Meta-learning PINN loss functions

Apostolos F Psaros[a], Kenji Kawaguchi[b], George Em Karniadakis[a,*]

[a]*Division of Applied Mathematics, Brown University, Providence, RI 02906, USA*
[b]*Center of Mathematical Sciences and Applications, Harvard University, Cambridge, MA 02138, USA*

Abstract

We propose a meta-learning technique for offline discovery of physics-informed neural network (PINN) loss functions. We extend earlier works on meta-learning, and develop a gradient-based meta-learning algorithm for addressing diverse task distributions based on parametrized partial differential equations (PDEs) that are solved with PINNs. Furthermore, based on new theory we identify two desirable properties of meta-learned losses in PINN problems, which we enforce by proposing a new regularization method or using a specific parametrization of the loss function. In the computational examples, the meta-learned losses are employed at test time for addressing regression and PDE task distributions. Our results indicate that significant performance improvement can be achieved by using a shared-among-tasks offline-learned loss function even for out-of-distribution meta-testing. In this case, we solve for test tasks that do not belong to the task distribution used in meta-training, and we also employ PINN architectures that are different from the PINN architecture used in meta-training. To better understand the capabilities and limitations of the proposed method, we consider various parametrizations of the loss function and describe different algorithm design options and how they may affect meta-learning performance.

Keywords: physics-informed neural networks, meta-learning, meta-learned loss function

1. Introduction

1.1. Related work and motivation

The physics-informed neural network (PINN) is a recently proposed method for solving forward and inverse problems involving partial differential equations (PDEs); see, e.g., [1–10] for different versions of PINNs. PINNs are based on (a) constructing a neural network (NN) approximator for the PDE solution that is inserted via automatic differentiation in the nonlinear operators describing the PDE, and (b) learning the solution by minimizing a composite objective function comprised of the residual terms for PDEs and boundary and initial conditions in the strong form; other PINN types in variational (weak) form have also been developed in [6].

Similar to solving supervised learning tasks with NNs, optimally solving PDEs with PINNs requires selecting the architecture, the optimizer, the learning rate schedule, and other hyperparameters (considered as such in a broad sense), each of which plays a different role in training. Moreover, optimally enforcing the physics-based constraints, e.g., by controlling the number and locations of residual points for each term in the composite objective function introduces additional PINN-specific hyperparameters

*Corresponding Author
Email address: george_karniadakis@brown.edu (George Em Karniadakis)

to be selected. Overall, partly because of the above and despite the conceptual simplicity of PINNs, the resulting learning problem is theoretically and practically challenging; see, e.g., [11].

In general, the loss function in NN training interacts with the optimization algorithm and affects both the convergence rate and the performance of the obtained minimum. In addition, the loss function interacts with the NN for shaping the loss landscape; i.e., the training objective as a function of the trainable NN parameters. As a result, from this point of view, deciding whether to use mean squared error (MSE), mean absolute error (MAE) or a different loss function can be considered as an additional hyperparameter to be selected; trial and error is often employed in practice, with MSE being the most popular option for PINNs. For facilitating automatic selection, a parametrized loss function has been proposed in [12], which includes standard losses as special cases and can be optimized online for improving performance. Although such an adaptive loss is shown in [12] to be highly effective in the computer vision problems considered therein, it increases training computational cost and does not benefit from prior knowledge regarding the particular problem being solved. The following question, therefore, arises in the context of PINNs: *Can we develop a framework for encoding the underlying physics of a parametrized PDE in a loss function optimized offline?*

Meta-learning is an emerging field that aims to optimize various parts of learning by infusing prior knowledge of a given task distribution such that new tasks drawn from the distribution can be solved faster and more accurately; see [13] for a comprehensive review. One form of meta-learning, namely gradient-based, alternates between an inner optimization in which dependence of updates on meta-learned parameters is tracked, and an outer optimization in which meta-learned parameters are updated based on differentiating the inner optimization paths. Such differentiation can be performed either exactly or approximately for reducing computational cost; see, e.g., [14–17].

In this regard, a recent research direction is concerned with loss function meta-learning, with diverse applications in supervised and reinforcement learning [18–31]. Although different works utilize different meta-learning techniques and have different goals, it has been shown that loss functions obtained via meta-learning can lead to an improved convergence of the gradient-descent-based optimization. Moreover, meta-learned loss functions can improve test performance under few-shot and semi-supervised conditions as well as cases involving a mismatch between train and test distributions, or between train loss functions and test evaluation metrics (e.g., because of non-differentiability of the latter). For simplicity, we refer to meta-learned loss functions as *learned losses* in this paper, while loss functions optimized during training are referred to as *online adaptive losses*.

1.2. Overview of the proposed method

In this work, we propose a method for offline discovery via meta-learning of PINN loss functions by utilizing information from task distributions defined based on parametrized PDEs. In the learned loss function, we encode information specific to the considered PDE task distribution by differentiating the PINN optimization path with respect to the loss function parametrization. Discovering loss functions by differentiating the physics-informed optimization path can enhance our understanding about the complex problem of solving PDEs with PINNs.

Following the PDE task distribution definition and the learned loss parametrization, the learned loss parameters are optimized via meta-training by repeating the following steps until a stopping criterion is

met: (a) PDE tasks are drawn from the task distribution; (b) in the inner optimization part, they are solved with PINNs for a few iterations using the current learned loss, and the gradient of the learned loss parameters is tracked throughout optimization; and (c) in the outer optimization part, the learned loss parameters are updated based on MSE of the final (optimized) PINN parameters. After meta-training, the obtained learned loss is used for meta-testing, which entails solving unseen tasks until convergence. A schematic illustration of the above alternating optimization procedure is provided in Fig. 1.

As we demonstrate in the computational examples of the present paper, the proposed loss function meta-learning technique can improve PINN performance significantly even compared with the online adaptive loss proposed in [12], while also allocating the loss function training computational cost to the offline phase.

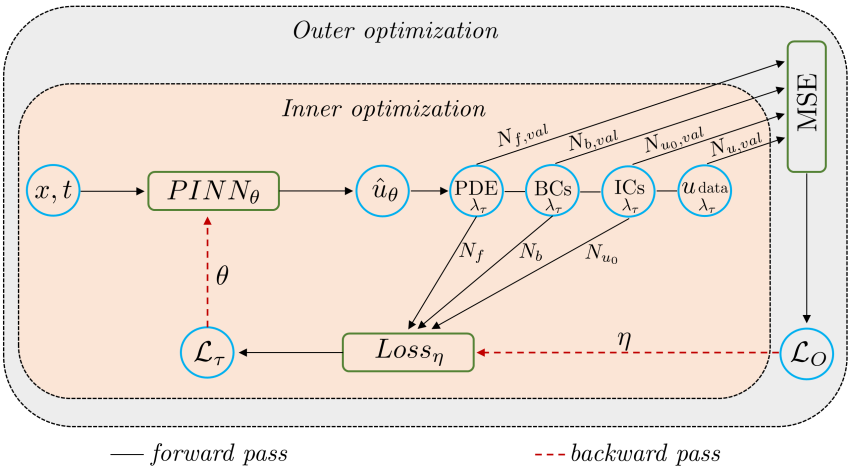


Fig. 1. Schematic illustration of the proposed meta-learning method for discovering PINN loss functions. In the inner optimization part, the PINN parameters θ are updated based on $\mathcal{L}_\tau$, i.e., the current learned loss evaluated on $\{N_f, N_b, N_{u_0}\}$ datapoints corresponding to task λ_τ; N_f, N_b, N_{u_0} correspond to number of points for evaluating the residuals for PDE, boundary conditions (BCs), and initial conditions (ICs), respectively. In the outer optimization part, the learned loss parameters η are updated based on $\mathcal{L}_O$, i.e., the MSE of the optimized PINN parameters evaluated on $\{N_{f,val}, N_{b,val}, N_{u_0,val}, N_{u,val}\}$ datapoints; $N_{f,val}, N_{b,val}, N_{u_0,val}, N_{u,val}$ correspond to number of points for evaluating the PDE, BCs, ICs, and solution data (if available) residuals, respectively.

1.3. Summary of innovative claims

- We propose a gradient-based meta-learning algorithm for offline discovery of PINN loss functions pertaining to diverse PDE task distributions.
- We extend the loss function meta-learning technique of [27] by considering alternative loss parametrizations and various algorithm design options.
- By proving two new theorems and a corollary, we identify two desirable properties of learned loss functions and propose a new regularization method to enforce them. We also prove that the loss function parametrization proposed in [12] is guaranteed to satisfy the two desirable properties.
- We define several representative benchmarks for demonstrating the performance of the considered algorithm design options as well as the applicability and the limitations of the proposed method.

1.4. Organization of the paper

We organize the paper as follows. In Section 2 we provide a brief overview of PINNs for solving and discovering PDEs as well as of standard NN training. In Section 3 we summarize meta-learning for PINNs, discuss our PINNs loss function meta-learning technique in detail, and present the theoretical results. In Section 4, we perform various computational experiments involving diverse PDE task distributions. Finally, we summarize our findings in Section 5, while the theorem proofs as well as additional design options and computational results are included in Appendices A-D.

2. Preliminaries

2.1. PINN solution technique overview

Consider a general problem defined as

$$\mathcal{F}_\lambda[u](t,x) = 0,\ (t,x) \in [0,T] \times \Omega \tag{1a}$$

$$\mathcal{B}_\lambda[u](t,x) = 0,\ (t,x) \in [0,T] \times \partial\Omega \tag{1b}$$

$$u(0,x) = u_{0,\lambda}(x),\ x \in \Omega, \tag{1c}$$

where $\Omega \subset \mathbb{R}^{D_x}$ is a bounded domain with boundary $\partial\Omega$, $T > 0$, and $u(t,x) \in \mathbb{R}^{D_u}$ denotes the solution at (t,x). Eq. (1a) is a PDE expressed with a nonlinear operator $\mathcal{F}_\lambda[\cdot]$ that contains identity and differential operators as well as source terms; Eq. (1b) represents the boundary conditions (BCs) expressed with an operator $\mathcal{B}_\lambda[\cdot]$, and Eq. (1c) represents the initial conditions (ICs) expressed with a function $u_{0,\lambda}$. In Eqs. (1a)-(1c), λ represents the parametrization of the problem and is considered as shared among the operators $\mathcal{F}_\lambda$, $\mathcal{B}_\lambda$ and the function $u_{0,\lambda}$. Furthermore, in practice Eqs. (1a)-(1c) are often given in the form of collected data; i.e., they are only specified on discrete sets of locations that are subsets of $[0,T] \times \Omega$, $[0,T] \times \partial\Omega$ and Ω, respectively.

In this regard, one problem scenario pertains to fixing the model parameters λ and aiming to obtain the solution $u(t,x)$ for every $(t,x) \in [0,T] \times \Omega$. This problem is henceforth referred to as solving the PDE or as forward problem. Another problem scenario pertains to having observations from the solution $u(t,x)$ and aiming to obtain the parameters λ that best describe the data. This problem is henceforth referred to as discovering the PDE or as inverse problem. PINNs were proposed in [1] for addressing both problem scenarios. In the case of a forward problem, the solution u is represented with a NN $\hat{u}$, which is trained such that a composite objective comprised of the strong-form (PDE), boundary and initial residuals corresponding to Eqs. (1a)-(1c), respectively, on a discrete set of points is minimized. For addressing the inverse problem with the PINNs solution technique of [1], an additional term corresponding to the solution u data misfit is added to the composite objective and $\hat{u}$ is trained simultaneously with λ.

Equivalently, we can view PINNs from a data-driven perspective. By defining $f(t,x)$ as the output of the left-hand side of Eq. (1) for an arbitrary function $u(t,x)$, i.e.,

$$f(t,x) := \mathcal{F}_\lambda[u](t,x), \tag{2}$$

the operator $\mathcal{F}_\lambda : u \mapsto f$ can be construed as a map from V_u, the set of all admissible functions u to V_f, the image of V_u under $\mathcal{F}_\lambda$. In this context, the function u that satisfies Eq. (1a) corresponds to a mapped

function $f \in V_f$ that is zero for every $(t, x) \in [0, T] \times \Omega$. As a result, satisfying Eq. (1a) is equivalent to having observations from this zero-outputting f at every $(t, x) \in [0, T] \times \Omega$ (or in a subset of it). Furthermore, satisfying Eq. (1b) is equivalent to having observations in $[0, T] \times \Omega$ from a zero-outputting b function defined as

$$b(t, x) := \mathcal{B}_\lambda[u](t, x) \tag{3}$$

and satisfying Eq. (1c) to having observations in $\{t = 0\} \times \Omega$ from the solution u. If Eqs. (1a)-(1c) are defined analytically for every point of the domains $[0, T] \times \Omega$, $[0, T] \times \partial\Omega$ and Ω, respectively, a finite dataset is considered in practice.

In this context, PINNs address the forward problem by (a) constructing three NN approximators $\hat{u}$, $\hat{f}$, and $\hat{b}$ that are connected via both the operators $\mathcal{F}_\lambda$ and $\mathcal{B}_\lambda$ and parameter sharing, i.e., $\hat{f}_{\theta,\lambda} = \mathcal{F}_\lambda[\hat{u}_\theta]$ and $\hat{b}_{\theta,\lambda} = \mathcal{B}_\lambda[\hat{u}_\theta]$, where θ are the shared parameters of the NN approximators, and (b) by training $\hat{u}$, $\hat{f}$, and $\hat{b}$ (simultaneously because of parameter sharing) to fit the complete dataset. For the inverse problem, in which observations of the solution u for times t other than zero are also available, the same three connected approximators $\hat{u}$, $\hat{f}$, and $\hat{b}$ are constructed, but the additional constraint of fitting the u observations is also included for learning λ too.

Overall, learning the approximators $\hat{u}$, $\hat{f}$, and $\hat{b}$ (and, potentially, λ too) takes the form of a minimization problem expressed as

$$\min_{\theta\ (,\ \lambda)} \mathcal{L}_f(\theta, \lambda) + \mathcal{L}_b(\theta, \lambda) + \mathcal{L}_{u_0}(\theta, \lambda) + \mathcal{L}_u(\theta), \tag{4}$$

where $\mathcal{L}_f$ represents the loss related to the f data, i.e., the physics of Eq. (1a), $\mathcal{L}_b(\theta, \lambda)$ and $\mathcal{L}_{u_0}(\theta, \lambda)$ the losses related to the BCs and ICs, respectively, and $\mathcal{L}_u$ the loss related to the u data. Next, considering $\{N_f, N_b, N_{u_0}, N_u\}$ datapoints, the terms $\{\mathcal{L}_f, \mathcal{L}_b, \mathcal{L}_{u_0}, \mathcal{L}_u\}$ in Eq. (4) expressed as the weighted average dataset errors become

$$\mathcal{L}_f = \frac{w_f}{N_f} \sum_{i=1}^{N_f} \ell(\hat{f}_{\theta,\lambda}(t_i, x_i), 0) \tag{5a}$$

$$\mathcal{L}_b = \frac{w_b}{N_b} \sum_{i=1}^{N_b} \ell(\hat{b}_{\theta,\lambda}(t_i, x_i), 0) \tag{5b}$$

$$\mathcal{L}_{u_0} = \frac{w_{u_0}}{N_{u_0}} \sum_{i=1}^{N_{u_0}} \ell(\hat{u}_\theta(0, x_i), u_{0,\lambda}(x_i)) \tag{5c}$$

$$\mathcal{L}_u = \frac{w_u}{N_u} \sum_{i=1}^{N_u} \ell(\hat{u}_\theta(t_i, x_i), u(t_i, x_i)). \tag{5d}$$

In Eq. (5), $\ell(prediction, target)$, with $\ell : D_u \times D_u \to \mathbb{R}_{\geq 0}$, is a loss function that takes as input the NN prediction at a domain point and the target value, and outputs the corresponding loss; $\hat{u}_\theta(t_i, x_i)$, $\hat{f}_{\theta,\lambda}(t_i, x_i)$, $\hat{b}_{\theta,\lambda}(t_i, x_i)$ denote the NN predictions $\hat{u}_\theta$ and $\hat{f}_{\theta,\lambda} = \mathcal{F}_\lambda[\hat{u}_\theta]$, $\hat{b}_{\theta,\lambda} = \mathcal{B}_\lambda[\hat{u}_\theta]$ evaluated at the i^{th} domain point (t_i, x_i), respectively; $u_{0,\lambda}(x_i)$ denotes i^{th} target value corresponding to the ICs function $u_{0,\lambda}$; and $u(t_i, x_i)$ denotes the i^{th} target value corresponding to solution u (if available). Clearly, the point sets $\{t_i, x_i\}_{i=1}^{N_f}$, $\{t_i, x_i\}_{i=1}^{N_b}$, $\{t_i, x_i\}_{i=1}^{N_{u_0}}$, and $\{t_i, x_i\}_{i=1}^{N_u}$ represent domain points at different locations, although the same symbol (t_i, x_i) has been used for all of them, for notation simplicity. Note that optimal weights $\{w_f, w_b, w_{u_0}, w_u\}$ to be used in Eq. (4) are not known a priori and are often set in practice as equal to

one or obtained via trial and error; see [11] for discussion and an adaptive method for addressing this issue.

Finally, by considering as ℓ the squared ℓ_2-norm of the discrepancy between predictions and targets, i.e., MSE if it is averaged over the dataset, Eq. (5) reduces to

$$\mathcal{L}_f = \frac{w_f}{N_f} \sum_{i=1}^{N_f} ||\hat{f}_{\theta,\lambda}(t_i, x_i)||_2^2 \tag{6a}$$

$$\mathcal{L}_b = \frac{w_b}{N_b} \sum_{i=1}^{N_b} ||\hat{b}_{\theta,\lambda}(t_i, x_i)||_2^2 \tag{6b}$$

$$\mathcal{L}_{u_0} = \frac{w_{u_0}}{N_{u_0}} \sum_{i=1}^{N_{u_0}} ||\hat{u}_\theta(0, x_i) - u_{0,\lambda}(x_i)||_2^2 \tag{6c}$$

$$\mathcal{L}_u = \frac{w_u}{N_u} \sum_{i=1}^{N_u} ||\hat{u}_\theta(t_i, x_i) - u(t_i, x_i)||_2^2. \tag{6d}$$

We note that the terms MSE and squared ℓ_2-norm of the discrepancy are used interchangeably in this paper depending on the context.

2.2. Loss functions in neural network training

This section serves as a brief summary of standard NN training and as an introduction to the meta-learning loss functions Section 3. As described in Section 2.1, addressing forward and inverse PDE problems with PINNs requires solving the minimization problem of Eq. (4) with a composite objective function comprised of the loss terms of Eq. (5). All of the functionals $\mathcal{L}_f(\theta, \lambda)$, $\mathcal{L}_b(\theta, \lambda)$, $\mathcal{L}_{u_0}(\theta, \lambda)$ and $\mathcal{L}_u(\theta)$ in Eq. (5) are given as the average discrepancies over f, b, u_0 and u data, respectively, and the total loss $\mathcal{L}$ is expressed as the weighted sum of the individual terms. Therefore, we can study in this section, without loss of generality, each part of the sum separately by only considering the objective function

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{u}_\theta(t_i, x_i), u(t_i, x_i)), \tag{7}$$

where $\hat{u}_\theta(t_i, x_i)$ denotes the prediction value for each (t_i, x_i) (i.e., either $\hat{f}_{\theta,\lambda}(t_i, x_i)$, $\hat{b}_{\theta,\lambda}(t_i, x_i)$, $\hat{u}_{\theta,\lambda}(0, x_i)$, or $\hat{u}_\theta(t_i, x_i)$ in Eq. (5)) and $u(t_i, x_i)$ denotes the target (i.e., either 0, $u_{0,\lambda}(x_i)$, or $u(t_i, x_i)$ in Eq. (5)). For each (t, x) in Eq. (1), $u(t, x)$ as well as $\mathcal{F}_\lambda[u](t, x)$ and $\mathcal{B}_\lambda[u](t, x)$ belong to $\mathbb{R}^{D_u}$; thus, $\hat{u}_\theta(t_i, x_i)$ and $u(t_i, x_i)$ in Eq. (7) are also considered D_u-dimensional. Furthermore, N represents the size of the corresponding dataset, i.e., N_f, N_b, N_{u_0}, or N_u.

An optimization technique using only first-order information of the objective function is the (stochastic) gradient descent, which is typically denoted as SGD, whether or not stochastic gradients are used, by considering mini-batches of the data in Eq. (7). Using SGD, the NN parameters θ are updated based on

$$\theta \leftarrow \theta - \epsilon \nabla_\theta \mathcal{L}, \tag{8}$$

where ϵ is the learning rate,

$$\nabla_\theta \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \theta} = \left[\frac{\partial \mathcal{L}}{\partial \theta_1}, \ldots, \frac{\partial \mathcal{L}}{\partial \theta_{D_\theta}} \right] \tag{9}$$

is the gradient of $\mathcal{L}$ with respect to θ, and D_θ is the number of NN parameters (weights and biases). Employing the chain rule for each $\frac{\partial \mathcal{L}}{\partial \theta_j}$, $j \in \{1, \ldots, D_\theta\}$, Eq. (9) becomes

$$\nabla_\theta \mathcal{L} = \left[\frac{1}{N} \sum_{i=1}^{N} \nabla_q \ell(q, u(t_i, x_i)) \Big|_{q=\hat{u}_\theta(t_i,x_i)} \right] J_{\hat{u}_\theta,\theta}, \tag{10}$$

where $\nabla_q \ell(q, u(t_i, x_i)) \in \mathbb{R}^{1 \times D_u}$ is the gradient of the scalar output $\ell(q, u(t_i, x_i))$ with respect to the prediction $q = \hat{u}_\theta(t_i, x_i)$. Furthermore, $J_{\hat{u}_\theta,\theta} \in \mathbb{R}^{D_u \times D_\theta}$ is the Jacobian matrix

$$J_{\hat{u}_\theta,\theta} = [\nabla_\theta(\hat{u}_{\theta,1}), \ldots, \nabla_\theta(\hat{u}_{\theta,D_u})]^\top \tag{11}$$

of the NN transformation $\theta \mapsto \hat{u}_\theta$. If $\hat{u}_\theta$ is one-dimensional, Eq. (10) reduces to

$$\nabla_\theta \mathcal{L} = \left[\frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell(q, u(t_i, x_i))}{\partial q} \Big|_{q=\hat{u}_\theta(t_i,x_i)} \right] \nabla_\theta \hat{u}_\theta \tag{12}$$

and if, in addition, ℓ is the squared ℓ_2-norm, to

$$\nabla_\theta \mathcal{L} = \left[\frac{1}{N} \sum_{i=1}^{N} 2(\hat{u}_\theta(t_i, x_i) - u(t_i, x_i)) \right] \nabla_\theta \hat{u}_\theta. \tag{13}$$

The term enclosed in brackets in Eqs. (10)-(13), and more specifically the loss function ℓ, controls how the objective function behaves for increasing discrepancies from the target. If $\hat{u}_\theta$ is multi-dimensional, the loss function through $\nabla_q \ell(q, u(t_i, x_i))|_{q=\hat{u}_\theta(t_i,x_i)}$ also controls how the discrepancy in each dimension affects the final gradient $\nabla_\theta \mathcal{L}$; note that each component of $\nabla_\theta \mathcal{L}$ is an inner product between the term inside brackets and $\frac{\partial \hat{u}_\theta}{\partial \theta_j}$, $j \in \{1, \ldots, D_\theta\}$. For instance, by using the squared ℓ_2-norm as ℓ, which is given as the sum of squared discrepancies across dimensions, all dimensions are treated uniformly; see Appendix A regarding how a parametrized loss function for multi-dimensional inputs can be constructed.

For other standard first-order optimization algorithms, such as AdaGrad and Adam, the parameter θ updates depend not only on the current iteration gradient $\nabla_\theta \mathcal{L}$ of Eqs. (10)-(13), but also on the θ updates history. For standard second-order algorithms, such as Newton's method and BFGS, θ updates depend not only on $\nabla_\theta \mathcal{L}$, but also on the Hessian or the approximate Hessian, respectively, of $\mathcal{L}$ with respect to θ.

3. Meta-learning loss functions for PINNs

3.1. Defining PDE task distributions

In this section, we consider only the forward problem scenario as defined in Eq. (1). As explained in Section 2.1, solving Eq. (1) with PINNs for a value of λ requires learning the PINN parameters θ by solving the minimization problem of Eq. (4). Overall, a NN is constructed, an optimization strategy is selected and the optimization algorithm is run until some convergence criterion is met.

However, all of the above steps, from defining the optimization problem to solving it, require the user to make certain design choices, which affect the overall training procedure and the final approximation accuracy. For example, they affect the convergence rate of the optimizer as well as the training and test error at the obtained minimum. For simplicity, all these aspects of training that depend on the selected

hyperparameters are henceforth collectively called *performance or efficiency* of the selected hyperparameters and of the training in general. For example, we may refer to a set of hyperparameters as being *more efficient* than another set.

Indicatively, the PINN architecture and activation function, the optimization algorithm, and the number of collocation points (points at which the PDE residual is evaluated) correspond to hyperparameters that must be tuned/selected a priori or be optimized in an online manner (see, e.g., [3] for adaptive activation functions). In this regard, it is standard practice to experiment with many different hyperparameter settings by performing a few iterations of the optimization algorithm and by evaluating performance based on validation error; i.e., to perform hold-out validation. In the context of PINNs, the validation error can be computed over collocation points not used in training or testing. After performing this trial-and-error procedure, the problem can then be fully solved; i.e., the optimization is run until convergence.

For solving a novel, different PDE of the form of Eq. (1), hold-out validation is either repeated from scratch, or search is limited to a tight range of hyperparameter settings, depending on the experience of the user with the novel parameter λ in Eq. (1). Thus, it becomes clear that solving novel problems more efficiently requires:

(a) To define families of related PDEs such that hyperparameters selected for solving one member of the family are expected to perform well also for other members.

(b) To effectively utilize information acquired from solving a few representative members of the family in order to solve other members efficiently.

Families of related machine learning problems are typically called task distributions in the literature; see, e.g., [13]. For example, approximating the function $y = sin(x + \pi)$ is related to approximating $y = sin(x)$, in the sense that a hyperparameter setting found efficient for the former problem can be used as is, or with minimal modifications, for solving the latter one efficiently. Therefore, a task distribution can, indicatively, be defined by functions of the form of $y = sin(x + \lambda)$, with λ drawn uniformly from $[-\pi, \pi]$. More generally, it is assumed that tasks are parametrized by λ, which is drawn from some distribution $p(\lambda)$. In addition, each λ defines a learning task, which can be shown experimentally or theoretically that is related to other learning tasks drawn from $p(\lambda)$.

Note that although the discussion up to this point has been motivated by the burden of repetitive hyperparameter tuning even for related problems, meta-learning has a rich range of applications that goes well beyond selecting NN architectures, learning rates and activation functions. For example, optimizing the loss function or selecting a shared-among tasks NN initialization are not generally considered hyperparameter tuning. It is useful, however, to interpret all the different factors that affect the NN optimization procedure and the final performance as hyperparameters, some of which we try to optimize and some of which we fix.

3.2. Meta-learning as a bi-level minimization problem

As explained in Section 3.1, a task can be defined as a fixed PDE problem associated with a parameter λ that is drawn from a pre-specified task distribution $p(\lambda)$. Furthermore, in conjunction with task-specific training data, a task is solved with PINNs until convergence via Eqs. (4)-(6). Final accuracy or overall

performance of the training procedure can, indicatively, be evaluated based on final validation data error after training (e.g., PDE residual on a large set of points in the domain). Next, given a task distribution defined via $p(\lambda)$ and a set of optimizable hyperparameters η, meta-learning seeks for the optimal setting of η, where optimality is defined in terms of average performance across tasks drawn from $p(\lambda)$. For example, if performance of PINN training is evaluated based on the ℓ_2-norm of PDE residual on validation points, average performance of η refers to average across λ final ℓ_2 error.

Following [13] and considering a finite set $\Lambda = \{\lambda_\tau\}_{\tau=1}^T$ of $\lambda \sim p(\lambda)$ values, meta-learning is commonly formalized as a bi-level minimization problem expressed as

$$\min_{\eta} \mathcal{L}_O(\theta^*(\eta), \eta) \tag{14a}$$

$$\text{s.t. } \theta^*(\eta) = \{\theta^*_\tau(\eta)\}_{\tau=1}^T \tag{14b}$$

$$\text{with } \theta^*_\tau(\eta) = \arg\min_{\theta} \mathcal{L}_\tau(\theta, \eta), \tag{14c}$$

where Eq. (14a) is referred to as outer optimization, whereas Eqs. (14b)-(14c) as inner optimization. In Eq. (14), $\theta^*(\eta) = \{\theta^*_\tau(\eta)\}_{\tau=1}^T$ consists of the final NN parameters θ^* obtained after solving all tasks in the set Λ using hyperparameters η, and $\mathcal{L}_\tau(\theta, \eta)$, which is given by Eq. (4)-(6), is called inner- or base-objective. The subscript τ and the arguments η, θ, and θ^* in Eq. (14) are used to signify task-dependent quantities as well as dependencies on (optimizable) η and θ, and (optimum) $\theta^*_\tau(\eta)$, respectively. Furthermore, $\mathcal{L}_O(\theta^*(\eta), \eta)$, which is called outer- or meta-objective and pertains to the ultimate goal of meta-learning, is commonly considered to be the average performance of η. For example, if validation error is measured based on final ℓ_2-norm of PDE residual on a validation set of size $N_{f,val}$, the outer-objective $\mathcal{L}_O(\theta^*(\eta), \eta)$ can be expressed as

$$\mathcal{L}_O(\theta^*(\eta), \eta) = \mathbb{E}_{\lambda_\tau \in \Lambda} \left[\frac{1}{N_{f,val}} \sum_{i=1}^{N_{f,val}} ||\hat{f}_{\theta^*_\tau, \lambda_\tau}(t_i, x_i)||_2^2 \right], \tag{15}$$

where $N_{f,val}$ and domain points $\{(t_i, x_i)\}_{i=1}^{N_{f,val}}$ are considered the same for all tasks for notation simplicity and without loss of generality. For more design choices regarding outer-objective $\mathcal{L}_O$ see Section 3.3 and [13].

Next, for addressing the bi-level minimization problem of Eq. (14), an alternating approach between outer and inner optimization can be considered, which in practice can be achieved by performing one or only a few steps for each optimization. Three main families of techniques exist ([13]): gradient-based, reinforcement-learning-based, and evolutionary meta-learning. Discussions in this paper are limited to gradient-based approaches. Although inner optimization corresponds to standard PINN training of Eqs. (4)-(6), gradient-based outer optimization requires computing the total derivative

$$\boldsymbol{d}_\eta \mathcal{L}_O(\theta^*(\eta), \eta) = \nabla_\eta \mathcal{L}_O + [\nabla_{\theta^*} \mathcal{L}_O] \, J_{\theta^*(\eta), \eta}, \tag{16}$$

where $\boldsymbol{d}_\eta$ represents total derivative with respect to η, ∇_η and ∇_{θ^*} here represent partial derivatives with respect to η and $\theta^*(\eta)$, respectively, and $J_{\theta^*(\eta),\eta}$ is the Jacobian matrix of the transformation from η to $\theta^*(\eta)$. The first term on the right-hand side of Eq. (16) refers to direct dependence of $\mathcal{L}_O$ on η (e.g., as is the case for neural architecture search; see [32]), whereas the second term refers to dependence of $\mathcal{L}_O$ on η through the obtained optimal $\theta^*(\eta)$. Because $\theta^*(\eta)$ is the result of a number of inner optimization steps,

obtaining the Jacobian $J_{\theta^*(\eta),\eta}$ via chain rule is often referred to as *differentiating over the optimization path*. In this regard, see Section 3.4.2 for addressing the exploding gradients pathology arising because of path differentiation in loss function meta-learning as well as [15–17] on approximate ways of computing Eq. (16). Overall, the computation of Eq. (16) can be performed by designing for this purpose automatic differentiation algorithms; see [14] for more information and open-source code. Of course, for utilizing Eq. (16) the outer-objective $\mathcal{L}_O$ and the inner optimization steps must be differentiable. For instance, a single SGD step given as $\theta \leftarrow \theta - \epsilon\nabla_\theta\mathcal{L}(\theta,\eta)$, where ϵ is the learning rate, is differentiable with respect to η if $\mathcal{L}$ is also differentiable with respect to η, meaning that the Jacobian in Eq. (16) can be computed. The same holds for multiple SGD steps as well as for other optimization algorithms, such as AdaGrad and Adam.

Algorithm 1 is a general gradient-based meta-learning algorithm for arbitrary, admissible η. The input to the meta-learning algorithm includes the number of outer and inner iterations, I and J, respectively, and the outer and inner learning rates, ϵ_1, ϵ_2, respectively; see Appendix C for stopping criteria. Furthermore, the algorithm input includes the task distribution $p(\lambda)$ to be used for resampling during training and the number of tasks T. As shown in Algorithm 1, the task set Λ is not required to remain the same during optimization. Optimizing η using a meta-learning algorithm such as Algorithm 1 is typically called *meta-training*, and using the obtained η for solving unseen tasks from the task distribution is called *meta-testing*.

Algorithm 1: General gradient-based meta-learning algorithm for PINNs

input: ϵ_1, ϵ_2, I, T, J, and task distribution $p(\lambda)$
initialize η with $\eta^{(0)}$
for $i \in \{1,\dots,I\}$ **do** outer loop
  sample set Λ of T tasks from $p(\lambda)$
  for $\tau \in \{1,\dots,T\}$ **do** tasks
    initialize θ_τ with $\theta_\tau^{(0)}$
    for $j \in \{1,\dots,J\}$ **do** inner loop
      ▷ Inner step for each task
      $\theta_\tau^{(j)} = \theta_\tau^{(j-1)} - \epsilon_2\nabla_\theta\mathcal{L}_\tau(\theta,\eta)\Big|_{\theta=\theta_\tau^{(j-1)},\ \eta=\eta^{(i-1)}}$ ▷ PINN step
      ▷ $\mathcal{L}_\tau$ given by Eq. (17)
    end
    set $\theta_\tau^{*(i)} = \theta_\tau^{(J)}$
  end
  $\eta^{(i)} = \eta^{(i-1)} - \epsilon_1 \boldsymbol{d}_\eta\mathcal{L}_O(\theta^*(\eta),\eta)\Big|_{\theta^*=\{\theta_\tau^{*(i)}\}_{\tau=1}^T,\ \eta=\eta^{(i-1)}}$ ▷ Outer step
  meta learning
  ▷ $\boldsymbol{d}_\eta\mathcal{L}_O$ given by Eqs. (16) and (18)
end
return $\eta^{(I)}$

3.3. An algorithm for meta-learning PINN loss functions

Meta-learning PINN loss functions by utilizing the concepts of Section 3.2 requires defining an admissible hyperparameter η that can be used in conjunction with Algorithm 1. In this regard, a parametrization η can be used for the loss function ℓ of Eq. (5) for which $\mathcal{L}_O$ and the inner optimization steps are differentiable. Indicatively, ℓ can be represented with a feed-forward NN (FFN) and thus η represents the weights and biases of the FFN. Such a parametrization in conjunction with Algorithm 1 has been proposed in [27] for supervised learning and reinforcement learning problems. Alternatively, ℓ can be the adaptive loss function proposed in [12] (see Eqs. (19) and (21)) and thus η can be the robustness and scale parameters. More discussion and options regarding loss function parametrization can be found in Section 3.4.

Following parametrization, the PINN objective function for each task τ is the same as in standard PINNs training and given as

$$\mathcal{L}_\tau(\theta, \eta) = \mathcal{L}_f(\theta, \lambda_\tau) + \mathcal{L}_b(\theta, \lambda_\tau) + \mathcal{L}_{u_0}(\theta, \lambda_\tau), \tag{17}$$

with the terms $\{\mathcal{L}_f, \mathcal{L}_b, \mathcal{L}_{u_0}\}$ given from Eq. (5) and evaluated on the training datasets of sizes $\{N_f, N_b, N_{u_0}\}$ by using the parametrized loss function ℓ_η instead of a fixed ℓ. The objective function $\mathcal{L}_\tau(\theta, \eta)$ of Eq. (17) is optimized with respect to θ in the inner optimization step of Algorithm 1, while also tracking the optimization path dependence on η (see [14]). Next, the outer-objective $\mathcal{L}_O$, which is used for optimizing the learned loss ℓ_η, can be defined as the MSE on validation data, i.e.,

$$\mathcal{L}_O(\theta^*(\eta), \eta) = \mathbb{E}_{\lambda_\tau \in \Lambda} \left[\mathcal{L}_f(\theta^*_\tau, \lambda_\tau) + \mathcal{L}_b(\theta^*_\tau, \lambda_\tau) + \mathcal{L}_{u_0}(\theta^*_\tau, \lambda_\tau) \right], \tag{18}$$

with the terms $\{\mathcal{L}_f, \mathcal{L}_b, \mathcal{L}_{u_0}\}$ given from Eq. (5) and evaluated on the validation datasets of sizes $\{N_{f,val}, N_{b,val}, N_{u_0,val}, N_{u,val}\}$ by using the task parameters λ_τ and the optimal task-specific parameters θ^*_τ for every τ. Clearly, Eq. (18) has the same form as Eq. (17), except for the fact that (a) the number of validation points $\{N_{f,val}, N_{b,val}, N_{u_0,val}, N_{u,val}\}$ can be different from $\{N_f, N_b, N_{u_0}, N_u\}$, (b) MSE is used in Eq. (18) as a loss function, whereas ℓ_η is used in Eq. (17), and (c) $\mathcal{L}_O$ is an average loss across tasks, whereas $\mathcal{L}_\tau$ is task-specific. Optimizing ℓ_η utilizing Algorithm 1 with $\mathcal{L}_O$ defined based on Eq. (18) aims to answer the following question: *Is there a loss function ℓ_η which if used for PINN training (Eq. (17)) will perform better in terms of average across tasks MSE error (Eq. (18))?*

Note that in Eq. (18) additional data not used for inner training can also be utilized. For example, we may have solution data u_τ corresponding to λ_τ values sampled from the task distribution $p(\lambda)$ (e.g., produced by traditional numerical solvers or measurements), which can be used in the outer optimization step. By utilizing such additional data, the aforementioned question that loss function meta-learning aims to answer is augmented with the following: *Is there a loss function ℓ_η that, in addition, leads to better solution u performance error (Eq. (18)), although training has been performed based only on PDE residual and BCs/ICs data (Eq. (17))?* Finally, see Section 3.4.3 for more information regarding imposing additional properties to the loss function ℓ_η through penalties in Eq. (18).

3.4. Algorithm design and theory

3.4.1. Loss function parametrization and initialization

3.4.1.1 Adaptive loss function. A loss function parametrization that can be used in conjunction with Algorithm 1 has been proposed in [12]. In this regard, the one-dimensional loss function is

parametrized by the shape parameter $\alpha \in \mathbb{R}$, which controls robustness to outliers (see Section 2.2) and the scale parameter $c > 0$ that controls the size of the quadratic bowl near zero. Specifically, the loss function is expressed as

$$\rho_{\alpha,c}(d) = \frac{|\alpha - 2|}{\alpha} \left(\left(\frac{(d/c)^2}{|\alpha - 2|} + 1 \right)^{\alpha/2} - 1 \right), \tag{19}$$

where d denotes the discrepancy between each dimension of the prediction and the target; e.g., $d = \hat{u}_{\theta,j}(t,x) - u(t,x)$ or $d = \hat{f}_{\theta,\lambda,j}(t,x)$ for each $j \in \{1, \ldots, D_u\}$ in PINNs. For fixed values of α, Eq. (19) yields known losses; see [12] and Table A.1. An extension to multi-dimensional inputs can be achieved via Eqs (A.2)-(A.3).

Nevertheless, the loss function of Eq. (19) cannot be used directly as an adaptive loss function to be optimized in the online manner (i.e., simultaneously with NN parameters) proposed in [12]. Specifically, $\rho_{\alpha,c}(d)$ in Eq. (19) is monotonic with respect to α, and thus attempting to optimize α by minimizing Eq. (19) trivially sets α to be as small as possible. To address this issue, [12] defined also the corresponding probability density function, i.e.,

$$p_{\alpha,c}(d) = \frac{1}{cZ(\alpha)} \exp(-\rho_{\alpha,c}(d)), \tag{20}$$

which is valid only for $\alpha \geq 0$ as $Z(\alpha)$ is divergent for $\alpha < 0$. Furthermore, [12] defined a loss function based on the negative log likelihood of $p_{\alpha,c}(d)$, i.e.,

$$\hat{\ell}_{\alpha,c}(d) = \log(c) + \log(Z(\alpha)) + \rho_{\alpha,c}(d), \tag{21}$$

which is simply a shifted version of Eq. (19). Because the partition function $Z(\alpha)$ is difficult to evaluate and differentiate, $\log(Z(\alpha))$ is approximated with a cubic Hermite spline, which induces an added computational cost.

The loss function of Eq. (21) has been used in [12] in conjunction with Eqs. (A.2)-(A.3) as an adaptive loss function that is optimized online. Specifically, either the same pair (α, c) is used for each dimension, i.e., Eq. (A.2) is employed with unit weights and $\eta = \{\alpha, c\}$ or a different pair is used, i.e., Eq. (A.2) with $\eta = \{\alpha_1, c_1, \ldots, \alpha_{D_u}, c_{D_u}\}$ is employed. Regarding implementation of the constraints $\alpha \geq 0$ and $c > 0$, the parameters α and c for every dimension (if applicable) can be expressed as

$$\begin{aligned} \alpha &= (\alpha_{max} - \alpha_{min})\ sigmoid(\hat{\alpha}) + \alpha_{min} \\ c &= softplus(\hat{c}) + c_{min} \end{aligned} \tag{22}$$

and $\hat{\alpha}$, $\hat{c}$ are optimized simultaneously with the NN parameters. In Eq. (22), the sigmoid function limits α in the range $[\alpha_{min}, \alpha_{max}]$, whereas the softplus function constrains c to being greater than c_{min}; e.g., $c_{min} = 10^{-8}$ for avoiding degenerate optima.

Note that Algorithm 1 uses the outer objective $\mathcal{L}_O$ of Eq. (18) for optimizing the loss function, which is different from the inner objectives $\mathcal{L}_\tau$ of Eq. (17). As a result, using Eq. (19) as a loss function parametrization in Algorithm 1 does not lead to trivial α solutions as in [12]. Thus, either Eq. (19) or Eq. (21) can be considered as loss function parametrizations for Algorithm 1.

In Section 3.4.3, we prove that the loss function of Eq. (19) automatically satisfies the specific conditions required for successful training according to our new theorems, without adding any regularization terms in meta-training. In the present paper, the loss function parametrization of this section is referred

to as LAL when used as a meta-learning parametrization, and as OAL when used as an online adaptive loss. For initialization, we consider the values $\alpha = 2.01$ and $c = 1/\sqrt{2}$, which approximate the squared ℓ_2-norm.

3.4.1.2 NN-parametrized loss function. An alternative parametrization based on FFN has been proposed in [27]. Specifically, for the one-dimensional supervised learning regression problem considered in [27], the most expressive representation $\ell_\eta = \hat{\ell}_\eta(\hat{u}_\theta, u)$ of Fig. A.1 has been utilized. In terms of parametrizing $\hat{\ell}_\eta$, 2 hidden layers with 40 neurons each without biases and with ReLU activations functions have been used, while the output is also passed through a softplus activation function for producing the final loss output. Finally, the NN parameters are initialized using the Xavier uniform initializer.

In this regard, we note that ensuring positivity of the loss function does not affect NN parameter optimization; i.e., the softplus output activation function affects the results of [27] only through its nonlinearity and not by dictating positive loss outputs. Furthermore, instead of randomly initializing NN parameters η, one can alternatively initialize them so that ℓ_η approximates a known loss function such as the squared ℓ_2-norm. For obtaining such an initialization, it suffices to perform even a few Adam iterations with synthetic data obtained by computing the ℓ_2-norm of randomly sampled values in the considered domain; see computational examples in Section 4 for more information.

3.4.1.3 Meta-learning the composite objective function weights. Finally, the composite objective function weights $\{w_f, w_b, w_{u_0}\}$, corresponding to the PDE residual, BCs and ICs loss terms, respectively, in Eqs. (5)-(6), can also be included in the meta-learned parameters η. As a result, three different loss functions are learned that are equivalent up to a scaling factor; see computational examples in Section 4 for experiments. For restricting $\{w_f, w_b, w_{u_0}\}$ to values greater than zero or a minimum value, the softplus activation function can be used similarly to the c parameter in Eq. (22).

3.4.2. Inner optimization steps

As discussed in Sections 3.2-3.3, the gradient $\nabla_\eta \mathcal{L}_O$ required to update η in Algorithm 1 is obtained through inner optimization path differentiation, i.e., via Eq. (16). For each outer iteration i, the obtained NN parameters $\theta_\tau^{*(i)}$ for each task τ following J inner SGD steps with hyperparameters $\eta^{(i)}$ are given as

$$\theta_\tau^{*(i)} = \theta_\tau^{(J)} = \theta_\tau^{(0)} - \epsilon_2 \sum_{j=1}^{J} \nabla_\theta \mathcal{L}_\tau(\theta, \eta)\Big|_{\theta=\theta_\tau^{(j-1)},\ \eta=\eta^{(i-1)}}, \tag{23}$$

where $\mathcal{L}_\tau$ is given by Eq. (17). As a result, the Jacobian $J_{\theta^*(\eta),\eta}$ in Eq. (16), which is the average over tasks of the derivative of Eq. (23) with respect to η, is given as a sum of J gradient terms, i.e.,

$$J_{\theta^*(\eta),\eta} = \mathbb{E}_{\lambda_\tau \in \Lambda}\left[-\epsilon_2 \sum_{j=1}^{J} \nabla_\eta \left(\nabla_\theta \mathcal{L}_\tau(\theta, \eta)\Big|_{\theta=\theta_\tau^{(j-1)},\ \eta=\eta^{(i-1)}} \right)^\top \right]. \tag{24}$$

Clearly, using a large number of inner steps J enables η optimization to take into account larger parts of the θ optimization history, which can be construed as enriching the meta-learning dataset. In this regard, the increased computational cost associated with large J values can be addressed by considering approximations of Eq. (24); see, e.g., [15] for an introduction. However, if the η gradients for subsequent j values point to similar directions, the summation of Eq. (24) can lead to large Jacobian values. In

turn, large Jacobian values in Eq. (24) lead to large $\boldsymbol{d}_\eta \mathcal{L}_O(\theta^*(\eta), \eta)$ values in Eq. (16), which make the optimization of η unstable. For the computational example of Section 4.1 we demonstrate in Appendix D the effect of the number of inner steps as well as the exploding gradients pathology with an experiment.

Finally, to address this exploding η gradients issue, we have tested (a) dividing $\boldsymbol{d}_\eta \mathcal{L}_O(\theta^*(\eta), \eta)$ values by the number of inner iterations J, (b) normalizing the gradient by its norm, and (c) performing gradient clipping, i.e., setting a cap for the gradient norm. The fact that the norm of the η gradient does not explode in every outer iteration makes dividing by J too strict, while normalizing by the gradient norm deprives the η gradient of its capability to provide also an update magnitude apart from a direction. For these reasons, gradient clipping is expected to be a better option. This is corroborated by the experiments of Section 4.

3.4.3. Theoretical derivation of desirable loss function properties

In general, meta-learning ℓ_η via Algorithm 1 in conjunction with Eqs. (17)-(18) aims to maximize meta-testing performance by considering an expressive ℓ_η, which is learned during meta-training. However, because the loss function plays a central role in optimization as explained in Section 2.2, it would be important for the learned loss ℓ_η to satisfy certain conditions to allow efficient gradient-based training. In this section, we theoretically identify *the optimal stationarity condition* and *the MSE relation condition* as the two desirable conditions that enable efficient training in our problems. Moreover, we propose a novel regularization method to impose the conditions. The LAL parametrization of Section 3.4.1.1 is then proven to satisfy the two conditions without any regularization.

The results presented in this section are general and pertain to regression problems as well. For this reason, we consider the more general than a PINN scenario of having a training dataset $\{(x_i, u_i)\}_{i=1}^N$ of N samples, where the pair (x_i, u_i) with $x_i \in \mathcal{X} \subseteq \mathbb{R}^{D_x}$ and $u_i \in \mathcal{U} \subseteq \mathbb{R}^{D_u}$, for $i \in \{1, \ldots, N\}$, corresponds to the i-th (input, target) pair. For learning a NN approximator $\hat{u}_\theta$, we minimize the objective

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{u}_\theta(x_i), u_i), \tag{25}$$

over $\theta \in \mathbb{R}^{D_\theta}$, where $\ell : \mathbb{R}^{D_u} \times \mathcal{U} \to \mathbb{R}_{\geq 0}$ is the selected or learned loss function. Note that in the following, the function $\hat{u}_\theta$ is allowed to represent a wide range of network architectures, including ones with batch normalization, convolutions, and skip connections. In this section, we assume that the map $\hat{u}_i : \theta \mapsto \hat{u}_\theta(x_i)$ is differentiable for every $i \in \{1, \ldots, N\}$.

We define the output vector for all N data points by

$$\hat{u}_X(\theta) = \mathrm{vec}((\hat{u}_\theta(x_1), \ldots, \hat{u}_\theta(x_N))) \in \mathbb{R}^{ND_u}, \tag{26}$$

and let $\{\theta^{(r)}\}_{r=0}^\infty$ be the optimization sequence defined by

$$\theta^{(r+1)} = \theta^{(r)} - \epsilon^{(r)} \bar{g}^{(r)}, \tag{27}$$

with an initial parameter vector $\theta^{(0)}$, a learning rate $\epsilon^{(r)}$, and an update vector $\bar{g}^{(r)}$. One of the theorems in this section relies on the following assumption on the update vector $\bar{g}^{(r)}$:

Assumption 1. *There exist $\bar{c}, \underline{c} > 0$ such that $\underline{c} \|\nabla_\theta \mathcal{L}(\theta^{(r)})\|^2 \leq \nabla_\theta \mathcal{L}(\theta^{(r)})^\top \bar{g}^{(r)}$ and $\|\bar{g}^{(r)}\|^2 \leq \bar{c} \|\nabla_\theta \mathcal{L}(\theta^{(r)})\|^2$ for any $r \geq 0$.*

It is noted that Assumption 1 is satisfied by using $\bar{g}^{(r)} = D^{(r)}\nabla_\theta \mathcal{L}(\theta^{(r)})$, where $D^{(r)}$ is any positive definite symmetric matrix with eigenvalues in the interval $[\underline{c}, \sqrt{\bar{c}}]$. Setting $D^{(r)} = I$ corresponds to SGD and Assumption 1 is satisfied with $\underline{c} = \bar{c} = 1$. Next, we define *the optimal stationarity condition* as well as *the MSE relation condition* and provide the main Theorems 1-2 and Corollary 1 of this section; corresponding proofs can be found in Appendix B.

Definition 1. The learned loss ℓ is said to satisfy *the optimal stationarity condition* if the following holds: for all $u \in \mathcal{U}$ and $q \in \mathbb{R}^{D_u}$, $\nabla_q \ell(q, u)$ exists and $\nabla_q \ell(q, u) = 0$ implies that $\ell(q, u) \leq \ell(q', u)$ for all $q' \in \mathbb{R}^{D_u}$.

Theorem 1. *If the learned loss ℓ satisfies the optimal stationarity condition, then any stationary point θ of $\mathcal{L}$ is a global minimum of $\mathcal{L}$ when* $\operatorname{rank}(\frac{\partial \hat{u}_X(\theta)}{\partial \theta}) = ND_u$. *If the learned loss ℓ does not satisfy the optimal stationarity condition by having a point $q \in \mathbb{R}^{D_u}$ such that $\nabla_q \ell(q, u) = 0$ and $\ell(q, u) > \ell(q', u)$ for some $q' \in \mathbb{R}^{D_u}$, then there exists a stationary point θ of $\mathcal{L}$ that is not a global minimum of $\mathcal{L}$ when* $\{\hat{u}_X(\theta) \in \mathbb{R}^{ND_u} : \theta \in \mathbb{R}^{D_\theta}\} = \mathbb{R}^{ND_u}$.

Theorem 2. *Suppose Assumption 1 holds. Assume that the learned loss ℓ satisfies the optimal stationarity condition, $\|\nabla_\theta \mathcal{L}(\theta) - \nabla_\theta \mathcal{L}(\theta')\| \leq L\|\theta - \theta'\|$ for all θ, θ' in the domain of $\mathcal{L}$ for some $L \geq 0$, and the learning rate sequence $\{\epsilon^{(r)}\}_r$ satisfies either (i) $\zeta \leq \epsilon^{(r)} \leq \frac{\underline{c}(2-\zeta)}{L\bar{c}}$ for some $\zeta > 0$, or (ii)* $\lim_{r\to\infty} \epsilon^{(r)} = 0$ *and $\sum_{r=0}^{\infty} \epsilon^{(r)} = \infty$. Then, for any limit point θ of the sequence $\{\theta^{(r)}\}_r$, the limit point θ is a global minimum of $\mathcal{L}$ if* $\operatorname{rank}(\frac{\partial \hat{u}_X(\theta)}{\partial \theta}) = ND_u$.

Definition 2. The learned loss ℓ is said to satisfy *the MSE relation condition* if the following holds: for all $u \in \mathcal{U}$ and $q \in \mathbb{R}^{D_u}$, $\nabla_q \ell(q, u) = 0$ if and only if $q = u$.

Corollary 1. *If the learned loss ℓ satisfies the optimal stationarity condition and the MSE relation condition, then any stationary point θ of $\mathcal{L}$ is a global minimum of the MSE loss $\mathcal{L}_{\mathrm{MSE}}$ when* $\operatorname{rank}(\frac{\partial \hat{u}_X(\theta)}{\partial \theta}) = ND_u$, *where $\mathcal{L}_{\mathrm{MSE}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \|\hat{u}_\theta(x_i) - u_i\|_2^2$.*

For example, the rank condition $\operatorname{rank}(\frac{\partial \hat{u}_X(\theta)}{\partial \theta}) = ND_u$ (as well as the expressivity condition of $\{\hat{u}_X(\theta) \in \mathbb{R}^{ND_u} : \theta \in \mathbb{R}^{D_\theta}\} = \mathbb{R}^{ND_u}$) is guaranteed to be satisfied by using wide NNs (e.g., see 33–35). Nevertheless, the rank condition $\operatorname{rank}(\frac{\partial \hat{u}_X(\theta)}{\partial \theta}) = ND_u$ is more general than the condition of using wide NNs, in the sense that the latter implies the former but not vice versa. Moreover, the standard loss functions used in PINNs, such as squared ℓ_2-norm, satisfy the differentiability condition of Definition 1, and are convex with respect to q: i.e., $\ell_u : q \mapsto \ell(q, u)$ is convex for all $u \in \mathcal{U}$. It is known that for any differentiable convex function $\ell_u : \mathbb{R}^{D_u} \to \mathbb{R}$, we have $\ell_u(q') \geq \ell_u(q) + \nabla_q \ell_u(q)^\top (q' - q)$ for all $q, q' \in \mathbb{R}^{D_u}$. Because the latter implies the optimal stationarity condition, we conclude that standard loss functions typically used in PINNs satisfy the optimal stationarity condition.

Unlike standard loss functions, the flexibility provided by meta-learning the loss function allows the learned loss $\ell_u : q \mapsto \ell(q, u)$ to be non-convex in q. This flexibility allows the learned loss to be well tailored to the task distribution considered, while we can still check or impose the optimal stationarity condition; see also Section 4. Corollary 1 shows that for our PINN problems with the MSE measure, it is desirable for the learned loss to satisfy the optimal stationarity condition and the MSE relation condition. The MSE relation condition imposes the existence of the desired stationary point related

to MSE, whereas the optimal stationarity condition ensures the global optimality. Since the optimal stationarity condition does not guarantee the existence of a stationary point, it is possible that there is no stationary point without the MSE relation condition or this type of an additional condition. As a pathological example, we may have $\ell_\eta(q,u) = q - u$, for which there is no stationary point and Theorems 1–2 vacuously hold true. Therefore, depending on the measures used in applications (i.e., MSE for our case), it is beneficial to impose this type of an additional condition along with the optimal stationarity condition in order to impose an existence of a desirable stationary point.

Using this theoretical result, we now propose a novel penalty term $\mathcal{L}_{O,add}$ to be added to the outer objective $\mathcal{L}_O$ of Eq. (28) in order to penalize the deviation of the learned loss from the conditions in Corollary 1. More concretely, the novel penalty term $\mathcal{L}_{O,add}$ based on Corollary 1 is expressed as

$$\mathcal{L}_{O,add}(\eta) = \mathbb{E}_q[\|\nabla_q \ell_\eta(q,q)\|_2^2] + \mathbb{E}_{q\neq q'}[\max(0, c - \|\nabla_q \ell_\eta(q,q')\|_2^2)], \tag{28}$$

where $q, q' \in \mathbb{R}^{D_u}$ are arbitrary inputs to the loss function, and c is a hyperparameter that can be set equal to a small value (e.g., 10^{-2}). The first term on the right-hand side of Eq. (28) promotes the first-order derivative of the learned loss to be zero for zero discrepancy, for obtaining a learned loss that satisfies the MSE relation condition. Furthermore, in the second term on the right-hand side of Eq. (28), the additional penalty term maximizes the derivative away from zero up to a constant c, for obtaining a learned loss that satisfies the optimal stationarity condition. The terms $\mathbb{E}_q[\|\nabla_q \ell_\eta(q,q)\|_2^2]$ and $\mathbb{E}_{q\neq q'}[\max(0, c-\|\nabla_q \ell_\eta(q,q')\|_2^2)]$ can, in practice, be computed by drawing some q and a random pair (q,q') such that $q \neq q'$, in each outer iteration, and by computing $\|\nabla_q \ell_\eta(q,q)\|_2^2$ and $\max(0, c - \|\nabla_q \ell_\eta(q,q')\|_2^2)$, respectively. Alternatively, we can define an empirical distribution on q and on (q,q'), and replace the expectations by summations over finite points. By following the same rationale and by augmenting the outer objective $\mathcal{L}_O$ of Eq. (28), other problem-specific constraints can be imposed to the loss function as well.

Finally, we prove that a learned loss with the LAL parametrization of Section 3.4.1.1 automatically satisfies the optimal stationarity condition and the MSE relation condition without adding the regularization term:

Proposition 1. *Any LAL loss of the form* $\ell(q,u) = \rho_{\alpha,c}(q-u) = \frac{|\alpha-2|}{\alpha}((\frac{((q-u)/c)^2}{|\alpha-2|}+1)^{\alpha/2}-1)$ *satisfies the optimal stationarity condition and the MSE relation condition if* $c > 0$, $\alpha \neq 0$, *and* $\alpha \neq 2$.

4. Computational examples

We consider four computational examples in order to demonstrate the applicability and the performance of Algorithm 1 for meta-learning PINN loss functions. In Section 4.1, we address the problem of discontinuous function approximation with varying frequencies. Because the function approximation problem is conceptually simpler than solving PDEs and computationally cheaper, Section 4.1 serves not only as a pedagogical example but also as a guide for understanding the behavior of Algorithm 1 when different loss function parametrizations and initializations, inner optimizers and other design options are used; see Appendices C and D. In Section 4.2 we address the problem of solving the advection equation with varying initial conditions and discontinuous solutions, in Section 4.3 we solve a steady-state version

of the reaction-diffusion equation with varying source term, and in Section 4.4 we solve the Burgers equation with varying viscosity in two regimes.

For each computational example, both FFN and LAL parametrizations from Section 3.4.1 are studied. For the FFN parametrization, we perform meta-training with and without the theoretically-driven regularization terms of Eq. (28) as developed in Section 3.4.3. We present the regularization results explicitly only when the terms of Eq. (28) are not zero, otherwise the respective results are identical. For the LAL parametrization, the regularization is not required because the desirable conditions of Section 3.4.3 are automatically satisfied according to Proposition 1.

In all the examples, meta-training is performed using Adam as the outer optimizer for 10,000 iterations. During meta-training, 6 snapshots of the learned loss are captured, with 0 corresponding to initialization. Furthermore, only one task is used in each outer step throughout this section ($T = 1$ in Algorithm 1); increasing this number up to 5 does not provide any significant performance increase in the considered cases. In meta-testing, we compare the performance of the snapshots of the learned loss captured during meta-training with standard loss functions from Table A.1. Specifically, we compare 12 learned losses (6 snapshots of the FFN and 6 of the LAL parametrization; see Sections 3.4.1.1-3.4.1.2) with the squared ℓ_2-norm (MSE), the absolute error (L1), the Cauchy, and the Geman-McClure (GMC) loss functions. In addition, we compare with the OAL of [12] (see Section 3.4.1.1) with 2 learning rates (0.01 and 0.1, denoted as OAL 1 and OAL 2) for its trainable parameters (robustness and scale); only the robustness parameter is trained in the computational examples because this setting yielded better performance. For evaluating the performance of the considered loss functions we use them for meta-testing on either 5 or 10 unseen tasks, either in-distribution (ID) or out-of-distribution (OOD), and record the relative ℓ_2 test error (rl2) on exact solution datapoints averaged over tasks.

4.1. Discontinuous function approximation with varying frequencies and heteroscedastic noise

We first consider a distribution of functions in $[0, 4\pi]$ defined as

$$u(x) = \begin{cases} \sin(\omega_1 x) + \epsilon, & \text{if } 0 \leq x \leq 2\pi \\ k(1 + \sin(\omega_2(x - 2\pi))), & \text{if } 2\pi < x \leq 4\pi, \end{cases} \tag{29}$$

where k denotes the magnitude of discontinuity and ϵ represents a zero-mean Gaussian noise term defined only in $[0, 2\pi]$ with standard deviation σ_ϵ. In this regard, a task distribution $p(\lambda)$ can be defined by drawing randomly frequency values $\lambda = \{\omega_1, \omega_2\}$ from $\mathscr{U}_{[\omega_{1,min}, \omega_{1,max}]}$ and $\mathscr{U}_{[\omega_{2,min}, \omega_{2,max}]}$, respectively, where $\mathscr{U}$ denotes a uniform distribution. The defined task distribution is used in this section for function approximation. The values of the fixed parameters used in this example can be found in Table 1.

Table 1
Function approximation: Task distribution values pertaining to Eq. (29), used in meta-training and OOD meta-testing.

	k	$\omega_{1,min}$	$\omega_{1,max}$	$\omega_{2,min}$	$\omega_{2,max}$	σ_ϵ
meta-training	1	1	3	5	6	0.2
OOD meta-testing	1	0.5	4	6	7	0.2

Meta-training. Following the design options experiment in Appendix D, we fix the design options of Algorithm 1 to $J = 20$ inner steps and to resampling and re-initializing every outer iteration. The approximator NN architecture consists of 3 hidden layers with 40 neurons each and *tanh* activation function. In the inner objective we consider $N_u = 100$ noisy datapoints with $\sigma_\epsilon = 0.2$, whereas in the outer objective $N_{u,val} = 1{,}000$ and $\sigma_\epsilon = 0$. This can be interpreted as leveraging in the offline phase clean historical data that we synthetically corrupt with noise in order to meta-learn a loss function that can work well at test time also for the case of noisy data. Moreover, both parametrizations (FFN and LAL) are used for comparison, and Adam is used as both inner and outer optimizer, with learning rates 10^{-3} and 10^{-4}, respectively. In Fig. 2 we show the learned loss snapshots and their first-order derivatives and compared with MSE for the FFN and LAL parametrizations. Both FFN and LAL parametrizations with MSE initialization yield highly different learned losses as compared to MSE. Being more flexible than LAL, FFN leads to more complex learned losses as depicted especially in the first-order derivative plots (Figs. 2b and 2d).

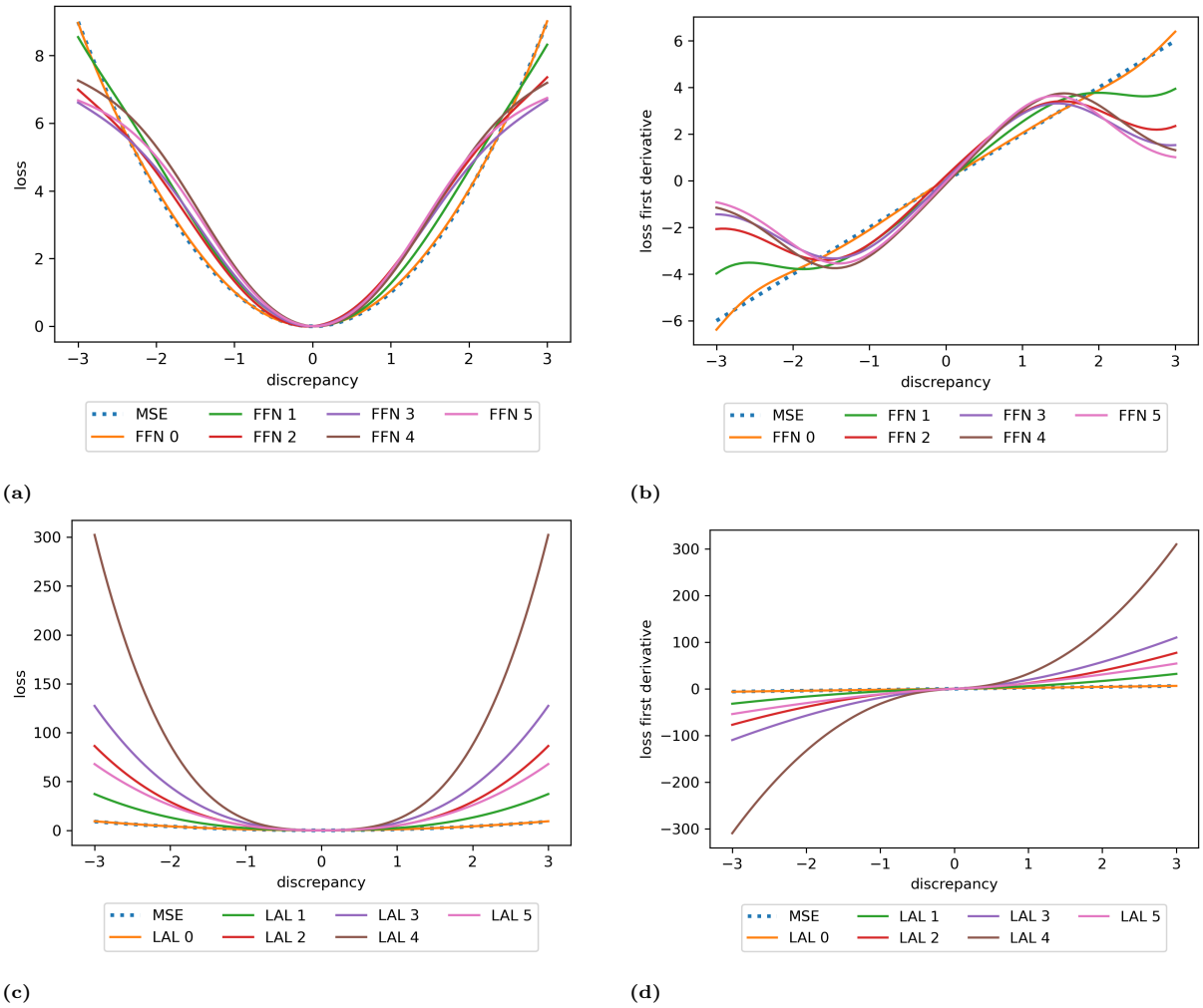


Fig. 2. Function approximation: Learned loss snapshots (a, c) and corresponding first-order derivatives (b, d), as captured during meta-training (distributed evenly in 10,000 outer iterations with 0 referring to initialization). Results obtained with FFN (a, b) and LAL (c, d) parametrizations. Both FFN and LAL parametrizations with MSE initialization yield highly different learned losses as compared to MSE. Being more flexible than LAL, FFN leads to more complex learned losses as depicted especially in the first-order derivative plots (b, d).

Meta-testing. For evaluating the performance of the captured learned loss snapshots, we use them for meta-testing on 10 OOD tasks and compare with standard loss functions from Table A.1. Specifically, we train with Adam for 50,000 iterations 10 tasks using 18 different loss functions (6 FFN, 6 LAL and 6 standard) and record the rl2 error on 1,000 exact solution datapoints. The test tasks are sampled from a distribution defined by combining Eq. (29) with $k = 1$ and $\sigma_\epsilon = 0.2$, and with the uniform distributions $\mathscr{U}_{[\omega_{1,min},\omega_{1,max}]}$ and $\mathscr{U}_{[\omega_{2,min},\omega_{2,max}]}$, where $\{\omega_{1,min}, \omega_{1,max}, \omega_{2,min}, \omega_{2,max}\}$ are shown in Table 1. The minimum rl2 error results are shown in Fig. 3. We see that the loss functions learned with the FFN parametrization do not generalize well, whereas the ones learned with the LAL parametrization achieve an average minimum rl2 error that is smaller than the error corresponding to all the other considered loss functions by at least 15%. For example, the average minimum rl2 error for LAL 4 is approximately 17% and is obtained (on average) close to iteration 20,000, whereas the corresponding error for MSE is approximately 20% and is obtained (on average) close to iteration 50,000. Despite the improvement

demonstrated in this example, in Fig. 10 we show for the advection equation example that the performance of the learned loss depends on the exponential decay parameters of Adam that control the dependence of the updates on the gradient history.

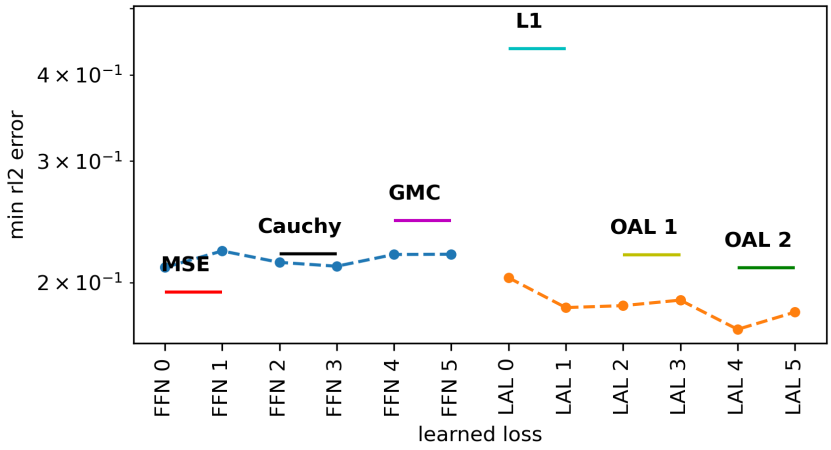


Fig. 3. Function approximation: Minimum relative test ℓ_2 error (rl2) averaged over 10 OOD tasks during meta-testing with Adam for 50,000 iterations. Learned loss snapshots (FFN 0-6 and LAL 0-6) are compared with standard loss functions of Table A.1 and with online adaptive loss functions OAL 1 and OAL 2 (2 loss-specific learning rates). The loss functions learned with the FFN parametrization do not generalize well, whereas the ones learned with the LAL parametrization achieve an average minimum rl2 error that is smaller than the error corresponding to all the other considered loss functions by at least 15%.

4.2. Task distributions defined based on advection equation with varying initial conditions and discontinuous solutions

Next, we consider the $(1+1)$-dimensional advection equation given as

$$\partial_t u + V \partial_x u = 0, \tag{30}$$

where V is the constant advection velocity, $x \in [-1, 1]$ and $t \in [0, 1]$. The considered Dirichlet BCs are given as

$$u(-1, t) = u(1, t) = 0 \tag{31}$$

and the ICs as

$$u(x, 0) = u_{0,\lambda}(x) = \begin{cases} \frac{1}{\lambda}, & \text{if } -1 \leq x \leq -1 + \lambda \\ 0, & \text{if } -1 + \lambda < x \leq 1, \end{cases} \tag{32}$$

i.e., $u_{0,\lambda}(x)$ is a normalized box function of length λ. The exact solution for this problem is given as $u_{0,\lambda}(x - Vt)$, which is also a box function that advects in time. In this regard, we can define a PDE task distribution comprised of problems of the form of Eq. (30) with ICs given by Eq. (32) with varying λ. A task distribution $p(\lambda)$ can be defined by drawing randomly λ values from $\mathscr{U}_{[\lambda_{min}, \lambda_{max}]}$, which correspond to different initial conditions $u_{0,\lambda}(x)$ in Eq. (32). In this example, $V = 1$, $\lambda_{min} = 0.5$, and the maximum value of λ that can be considered so that the BCs are not violated is 1; thus we consider $\lambda_{max} = 1$.

Meta-training. During meta-training, Algorithm 1 is employed with either SGD or Adam as inner optimizer with learning rate 10^{-2} or 10^{-3}, respectively. Both FFN and LAL are initialized as MSE

approximations, the number of inner iterations is 20, and tasks are resampled and approximator NNs (PINNs in this case) are randomly re-initialized in every outer iteration. The number of datapoints $\{N_f, N_b, N_{u_0}\}$ and $\{N_{f,val}, N_{b,val}, N_{u_0,val}\}$ used for evaluating both the inner objective of Eq. (17) and the outer objective of Eq. (18), respectively, are the same and equal to $\{1{,}000, 100, 200\}$, and $N_{u,val} = 0$. Furthermore, the PINN architecture consists of 4 hidden layers with 20 neurons each and *tanh* activation function.

We show in Fig. 4, the final loss functions (FFN and LAL parametrizations) as obtained with SGD as inner optimizer, with and without meta-learning the objective function weights (Section 3.4.1.3), and as obtained with Adam as inner optimizer. For SGD as inner optimizer, both FFN and LAL parametrizations with MSE initialization yield highly different learned losses as compared to MSE, with FFN yielding more complex learned losses. Furthermore, objective function weights meta-learning leads to an asymmetric final learned loss and is found in meta-testing to deteriorate performance. For Adam as inner optimizer, the final learned losses are close to MSE. The corresponding objective function weights trajectories are shown in Fig. 5. All objective function weights increase for both parametrizations, which translates into learning rate increase, while FFN and LAL disagree on how they balance ICs. The meta-testing results (100 test iterations) obtained while meta-training for the case of SGD as inner optimizer are shown in Fig. 6. Although initially objective function weights meta-learning improves performance, the corresponding final learned losses in conjunction with the final learned weights eventually deteriorate performance.

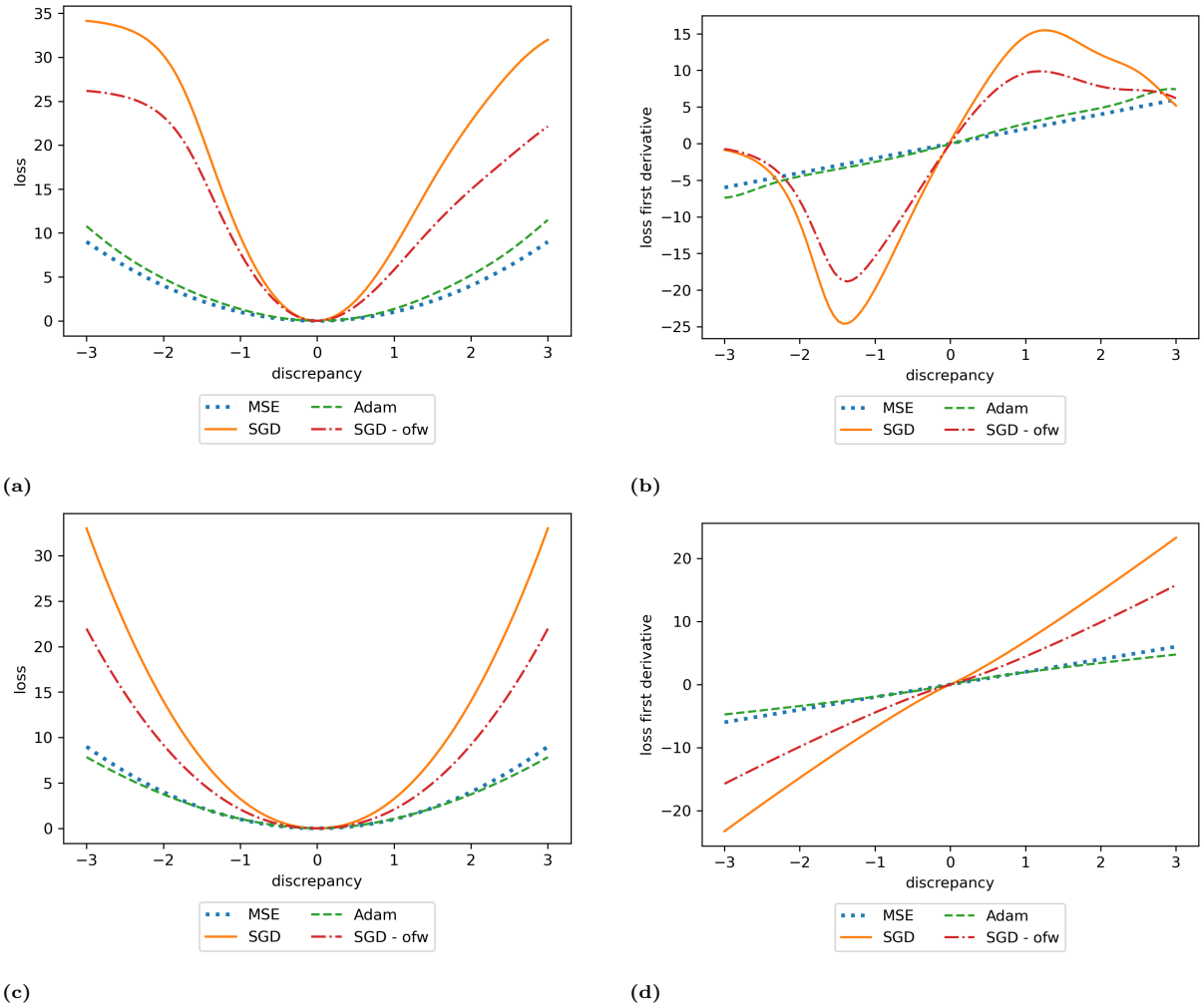
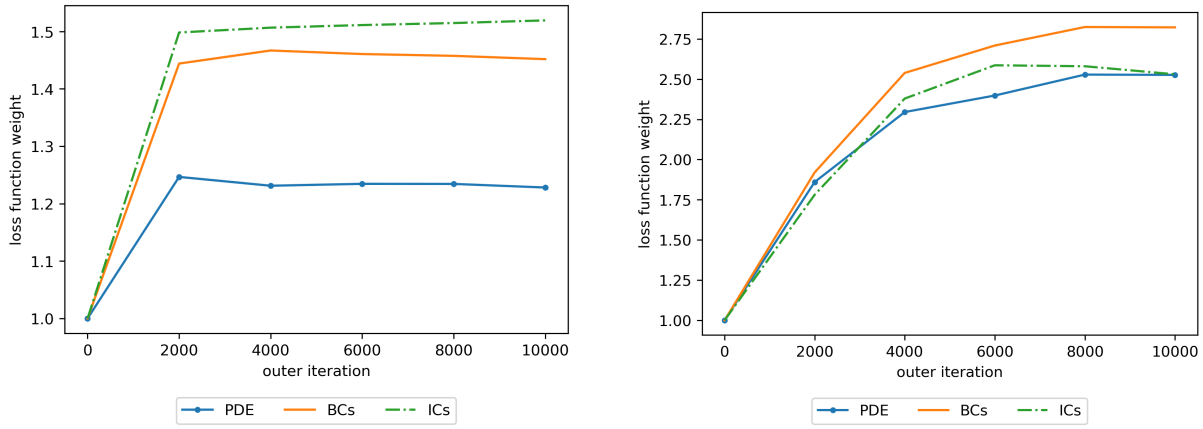


Fig. 4. Advection equation: Final learned losses (a, c) and corresponding first-order derivatives (b, d), with FFN (a, b) and LAL (c, d) parametrizations. Results obtained via meta-training with SGD as inner optimizer (without and with meta-learning the objective function weights; SGD and SGD - ofw) and with Adam optimizer; comparisons with MSE are also included. For SGD as inner optimizer, both FFN and LAL parametrizations with MSE initialization yield highly different learned losses as compared to MSE, with FFN yielding more complex learned losses. SGD - ofw leads to an asymmetric final learned loss and is found in meta-testing to deteriorate performance. For Adam as inner optimizer, final learned losses are close to MSE.



(a) **(b)**

Fig. 5. Advection equation: Learned objective function weights, pertaining to PDE, BCs, and ICs residuals, as a function of outer iteration in meta-training; see Section 3.4.1.3. Results obtained with FFN (a) and LAL (b) parametrizations. All objective function weights increase for both FFN and LAL parametrizations, which translates into learning rate increase, while FFN and LAL disagree on how they balance ICs.

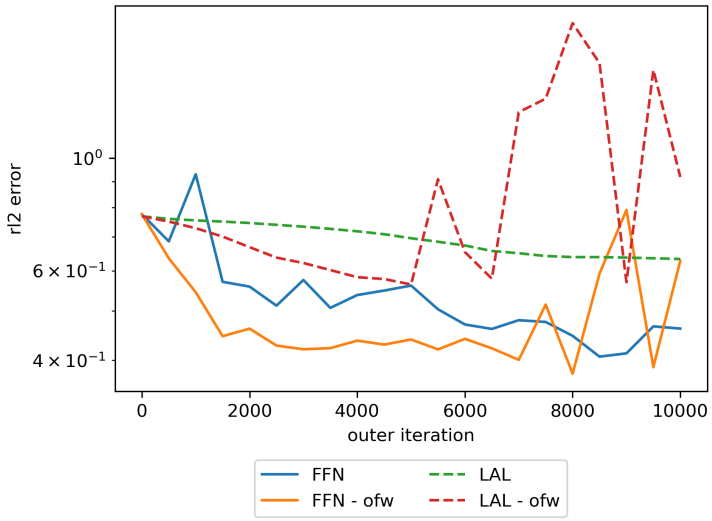
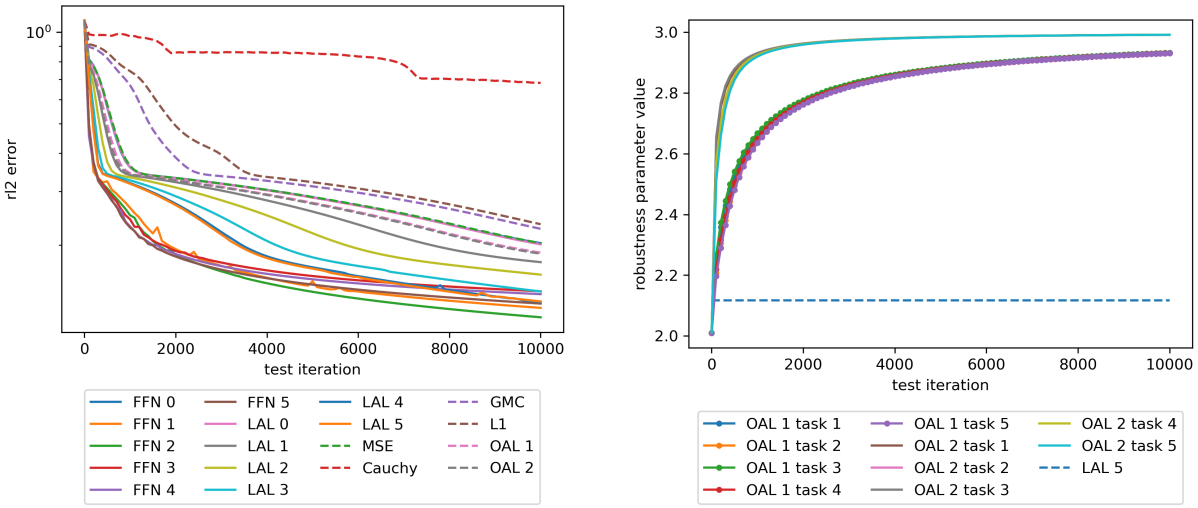


Fig. 6. Advection equation: Meta-testing results (relative ℓ_2 test error on 1 unseen task after 100 iterations) obtained using learned loss snapshots and performed during meta-training (every 500 outer iterations); these can be construed as *meta-validation error* trajectories. Results related to FFN (without and with objective function weights meta-learning; FFN and FFN - ofw) and to LAL (without and with objective function weights meta-learning; LAL and LAL - ofw) are included. In this experiment, although initially objective function weights meta-learning improves performance, the corresponding final learned losses in conjunction with the final learned weights eventually deteriorate performance.

Meta-testing with SGD. For evaluating the performance of the captured learned loss snapshots, we employ them for meta-testing on 5 ID tasks and compare with standard loss functions from Table A.1. Specifically, we train with SGD for 10,000 iterations with learning rate 0.01 (same as in meta-training), 5 tasks using learned and standard loss functions, and record the rl2 error on 10,000 exact solution datapoints. As learned losses, we use the ones obtained with SGD as inner optimizer and without objective function weights meta-learning. The test error histories as well as the OAL parameters during

training are shown in Fig. 7, and the minimum rl2 error results are shown in Fig. 8. As shown in Fig. 7b, the final learned loss LAL 5 is much different than both OAL 1 and 2, which converge to a robustness parameter value close to 3 for all tasks. In Figs. 7a-8, we see that the loss functions learned with both parametrizations achieve an average minimum rl2 error during 10,000 iterations that is significantly smaller than all the considered losses (even the online adaptive ones) although they have been meta-trained with only 20 iterations.



(a) **(b)**

Fig. 7. Advection equation: In-distribution (ID) meta-testing results. Results obtained using SGD with learning rate 0.01 for 10,000 iterations. (a) shows the meta-testing relative ℓ_2 test error (rl2) trajectories for all loss functions considered. (b) shows the robustness parameter trajectories for online adaptive loss functions OAL 1 and OAL 2, with loss-specific learning rates 0.01 and 0.1, respectively (see Section 3.4.1.1); comparison with final learned loss obtained via meta-training with LAL parametrization also included. As shown in (a), learned losses perform better than considered standard and adaptive losses. Furthermore, as shown in (b), final learned loss LAL 5 is much different than both OAL 1 and 2, which converge to a robustness parameter value close to 3 for all tasks.

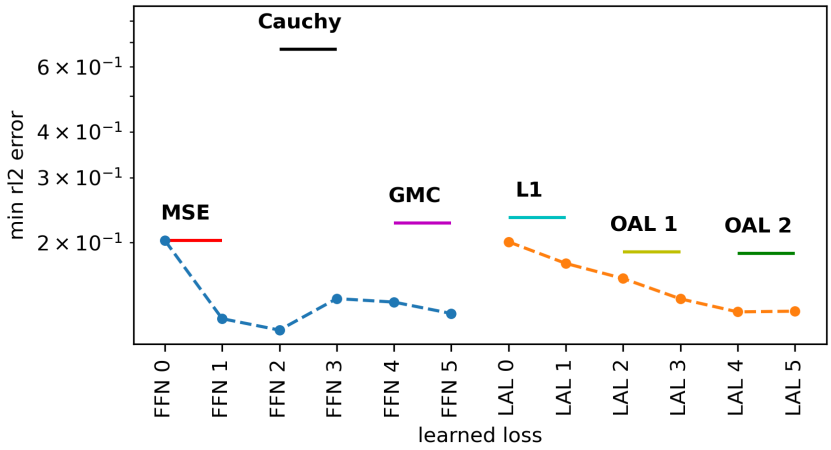


Fig. 8. Advection equation: Minimum relative test ℓ_2 error (rl2) averaged over 5 ID tasks during meta-testing with SGD for 10,000 iterations. Learned loss snapshots (FFN 0-6 and LAL 0-6, obtained with SGD as inner optimizer) are compared with standard loss functions of Table A.1 and with online adaptive loss functions OAL 1 and OAL 2 (2 loss-specific learning rates). Loss functions learned with both FFN and LAL parametrizations achieve an average minimum rl2 error during 10,000 iterations that is significantly smaller than all the considered losses (even the online adaptive ones), although they have been meta-trained with only 20 iterations.

Meta-testing with Adam. Next, we employ the learned losses obtained using Adam as inner optimizer in the same meta-testing experiment as the one of Figs. 7-8; except for the fact that Adam with learning rate 10^{-3} is used in meta-testing instead of SGD. The minimum rl2 error results are shown in Fig. 9, where we see that the learned losses do not improve performance as compared to MSE; same results have been observed for the examples of Sections 4.3-4.4 but these results are not included in this paper. One reason for this result is the fact that Adam depends on the whole history of gradients during optimization through an exponentially decaying average that only discards far in the past gradients. However, our learned losses have been meta-trained with only 20 inner iterations, and thus it is unlikely that they could have learned this memory property of Adam. To illustrate the validity of the above explanation, we perform meta-training with Adam as inner optimizer with varying exponential decay parameters and subsequently meta-testing with the obtained learned loss snapshots (see Fig. 10). Specifically, we use values for the pair (β_1, β_2), corresponding to the decay parameters for the first and second moment estimates in Adam (see [36]) in the set $\{(0.5, 0.5), (0.8, 0.8), (0.9, 0.999) = \text{default}, (0.99, 0.9999)\}$ with higher numbers corresponding to higher dependency on the far past. Note that the decay factors multiplying the $21st$ gradient in the past (i.e., 1 gradient beyond the history used in meta-training) are approximately 10^{-6} and 10^{-2} for the pairs $(0.5, 0.5)$ and $(0.8, 0.8)$, respectively. As expected and shown in Fig. 10, higher β values corresponding to higher dependency on the far past yield deteriorating performance of the learned losses. In this regard, we use only SGD as inner optimizer in the rest of the computational examples and leave the task of improving the performance of the technique for addressing inner optimizers with memory, such as SGD with momentum, AdaGrad, RMSProp and Adam, as future work.

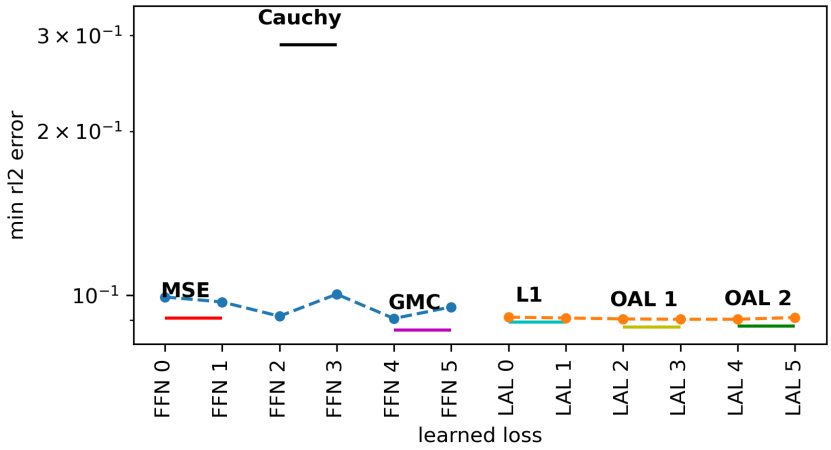


Fig. 9. Advection equation: Minimum relative test ℓ_2 error (rl2) averaged over 5 ID tasks during meta-testing with Adam for 10,000 iterations. Learned loss snapshots (FFN 0-6 and LAL 0-6, obtained with Adam as inner optimizer) are compared with standard loss functions of Table A.1 and with online adaptive loss functions OAL 1 and OAL 2 (2 loss-specific learning rates). Loss functions learned with both FFN and LAL parametrizations do not improve performance as compared to MSE. This is attributed to the fact that our learned losses have been meta-trained with only 20 inner iterations, whereas Adam depends on the whole history of gradients during optimization through an exponentially decaying average; see Fig. 10 for an experiment with various exponential decay parameters.

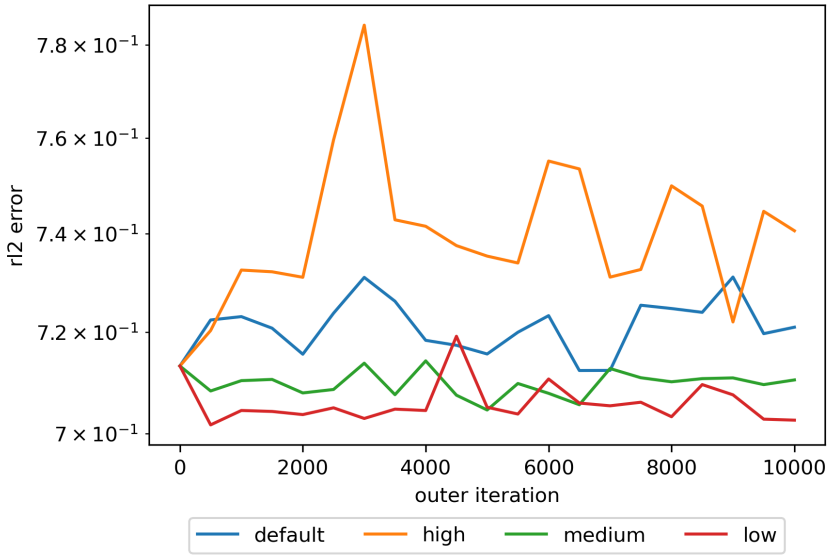


Fig. 10. Advection equation: Meta-testing results (relative ℓ_2 test error on 10 unseen task after 100 iterations) obtained using learned loss snapshots and performed during meta-training (every 500 outer iterations). Results are related to meta-training with FFN parametrization and Adam as inner optimizer with 4 different β pair values: low $= (0.5, 0.5)$, medium $= (0.8, 0.8)$, default $= (0.9, 0.999)$, and high $= (0.99, 0.9999)$; levels low, medium, high indicate degree of optimizer dependence on gradient history older than 20 iterations, where 20 is the number of inner iterations used in meta-training. Higher β values corresponding to higher dependency on the far past yield deteriorating performance of the learned losses.

4.3. Task distributions defined based on reaction-diffusion equation with varying source term

Reaction-diffusion equations are used to describe diverse systems ranging from population dynamics to chemical reactions and have the general form

$$\partial_t u = D\Delta u + z(x, u, \nabla u), \tag{33}$$

where Δ denotes the Laplace operator and D is called diffusion coefficient. In Eq. (33), $D\Delta u$ represents the diffusion term whereas $z(x, u, \nabla u)$ the reaction term. In the following, without loss of generality, we consider a two-dimensional nonlinear, steady-state version of Eq. (33) given as

$$k(\partial^2_{x_1} u + \partial^2_{x_2} u) + u(1 - u^2) = z, \tag{34}$$

where $x_1, x_2 \in [-1, 1]$ refer to space dimensions, z can be interpreted as a source term, and u is considered as known at all boundaries.

To demonstrate the role of task distributions in the context of varying excitation terms, we consider a family of fabricated solutions u that after differentiation produce a family of z terms in Eq. (34). As an illustrative example, the task distribution $p(z_\lambda)$ can be defined by drawing $\lambda = \{\alpha_1, \alpha_2, \omega_1, \omega_2, \omega_3, \omega_4\}$, with $\lambda \sim p(\lambda)$, constructing an analytical solution $u = \alpha_1 \tanh(\omega_1 x_1) \tanh(\omega_2 x_2) + \alpha_2 \sin(\omega_3 x_1) \sin(\omega_4 x_2)$, and constructing z_λ via Eq. (34). Obviously, in practice the opposite is true; different excitation terms z pose a novel problem of the form of Eq. (34) to be solved. Nevertheless, defining z by using fabricated u solutions that are by construction related to each other helps to demonstrate the concepts of this work in a more straightforward manner.

Meta-training. The task distribution parameters used in meta-training are shown in Table 2. The meta-training design options are the same as in Section 4.2 except for the fact that objective function weights are not meta-learned. Furthermore, the PINN architecture consists of 3 hidden layers with 20 neurons each and *tanh* activation function.

In Fig. 11, we show the final loss functions (FFN and LAL parametrizations) as obtained with SGD and Adam as inner optimizers. In addition, in Fig. 11 we include the loss functions obtained using the exact solution data in the outer objective instead of the composite PINNs loss of Eq. (18). In the considered example, this data is available because a fabricated solution is used, whereas in practice this data may be originating from a numerical solver or from measurements. This is referred to as *double data (DD)* in the plots because different data is used for the inner and for the outer objective; single data is denoted as SD. The number of datapoints used for evaluating the outer objective of Eq. (18) is shown in Table 3, whereas the corresponding number for the inner objective of Eq. (17) is the same as the single-data case in Table 3.

Connection with the theory of Section 3.4.3. Whereas regularization is not required for LAL (see Proposition 1), as shown in Fig. 11 the loss function corresponding to SGD with FFN and no regularization is shifted to the right; i.e., its first-order derivative at 0 discrepancy is not zero. As a result, the MSE relation condition of Corollary 1 is not satisfied and the learned loss leads to divergence in optimization when used for meta-testing. Thus, we also include in Fig. 11 the regularized loss function obtained via meta-training with the theoretically-driven gradient penalty of Eq. (28), which solves this issue; see

also Fig. 12 for the loss function snapshots captured during meta-training for the non-regularized and regularized cases with the FFN parametrization.

Table 2

Reaction-diffusion equation: Task distribution values used in meta-training and OOD meta-testing. Parameters α_1, α_2 and parameters ω_1, ω_2, ω_3, ω_4 share the same limits α_{min}, α_{max} and ω_{min}, ω_{max}, respectively.

	α_{min}	α_{max}	ω_{min}	ω_{max}
meta-training	0.1	1	1	5
OOD meta-testing	0.1	2	0.5	7

Table 3

Reaction-diffusion equation: PDE, BCs, ICs, and solution data considered in the outer objective of Eq. (18) for single data (outer objective data same as inner objective) and for double data (solution data considered available in the outer objective) in meta-training ($\{N_{f,val}, N_{b,val}, N_{u_0,val}, N_{u,val}\}$), as well as for OOD meta-testing ($\{N_f, N_b, N_{u_0}, N_u\}$).

	$N_{f(,val)}$	$N_{b(,val)}$	$N_{u_0(,val)}$	$N_{u(,val)}$
single data (SD) meta-training	1,600	160	NA	0
double data (DD) meta-training	0	0	NA	1,600
OOD meta-testing	2,500	200	NA	0

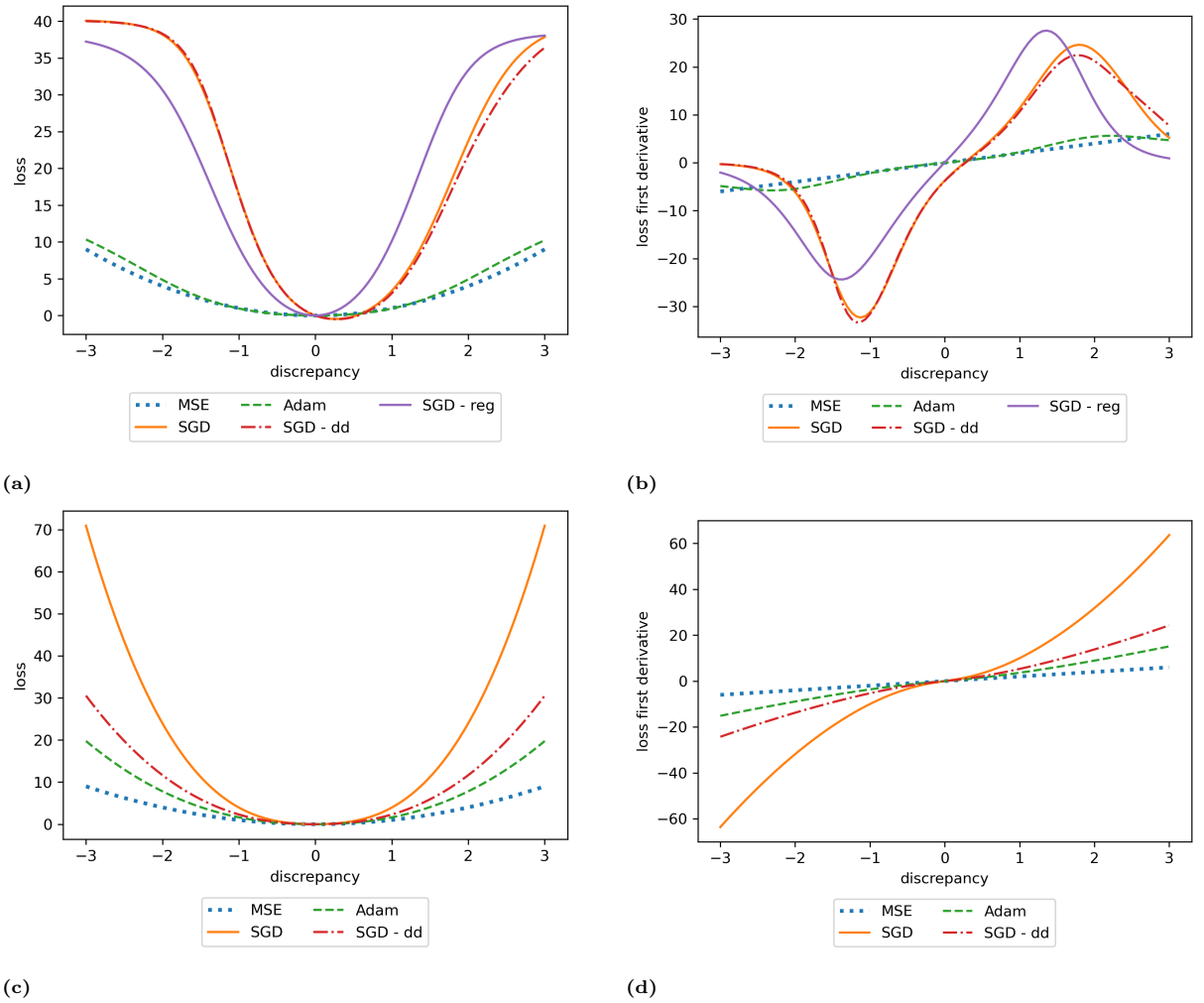
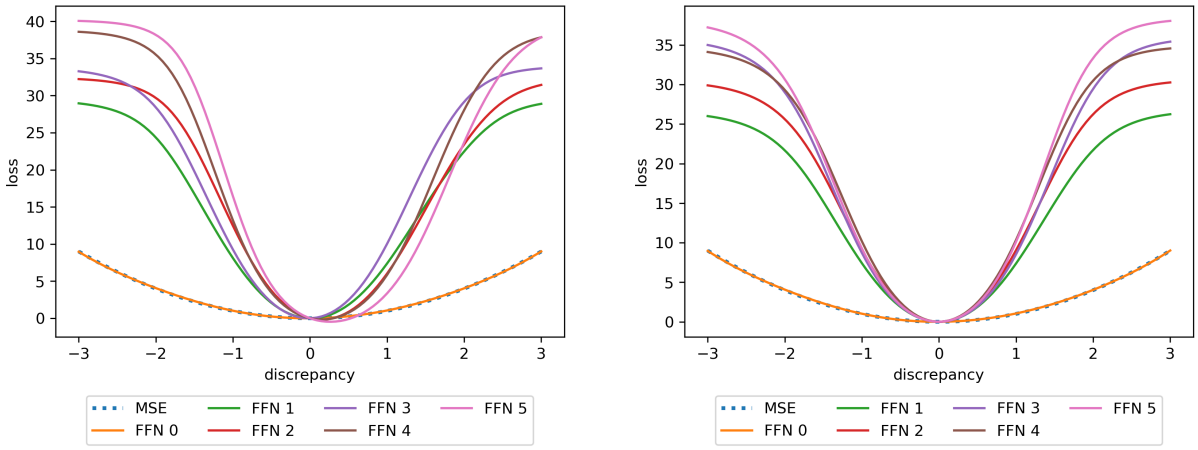


Fig. 11. Reaction-diffusion equation: Final learned losses (a, c) and corresponding first-order derivatives (b, d), with FFN (a, b) and LAL (c, d) parametrizations. Results obtained via meta-training with SGD as inner optimizer (with single data, without and with regularization, and with double data; SGD, SGD - reg, SGD - dd) and with Adam optimizer; comparisons with MSE are also included. Whereas regularization is not required for LAL (Proposition 1), the loss function corresponding to SGD with FFN and no regularization is shifted to the right, i.e., its first-order derivative at 0 discrepancy is not zero. Theory-driven regularization as discussed in Section 3.4.3 fixes this issue.

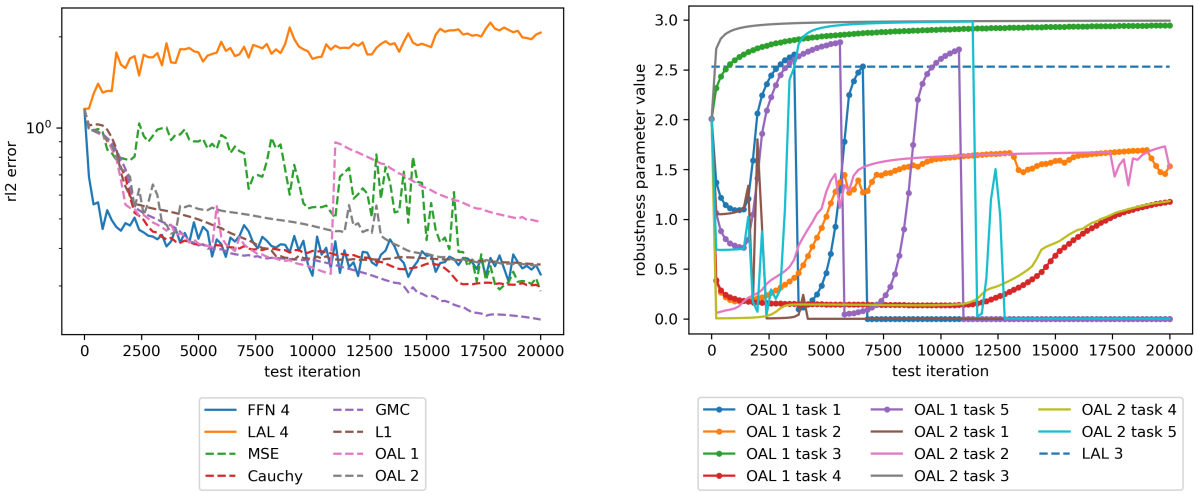


(a) **(b)**

Fig. 12. Reaction-diffusion equation: Learned loss snapshots captured during meta-training (distributed evenly in 10,000 outer iterations and 0 corresponds to initialization), without (a) and with regularization (b) via the penalty of Eq. (28), with FFN parametrization. Learned losses corresponding to SGD with FFN and no regularization are shifted to the right, i.e., their first-order derivative at 0 discrepancy is not zero. Theory-driven regularization as discussed in Section 3.4.3 fixes this issue.

Meta-testing. For evaluating the performance of the captured learned loss snapshots, we employ them for meta-testing on 5 OOD tasks and compare with standard loss functions from Table A.1. Specifically, we train with SGD for 20,000 iterations with learning rate 0.01 (same as in meta-training) 5 tasks using learned and standard loss functions and record the rl2 error on 2,500 exact solution datapoints. The OOD test tasks are drawn based on the parameter limits shown in Table 2. In addition, to increase the difficulty of OOD meta-testing, we also draw random architectures to be used in meta-testing that are different than the meta-training architecture; the number of hidden layers is drawn from $\mathscr{U}_{[2,5]}$ and the number of neurons in each layer from $\mathscr{U}_{[15,55]}$. A learned loss that performs equally well for architectures not used in meta-training is desirable if, for example, we seek to optimize the PINN architecture with a fixed learned loss.

The test error histories as well as the OAL parameters during meta-training are shown in Fig. 13, and the minimum rl2 error results are shown in Fig. 14. As shown in Fig. 13b, the online adaptive losses OAL converge to different loss functions for each task, whereas LAL provides a shared learned loss for all tasks. In Fig. 14 we see that the loss functions learned with the FFN parametrization achieve an average minimum rl2 error during 20,000 iterations that is significantly smaller than most considered standard losses although they have been meta-trained with only 20 iterations, with a different PINN architecture and on a different task distribution. On the other hand, LAL does not generalize well. This is attributed to the fact that, for this example, potentially a task-specific loss would be more appropriate (as suggested by Fig. 13b), and the LAL parametrization is not flexible enough to provide a shared loss function that performs well across tasks (as opposed to FFN).



(a) **(b)**

Fig. 13. Reaction-diffusion equation: Out-of-distribution meta-testing results obtained using SGD with learning rate 0.01 for 20,000 iterations. (a) shows the meta-testing relative ℓ_2 test error (rl2) trajectories for all standard loss functions considered and for selected learned loss snapshots captured during meta-training. (b) shows the robustness parameter trajectories for online adaptive loss functions OAL 1 and OAL 2, with loss-specific learning rates 0.01 and 0.1, respectively (see Section 3.4.1.1); comparison with LAL 3 also included. As shown in (a), the FFN learned loss performs better than most considered standard and adaptive losses, whereas LAL does not generalize well. Furthermore, as shown in (b), the online adaptive losses OAL converge to different loss functions for each task, whereas LAL provides a shared learned loss for all tasks.

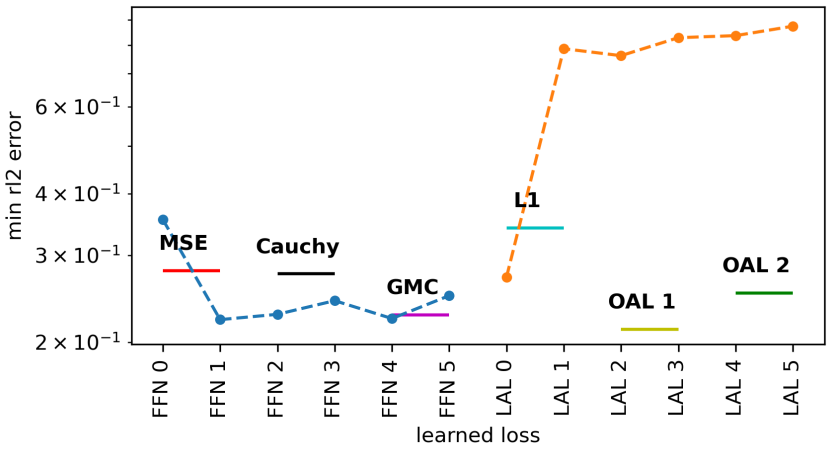


Fig. 14. Reaction-diffusion equation: Minimum relative test ℓ_2 error (rl2) averaged over 5 OOD tasks during meta-testing with SGD for 20,000 iterations. Learned loss snapshots (FFN 0-6 and LAL 0-6) are compared with standard loss functions of Table A.1 and with online adaptive loss functions OAL 1 and OAL 2 (2 loss-specific learning rates). Although FFN learned losses perform better than most standard losses, LAL does not generalize well. This is attributed to the fact that, for this example, potentially a task-specific loss would be more appropriate (as suggested by Fig. 13b), and the LAL parametrization is not flexible enough to provide a shared loss function that performs well across tasks (as opposed to FFN).

4.4. Task distributions defined based on Burgers equation with varying viscocity

Finally, we consider the Burgers equation defined by

$$\partial_t u + u\partial_x u = \lambda \partial_x^2 u,\ x \in [-1, 1], t \in [0, 1], \tag{35}$$

$$u(x, 0) = -\sin(\pi x), u(-1, t) = u(1, t) = 0, \tag{36}$$

where u denotes the flow velocity and λ the viscosity of the fluid. From a function approximation point of view, solutions corresponding to values of λ very close to zero (e.g., $\lambda \approx 10^{-3}$), are expected to have some common characteristics such as steep gradients after some time t. This fact justifies defining a task distribution comprised of PDEs of the form of Eq. (35) with, indicatively, $\lambda < 2 \times 10^{-3}$ and with the same ICs/BCs. A similar explanation can be given for smoother solutions corresponding to values of $\lambda > 10^{-1}$, for example.

Meta-training. For meta-training, we use two regimes for λ as shown in Table 4. The design options are the same as in Section 4.2 except for the fact that only SGD is used in this example as an inner optimizer candidate. The number of datapoints used for evaluating the outer objective of Eq. (18) is shown in Table 5, whereas the corresponding number for the inner objective of Eq. (17) is the same as the single-data case in Table 5. Furthermore, the PINN architecture consists of 3 hidden layers with 20 neurons each and *tanh* activation function.

In Fig. 15 we show the final loss functions (FFN and LAL parametrizations) as obtained with SGD as inner optimizer with single data with and without objective function weights meta-learning, and with double data. The corresponding objective function weights trajectories are shown in Fig. 16. The learned losses with single data have steeper derivatives because they lack the objective function weights, which are shown in Fig. 16 to be greater than 1. Furthermore, for single data the loss functions obtained for the two regimes are slightly different when objective function weights are not meta-learned but almost identical when they are. This means that the meta-learning algorithm compensates for the difference in the two regimes by yielding the same learned loss with different balancing of the PDE, BCs, and ICs terms in the PINNs objective function. On the other hand, when solution data is used in the outer objective (available via the analytical solution), the obtained loss functions are highly different; see also Fig. 17 for the loss function snapshots captured during meta-training for both regimes and for the single and double data cases with the FFN parametrization.

Connection with theory Section 3.4.3. Finally, see Fig. 18 for the outer objective trajectories for both single and double data training with the FFN parametrization, as well as Fig. 19 for the the corresponding test performance on 5 unseen tasks while meta-training for 20 iterations. It is shown in Fig. 18c that the outer objective drops significantly during training for double data and regime 1; this can be construed as *meta-training error*. Furthermore, for the same case it is shown in Fig. 19b that rl2 drops significantly during training; this can be construed as *meta-validation error*. However, the learned loss obtained during this training with no regularization does not satisfy the optimal stationarity condition of Section 3.4.3; see Figs. 15, 17 and notice that the stationary point at 0 is not a global minimum. In line with our theoretical results of Section 3.4.3, these learned losses have also been found in our experiments to lead to divergence if used for full meta-testing (20,000 iterations), i.e., they do not

generalize well although meta-training and meta-validation performance is satisfactory. Finally, we also include in Fig. 12 the regularized loss functions obtained via meta-training with the theoretically-driven gradient penalty of Eq. (20), which solves this issue with the FFN parametrization and double data meta-training.

Table 4

Burgers equation: Task distribution values used in meta-training and OOD meta-testing for both task regimes r1 and r2.

	r1 λ_{min}	r1 λ_{max}	r2 λ_{min}	r2 λ_{max}
meta-training	10^{-3}	2×10^{-3}	10^{-1}	1
OOD meta-testing	10^{-3}	10^{-2}	10^{-2}	2

Table 5

Burgers equation: PDE, BCs, ICs, and solution data considered in the outer objective of Eq. (18) for single data (outer objective data same as inner objective) and for double data (solution data considered available in the outer objective) in meta-training ($\{N_{f,val}, N_{b,val}, N_{u_0,val}, N_{u,val}\}$), as well as for OOD meta-testing ($\{N_f, N_b, N_{u_0}, N_u\}$).

	$N_{f(,val)}$	$N_{b(,val)}$	$N_{u_0(,val)}$	$N_{u(,val)}$
single data (SD) meta-training	1,000	200	100	0
double data (DD) meta-training	0	0	0	10,000
OOD meta-testing	2,000	200	100	0

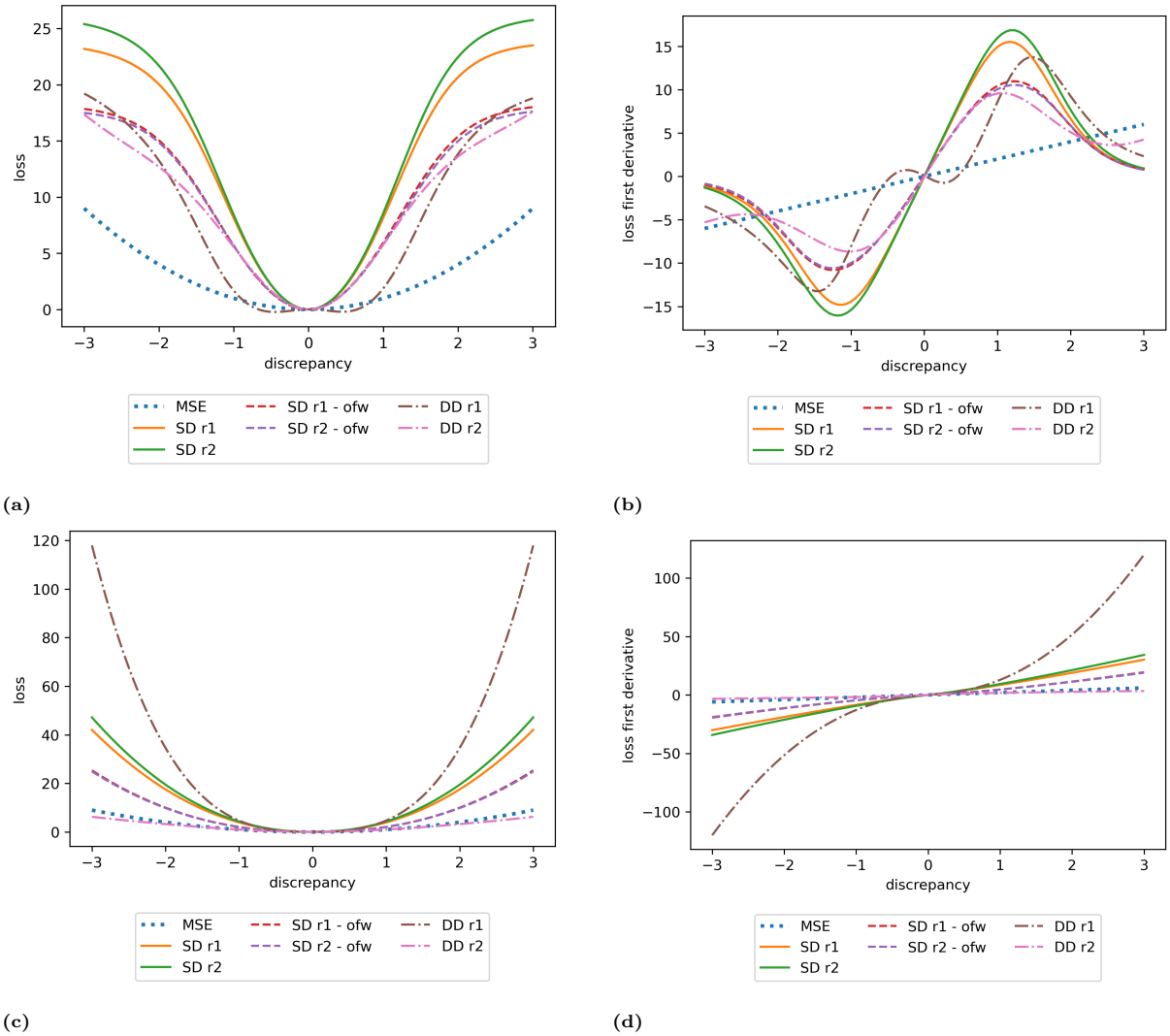
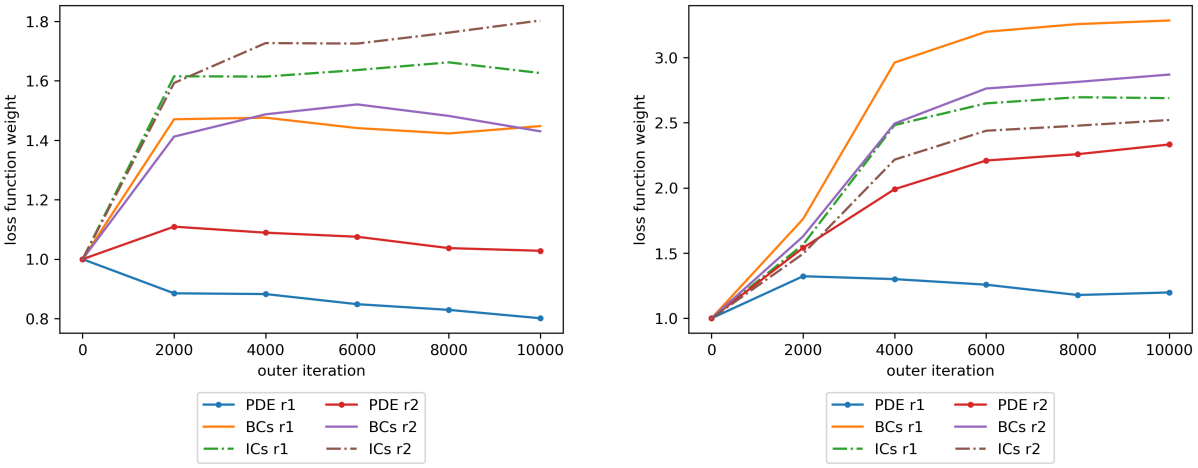


Fig. 15. Burgers equation: Final learned losses (a, c) and corresponding first-order derivatives (b, d), with FFN (a, b) and LAL (c, d) parametrizations for task regimes 1 and 2 (r1 and r2). Results obtained via meta-training with SGD as inner optimizer (with single data, with and without objective function weights meta-learning, and with double data; SD r1-r2, SD r1-r2 - ofw, DD r1-r2); comparisons with MSE are also included. Whereas regularization is not required for LAL (Proposition 1), the final learned loss with FFN parametrization and double data (DD r1) does not satisfy the optimal stationarity condition (notice that the stationary point at 0 is not a global minimum). Theory-driven regularization as discussed in Section 3.4.3 fixes this issue; see Fig. 20. Furthermore, learned losses with single data (SD r1-r2) have steeper derivatives because they lack the objective function weights, which are shown in Fig. 16 to be greater than 1.



(a) **(b)**

Fig. 16. Burgers equation: Learned objective function weights, pertaining to PDE, BCs, and ICs residuals, as a function of outer iteration in meta-training and task regimes (r1-r2); see Section 3.4.1.3. Results obtained with FFN (a) and LAL (b) parametrizations. Most objective function weights increase for both FFN and LAL parametrizations, which translates into learning rate increase, while FFN and LAL disagree on how they balance the various terms.

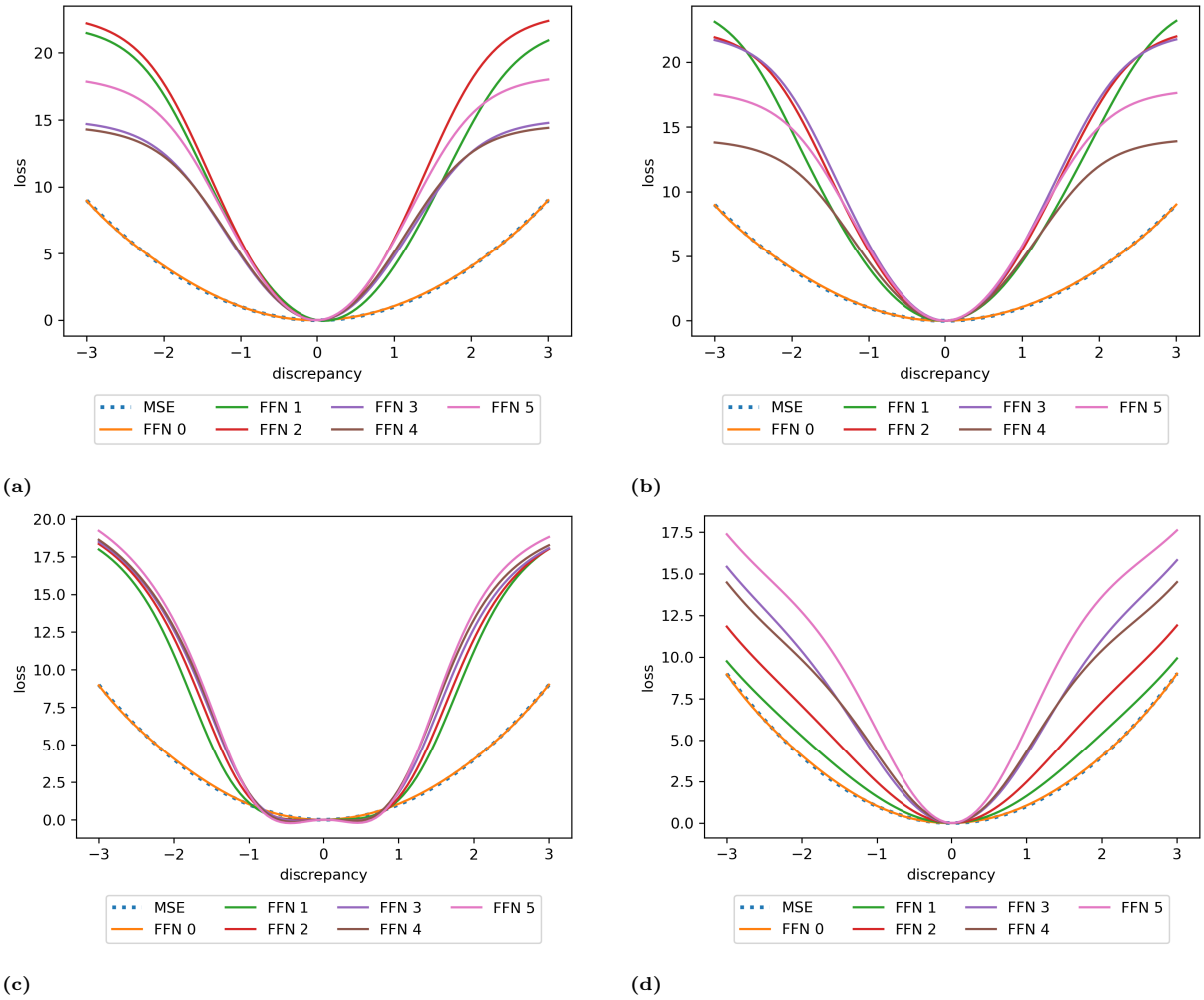


Fig. 17. Burgers equation: Learned loss snapshots captured during meta-training (distributed evenly in 10,000 outer iterations with 0 referring to initialization), with single (a, b) and double data (c, d; see Section 3.3). FFN parametrization only and both task regimes r1 (a, c) and r2 (b, d) are included. The learned losses for single and double data are in general different and the learned losses in part (c) do not satisfy the optimal stationarity condition of Section 3.4.3 (notice that the stationary point at 0 is not a global minimum). Theory-driven regularization as discussed in Section 3.4.3 fixes this issue; see Fig. 20.

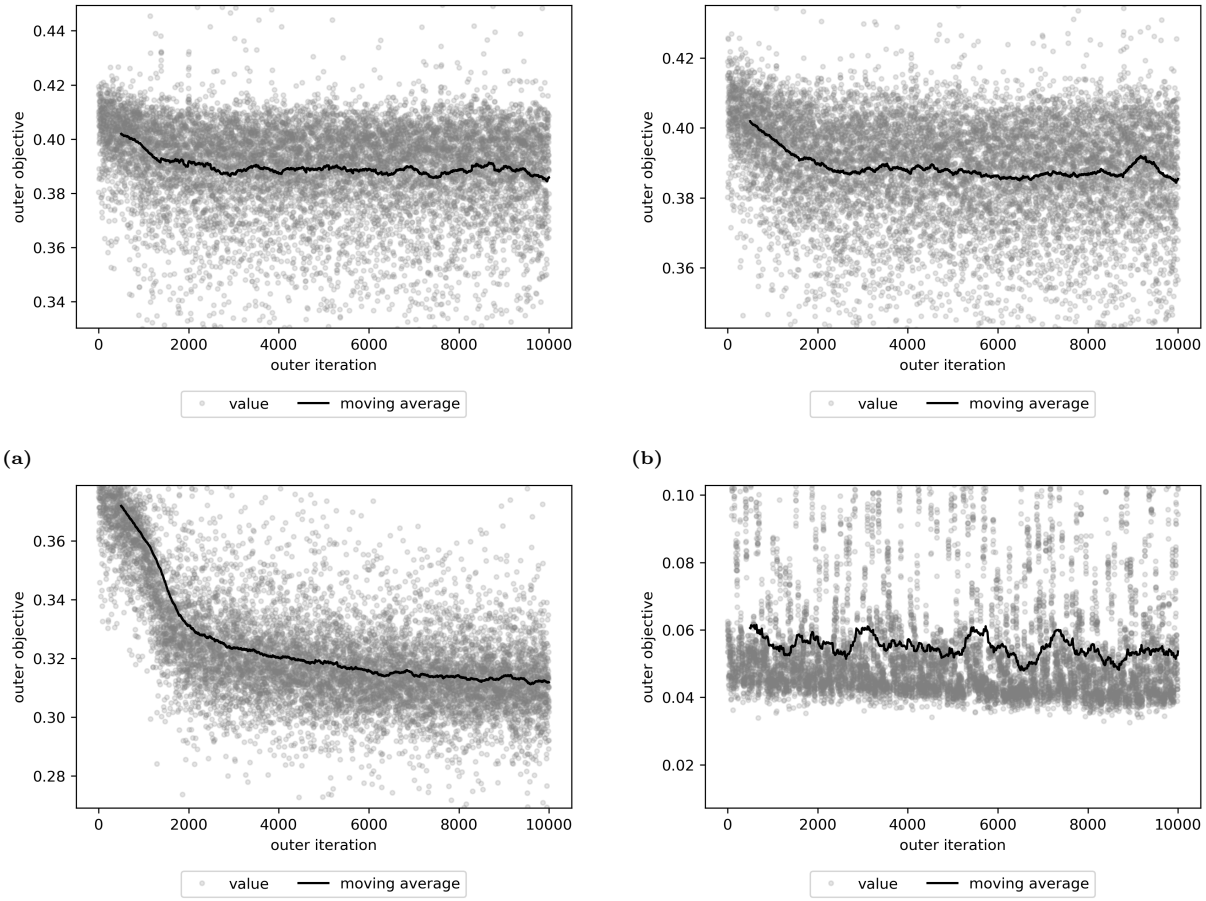
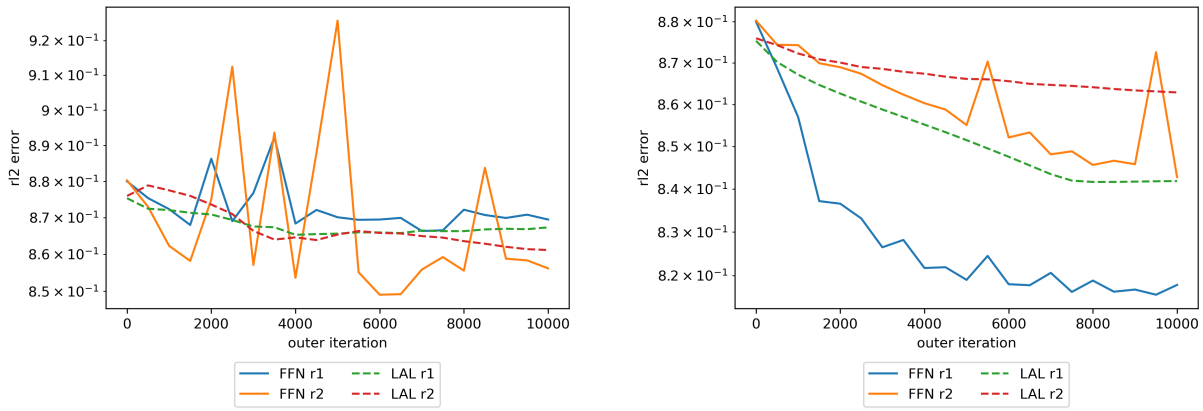
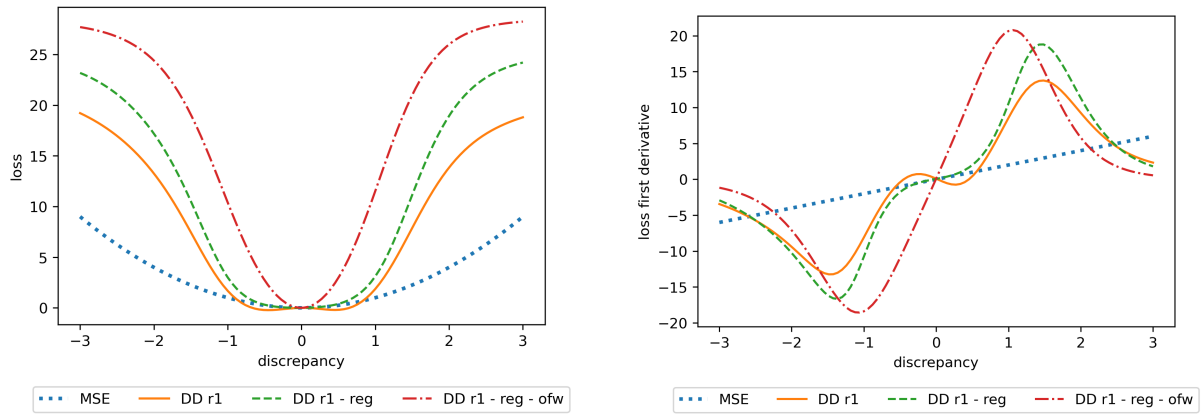


Fig. 18. Burgers equation: Outer objective values recorded during meta-training as well as corresponding moving averages (500 iterations window size); these can be construed as *meta-training error* trajectories. Results related to single data with objective function weights meta-learning (a, b) and double data without objective function weights meta-learning (c, d). FFN parametrization only and both task regimes r1 (a, c) and r2 (b, d) are included. It is shown in part (c) that the outer objective drops significantly during training for double data and r1; recall that it corresponds to loss after 20 iterations and thus, it cannot drop to very small values as typical machine learning objectives do. The learned loss obtained during the training of part (c) does not satisfy the optimal stationarity condition; theory-driven regularization as discussed in Section 3.4.3 fixes this issue.



(a) **(b)**

Fig. 19. Burgers equation: Meta-testing results (relative ℓ_2 test error on 5 unseen tasks after 20 iterations) obtained using learned loss snapshots and performed during meta-training (every 500 outer iterations); these can be construed as *meta-validation error* trajectories. Results related to single data with objective function weights meta-learning (a) and double data without objective function weights meta-learning (b). FFN and LAL parametrizations and both task regimes r1 and r2 are included. It is shown in part (b) that rl2 drops significantly during training for double data and r1. In line with our theoretical results of Section 3.4.3, this learned loss does not satisfy the optimal stationarity condition and leads to divergence if used for full meta-testing (20,000 iterations), i.e., it does not generalize well although training (Fig. 18c) and validation (part b of present figure) performance is satisfactory. Theory-driven regularization as discussed in Section 3.4.3 fixes this issue.



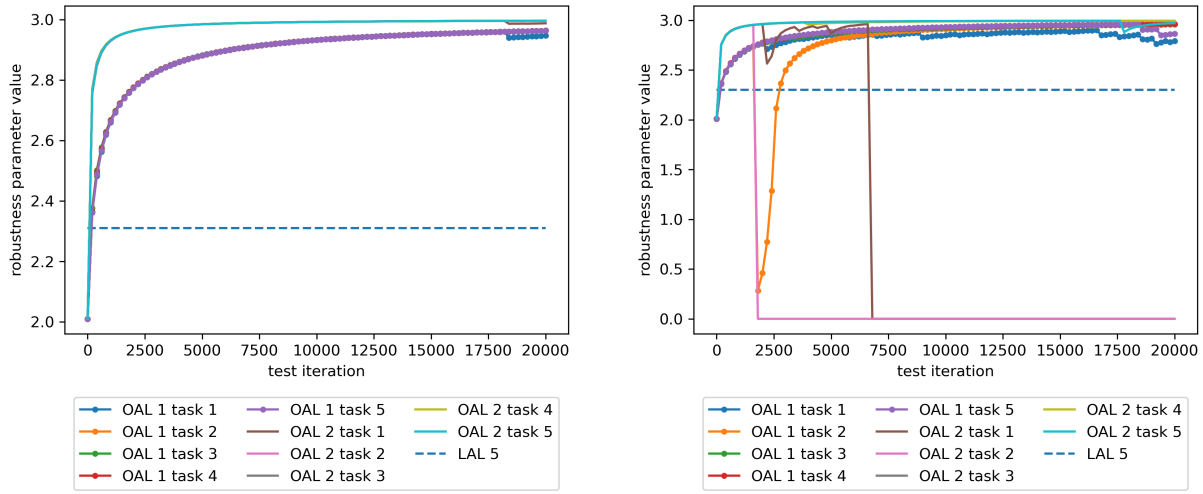
(a) **(b)**

Fig. 20. Burgers equation: Final learned losses (a) and corresponding first-order derivatives (b), with FFN parametrization for task regime 1 with double data meta-training. Results obtained with and without regularization (reg) and with and without objective function weights meta-learning (ofw); comparisons with MSE are also included. Whereas regularization is not required for LAL (Proposition 1), the final learned loss with FFN parametrization and double data (DD r1) does not satisfy the optimal stationarity condition and leads to divergence in optimization. Theory-driven regularization as discussed in Section 3.4.3 fixes this issue.

Meta-testing. For evaluating the performance of the captured learned loss snapshots, we train with SGD for 20,000 iterations with learning rate 0.01 (same as in meta-training) 5 OOD tasks using learned and standard loss functions and record the rl2 error on 10,000 exact solution datapoints. Regarding learned losses, we employ the ones obtained with single data and with objective function weights meta-

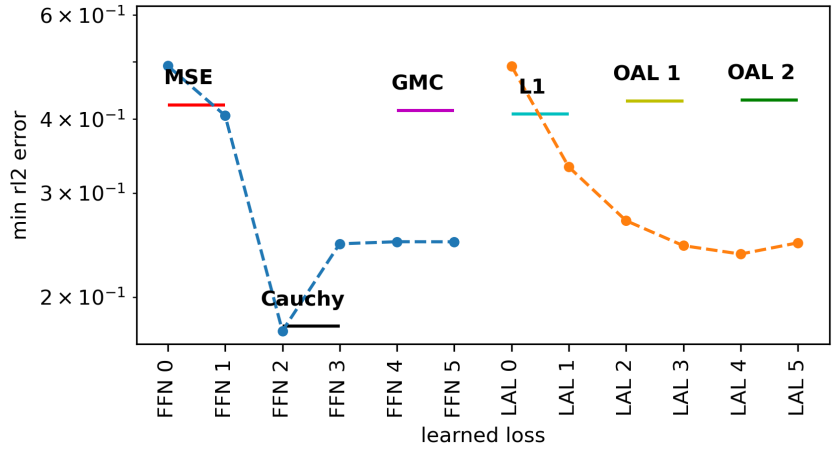
learning, as they performed better in our experiments. The OOD test tasks are drawn based on the parameter limits shown in Table 2 and the random architectures used are drawn in the same way as in Section 4.3.

The OAL parameters during training are shown in Fig. 21 and the minimum rl2 error results are shown in Fig. 22. For both regimes, the final learned loss LAL 5 is different than both OAL 1 and 2, which converge to a robustness parameter value close to 3 for all tasks. Finally, the loss functions learned with both parametrizations achieve an average minimum rl2 error during 20,000 iterations that is significantly smaller than most considered losses (even the online adaptive ones), although they have been meta-trained with only 20 iterations with a different PINN architecture and on a different task distribution.

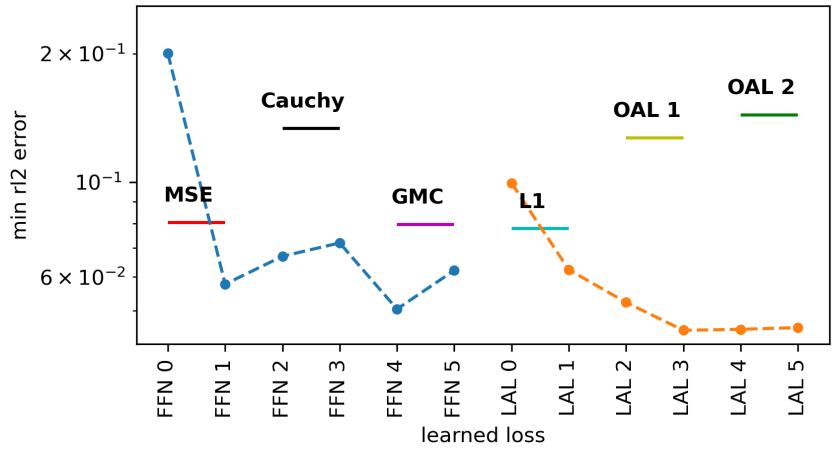


(a) **(b)**

Fig. 21. Burgers equation: Robustness parameter trajectories for online adaptive loss functions OAL 1 and OAL 2 and for both test task regimes. Loss-specific learning rates 0.01 (OAL 1) and 0.1 (OAL 2); see Section 3.4.1.1. Comparison with final learned loss obtained via meta-training with LAL parametrization also included. Results correspond to test regimes 1 (a; $10^{-4} \leq \lambda \leq 10^{-2}$) and 2 (b; $10^{-2} \leq \lambda \leq 2$). For both regimes, final learned loss LAL 5 is different than both OAL 1 and 2, which converge to a robustness parameter value close to 3 for all tasks.



(a)



(b)

Fig. 22. Burgers equation: Minimum relative test ℓ_2 error (rl2) averaged over 5 OOD tasks during meta-testing with SGD for 20,000 iterations. Learned loss snapshots (FFN 0-6 and LAL 0-6) are compared with standard loss functions of Table A.1 and with online adaptive loss functions OAL 1 and OAL 2 (2 loss-specific learning rates). Results correspond to test regimes 1 (a; $10^{-4} \leq \lambda \leq 10^{-2}$) and 2 (b; $10^{-2} \leq \lambda \leq 2$). For both regimes, the learned loss functions with both FFN and LAL parametrizations achieve an average minimum rl2 error that is significantly smaller than most considered losses (even the online adaptive ones), although they have been meta-trained with only 20 iterations, with a different PINN architecture and on a different task distribution.

5. Summary

We have presented a meta-learning method for offline discovery of physics-informed neural network (PINN) loss functions, addressing diverse task distributions defined based on parametrized partial differential equations (PDEs) that are solved with PINNs. For employing our technique given a PDE task distribution definition, we parametrize and optimize the learned loss via meta-training by following an alternating optimization procedure until a stopping criterion is met. Specifically, in each loss function update step, (a) PDE tasks are drawn from the task distribution; (b) in the inner optimization, they are solved with PINNs for a few iterations using the current learned loss and the gradient of the learned loss parameters is tracked throughout optimization; and (c) in the outer optimization, the learned loss parameters are updated based on MSE of the final (semi-optimized) PINN parameters.

Furthermore, we have presented and proven two new theorems, involving a condition, namely the optimal stationarity condition, that the learned loss should satisfy for successful training. If satisfied, this condition assures that under certain assumptions, a global minimum is reached by using gradient descent with the learned loss. In addition, we have proven that under a mean squared error (MSE) relation condition combined with the optimal stationarity condition, any stationary point obtained based on the learned loss function is a global minimum of the MSE-based loss as well. Driven by these theoretical results, we have also proposed a novel regularization method for imposing the above desirable conditions. Finally, we have proven that one of the two parametrizations used in this paper for the learned loss, namely the learned adaptive loss (LAL) discussed in Section 3.4.1.1 and proposed in [12], satisfies automatically these two conditions without any regularization.

In the computational examples, the learned losses have been employed at test time for addressing regression and PDE task distributions. Our results have demonstrated that significant performance improvement can be achieved by using a shared-among-tasks offline-learned loss function even for out-of-distribution meta-testing; i.e., solving test tasks not belonging to the task distribution used in meta-training and utilizing PINN architectures that are different than the PINN architecture used in meta-training. Note that a learned loss that performs equally well for architectures not used in meta-training is desirable if, for example, we seek to optimize the PINN architecture with a fixed learned loss. Moreover, improved performance has been demonstrated even compared with adapting the loss function online as proposed in [12]. In this regard, we have considered the problems of discontinuous function approximation with varying frequencies, of the advection equation with varying initial conditions and discontinuous solutions, of the reaction-diffusion equation with varying source term, and of the Burgers equation with varying viscosity in two parametric regimes. We have demonstrated the importance of different loss function parametrizations, as well as of other meta-learning algorithm design options discussed in Section 3.4.

As of future work, an interesting direction pertains to improving the performance of the technique for addressing inner optimizers with memory, such as RMSProp and Adam. Specifically, although the LAL learned losses coupled with Adam performed better than MSE in the function approximation case of Section 4.1, neither feed-forward NN (FFN) nor LAL learned losses exhibited satisfactory generalization capabilities in the PINN examples of Sections 4.2-4.4. One reason for this result is the fact that Adam depends on the whole history of gradients during optimization through an exponentially decaying average that only discards far in the past gradients. To illustrate this, we have shown in Section 4.2 that higher β values in Adam corresponding to higher dependency on the far past yield deteriorating performance of the learned losses.

6. Acknowledgment

This work was supported by OSD/AFOSR MURI grant FA9550-20-1-0358.

References

[1] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics 378 (2019) 686–707.

[2] L. Lu, X. Meng, Z. Mao, G. E. Karniadakis, DeepXDE: A deep learning library for solving differential equations, SIAM Review 63 (2021) 208–228.

[3] A. D. Jagtap, K. Kawaguchi, G. E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, Journal of Computational Physics 404 (2020) 109136.

[4] X. Meng, G. E. Karniadakis, A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems, Journal of Computational Physics 401 (2020) 109020.

[5] G. Pang, M. D'Elia, M. Parks, G. E. Karniadakis, nPINNs: Nonlocal physics-informed neural networks for a parametrized nonlocal universal Laplacian operator. Algorithms and applications, Journal of Computational Physics 422 (2020) 109760.

[6] E. Kharazmi, Z. Zhang, G. E. M. Karniadakis, Hp-VPINNs: Variational physics-informed neural networks with domain decomposition, Computer Methods in Applied Mechanics and Engineering 374 (2021) 113547.

[7] L. Yang, X. Meng, G. E. Karniadakis, B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data, Journal of Computational Physics 425 (2021) 109913.

[8] S. Cai, Z. Wang, S. Wang, P. Perdikaris, G. E. Karniadakis, Physics-Informed Neural Networks for Heat Transfer Problems, Journal of Heat Transfer 143 (2021).

[9] K. Shukla, A. D. Jagtap, G. E. Karniadakis, Parallel Physics-Informed Neural Networks via Domain Decomposition, arXiv:2104.10013 [cs] (2021). arXiv:2104.10013.

[10] X. Jin, S. Cai, H. Li, G. E. Karniadakis, NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations, Journal of Computational Physics 426 (2021) 109951.

[11] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient pathologies in physics-informed neural networks, arXiv preprint arXiv:2001.04536 (2020). arXiv:2001.04536.

[12] J. T. Barron, A general and adaptive robust loss function, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 4331–4339.

[13] T. Hospedales, A. Antoniou, P. Micaelli, A. Storkey, Meta-Learning in Neural Networks: A Survey, arXiv:2004.05439 [cs, stat] (2020). arXiv:2004.05439.

[14] E. Grefenstette, B. Amos, D. Yarats, P. M. Htut, A. Molchanov, F. Meier, D. Kiela, K. Cho, S. Chintala, Generalized Inner Loop Meta-Learning, arXiv:1910.01727 [cs, stat] (2019). arXiv:1910.01727.

[15] A. Nichol, J. Achiam, J. Schulman, On First-Order Meta-Learning Algorithms, arXiv:1803.02999 [cs] (2018). arXiv:1803.02999.

[16] A. Rajeswaran, C. Finn, S. Kakade, S. Levine, Meta-Learning with Implicit Gradients, arXiv:1909.04630 [cs, math, stat] (2019). arXiv:1909.04630.

[17] J. Lorraine, P. Vicol, D. Duvenaud, Optimizing Millions of Hyperparameters by Implicit Differentiation, International conference on machine learning (2020) 12.

[18] F. Sung, L. Zhang, T. Xiang, T. Hospedales, Y. Yang, Learning to learn: Meta-critic networks for sample efficient learning, arXiv preprint arXiv:1706.09529 (2017). arXiv:1706.09529.

[19] R. Houthooft, R. Y. Chen, P. Isola, B. C. Stadie, F. Wolski, J. Ho, P. Abbeel, Evolved Policy Gradients, arXiv:1802.04821 [cs] (2018). arXiv:1802.04821.

[20] L. Wu, F. Tian, Y. Xia, Y. Fan, T. Qin, L. Jian-Huang, T.-Y. Liu, Learning to Teach with Dynamic Loss Functions, in: Advances in Neural Information Processing Systems 32, 2018, p. 12.

[21] Z. Xu, H. van Hasselt, D. Silver, Meta-Gradient Reinforcement Learning, in: Advances in Neural Information Processing Systems 31, 2018, p. 12.

[22] Z. Zheng, J. Oh, S. Singh, On Learning Intrinsic Rewards for Policy Gradient Methods, arXiv:1804.06459 [cs, stat] (2018). arXiv:1804.06459.

[23] A. Antoniou, A. Storkey, Learning to Learn via Self-Critique, Advances in Neural Information Processing Systems 33 (2019) 11.

[24] J. Grabocka, R. Scholz, L. Schmidt-Thieme, Learning Surrogate Losses, arXiv:1905.10108 [cs, stat] (2019). arXiv:1905.10108.

[25] C. Huang, S. Zhai, W. Talbott, M. A. Bautista, S.-Y. Sun, C. Guestrin, J. Susskind, Addressing the Loss-Metric Mismatch with Adaptive Loss Alignment, International conference on machine learning (2019) 10.

[26] H. Zou, T. Ren, D. Yan, H. Su, J. Zhu, Reward shaping via meta-learning, arXiv preprint arXiv:1901.09330 (2019). arXiv:1901.09330.

[27] S. Bechtle, A. Molchanov, Y. Chebotar, E. Grefenstette, L. Righetti, G. Sukhatme, F. Meier, Meta-Learning via Learned Loss, arXiv:1906.05374 [cs, stat] (2020). arXiv:1906.05374.

[28] S. Gonzalez, R. Miikkulainen, Effective Regularization Through Loss-Function Metalearning, arXiv:2010.00788 [cs, stat] (2020). arXiv:2010.00788.

[29] S. Gonzalez, R. Miikkulainen, Improved Training Speed, Accuracy, and Data Utilization Through Loss Function Optimization, arXiv:1905.11528 [cs, stat] (2020). arXiv:1905.11528.

[30] L. Kirsch, S. van Steenkiste, J. Schmidhuber, Improving Generalization in Meta Reinforcement Learning using Learned Objectives, arXiv:1910.04098 [cs, stat] (2020). arXiv:1910.04098.

[31] A. Sicilia, X. Zhao, D. Minhas, E. O'Connor, H. Aizenstein, W. Klunk, D. Tudorascu, S. J. Hwang, Multi-Domain Learning by Meta-Learning: Taking Optimal Steps in Multi-Domain Loss Landscapes by Inner-Loop Learning, arXiv:2102.13147 [cs, eess] (2021). arXiv:2102.13147.

[32] H. Liu, K. Simonyan, Y. Yang, DARTS: Differentiable Architecture Search, arXiv:1806.09055 [cs, stat] (2019). arXiv:1806.09055.

[33] K. Kawaguchi, J. Huang, Gradient descent finds global minima for generalizable deep neural networks of practical sizes, in: 2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton), IEEE, 2019, pp. 92–99.

[34] J. Huang, H.-T. Yau, Dynamics of deep neural networks and neural tangent hierarchy, in: International Conference on Machine Learning, PMLR, 2020, pp. 4542–4551.

[35] K. Kawaguchi, Q. Sun, A Recipe for Global Convergence Guarantee in Deep Neural Networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35, 2021, pp. 8074–8082.

[36] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014). arXiv:1412.6980.

Appendices

A. Loss functions with multi-dimensional inputs

For each (t, x) in Eq. (1), $u(t, x)$, $\mathcal{F}_\lambda[u](t, x)$, and $\mathcal{B}_\lambda[u](t, x)$ belong to $\mathbb{R}^{D_u}$. The same holds for the outputs of the NN approximators $\hat{u}_\theta$, $\hat{f}_{\theta,\lambda}$ and $\hat{b}_{\theta,\lambda}$. As a result, the loss function ℓ_η, with $\ell_\eta : D_u \times D_u \to \mathbb{R}_{\geq 0}$, outputting the distance between the predicted $\mathcal{F}_\lambda[\hat{u}_\theta](t, x)$ and 0, between the predicted $\mathcal{B}_\lambda[\hat{u}_\theta](t, x)$ and 0, and between the predicted $u_\theta(0, x)$ and $u_{0,\lambda}(x)$ in Eq. (5), takes as input two D_u-dimensional vectors and outputs a scalar loss value. Considering for simplicity, as in Section 2.2, each part of the objective function of Eqs. (4)-(6) separately, the squared ℓ_2-norm loss between $\hat{u}_\theta(t, x)$ and $u(t, x)$ for each datapoint $((t, x), u(t, x))$ is given as

$$||\hat{u}_\theta(t, x) - u(t, x)||_2^2 = \sum_{j=1}^{D_u} (\hat{u}_{\theta,j}(t, x) - u_j(t, x))^2. \tag{A.1}$$

In Eq. (A.1), the loss for each $j \in \{1, \ldots, D_u\}$ depends only on the discrepancy between $\hat{u}_{\theta,j}(t, x)$ and $u_j(t, x)$, and the total loss $||\hat{u}_\theta(t, x) - u(t, x)||_2^2$ is given as the sum of the one-dimensional losses (i.e., of the losses pertaining to one-dimensional inputs). In this regard, see Table A.1 for other than squared error candidates for the one-dimensional loss of Eq. (A.1).

As a first attempt towards constructing a loss function with multi-dimensional inputs, one can generalize Eq. (A.1) by considering a parametrized function $\hat{\ell}_\eta$ instead of the squared error in the summation. This gives rise to a parametrized loss function given as

$$\ell_\eta(\hat{u}_\theta(t, x), u(t, x)) = \sum_{j=1}^{D_u} a_j \hat{\ell}_\eta(\hat{u}_{\theta,j}(t, x) - u_j(t, x)), \tag{A.2}$$

where, in addition, each directional loss is multiplied by a weight a_j for making the loss function ℓ_η more flexible/expressive. The weighted sum of Eq. (A.2) in conjunction with the meta-learning Algorithm 1 can lead to a loss function that normalizes each directional loss optimally. Instead of using the same $\hat{\ell}_\eta$ for each dimension as in Eq. (A.2), more expressive loss functions can be constructed by considering a different loss function $\hat{\ell}_\eta(\hat{u}_\theta(t, x) - u(t, x))$ for each dimension, i.e.,

$$\ell_\eta(\hat{u}_\theta(t, x), u(t, x)) = \sum_{j=1}^{D_u} \hat{\ell}_{\eta,j}(\hat{u}_{\theta,j}(t, x) - u_j(t, x)). \tag{A.3}$$

The latter can be extended further by parametrizing $\hat{\ell}_\eta$ in such a way that both $\hat{u}_\theta(t, x)$, $u(t, x)$ are given as inputs, and not only the discrepancy $\hat{u}_\theta(t, x) - u(t, x)$. For obtaining even more expressive loss functions ℓ_η, one can replace the summation formulas of Eqs. (A.2)-(A.3) by a more general function $\hat{\ell}_\eta$ that takes as inputs the vectors $\hat{u}_\theta(t, x)$, $u(t, x)$ instead of the corresponding values in each dimension. See Fig. A.1 for a schematic illustration of the aforementioned indicative options.

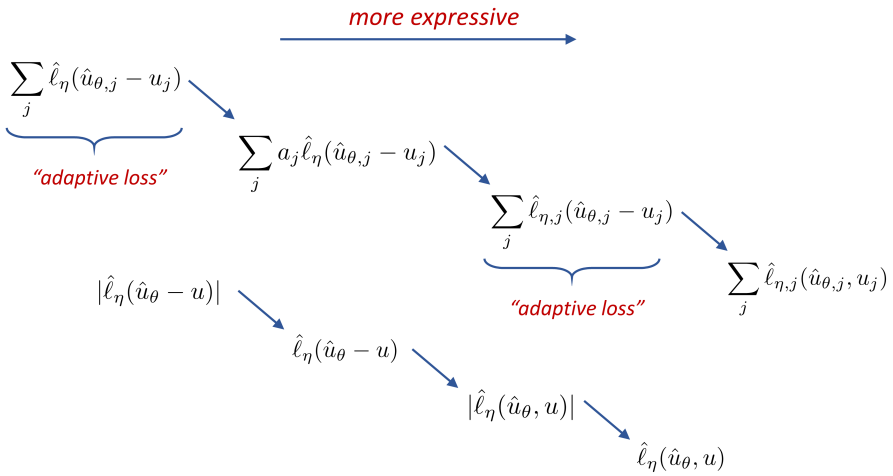
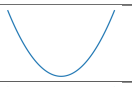
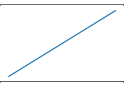
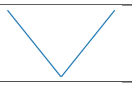
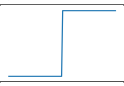
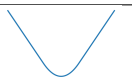
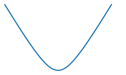
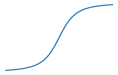
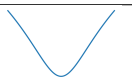
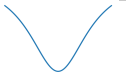
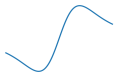
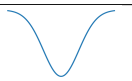
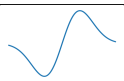


Fig. A.1. Various indicative ways to represent loss function $\ell_\eta(\hat{u}_\theta(t,x), u(t,x))$, ordered based on increasing expressiveness. "Adaptive loss" corresponds to the loss function of [12]; see also Section 3.4.1.

Table A.1

Standard one-dimensional loss functions (i.e., pertaining to one-dimensional inputs $d = \hat{u}_{\theta,j}(t,x) - u_j(t,x)$ for each $j \in \{1, \ldots, D_u\}$), accompanied by the corresponding first-order derivatives. Loss function general forms are adopted by [12]; see more information in Section 3.4.1.

Name	General form	Loss sketch	Derivative sketch
Squared error	$\frac{1}{2}(d/c)^2$		
Absolute error	$\lvert d \rvert$		
Huber	$0.5d^2/c$, if $\lvert x \rvert < c$ else $\lvert x \rvert - 0.5c$		
pseudo-Huber	$\sqrt{(d/c)^2 + 1} - 1$		
Cauchy	$\log\left(\frac{1}{2}(d/c)^2 + 1\right)$		
Geman-McClure	$\frac{2(d/c)^2}{(d/c)^2+4}$		
Welsch	$1 - \exp\left(-\frac{1}{2}(d/c)^2\right)$		

B. Proofs

B.1. Proof of Theorem 1

Proof. We now prove the first statement of this theorem. Let θ be an arbitrary stationary point. Then, we have that

$$0 = \frac{\partial \mathcal{L}(\theta)}{\partial \theta} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\partial \ell(q, u_i)}{\partial q}\Big|_{q=\hat{u}_\theta(x_i)}\right)\frac{\partial \hat{u}_\theta(x_i)}{\partial \theta} \tag{B.1}$$

$$= \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{D_u}\left(\frac{\partial \ell(q, u_i)}{\partial q_j}\Big|_{q=\hat{u}_\theta(x_i)}\right)\frac{\partial \hat{u}_\theta(x_i)_j}{\partial \theta}. \tag{B.2}$$

By rearranging for the gradient,

$$0 = N\nabla\mathcal{L}(\theta) = \sum_{i=1}^{N}\sum_{j=1}^{D_u}\left(\frac{\partial \hat{u}_\theta(x_i)_j}{\partial \theta}\right)^\top\left(\frac{\partial \ell(q, u_i)}{\partial q_j}\Big|_{q=\hat{u}_\theta(x_i)}\right)^\top. \tag{B.3}$$

By rearranging the double sum into the matrix-vector product,

$$0 = \left(\frac{\partial \hat{u}_X(\theta)}{\partial \theta}\right)^\top v, \tag{B.4}$$

where

$$v = \mathrm{vec}\left(\left(\frac{\partial \ell(q, u_1)}{\partial q}\Big|_{q=\hat{u}_\theta(x_1)}\right)^\top, \ldots, \left(\frac{\partial \ell(q, u_N)}{\partial q}\Big|_{q=\hat{u}_\theta(x_N)}\right)^\top\right) \in \mathbb{R}^{ND_u}. \tag{B.5}$$

Therefore, if $\mathrm{rank}(\frac{\partial \hat{u}_X(\theta)}{\partial \theta}) = ND_u$, we have that $v = 0$. Here, by the definition of the v, the fact of $v = 0$ implies that

$$\frac{\partial \ell(q, u_i)}{\partial q}\Big|_{q=\hat{u}_\theta(x_i)} = 0, \tag{B.6}$$

for all $i = 1, \ldots, N$. Since the loss ℓ satisfies the optimal stationarity condition, this implies that for all $i = 1, \ldots, N$,

$$\ell(\hat{u}_\theta(x_i), u_i) \leq \ell(q', u_i), \qquad \forall q' \in \mathbb{R}^{D_u}. \tag{B.7}$$

Since the objective function $\mathcal{L}$ is the sum of these terms,

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\ell(\hat{u}_\theta(x_i), u_i) \leq \frac{1}{N}\sum_{i=1}^{N}\ell(q'_i, u_i), \qquad \forall q'_1, \ldots, q'_N \in \mathbb{R}^{D_u}. \tag{B.8}$$

This implies that

$$\mathcal{L}(\theta) \leq \inf_{q'_1, \ldots, q'_N \in \mathbb{R}^{D_u}} \frac{1}{N}\sum_{i=1}^{N}\ell(q'_i, u_i) \leq \mathcal{L}(\theta'), \qquad \forall \theta \in \mathbb{R}^{D_\theta}, \tag{B.9}$$

where D_θ is the dimension of θ. This shows that an arbitrary stationary point θ is a global minimum of $\mathcal{L}$ if $\mathrm{rank}(\frac{\partial \hat{u}_X(\theta)}{\partial \theta}) = ND_u$. This proves the first statement of this theorem.

We now proceed to prove the second statement of this theorem. Using Eq. (B.3), for any θ such that $(\frac{\partial \ell(q,u_i)}{\partial q_j}|_{q=\hat{u}_\theta(x_i)}) = 0$ for all $i \in \{1, \ldots, N\}$, we have $\nabla\mathcal{L}(\theta) = 0$. In other words, every θ such that $(\frac{\partial \ell(q,u_i)}{\partial q_j}|_{q=\hat{u}_\theta(x_i)}) = 0$ for all $i \in \{1, \ldots, N\}$ is a stationary point of $\mathcal{L}$. Using the assumption of $\{\hat{u}_X(\theta) \in \mathbb{R}^{ND_u} : \theta \in \mathbb{R}^{D_\theta}\} = \mathbb{R}^{ND_u}$, if there exists a global minimum of $\mathcal{L}$ (it is possible that a global minimum does not exist), then it achieves the global minimum values of $\ell(\cdot, u_i)$ for all $i \in \{1, \ldots, N\}$. Thus, all we need to show is the existence of a stationary point of $\mathcal{L}$ that does not achieve the global minimum values

of $\ell(\cdot, u_i)$ for all $i \in \{1, \dots, N\}$. Using the assumption of $\{\hat{u}_X(\theta) \in \mathbb{R}^{ND_u} : \theta \in \mathbb{R}^{D_\theta}\} = \mathbb{R}^{ND_u}$ and the assumption of having a point $q \in \mathbb{R}^{D_u}$ such that $\nabla_q \ell(q, u) = 0$ and $\ell(q, u) > \ell(q', u)$ for some $q' \in \mathbb{R}^{D_u}$, there exists a $\bar{\theta}$ such that for all $i \in \{1, \dots, N\}$,

$$\frac{\partial \ell(q, u_i)}{\partial q_j}\Big|_{q=\hat{u}_{\bar{\theta}}(x_i)} = 0 \text{ and } \ell(\hat{u}_{\bar{\theta}}(x_i), u_i) > \ell(q_i', u_i) \text{ for some } q_i' \in \mathbb{R}^{D_u}.$$

Here, $\bar{\theta}$ is a stationary point of $\mathcal{L}$ since $\frac{\partial \ell(q,u_i)}{\partial q_j}|_{q=\hat{u}_{\bar{\theta}}(x_i)} = 0$ for all $i \in \{1, \dots, N\}$. Moreover, $\bar{\theta}$ is not a global minimum of $\mathcal{L}$ since $\ell(\hat{u}_{\bar{\theta}}(x_i), u_i) > \ell(q_i', u_i)$ for some $q_i' \in \mathbb{R}^{D_u}$. This proves the second statement of this theorem. □

B.2. Proof of Theorem 2

To prove this theorem, we utilize the following known lemma:

Lemma 1. *For any differentiable function $\varphi : \mathrm{dom}(\varphi) \to \mathbb{R}$ with an open convex domain $\mathrm{dom}(\varphi) \subseteq \mathbb{R}^{D_\varphi}$, if $\|\nabla\varphi(z') - \nabla\varphi(z)\| \le L_\varphi \|z' - z\|$ for all $z, z' \in \mathrm{dom}(\varphi)$, then*

$$\varphi(z') \le \varphi(z) + \nabla\varphi(z)^\top (z' - z) + \frac{L_\varphi}{2}\|z' - z\|^2 \quad \textit{for all } z, z' \in \mathrm{dom}(\varphi). \tag{B.10}$$

Proof of Lemma 1. Fix $z, z' \in \mathrm{dom}(\varphi) \subseteq \mathbb{R}^{D_\varphi}$. Since $\mathrm{dom}(\varphi)$ is a convex set, $z + r(z' - z) \in \mathrm{dom}(\varphi)$ for all $r \in [0, 1]$. Since $\mathrm{dom}(\varphi)$ is open, there exists $\zeta > 0$ such that $z + (1 + \zeta')(z' - z) \in \mathrm{dom}(\varphi)$ and $z + (0 - \zeta')(z' - z) \in \mathrm{dom}(\varphi)$ for all $\zeta' \le \zeta$. Fix $\zeta > 0$ to be such a number. Combining these, $z + r(z' - z) \in \mathrm{dom}(\varphi)$ for all $r \in [0 - \zeta, 1 + \zeta]$.

Accordingly, we can define a function $\bar{\varphi} : [0 - \zeta, 1 + \zeta] \to \mathbb{R}$ by $\bar{\varphi}(r) = \varphi(z + r(z' - z))$. Then, $\bar{\varphi}(1) = \varphi(z')$, $\bar{\varphi}(0) = \varphi(z)$, and $\nabla\bar{\varphi}(r) = \nabla\varphi(z + r(z' - z))^\top (z' - z)$ for $r \in [0, 1] \subset (0 - \zeta, 1 + \zeta)$. Since $\|\nabla\varphi(z') - \nabla\varphi(z)\| \le L_\varphi \|z' - z\|$,

$$\begin{aligned}
\|\nabla\bar{\varphi}(r') - \nabla\bar{\varphi}(r)\| &= \|[\nabla\varphi(z + r'(z' - z)) - \nabla\varphi(z + r(z' - z))^\top (z' - z)\| && \text{(B.11)}\\
&\le \|z' - z\| \|\nabla\varphi(z + r'(z' - z)) - \nabla\varphi(z + r(z' - z))\| && \text{(B.12)}\\
&\le L_\varphi \|z' - z\| \|(r' - r)(z' - z)\| && \text{(B.13)}\\
&\le L_\varphi \|z' - z\|^2 \|r' - r\|. && \text{(B.14)}
\end{aligned}$$

Thus, $\nabla\bar{\varphi} : [0, 1] \to \mathbb{R}$ is Lipschitz continuous with the Lipschitz constant $L_\varphi \|z' - z\|^2$, and hence $\nabla\bar{\varphi}$ is continuous.

By using the fundamental theorem of calculus with the continuous function $\nabla\bar{\varphi} : [0, 1] \to \mathbb{R}$,

$$\begin{aligned}
\varphi(z') &= \varphi(z) + \int_0^1 \nabla\varphi(z + r(z' - z))^\top (z' - z) dr && \text{(B.15)}\\
&= \varphi(z) + \nabla\varphi(z)^\top (z' - z) + \int_0^1 [\nabla\varphi(z + r(z' - z)) - \nabla\varphi(z)]^\top (z' - z) dr && \text{(B.16)}\\
&\le \varphi(z) + \nabla\varphi(z)^\top (z' - z) + \int_0^1 \|\nabla\varphi(z + r(z' - z)) - \nabla\varphi(z)\| \|z' - z\| dr && \text{(B.17)}\\
&\le \varphi(z) + \nabla\varphi(z)^\top (z' - z) + \int_0^1 r L_\varphi \|z' - z\|^2 dr && \text{(B.18)}\\
&= \varphi(z) + \nabla\varphi(z)^\top (z' - z) + \frac{L_\varphi}{2}\|z' - z\|^2. && \text{(B.19)}
\end{aligned}$$

□

Proof of Theorem 2. The function $\mathcal{L}$ is differentiable since ℓ_i is differentiable, $\theta \mapsto \hat{u}_\theta(x_i)$ is differentiable, and a composition of differentiable functions is differentiable. We will first show that in both cases of (i) and (ii) for the learning rates, we have $\lim_{r\to\infty} \nabla\mathcal{L}(\theta^{(r)}) = 0$. If $\nabla\mathcal{L}(\theta^{(r)}) = 0$ at any $r \geq 0$, then Assumption 1 ($\|\bar{g}^{(r)}\|_2^2 \leq \bar{c}\|\nabla\mathcal{L}(\theta^{(r)})\|_2^2$) implies $\bar{g}^{(r)} = 0$, which implies

$$\theta^{(r+1)} = \theta^{(r)} \text{ and } \nabla\mathcal{L}(\theta^{(r+1)}) = \nabla\mathcal{L}(\theta^{(r)}) = 0. \tag{B.20}$$

This means that if $\nabla\mathcal{L}(\theta^{(r)}) = 0$ at any $r \geq 0$, we have that $\bar{g}^{(r)} = 0$ and $\nabla\mathcal{L}(\theta^{(r')}) = 0$ for all $r' \geq r$ and hence

$$\lim_{r\to\infty} \nabla\mathcal{L}(\theta^{(r)}) = 0, \tag{B.21}$$

as desired. Therefore, we now focus on the remaining scenario where $\nabla\mathcal{L}(\theta^{(r)}) \neq 0$ for all $r \geq 0$.

By using Lemma 1,

$$\mathcal{L}(\theta^{(r+1)}) \leq \mathcal{L}(\theta^{(r)}) - \epsilon^{(r)}\nabla\mathcal{L}(\theta^{(r)})^\top \bar{g}^{(r)} + \frac{L(\epsilon^{(r)})^2}{2}\|\bar{g}^{(r)}\|^2. \tag{B.22}$$

By rearranging and using Assumption 1,

$$\mathcal{L}(\theta^{(r)}) - \mathcal{L}(\theta^{(r+1)}) \geq \epsilon^{(r)}\nabla\mathcal{L}(\theta^{(r)})^\top \bar{g}^{(r)} - \frac{L(\epsilon^{(r)})^2}{2}\|\bar{g}^{(r)}\|^2 \tag{B.23}$$

$$\geq \epsilon^{(r)}\underline{c}\|\nabla\mathcal{L}(\theta^{(r)})\|^2 - \frac{L(\epsilon^{(r)})^2}{2}\bar{c}\|\nabla\mathcal{L}(\theta^{(r)})\|^2. \tag{B.24}$$

By simplifying the right-hand-side,

$$\mathcal{L}(\theta^{(r)}) - \mathcal{L}(\theta^{(r+1)}) \geq \epsilon^{(r)}\|\nabla\mathcal{L}(\theta^{(r)})\|^2(\underline{c} - \frac{L\epsilon^{(r)}}{2}\bar{c}). \tag{B.25}$$

Let us now focus on case (i). Then, using $\epsilon^{(r)} \leq \frac{\underline{c}\ (2-\zeta)}{L\bar{c}}$,

$$\frac{L\epsilon^{(r)}}{2}\bar{c} \leq \frac{L\underline{c}\ (2-\zeta)}{2L\bar{c}}\bar{c} = \underline{c} - \frac{\zeta}{2}\underline{c}\ . \tag{B.26}$$

Using this inequality and using $\zeta \leq \epsilon^{(r)}$ in Eq. (B.25),

$$\mathcal{L}(\theta^{(r)}) - \mathcal{L}(\theta^{(r+1)}) \geq \frac{\underline{c}\zeta^2}{2}\|\nabla\mathcal{L}(\theta^{(r)})\|^2. \tag{B.27}$$

Since $\nabla\mathcal{L}(\theta^{(r)}) \neq 0$ for any $r \geq 0$ (see above) and $\zeta > 0$, this means that the sequence $(\mathcal{L}(\theta^{(r)}))_r$ is monotonically decreasing. Since $\mathcal{L}(q) \geq 0$ for any q in its domain, this implies that the sequence $(\mathcal{L}(\theta^{(r)}))_r$ converges. Therefore, $\mathcal{L}(\theta^{(r)}) - \mathcal{L}(\theta^{(r+1)}) \to 0$ as $r \to \infty$. Using Eq. (B.27), this implies that

$$\lim_{r\to\infty} \nabla\mathcal{L}(\theta^{(r)}) = 0, \tag{B.28}$$

which proves the desired result for the case (i).

We now focus on the case (ii). Then, we still have Eq. (B.25). Since $\lim_{r\to\infty} \epsilon^{(r)} = 0$ in Eq. (B.25), the first order term in $\epsilon^{(r)}$ dominates after sufficiently large r: i.e., there exists $\bar{r} \geq 0$ such that for any $r \geq \bar{r}$,

$$\mathcal{L}(\theta^{(r)}) - \mathcal{L}(\theta^{(r+1)}) \geq c\epsilon^{(r)}\|\nabla\mathcal{L}(\theta^{(r)})\|^2 \tag{B.29}$$

for some constant $c > 0$. Since $\nabla\mathcal{L}(\theta^{(r)}) \neq 0$ for any $r \geq 0$ (see above) and $c\epsilon^{(r)} > 0$, this means that the sequence $(\mathcal{L}(\theta^{(r)}))_r$ is monotonically decreasing. Since $\mathcal{L}(q) \geq 0$ for any q in its domain, this implies that the sequence $(\mathcal{L}(\theta^{(r)}))_r$ converges to a finite value. Thus, by adding Eq. (B.29) both sides over all $r \geq \bar{r}$,

$$\infty > \mathcal{L}(\theta^{(\bar{r})}) - \lim_{r\to\infty} \mathcal{L}(\theta^{(r)}) \geq c \sum_{r=\bar{r}}^{\infty} \epsilon^{(r)} \|\nabla\mathcal{L}(\theta^{(r)})\|^2. \tag{B.30}$$

Since $\sum_{r=0}^{\infty} \epsilon^{(r)} = \infty$, this implies that $\liminf_{r\to\infty} \|\nabla\mathcal{L}(\theta^{(r)})\| = 0$. We now show that $\limsup_{r\to\infty} \|\nabla\mathcal{L}(\theta^{(r)})\| = 0$ by contradiction. Suppose that $\limsup_{r\to\infty} \|\nabla\mathcal{L}(\theta^{(r)})\| > 0$. Then, there exists $\delta > 0$ such that $\limsup_{r\to\infty} \|\nabla\mathcal{L}(\theta^{(r)})\| \geq \delta$. Since $\liminf_{r\to\infty} \|\nabla\mathcal{L}(\theta^{(r)})\| = 0$ and $\limsup_{r\to\infty} \|\nabla\mathcal{L}(\theta^{(r)})\| \geq \delta$, let $\rho_{j\,j}$ and $\rho'_{j\,j}$ be sequences of indexes such that $\rho_j < \rho'_j < \rho_{j+1}$, $\|\nabla\mathcal{L}(\theta^{(r)})\| > \frac{\delta}{3}$ for $\rho_j \leq r < \rho'_j$, and $\|\nabla\mathcal{L}(\theta^{(r)})\| \leq \frac{\delta}{3}$ for $\rho'_j \leq r < \rho_{j+1}$. Since $\sum_{r=\bar{r}}^{\infty} \epsilon^{(r)} \|\nabla\mathcal{L}(\theta^{(r)})\|^2 < \infty$, let $\bar{j}$ be sufficiently large such that $\sum_{r=\rho_{\bar{j}}}^{\infty} \epsilon^{(r)} \|\nabla\mathcal{L}(\theta^{(r)})\|^2 < \frac{\delta^2}{9L\sqrt{\bar{c}}}$. Then, for any $j \geq \bar{j}$ and any ρ such that $\rho_j \leq \rho \leq \rho'_j - 1$, we have that

$$\|\nabla\mathcal{L}(\theta^{(\rho)})\| - \|\nabla\mathcal{L}(\theta^{(\rho'_j)})\| \leq \|\nabla\mathcal{L}(\theta^{(\rho'_j)}) - \nabla\mathcal{L}(\theta^{(\rho)})\| \tag{B.31}$$

$$= \left\| \sum_{r=\rho}^{\rho'_j-1} \nabla\mathcal{L}(\theta^{(r+1)}) - \nabla\mathcal{L}(\theta^{(r)}) \right\| \tag{B.32}$$

$$\leq \sum_{r=\rho}^{\rho'_j-1} \left\| \nabla\mathcal{L}(\theta^{(r+1)}) - \nabla\mathcal{L}(\theta^{(r)}) \right\| \tag{B.33}$$

$$\leq L \sum_{r=\rho}^{\rho'_j-1} \left\| \theta^{(r+1)} - \theta^{(r)} \right\| \tag{B.34}$$

$$\leq L\sqrt{\bar{c}} \sum_{r=\rho}^{\rho'_j-1} \epsilon^{(r)} \left\| \nabla\mathcal{L}(\theta^{(r)}) \right\|, \tag{B.35}$$

where the first and third lines use the triangle inequality (and symmetry), the forth line uses the assumption that $\|\nabla\mathcal{L}(\theta) - \nabla\mathcal{L}(\theta')\| \leq L\|\theta - \theta'\|$, and the last line follows the definition of $\theta^{(r+1)} - \theta^{(r)} = -\epsilon^{(r)}\bar{g}$ and the assumption of $\|\bar{g}^{(r)}\|^2 \leq \bar{c}\|\nabla\mathcal{L}(\theta^{(r)})\|^2$. Then, by using the definition of the sequences of the indexes,

$$\|\nabla\mathcal{L}(\theta^{(\rho)})\| - \|\nabla\mathcal{L}(\theta^{(\rho'_j)})\| \leq \frac{3L\sqrt{\bar{c}}}{\delta} \sum_{r=\rho}^{\rho'_j-1} \epsilon^{(r)} \left\| \nabla\mathcal{L}(\theta^{(r)}) \right\|^2 \leq \frac{\delta}{3}. \tag{B.36}$$

Here, since $\|\nabla\mathcal{L}(\theta^{(\rho'_j)})\| \leq \frac{\delta}{3}$, by rearranging the inequality, we have that for any $\rho \geq \rho_{\bar{j}}$,

$$\|\nabla\mathcal{L}(\theta^{(\rho)})\| \leq \frac{2\delta}{3}. \tag{B.37}$$

This contradicts the inequality of $\limsup_{r\to\infty} \|\nabla\mathcal{L}(\theta^{(r)})\| \geq \delta$. Thus, we have

$$\limsup_{r\to\infty} \|\nabla\mathcal{L}(\theta^{(r)})\| = \liminf_{r\to\infty} \|\nabla\mathcal{L}(\theta^{(r)})\| = 0. \tag{B.38}$$

This implies that

$$\lim_{r\to\infty} \nabla\mathcal{L}(\theta^{(r)}) = 0, \tag{B.39}$$

which proves the desired result for the case (ii). Therefore, in both cases of (i) and (ii) for the learning rates, we have $\lim_{r\to\infty} \nabla\mathcal{L}(\theta^{(r)}) = 0$. From Theorem 1, this implies that an arbitrary limit point θ of the sequence $\{\theta^{(r)}\}_{r=0}$ is a global minimum of $\mathcal{L}$ if $\text{rank}(\frac{\partial \hat{u}_X(\theta)}{\partial\theta}) = ND_u$.

□

B.3. Proof of Corollary 1

Proof. By following the proof of the first statement of Theorem 1 (see Eq. (B.40)), we have that at any stationary point θ of $\mathcal{L}$,

$$\frac{\partial \ell(q, u_i)}{\partial q}\Big|_{q=\hat{u}_\theta(x_i)} = 0 \ \text{ for all } i = 1, \ldots, N. \tag{B.40}$$

By the assumption of $\nabla_q \ell(q, u) = 0$ if and only if $q = u$, this implies that at any stationary point θ of $\mathcal{L}$,

$$\hat{u}_\theta(x_i) = u_i \ \text{ for all } i = 1, \ldots, N.$$

This implies that at any stationary point θ of $\mathcal{L}$, we have that $\mathcal{L}_{\mathrm{MSE}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \|\hat{u}_\theta(x_i) - u_i\|_2^2 = 0$, which is the global minimum value of $\mathcal{L}_{\mathrm{MSE}}$. □

B.4. Proof of Proposition 1

Proof. Let $\ell(q, u) = \rho_{\alpha,c}(q - u) = \frac{|\alpha-2|}{\alpha}((\frac{((q-u)/c)^2}{|\alpha-2|} + 1)^{\alpha/2} - 1)$. Let c and α be real numbers such that $c > 0$, $\alpha \neq 0$, and $\alpha \neq 2$. Then,

$$\frac{\partial \ell(q, u)}{\partial q} = \frac{|\alpha - 2|}{\alpha}\left(\frac{\alpha}{2}\left(\frac{((q-u)/c)^2}{|\alpha-2|} + 1\right)^{(\alpha/2)-1}\right)\frac{1}{|\alpha-2|}\frac{1}{c^2}2(q-u) \tag{B.41}$$

$$= \left(\frac{1}{c^2}\left(\frac{((q-u)/c)^2}{|\alpha-2|} + 1\right)^{(\alpha/2)-1}\right)(q-u). \tag{B.42}$$

Here, since any (real) power of strictly positive real number is strictly positive, we have

$$\frac{1}{c^2}\left(\frac{((q-u)/c)^2}{|\alpha-2|} + 1\right)^{(\alpha/2)-1} > 0. \tag{B.43}$$

By combining Eq. (B.42) and Eq. (B.43), we have that $\frac{\partial \ell(q,u)}{\partial q} = 0$ implies $q - u = 0$. On the other hand, using Eq. (B.42), we have that $q - u = 0$ implies $\frac{\partial \ell(q,u)}{\partial q} = 0$. In other words,

$$\frac{\partial \ell(q, u)}{\partial q} = 0 \qquad \Longleftrightarrow \qquad q = u. \tag{B.44}$$

This proves the statement for the second condition that $\nabla_q \ell(q, u) = 0$ if and only if $q = u$. We now prove the statement for the optimal stationarity condition by showing that $q = u$ implies that $\ell(q, u) \leq \ell(q', u)$ for all $q' \in \mathbb{R}$. If $q = u$, then

$$\ell(q, u) = \frac{|\alpha - 2|}{\alpha}\left((1)^{\alpha/2} - 1\right) = 0. \tag{B.45}$$

On the other hand, we have that

$$\ell(q, u) \geq 0, \qquad \forall q, u \in \mathbb{R}, \tag{B.46}$$

since $((\frac{(d/c)^2}{|\alpha-2|} + 1)^{\alpha/2} - 1) \geq 0$ if $\alpha \geq 0$ and $((\frac{(d/c)^2}{|\alpha-2|} + 1)^{\alpha/2} - 1) \leq 0$ if $\alpha \leq 0$. Therefore, for any $q, u \in \mathbb{R}$, having $q = u$ implies that $\ell(q, u) \leq \ell(q', u)$ for all $q' \in \mathbb{R}$. By using Eq. (B.44), this proves the statement for the optimal stationarity condition. □

C. Other algorithm design options

Apart from the most important options presented in Section 3.4 and Appendix A that pertain to the loss function parametrization, the number of inner optimization steps and imposing desirable loss function properties, some additional design options are presented in this section. First, consider the options of sampling new tasks Λ from the task distribution $p(\lambda)$ (line 4 in Algorithm 1) and initializing the approximator NN parameters θ_τ for $\tau \in \{1, \ldots, T\}$ (line 6 in Algorithm 1). Resampling a set Λ of T tasks in every outer iteration exposes the learned loss to more samples from the task distribution. Similarly, solving these tasks with T new randomly initialized NNs exposes the loss function to more samples from the NN initialization distribution. Although such introduced randomness is generally expected to improve test performance, it leads to unstable training that depends also on the number of inner optimization steps. As a result, we have the option to resample new tasks and new NN parameter initializations every I' and I'' outer iterations, respectively, instead of every single iteration; i.e., θ_τ is re-initialized in line 6 of Algorithm 1 with a setting $\theta_\tau^{(0)}$ that is replaced every I'' iterations. An indicative experiment for demonstrating the effect of these options on training and test performance of the learned loss is performed in Appendix D for the regression example of Section 4.1.

Finally, as a stopping criterion for Algorithm 1, i.e., for selecting the maximum number of outer iterations I, a performance metric can be recorded during meta-training and the algorithm can be stopped if no progress is observed for a number of iterations. Two candidate options for this metric are the following: (a) the outer objective of Eq. (18), corresponding to a *meta-training error*, and (b) the meta-test performance on a few test iterations with learned loss snapshots captured during meta-training, corresponding to a *meta-validation error*. However, because of the aforementioned induced randomness during meta-training, the outer objective may be too noisy for it to be a useful metric, especially when only one task is used for each outer update ($T = 1$ in Algorithm 1); thus, either a less noisy option can be utilized (see Fig. D.3) or a moving average can be recorded (see Fig. (18)). Furthermore, depending on the parametrization and other design options, the meta-validation error can also be noisy (see Figs. D.4, 6, and 19). For this reason, in the computational examples of Section 4, we meta-train for a sufficiently large number of outer iterations (10,000) that works reasonably well according to both metrics. We also capture snapshots of the learned loss and use all of them in meta-testing for gaining a deeper understanding of the algorithm's applicability for solving PDEs with PINNs.

D. Additional computational results related to the function approximation example

Effect of loss function initialization. To demonstrate the effect of the initialization of the NN-parametrized loss function, the outer learning rate, as well as the gradient clipping approach discussed in Section 3.4.2, we first consider employing Algorithm 1 with a randomly initialized loss function, a large learning rate equal to 5×10^{-2} and $J = 20$ inner optimization steps. SGD is considered as both the inner and outer optimizer, the approximator NN architecture consists of 3 hidden layers with 40 neurons each and *tanh* activation function, the number of datapoints used for inner training is $N_u = 100$, and the number of datapoints used for updating the loss is $N_{u,val} = 1{,}000$; the approximator NN as well as N_u and $N_{u,val}$ are the same for all experiments in this section. Note that this is the only case in the computational examples that SGD is used as outer optimizer; in all other cases Adam is used. In

Fig. D.1, the norm of the loss parameters as well as the norm and the maximum of their gradient for each outer iteration are shown. The gradient on iteration 4 explodes and leads to a large jump on the loss parameters norm as well. Furthermore, Fig. D.2 shows the loss gradient norm for each outer iteration and as a function of inner iterations. Clearly, although for increasing inner steps J we differentiate over a longer optimization path as explained in Section 3.4.2, the loss gradient norm does not necessarily increase with increasing J. For this reason, throughout the rest of the computational examples we use a gradient clipping approach for addressing the exploding gradient issue, instead of, for example, uniformly dividing all gradients by J.

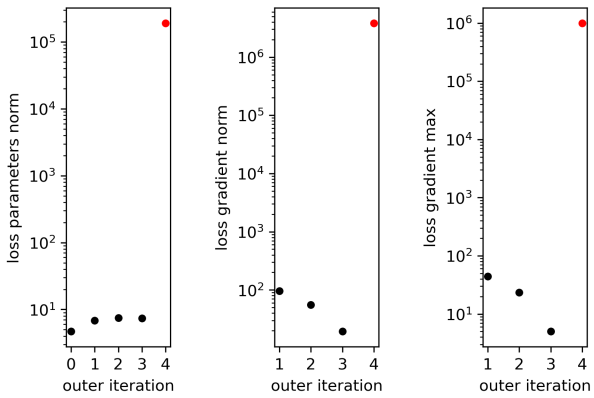


Fig. D.1. Function approximation: Loss parameters norm as well as norm and maximum of their gradient for each outer iteration for the case of randomly initializing the loss function parameters and not using gradient clipping. The gradient on iteration 4 explodes and leads to a large jump on the loss parameters norm as well.

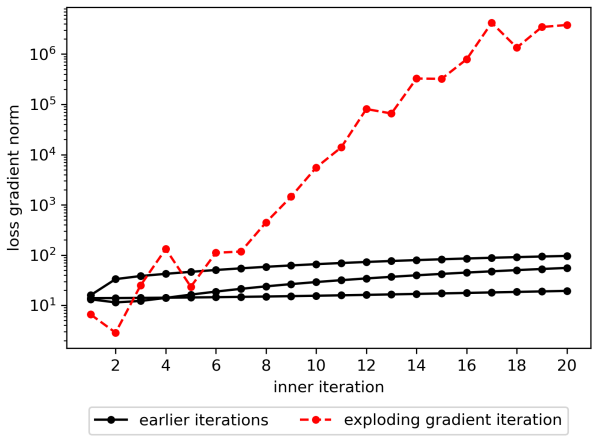
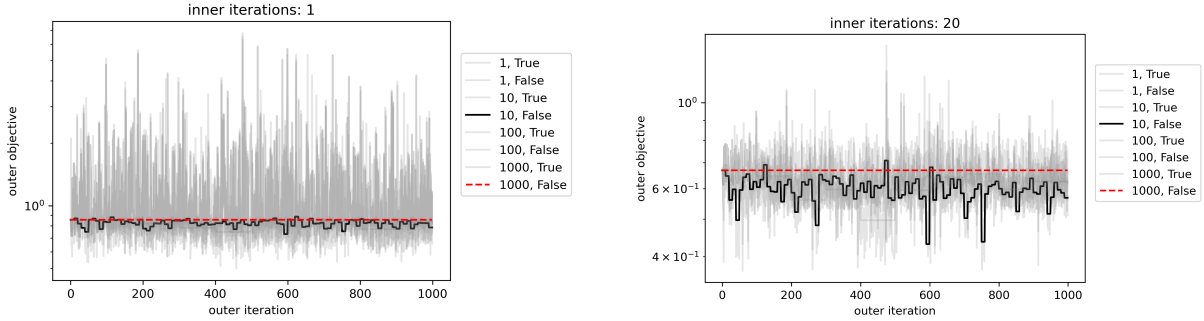


Fig. D.2. Function approximation: Loss parameters gradient norm as a function of inner and outer iterations. Gradient norm does not increase with increasing number of inner iterations for all outer iterations.

Design options experiment. Next, to demonstrate the effect of the design options discussed in Section 3.4 and to evaluate the generalization capabilities of the algorithm we consider the following experiment: Algorithm 1 is employed with an LAL-parametrized loss function initialized with an MSE approximation (see Section 3.4.1.1), with Adam as inner optimizer, and with $I = 1{,}000$ outer steps. Furthermore, the number of inner steps J varies between values in $\{1, 20\}$, the frequency according to

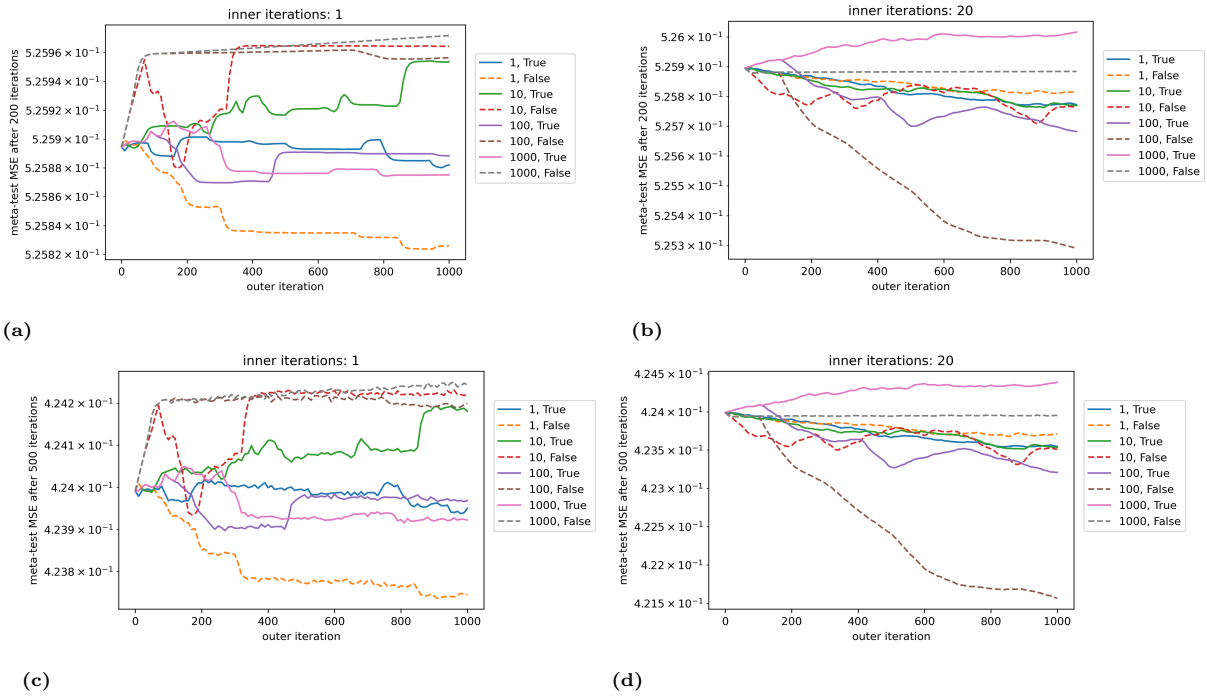
which we sample new tasks varies between values in $\{1, 10, 100, 1000 = \text{no resampling}\}$, and whether in each outer iteration we use a newly initialized approximator NN is either True or False; see Appendix C for relevant discussion. In Fig. D.3 we show the outer objective during meta-training and as a function of outer iteration for $J = 1$ and $J = 20$. Clearly, resampling tasks and re-initializing the approximator NN introduces noise to the training as depicted by the corresponding outer objective trajectories shown with gray lines in Fig. D.3; see also relevant results in Fig. 18 pertaining to the Burgers equation example.

Furthermore, in Fig. D.4, the generalization capacity of the obtained loss functions is evaluated by performing meta-testing on 5 ID unseen tasks for 100 and 500 test iterations. Note that each line in Fig. D.4 corresponds to the performance of 101 different loss functions and is obtained by meta-training with different design options; i.e., we perform a test every 10 outer iterations for each meta-training session. Overall, increasing the number of inner iterations improves the learned loss test performance as well as its robustness with increasing outer iterations and with varying design options. Moreover, the patterns observed for 100 test iterations are almost identical to the ones observed for 500 test iterations, which means that if we attempt to optimize the design options based on meta-testing with 100 or 500 iterations we will end up with the same optimal ones.



(a) **(b)**

Fig. D.3. Function approximation: Outer objective values recorded during meta-training (1,000 outer iterations) as a function of design options; these can be construed as *meta-training error* trajectories. Results obtained using 1 inner iteration (a) and 20 inner iterations (b) in meta-training. Design options considered are number of inner iterations $J \in \{1, 20\}$, frequency of resampling tasks $I' \in \{1, 10, 100, 1000 = \text{no resampling}\}$, and whether approximator NN is re-initialized with a new initialization setting or with the same one (True, False); see more information in Appendix C. Resampling tasks and re-initializing the approximator NN introduces noise to the training. Black lines (task resampling every 10 outer iterations) are more noisy than the red lines (no task resampling), whereas the rest of the lines are shown with the same gray color for indicating noisy lines.



(a) **(b)**

(c) **(d)**

Fig. D.4. Function approximation: Meta-testing performance (100 (a, b) and 500 (c, d) test iterations) as a function of outer iteration during meta-training (1,000 outer iterations) and of design options; these can be construed as *meta-validation error* trajectories. Results obtained using 1 inner iteration (a, c) and 20 inner iterations (b, d) in meta-training. Design options considered are number of inner iterations $J \in \{1, 20\}$, frequency of resampling tasks $I' \in \{1, 10, 100, 1000 = \text{no resampling}\}$, and whether approximator NN is re-initialized with a new initialization setting or with the same one (True, False); see more information in Appendix C. Clearly, increasing the number of inner iterations improves the learned loss test performance as well as its robustness with increasing outer iterations and with varying design options. Furthermore, the patterns observed for 100 test iterations are almost identical to the ones observed for 500 test iterations.