $\lambda_1 = 1, \lambda_2 = \lambda_3 = 0$), the next 1000 epochs training with an additional constraint on the divergence ($\lambda_1 \neq 0, \lambda_2 \neq 0, \lambda_3 = 0$), and finally followed by the introduction of the equation residual ($\lambda_1 \neq 0, \lambda_2 \neq 0, \lambda_3 \neq 0$). At every checkpoint introducing a new task,

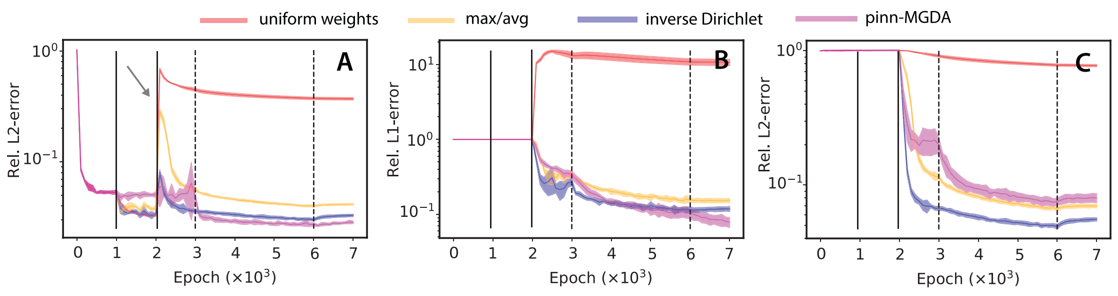

Figure C.6. Catastrophic forgetting in sequential PINN training of active turbulence from noisy measurement data. (A) Test error in the velocity field $\boldsymbol{u}$ during the three-step sequential training described in the main text with different weighting strategies (colors). The solid vertical lines at epochs 1000 and 2000 mark the introduction of the divergence constraint and PDE residual, respectively. The dashed vertical lines mark epochs at which the learning rate $\eta(\tau)$ is adjusted. (B) Average relative L1 errors in the inferred values for the coefficients ξ_0 (ξ_1 is absorbed into the effective pressure), α, β, Γ_0, and Γ_2 of equation (11) over training epochs. (C) Average relative L2 error for inferring the latent pressure gradient ∇p across training epochs. The simulated data $\boldsymbol{u}$ is corrupted with additive Gaussian noise $\hat{\boldsymbol{u}} = \boldsymbol{u} + 0.25\,\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \psi^2)$ is a vector of element-wise independent and identically distributed Gaussian random numbers with mean zero and variance $\psi^2 = \mathrm{Var}\{u_1, \ldots, u_N\}$.

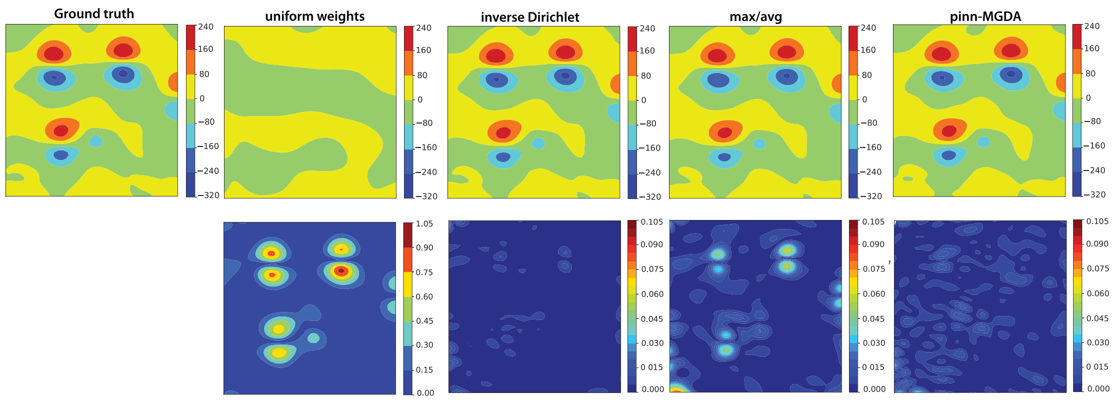

Figure C.7. Inferred effective pressure. The top row shows the prediction of the horizontal component of the effective pressure gradient $\partial_x p^*$ given the velocity field $\boldsymbol{u}$. The bottom row shows the corresponding point-wise relative error of the prediction. Different columns correspond to different weighting strategies as per the titles.

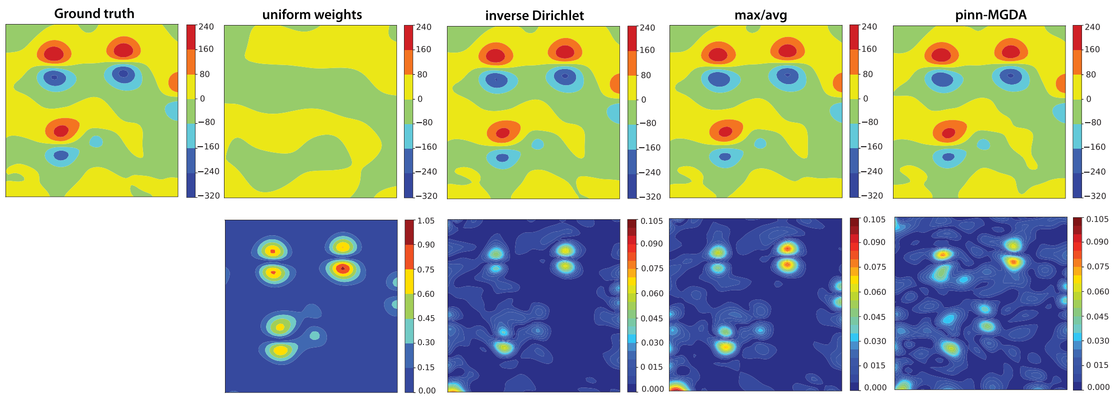

Figure C.8. Inferred effective pressure from noisy measurement data. The top row shows the prediction of the horizontal component of the effective pressure gradient $\partial_x p^*$ given the noisy corrupted velocity field $\hat{\boldsymbol{u}} = \boldsymbol{u} + 0.25\,\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \psi^2)$ is a vector of element-wise independent and identically distributed Gaussian random numbers with mean zero and variance $\psi^2 = \mathrm{Var}\{u_1, \ldots, u_N\}$. The bottom row shows the corresponding point-wise relative error of the prediction. Different columns correspond to different weighting strategies as per the titles.

we reset all λ_k for the computation of $\hat{\lambda}_k$ (see equation (7) main text) of the max/avg variant to $\lambda_k = 1$. For both, forward and inverse problems, we choose an initial learning rate $\eta = 10^{-3}$ and decrease it by a factor of 10 after 3000 and 6000 epochs (see figure C.3).

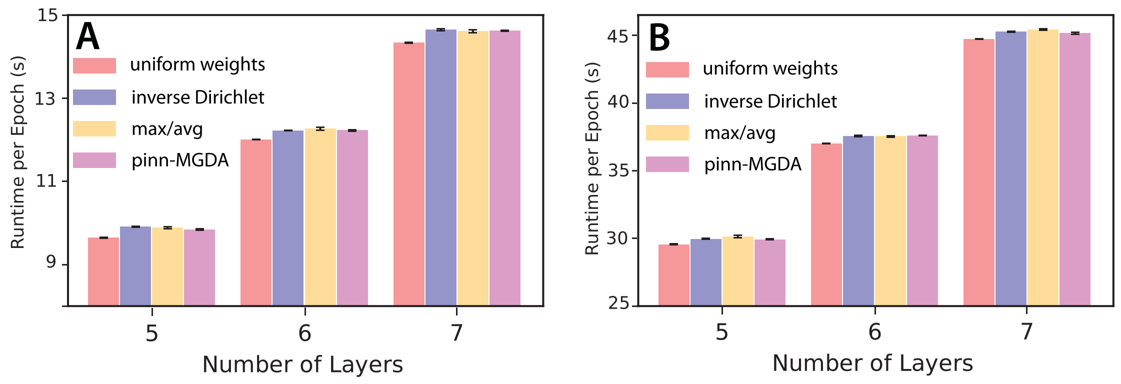

Figure C.9. Computational aspects of dynamic weighing for the active turbulence problem. (A) Training times per epoch for the inverse problem in primitive variables with 35 mini-batch updates per epoch of batch size $|B| = 4096$. (B) Training times per epoch for the forward problem in vorticity-streamline formulation with 49 mini-batch updates per epoch of batch size $|B| = 4000$. The higher computational cost in (B) stems from using more training points and from the additional derivative that need to be computed for evaluating the vorticity. The profiling was done on a Nvidia GTX 1080 graphics card with 8 GB VRAM.

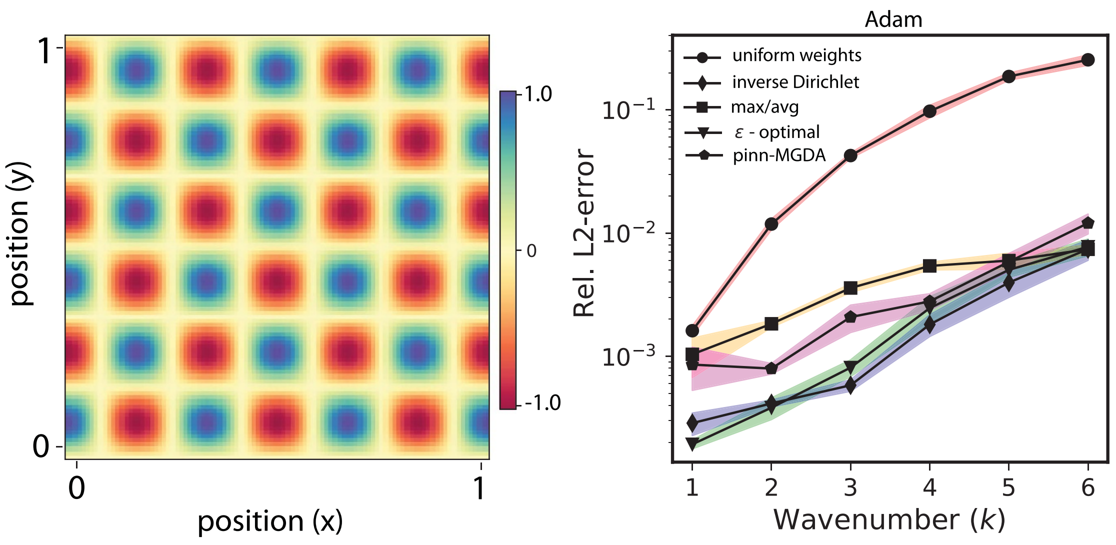

Figure D.1. Comparison between methods for solving the Poisson equation. Left: solution of the Poisson equation (equation (D1)) for the wavenumber $k = 6$. Right: comparison between relative L2 errors for different weighing strategies for increasing wavenumbers k using the Adam optimizer.

Appendix D. 2D poisson problem

We consider the problem of solving the 2D Poisson equation given as,

$$\Delta u = -2\omega^2 \cos(\omega x)\sin(\omega y), \tag{D1}$$

with the parameter ω controlling the frequency of the oscillatory solution $u(x, y)$. The analytical solution $u(x, y) = \cos(\omega x)\sin(\omega y)$ is computed on the domain $[0, 1] \times [0, 1]$ with a grid resolution of 100×100. The boundary condition is given as $\mathcal{B}(x, y) = \cos(\omega x)\sin(\omega y)$, $x, y \in \partial\Omega$. We set up the PINN with two objective functions one for handling boundary conditions and the other to compute the residual with the loss

$$\mathcal{L}(u, \theta) = \lambda_1 \mathcal{L}_{\text{res}} + \lambda_2 \mathcal{L}_{\text{boundary}}. \tag{D2}$$

We set $N_{\mathcal{B}} = 400$ sampled uniformly on the four boundaries, and randomly sample $N_{\text{int}} = 2500$ points from the interior of the domain. We choose a network $f_\theta : (x, y) \to (u)$ with five layers and 50 nodes per layer and the tanh activation function is used. The neural network is trained for 30 000 epochs using the Adam optimizer. We choose the initial learning rate $\eta = 10^{-3}$ and decrease it by a factor of 10 after 10 000, 20 000 and 30 000 epochs. In figure D.1 we show the relative L2 error for different weighting strategies across changing wavenumber ω. We notice a very similar trend as observed in the previous problems, with *inverse Dirichlet* scheme performing in par with the analytical ε optimal weighting strategy, followed by max/avg

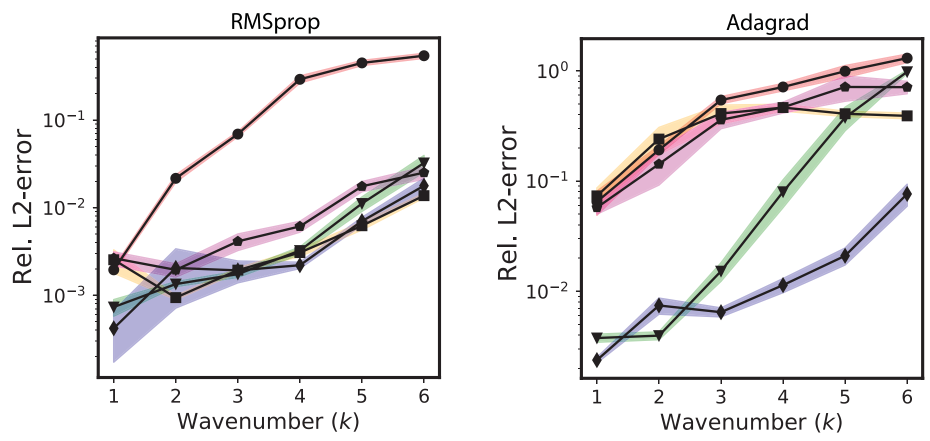

Figure D.2. Effect of the choice of optimizer on the performance of weighting strategies. Left: comparison between relative L2 errors for different weighing strategies for increasing wavenumbers k using the RMSprop optimizer. Right: the same comparison when using the Adagrad optimizer. The line markers and colors follow the legend of figure D.1.

and pinn-MGDA. Using the Poisson example, we also compare the effect of the choice of the optimizer on the performance of weighting strategies. In figure D.2, we show the performance of weighting strategies using the RMSprop optimizer (left) and Adagrad (right). We find that the inverse Dirichlet strategy outperforms the other strategies in both cases. Especially for Adagrad, we find our weighting strategy to be the most accurate by an order of magnitude. Interestingly, static ε-optimal weights fail to capture the solution for high wavenumbers when using Adagrad (see figure D.2 right).

Appendix E. Curvature-driven level set

The level-set method [52] is a popular method used for implicit representation of surfaces. In contrast to explicit methods for interface capturing, level-set methods offer a natural means to handle complex changes in the interface topology. For this reason, level-set methods have found extensive applications in computer vision [53], multiphase and computational fluid dynamics [54], and for solving PDEs on surfaces [55]. For this example, we consider the curvature driven flow problem that evolves the topology of a complex interface based on its local curvature, and can be tailored for image processing tasks like image smoothing and enhancement [53, 56]. Specifically, we look at the problem of a wound spiral which relaxes under its own curvature [52]. The parametrization of the initial spiral is given by $\theta = 2\,\pi D\sqrt{s}$ with $s = (k+a)/(np+a)$. The point locations of the zero contour can be computed as,

$$x_s = x_c + m\left(D/(1+D)\sqrt{s}\cos(\theta)\right),$$
$$y_s = y_c + m\left(D/(1+D)\sqrt{s}\sin(\theta)\right).$$

The zero level set function can then be computed using the distance function $d(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{x}_p\|$ as $\phi(\boldsymbol{x}, t=0) = d(\boldsymbol{x}) - w$ with w being the width of the spirals. The distance function $d(\boldsymbol{x})$ measures the shortest distance between two points $\boldsymbol{x}$ and $\boldsymbol{x}_p$. Given the level set function $\phi(\boldsymbol{x})$, we can compute the 2D curvature κ as,

$$\kappa = \frac{\phi_{xx}\phi_y\phi_y - 2\phi_{xy}\phi_x\phi_y + \phi_{yy}\phi_x\phi_x}{\left(\phi_x^2 + \phi_y^2\right)^{3/2}}. \tag{E1}$$

The evolution of the level set function driven by the local curvature κ and is computed using the PDE,

$$\frac{\partial \phi}{\partial t} = \kappa|\nabla\phi|.$$

The simulation was conducted on an 128×128 resolution spatial grid covering the domain $[-2, 2] \times [-2, 2]$ using the finite-difference method with necessary reinitialization to preserve the sign distance property of the level-set function ϕ. The spiral parameters are chosen to be $np = 400, D = 2.5, a = 3, m = 2$ [57]. Time integration is performed using a second-order Runge-Kutta scheme with a step size of $dt = 0.001$.

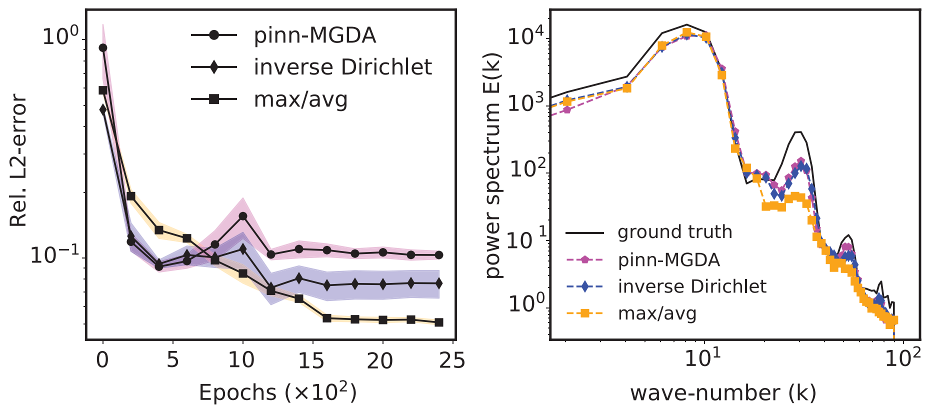

Figure E.1. Curvature-driven level-set problem. (A) Relative L2 error for for different methods for the curvature-driven level-set flow. (B) The Power spectrum of the PINN level-set function approximation after 2000 epochs compared with the Power spectrum of the ground truth, shown as solid black line.

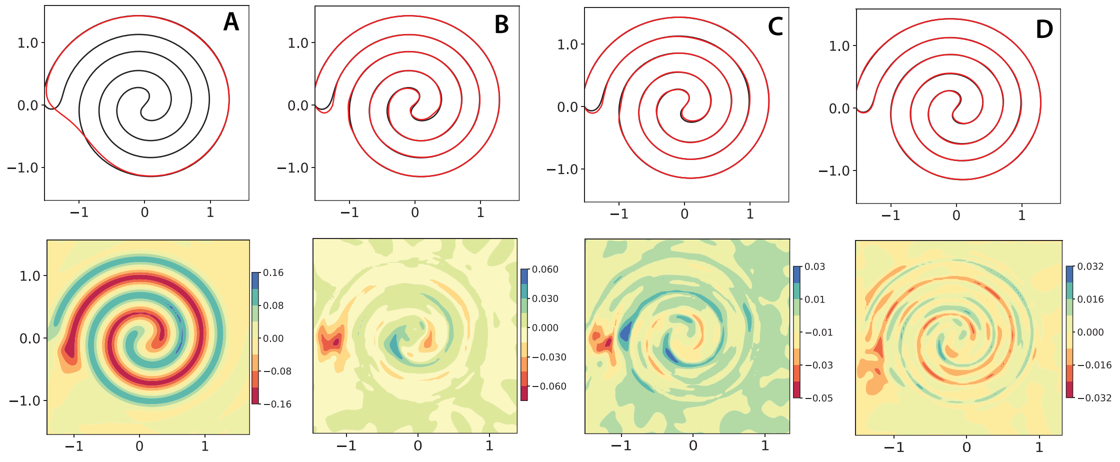

Figure E.2. Solution snapshot of curvature-driven level-set flow. Top row: zero contour of the level-set function visualized at simulation time $t = 0.01$ after 2000 epochs of training. Different columns correspond to different weighing strategies: (A) uniform weighing, (B) pinn-MGDA, (C) inverse Dirichlet, (D) max/avg. Bottom row: The corresponding point-wise error of the PINN approximation of the level-set function in comparison to the ground truth solution.

E.1. Training setup

The training data is generated from uniformly sampling in the domain $[-1.5, 1.5] \times [-1.5, 1.5]$ with a resolution of 100×100 in space and 80 time snapshots separated with time-step $dt = 0.001$. This lead to $N_\mathrm{I} = 10\,000, N_\mathrm{B} = 32\,000$ and $N_\mathrm{int} = 768\,000$ points in the interior of the space and time domain. The network is trained for 2500 epochs using the Adam optimizer with each epoch consisting of 187 mini-batch updates with batch size $|\mathrm{B}| = 4096$. The neural network $f_\theta : (x, y, t) \to (\phi)$ with 5 layers and 64 nodes is chosen for this problem. We found that using the ELU activation function resulted in the best performance. The total loss is given as,

$$\mathcal{L}(\phi) = \lambda_1\, \mathcal{L}_\mathrm{boundary} + \lambda_2\, \mathcal{L}_\mathrm{res} + \lambda_3\, \mathcal{L}_\mathrm{init}. \tag{E2}$$

We choose the initial learning rate as $\eta = 10^{-3}$ and decrease it after 1500 and 2000 epochs by a factor of 10.

Appendix F. Network initialization and parameterization

Let $\mathcal{X} \in \mathbb{R}^{N \times d_\mathrm{inp}}$ be the set of all input coordinates to train the neural network, where N denotes the total number of data points and d_inp the dimension of the input data, i.e. $d_\mathrm{inp} = 3$ corresponding to (x, y, t). Moreover, $\boldsymbol{X}_\mathrm{B} \in \mathbb{R}^{|\mathrm{B}| \times d_\mathrm{inp}}$ will denote a single batch, by randomly subsampling $\mathcal{X}$ into $N/|\mathrm{B}|$ subsets. We then define an n-layer deep neural network with parameters $\boldsymbol{\theta}$ as a composite function $f_{\boldsymbol{\theta}} : \mathbb{R}^{|\mathrm{B}| \times d_\mathrm{inp}} \to \mathbb{R}^{|\mathrm{B}| \times d_\mathrm{out}}$, where d_out corresponds to the dimension of the output variable. The output is usually computed as

$$f_{\boldsymbol{\theta}}(\boldsymbol{X}_\mathrm{B}) = T_n(\sigma(T_{n-1}(\sigma(\ldots\sigma(T_1(\boldsymbol{X}_\mathrm{B})))))), \tag{F1}$$

with the affine linear maps $T_i(\boldsymbol{z}_{i-1}) = \boldsymbol{z}_{i-1}\boldsymbol{W}_i^T + \boldsymbol{b}_i$, $i = 1, \ldots, n$, $\boldsymbol{z}_0 = \boldsymbol{X}_{\mathrm{B}}$, and output $\boldsymbol{Y}_{\mathrm{B}} = \boldsymbol{z}_{n-1}\boldsymbol{W}_n^T + \boldsymbol{b}_n$. Here, $\boldsymbol{W}_i \in \mathbb{R}^{q_i \times q_{i-1}}$ describes the weights of the incoming edges of layer i with q_i neurons, $\boldsymbol{b}_i \in \mathbb{R}^{q_i}$ contains a scalar bias for each neuron and $\sigma(\cdot)$ is a non-linear activation function. The network parameters $\boldsymbol{W}_i, \boldsymbol{b}_i \subset \boldsymbol{\theta}$ are initialized using normal Xavier initialization [39] with $\boldsymbol{W}_i \sim \mathcal{N}(0, \Psi^2)$, where $\mathcal{N}$ is a Gaussian distribution with zero mean and standard deviation

$$\Psi = g \cdot \sqrt{\frac{2}{q_i + q_{i-1}}}, \tag{F2}$$

and $\boldsymbol{b}_i = \boldsymbol{0}$. The gain g was chosen to be $g = 1$ for all activation functions except for tanh with a recommended gain of $g = \frac{5}{3}$. Before feeding $\boldsymbol{X}_{\mathrm{B}}$ into the first layer of the neural network, we normalize it with respect to the training data $\mathcal{X}$ as

$$\hat{\boldsymbol{X}}_{\mathrm{B}} = \frac{\boldsymbol{X}_{\mathrm{B}} - \boldsymbol{\mu}}{\boldsymbol{\psi}}, \tag{F3}$$

where $\boldsymbol{\mu} = ((\overline{\mathcal{X}_1}, \ldots, \overline{\mathcal{X}_{d_{\mathrm{inp}}}}))$ and $\boldsymbol{\psi} = ((\sqrt{\mathrm{Var}\{\mathcal{X}_1\}}, \ldots, \sqrt{\mathrm{Var}\{\mathcal{X}_{d_{\mathrm{inp}}}\}}))$ are the column-wise mean and standard deviations of the training data $\mathcal{X}$.

F.1. Weights computation

We initialize all weights to $\lambda_k = 1, k = 1, \ldots, K$. For all experiments we update the dynamic weights at the first batch of every fifth epoch. The moving average is then performed with $\alpha = 0.5$. The mini-batches $\boldsymbol{X}_{\mathrm{B}}$ are randomized after each epoch, such that for every update of the weights, λ_k will be computed with respect to a different subset $\boldsymbol{X}_{\mathrm{B}}$ of all training points $\mathcal{X}$. Furthermore, batching is only performed over the interior points N_{int}. As $N_{\mathrm{I}}, N_{\mathrm{B}} \ll N_{\mathrm{int}}$, the points for initial and boundary conditions do not require batching and remain fixed throughout the training.

ORCID iD

Ivo F Sbalzarini https://orcid.org/0000-0003-4414-4340

References

[1] He K, Zhang X, Ren S and Sun J 2016 Deep residual learning for image recognition *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* pp 770–8
[2] van den Oord A, Dieleman S, Zen H, Simonyan K, Vinyals O, Graves A, Kalchbrenner N, Senior A and Kavukcuoglu K 2016 WaveNet: a generative model for raw audio (arXiv:1609.03499)
[3] Silver D *et al* 2016 Mastering the game of go with deep neural networks and tree search *Nature* **529** 484
[4] Sirignano J and Spiliopoulos K 2018 Dgm: a deep learning algorithm for solving partial differential equations *J. Comput. Phys.* **375** 1339
[5] Raissi M, Perdikaris P and Karniadakis G E 2019 Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations *J. Comput. Phys.* **378** 686
[6] Gühring I, Kutyniok G and Petersen P 2020 Error bounds for approximations with deep ReLU neural networks in $W^{s,p}$ norms *Anal. Appl.* **18** 803
[7] Czarnecki W M, Osindero S, Jaderberg M, Swirszcz G and Pascanu R 2017 Sobolev training for neural networks *Advances in Neural Information Processing Systems* pp 4278–87 (arXiv:1706.04859)
[8] Raissi M, Wang Z, Triantafyllou M S and Karniadakis G E 2019 Deep learning of vortex-induced vibrations *J. Fluid Mech.* **861** 119
[9] Fang Z and Zhan J 2019 Deep physical informed neural networks for metamaterial design *IEEE Access* **8** 24506
[10] Liu D and Wang Y 2019 Multi-fidelity physics-constrained neural network and its application in materials modeling *J. Mech. Des.* **141** 121403
[11] Chen Y, Lu L, Karniadakis G E and Dal Negro L 2020 Physics-informed neural networks for inverse problems in nano-optics and metamaterials *Opt. Express* **28** 11618
[12] Raissi M, Yazdani A and Karniadakis G E 2020 Hidden fluid mechanics: learning velocity and pressure fields from flow visualizations *Science* **367** 1026
[13] Sun L, Gao H, Pan S and Wang J-X 2020 Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data *Comput. Methods Appl. Mech. Eng.* **361** 112732
[14] Mao Z, Jagtap A D and Karniadakis G E 2020 Physics-informed neural networks for high-speed flows *Comput. Methods Appl. Mech. Eng.* **360** 112789
[15] Raissi M, Babaee H and Givi P 2019 Deep learning of turbulent scalar mixing *Phys. Rev. Fluids* **4** 124501
[16] Jin X, Cai S, Li H and Karniadakis G E 2020 NSFnets (Navier-Stokesflow nets): physics-informed neural networks for the incompressible Navier-Stokes equations *J. Comput. Phys.* **426** 109951
[17] Yazdani A, Lu L, Raissi M and Karniadakis G E 2020 Systems biology informed deep learning for inferring parameters and hidden dynamics *PLoS Computat. Biol.* **16** e1007575
[18] Sahli Costabal F, Yang Y, Perdikaris P, Hurtado D E and Kuhl E 2020 Physics-informed neural networks for cardiac activation mapping *Front. Phys.* **8** 42

[19] Kissas G, Yang Y, Hwuang E, Witschey W R, Detre J A and Perdikaris P 2020 Machine learning in cardiovascular flows modeling: predicting arterial blood pressure from non-invasive 4D flowMRI data using physics-informed neural networks *Comput. Methods Appl. Mech. Eng.* **358** 112623
[20] Zheng Q, Zeng L and Karniadakis G E 2020 Physics-informed semantic inpainting: application to geostatistical modeling *J. Comput. Phys.* **419** 109676
[21] Rao C, Sun H and Liu Y 2021 Physics-informed deep learning for computational elastodynamics without labeled data *J. Eng. Mech.* **147** 04021043
[22] Yang L, Meng X and Karniadakis G E 2020 B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data *J. Comput. Phys.* **425** 109913
[23] Zhang D, Guo L and Karniadakis G E 2020 Learning in modal space: solving time-dependent stochastic PDEs using physics-informed neural networks *SIAM J. Sci. Comput.* **42** A639
[24] Han J, Jentzen A and Weinan E 2018 Solving high-dimensional partial differential equations using deep learning *Proc. Natl Acad. Sci.* **115** 8505
[25] Pang G, Lu L and Karniadakis G E 2019 fPINNs: fractional physics-informed neural networks *SIAM J. Sci. Comput.* **41** A2603
[26] Rahaman N, Baratin A, Arpit D, Draxler F, Lin M, Hamprecht F, Bengio Y and Courville A 2019 On the spectral bias of neural networks *Int. Conf. on Machine Learning* (PMLR) pp 5301–10 (arXiv:1806.08734v3)
[27] Xu Z-Q J, Zhang Y and Xiao Y 2019 Training behavior of deep neural network in frequency domain *Int. Conf. on Neural Information Processing* (Springer) pp 264–74
[28] Xu Z-Q J, Zhang Y, Luo T, Xiao Y and Ma Z 2019 Frequency principle: Fourier analysis sheds light on deep neural networks (arXiv:1901.06523)
[29] Xu Z J 2018 Understanding training and generalization in deep learning by Fourier analysis (arXiv:1808.04295)
[30] Wang S, Yu X and Perdikaris P 2021 When and why PINNs fail to train: a neural tangent kernel perspective *J. Comput. Phys.* **449** 110768
[31] Kendall A, Gal Y and Cipolla R 2018 Multi-task learning using uncertainty to weigh losses for scene geometry and semantics *IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)* pp 7482–91
[32] Chen Z, Badrinarayanan V, Lee C-Y and Rabinovich A 2018 Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks *Int. Conf. on Machine Learning* (PMLR) pp 794–803 (arXiv:1711.02257v4)
[33] Sener O and Koltun V 2018 Multi-task learning as multi-objective optimization *Adv. Neural Inf. Process. Syst.* **31** 525–36
[34] Wang S, Teng Y and Perdikaris P 2020 Understanding and mitigating gradient pathologies in physics-informed neural networks (arXiv:2001.04536)
[35] Rohrhofer F M, Posch S and Geiger B C 2021 On the Pareto front of physics-informed neural networks (arXiv:2105.00862)
[36] Zitzler E and Thiele L 1999 Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach *IEEE Trans. Evolutionary Computat.* **3** 257
[37] Boyd S, Boyd S P and Vandenberghe L 2004 *Convex Optimization* (Cambridge: Cambridge University Press) (https://doi.org/10.1017/CBO9780511804441)
[38] Kingma D P and Ba J 2015 Adam: a method for stochastic optimization *3rd Int. Conf. on Learning Representations, ICLR 2015 (San Diego, CA, USA, 7-9 May 2015) Conf. Track Proc.* ed Y Bengio and Y LeCun (arXiv:1412.6980)
[39] Glorot X and Bengio Y 2010 Understanding the difficulty of training deep feedforward neural networks *Proc. of the Thirteenth Int. Conf. on Artificial Intelligence and Statistics (JMLR Workshop Conf. Proc.)* pp 249–56
[40] Cox S M and Matthews P C 2002 Exponential time differencing for stiff systems *J. Comput. Phys.* **176** 430
[41] Désidéri J-A 2012 Multiple-gradient descent algorithm (MGDA) for multiobjective optimization *C. R. Math.* **350** 313
[42] van der Meer R, Oosterlee C and Borovykh A 2020 Optimally weighted loss functions for solving PDEs with neural networks (arXiv:2002.06269)
[43] Wensink H H, Dunkel J, Heidenreich S, Drescher K, Goldstein R E, Löwen H and Yeomans J M 2012 Meso-scale turbulence in living fluids *Proc. Natl Acad. Sci.* **109** 14308
[44] Dunkel J, Heidenreich S, Drescher K, Wensink H H, Bär M and Goldstein R E 2013 Fluid dynamics of bacterial turbulence *Phys. Rev. Lett.* **110** 228102
[45] Ramaswamy R and Jülicher F 2016 Activity induces traveling waves, vortices and spatiotemporal chaos in a model actomyosin layer *Sci. Rep.* **6** 1
[46] Kirkpatrick J *et al* 2017 Overcoming catastrophic forgetting in neural networks *Proc. Natl Acad. Sci.* **114** 3521
[47] Buchholz T-O, Prakash M, Schmidt D, Krull A and Jug F 2020 DenoiSeg: joint denoising and segmentation *Conf. on Computer Vision* (Springer) pp 324–37
[48] Both G-J, Choudhury S, Sens P and Kusters R 2021 Deepmod: deep learning for model discovery in noisy data *J. Comput. Phys.* **428** 109985
[49] Mandt S, Hoffman M D and Blei D M 2017 Stochastic gradient descent as approximate Bayesian inference *J. Mach. Learn. Res.* **18** 1
[50] Hardy G and Titchmarsh E 1931 A note on Parseval's theorem for Fourier transforms *J. London Math. Soc.* **1** 44
[51] Kassam A-K and Trefethen L N 2005 Fourth-order time-stepping for stiff PDEs *SIAM J. Sci. Comput.* **26** 1214
[52] Osher S and Fedkiw R 2006 *Level Set Methods and Dynamic Implicit Surfaces* vol 153 (Berlin: Springer Science & Business Media) (https://doi.org/10.1007/b98879)
[53] Malladi R and Sethian J A 1995 Image processing via level set curvature flow *Proc. Natl Acad. Sci.* **92** 7046
[54] Zhao H-K, Chan T, Merriman B and Osher S 1996 A variational level set approach to multiphase motion *J. Comput. Phys.* **127** 179
[55] Bergdorf M, Sbalzarini I F and Koumoutsakos P 2010 A Lagrangian particle method for reaction-diffusion systems on deforming surfaces *J. Math. Biol.* **61** 649
[56] Alvarez L, Guichard F, Lions P-L and Morel J-M 1993 Axioms and fundamental equations of image processing *Arch. Ration. Mech. Anal.* **123** 199
[57] Alamé K, Anantharamu S and Mahesh K 2020 A variational level set methodology without reinitialization for the prediction of equilibrium interfaces over arbitrary solid surfaces *J. Comput. Phys.* **406** 109184