



# Optimally weighted loss functions for solving PDEs with Neural Networks

Remco van der Meer<sup>a,b</sup>, Cornelis W. Oosterlee<sup>c</sup>, Anastasia Borovykh<sup>d,\*</sup>

<sup>a</sup> CWI, Science Park 123, 1098 XG Amsterdam, Netherlands

<sup>b</sup> Delft University of Technology, Van Mourik Broekmanweg 6, 2628 XE Delft, Netherlands

<sup>c</sup> Mathematical Institute, Utrecht University, Netherlands

<sup>d</sup> Warwick Business School, University of Warwick, Coventry CV4 7AL, UK

## ARTICLE INFO

### Article history:

Received 12 September 2020

Received in revised form 24 March 2021

### Keywords:

Partial differential equation

Neural network

Convection–diffusion equation

Poisson equation

Loss functional

High-dimensional problems

## ABSTRACT

Recent works have shown that deep neural networks can be employed to solve partial differential equations, giving rise to the framework of physics informed neural networks (Raissi et al., 2007). We introduce a generalization for these methods that manifests as a scaling parameter which balances the relative importance of the different constraints imposed by partial differential equations. A mathematical motivation of these generalized methods is provided, which shows that for linear and well-posed partial differential equations, the functional form is convex. We then derive a choice for the scaling parameter that is optimal with respect to a measure of relative error. Because this optimal choice relies on having full knowledge of analytical solutions, we also propose a heuristic method to approximate this optimal choice. The proposed methods are compared numerically to the original methods on a variety of model partial differential equations, with the number of data points being updated adaptively. For several problems, including high-dimensional PDEs the proposed methods are shown to significantly enhance accuracy.

© 2021 Elsevier B.V. All rights reserved.

## 1. Introduction

Advances in computing power and rapid growth of available data in recent years have invigorated the field of machine learning and data science. Although the theory to train deep learning models has been available since the early 60s, only recently has it become possible to train them on commonly available hardware [1]. This has led to exceptional achievements in a wide range of problems, including image recognition, natural language processing, genomics and reinforcement learning. Aside from these empirical achievements, the theoretical understanding of such methods is also advancing rapidly. This has sparked interest in solving more fundamental problems by utilizing these methods.

Previous work has shown the successes of Bayesian approaches for learning partial differential equation (PDE) representations in linear settings [2,3]; extensions to nonlinear regimes introduced certain limitations [4,5]. Motivated by the universal approximation theorems [6,7], recent studies have considered a different approach for solving (non)linear PDEs utilizing deep neural networks [8–14]. Specifically, the works of [9,10] minimize a loss function which represents the PDE constraints, while the work of [13] uses the backward stochastic differential equation (BSDE) representation. The work of [14] studies the performance of different network architectures. These studies present several different advantages

\* Corresponding author at: Warwick Business School, University of Warwick, Coventry CV4 7AL, UK.  
E-mail address: [borovykh\\_a@hotmail.com](mailto:borovykh_a@hotmail.com) (A. Borovykh).

that neural network based methods have over classical numerical methods. The most notable advantage mentioned is that neural network based methods do not require any form of discretization. Constructing a mesh can be especially prohibitive for high-dimensional problems, problems with complex geometries, or domains with tiny structures. By alleviating this requirement, neural network based methods may have great potential to outperform existing methods in these areas. As shown in e.g. [15–18] neural networks can provably overcome the curse of dimensionality in certain classes of PDEs. The study of [8] mentions additional advantages, which include the differentiability of the obtained solution and the possibility to efficiently parallelize the methods. While neural networks have been successful in solving a variety of PDEs, certain challenges remain. As mentioned in e.g. [19] neural networks can suffer from unbalanced back-propagated gradients. Alternatively, when the PDE is only partially known and data-collection is expensive, it is not trivial how to employ neural networks; the work of [20] proposes transfer learning from low- to high-fidelity models as a solution.

The key concept brought forward in these studies is to approximate (part of) the solution of the PDE one aims to solve with a deep neural network. Consider a general PDE for the scalar function  $u(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}$  on the domain  $\Omega \subset \mathbb{R}^d$  given by

$$\begin{cases} N(\mathbf{x}, u) = F(\mathbf{x}) & \text{in } \Omega, \\ B(\mathbf{x}, u) = G(\mathbf{x}) & \text{on } \partial\Omega, \end{cases} \quad (1)$$

with  $\mathbf{x} \in \Omega \subset \mathbb{R}^d$  and  $N$  and  $B$  the differential operators on the interior and boundary, resp., and  $F, G$  are source functions. These operators and functions define constraints on  $u$  that must be satisfied to solve the PDE. The neural network-based methods employ a deep neural network of which the input layer has  $d$  neurons and the output layer has a single neuron representing the entire solution of the PDE. To solve the PDE, the neural network must discover solutions that satisfy the constraints imposed by the PDE and the boundary conditions. This discovery is done without using any explicit information about the true solution; only the PDE and the boundary conditions that must be satisfied are provided. Therefore, these methods can be categorized as unsupervised learning methods.

There are different ways to satisfy the constraints imposed by the PDE and the boundary conditions. The study of [8] treats the boundary conditions of the PDE as a hard constraint by constructing an auxiliary function that satisfies the boundary conditions. The remaining constraint that is imposed by the PDE itself is treated as a soft constraint, which is approximately satisfied by minimizing some loss function. The studies [9–11] use a different approach and treat all the constraints as soft constraints. This is implemented by constructing a single loss function in which all of these constraints are included. These approaches are more general, as they do not require special functions that depend on the problem one aims to solve. The result is that the neural network directly serves as the approximation of the entire solution; feeding a location  $\mathbf{x}$  into the neural network results in an output  $\hat{u}(\mathbf{x})$  that approximates the true solution  $u(\mathbf{x})$ .

To solve the general PDE given in Eq. (1), the authors of [10] suggest rewriting the PDE into the form

$$\begin{cases} \mathcal{N}(\mathbf{x}, u) := N(\mathbf{x}, u) - F(\mathbf{x}) = 0 & \text{in } \Omega, \\ \mathcal{B}(\mathbf{x}, u) := B(\mathbf{x}, u) - G(\mathbf{x}) = 0 & \text{on } \partial\Omega, \end{cases} \quad (2)$$

and minimizing the loss functional

$$L(\hat{u}) = \frac{1}{n_I} \sum_{i=1}^{n_I} \mathcal{N}(\mathbf{x}_i^I, \hat{u})^2 + \frac{1}{n_B} \sum_{i=1}^{n_B} \mathcal{B}(\mathbf{x}_i^B, \hat{u})^2, \quad (3)$$

with a deep neural network. Here,  $\mathbf{x}_i^I \in \Omega$  and  $\mathbf{x}_i^B \in \partial\Omega$  are collocation points. The authors of [9–11] provide empirical evidence that for several instances of well-posed problems, e.g. the Burgers' equation and the Poisson equation, these methods yield accurate results. Although these empirical results are promising, it is still poorly understood why or when these methods work. The authors of [10] suggest that well-posedness may be what determines whether or not these methods work. However, in Section 6 we show that instances of well-posed PDEs for which minimizing Eq. (3) leads to inaccurate approximations are not uncommon.

The main aim of this work is to augment the generality of the methods introduced in [10] by extending the class of problems that may be solved with these methods. To this end, we aim to address the aforementioned problems. Herein our focus will be on research around the neural network engine; The scientific community has spent decades developing sophisticated training algorithms, and neural network training techniques are expected to improve further. Our focus is on translating the PDEs to a minimization problem that is well suited for neural networks to solve.

In Section 2, we give a mathematical motivation for the original methods that were introduced in [10]. Here, we show that in the asymptotic regime of large neural networks, these methods should indeed be able to solve linear, well-posed problems. Then, Section 3 introduces a scaling parameter which generalizes these methods. This parameter is optimized with respect to a measure of relative error. Section 4 proposes a heuristic method to perform this optimization which does not require the explicit knowledge of the analytical solution. These theoretical results are analyzed numerically in Section 6. Here, several linear well-posed model PDEs such as the Laplace equation and the convection–diffusion equation are considered. Instances of these PDEs for which the original methods worked well and instances for which they do not are both used to examine the impact of the proposed generalization.

## 2. Method motivation

This section aims to provide a mathematical motivation for the use of the methods introduced in [10]. This is done by showing that the loss function of Eq. (3) can be viewed as a Monte-Carlo approximation of a functional, and subsequently by showing that this loss functional satisfies several highly desirable properties for well-posed linear PDEs. Although these properties are challenging to exploit directly, they can be used to justify the use of approximate methods to solve such problems. Loss functionals are challenging to compute exactly and therefore rarely used in practice for the purpose of training neural networks. However, loss functionals are significantly easier to analyze mathematically; by focusing on them instead of their Monte-Carlo counterparts, one can avoid all problems related to the distribution of the collocation points. To emphasize the difference, loss functionals are denoted with a hat throughout this work, while their Monte-Carlo approximations are denoted without one.

For the sake of generality, the analysis is performed on a generalized loss functional. Instead of considering the mean squared error, i.e. the second power of the  $L^2$  norm, the  $p$ th power of the  $L^p$  norm is considered for  $p \geq 1$ . The resulting loss functional is given by

$$\hat{L}(\hat{u}) = \frac{1}{|\Omega|} \int_{\Omega} |\mathcal{N}(\mathbf{x}, \hat{u})|^p d\mathbf{x} + \frac{1}{|\partial\Omega|} \int_{\partial\Omega} |\mathcal{B}(\mathbf{x}, \hat{u})|^p d\mathbf{x}_r. \quad (4)$$

Note that for  $p = 2$  the Monte-Carlo approximation of this functional is given by Eq. (3). For notational convenience, the constants present in Eq. (4) are replaced by the constants  $0 < c_1, c_2 \in \mathbb{R}$ . This generalization leads to the loss functional

$$\hat{L}(\hat{u}) = c_1 \int_{\Omega} |\mathcal{N}(\mathbf{x}, \hat{u})|^p d\mathbf{x} + c_2 \int_{\partial\Omega} |\mathcal{B}(\mathbf{x}, \hat{u})|^p d\mathbf{x}_r. \quad (5)$$

The integrals present in Eqs. (4) and (5) are labeled as

$$\hat{L}_I(\hat{u}) := \int_{\Omega} |\mathcal{N}(\mathbf{x}, \hat{u})|^p d\mathbf{x} \quad (6)$$

$$\hat{L}_B(\hat{u}) := \int_{\partial\Omega} |\mathcal{B}(\mathbf{x}, \hat{u})|^p d\mathbf{x}_r, \quad (7)$$

and are referred to as the interior and boundary loss, respectively.

The functional given in Eq. (4) has a much clearer correspondence to the PDE given in Eq. (1) than the Monte-Carlo approximation. Since the loss functional is positive, and only zero when  $\mathcal{N}$  and  $\mathcal{B}$  are zero everywhere, its global minimizer coincides with the solution of the PDE. This is a property that the Monte-Carlo approximation of the loss functional lacks, since minimizing that loss function only ensures that the PDE is satisfied on a finite number of collocation points.

Despite this, minimizing the loss function of Eq. (3) to solve a PDE can still be motivated with the properties of the loss functional of Eq. (4) for well-posed PDEs. Definition 1 gives the formal definition of well-posedness as given in [21].

**Definition 1.** Consider a PDE of the form

$$\begin{cases} \mathcal{N}(\mathbf{x}, u) := N(\mathbf{x}, u) - F(\mathbf{x}) = 0 & \text{in } \Omega, \\ \mathcal{B}(\mathbf{x}, u) := B(\mathbf{x}, u) - G(\mathbf{x}) = 0 & \text{on } \partial\Omega, \end{cases} \quad (8)$$

on the finite and sufficiently smooth domain  $\Omega$ , with  $N, B$  the operators that define the PDE, and  $F, G$  source functions. Such a PDE is called well-posed if for all  $F, G$  there exists a unique solution, and if for every two sets of data  $F_1, G_1$  and  $F_2, G_2$ , the corresponding solutions  $u_1$  and  $u_2$  satisfy

$$\|u_1 - u_2\| \leq C\{\|F_1 - F_2\| + \|G_1 - G_2\|\} \quad (9)$$

for some fixed, finite constant  $C \in \mathbb{R}$ . Such a constant  $C$  will be referred to as the Lipschitz constant of the PDE. Here,  $\|\cdot\|$  denotes the  $L^1$  norm.

Motivating the use of the Monte-Carlo approximation of the loss functional of Eq. (4) requires a statement regarding the approximate optimization of these functionals; so far, it is only clear that minimizing the loss functional exactly yields a solution of the PDE. However, when one minimizes the Monte-Carlo approximation instead, it is highly unlikely that this exact minimum is attained. Even if one were able to find a solution for which the approximated loss function zeros out completely, the loss functional would likely remain nonzero. The following theorem bridges the gap between this approximate and exact optimization of the loss functionals.

**Theorem 1.** Consider the well-posed PDE of order  $k$  given by

$$\begin{cases} \mathcal{N}(\mathbf{x}, u) := N(\mathbf{x}, u) - F(\mathbf{x}) = 0 & \text{in } \Omega, \\ \mathcal{B}(\mathbf{x}, u) := B(\mathbf{x}, u) - G(\mathbf{x}) = 0 & \text{on } \partial\Omega. \end{cases} \quad (10)$$

Let the exact solution of this PDE be given by  $u$  and let the loss functional be given by Eq. (5) for some fixed  $p \geq 1$  and  $c_1, c_2 > 0$ . Consider some approximate solution  $\hat{u}$  of which the first  $k$  (partial) derivatives exist and have finite  $L^p$  norm. Then, for any  $\epsilon > 0$  there exists a  $\delta > 0$  such that for the approximate solution  $\hat{u}$ ,

$$\hat{L}(\hat{u}) < \delta \implies \|\hat{u} - u\| < \epsilon. \quad (11)$$

**Proof.** Let  $\epsilon > 0$  be arbitrary. The aim is to find a  $\delta$  for which Eq. (11) holds. Because the PDE given in Eq. (10) is assumed to be well-posed, there exists a finite Lipschitz constant  $C$  for this particular PDE. Given this Lipschitz constant, and given the constants  $p$ ,  $c_1$  and  $c_2$ , choose

$$\delta := \epsilon^p \left[ C \left( c_1^{-\frac{1}{p}} |\Omega|^{1-\frac{1}{p}} + c_2^{-\frac{1}{p}} |\partial\Omega|^{1-\frac{1}{p}} \right) \right]^{-p}. \quad (12)$$

and let  $\hat{u}$  be some approximate solution of the PDE for which  $\hat{L}(\hat{u}) < \delta$ . Because  $\hat{u}$  may not exactly satisfy the operators  $N$  and  $B$  of Eq. (10), one can write

$$N(\mathbf{x}, \hat{u}) = F + \hat{F}, \quad B(\mathbf{x}, \hat{u}) = G + \hat{G}. \quad (13)$$

In other words,  $\hat{u}$  satisfies a perturbed version of the PDE. Because the PDE is well-posed, it follows that

$$\|\hat{u} - u\| < C\{\|(\hat{F} + F) - F\| + \|(\hat{G} + G) - G\|\} = C\{\|\hat{F}\| + \|\hat{G}\|\}. \quad (14)$$

Using Hölder's inequality [22], one finds for  $\|\hat{F}\|$  the upper bound

$$\|\hat{F}\| = \int_{\Omega} |\hat{F}(\mathbf{x})| d\mathbf{x} \leq |\Omega|^{1-\frac{1}{p}} \left[ \int_{\Omega} |\hat{F}(\mathbf{x})|^p d\mathbf{x} \right]^{\frac{1}{p}} = |\Omega|^{1-\frac{1}{p}} \hat{L}_I(\hat{u})^{\frac{1}{p}}. \quad (15)$$

Similarly,  $\|\hat{G}\|$  can be bounded from above by

$$\|\hat{G}\| = \int_{\partial\Omega} |\hat{G}(\mathbf{x})| d\mathbf{x}_r \leq |\partial\Omega|^{1-\frac{1}{p}} \left[ \int_{\partial\Omega} |\hat{G}(\mathbf{x})|^p d\mathbf{x}_r \right]^{\frac{1}{p}} = |\partial\Omega|^{1-\frac{1}{p}} \hat{L}_B(\hat{u})^{\frac{1}{p}}. \quad (16)$$

Combining these results gives

$$\begin{aligned} \|\hat{u} - u\| &< C\{\|\hat{F}\| + \|\hat{G}\|\} \leq C\{|\Omega|^{1-\frac{1}{p}} \hat{L}_I(\hat{u})^{\frac{1}{p}} + |\partial\Omega|^{1-\frac{1}{p}} \hat{L}_B(\hat{u})^{\frac{1}{p}}\} \\ &\leq C \left[ c_1^{-\frac{1}{p}} |\Omega|^{1-\frac{1}{p}} + c_2^{-\frac{1}{p}} |\partial\Omega|^{1-\frac{1}{p}} \right] \hat{L}(\hat{u})^{\frac{1}{p}}. \end{aligned} \quad (17)$$

Finally, applying the inequality  $\hat{L}(\hat{u}) < \delta$  yields

$$\|\hat{u} - u\| < C \left[ c_1^{-\frac{1}{p}} |\Omega|^{1-\frac{1}{p}} + c_2^{-\frac{1}{p}} |\partial\Omega|^{1-\frac{1}{p}} \right] \delta^{\frac{1}{p}} = \epsilon,$$

completing the proof.  $\square$

**Remark 1.** When one approximates the solution  $u$  with a neural network, the activation function determines whether the assumptions made in Theorem 1 are satisfied. Activation functions that are in  $C^\infty$  such as the hyperbolic tangent result in neural networks that satisfy all assumptions.

Because Monte-Carlo approximations have a high probability of being close in value to the functions they approximate, it is unlikely that a small approximated value arises when the functional itself is large, given a sufficient number of collocation points. Theorem 1 can then be applied to conclude that a low loss function implies with high probability that the approximations are accurate, validating the use of the approximation for well-posed problems. This theorem also plays an important role in justifying the use of neural networks to minimize these loss functions or functionals; although neural networks are universal approximators, they are unable to exactly represent many functions, and may therefore not be able to exactly reach the minimum of the loss functions or functionals. Theorem 1 shows that this is not a problem, as approximate optimization is sufficient. Thus, Theorem 1 can be used to show that when neural networks reach a low loss value, the approximation they define is likely accurate.

This theorem cannot be used to show that it is reasonable to expect neural networks to reach such small loss values. Due to the significant gap between neural network theory and practice, it is beyond the scope of this work to formally prove this. Therefore, instead of looking at sufficient conditions to guarantee successful network training, one might look at necessary conditions. To this end, consider a loss functional for which local minima exist. It can be shown that the corresponding loss function that would be used to train the neural network would also have local minima for large neural networks with sufficiently smooth activation functions.

By the universal approximation theorem, any smooth function and its derivatives can be approximated to arbitrary accuracy on compact domains by large enough neural networks with sufficiently smooth activation functions. This also means that local minimum points can be approximated to arbitrary accuracy by such neural networks. Furthermore, note that these neural networks are continuous in their parameters, meaning that a small change in the parameters will result in a small change of the function that the network defines, as well as its derivatives. By assumption, functions close to the local minima have a loss that is larger than the local minimum. Thus, the parameter vectors that map to this region of functions also have larger losses. In other words, the local minimum also exists in the network's parameters.

Popular training algorithms such as gradient descent do not handle local minima well. Though there are methods to escape local minima, these methods typically have a performance cost associated and are not guaranteed to find the global minimum. It is therefore very important that the loss functional does not have local minima. Fortunately, showing that the loss functional does not have local minima can be done with relative ease for linear PDEs. In fact, it can be shown that a much stronger property holds for these PDEs. This is done in [Theorem 2](#).

**Theorem 2.** Consider a linear PDE of the form given in Eq. (10) of order  $k$ . Then, the loss functional  $L : \mathcal{F} \rightarrow \mathbb{R}$  defined in Eq. (5), where  $\mathcal{F}$  is the space of functions whose partial derivatives up to order  $k$  exist and have finite  $L^p$  norm, is convex.

**Proof.** To show that the loss functional is convex, it must be shown that for any two functions  $u_1, u_2 \in \mathcal{F}$  and for any  $t \in [0, 1]$ , the inequality

$$\hat{L}(tu_1 + (1-t)u_2) \leq t\hat{L}(u_1) + (1-t)\hat{L}(u_2) \quad (18)$$

holds. To this end, let  $u_1, u_2 \in \mathcal{F}$  and  $t \in [0, 1]$  be arbitrary. For notational convenience, the interpolation is labeled  $v(t) := tu_1 + (1-t)u_2$ . Note that  $v(t) \in \mathcal{F}$ . The loss functional at the interpolation is given by  $\hat{L}(v(t)) = c_1\hat{L}_I(v(t)) + c_2\hat{L}_B(v(t))$ , with  $c_1, c_2 > 0$ . The interior loss is considered first. It is given by

$$\hat{L}_I(v(t)) = \int_{\Omega} |\mathcal{N}(v(t))|^p d\mathbf{x} = \int_{\Omega} |N(v(t)) - F(\mathbf{x})|^p d\mathbf{x}. \quad (19)$$

For linear PDEs,  $N$  is linear, and hence, using additionally the triangle inequality, this can be bounded by

$$\hat{L}_I(v(t)) = \int_{\Omega} |N(tu_1 + (1-t)u_2) - F(\mathbf{x})|^p d\mathbf{x} \leq \int_{\Omega} (t|\mathcal{N}(u_1)| + (1-t)|\mathcal{N}(u_2)|)^p d\mathbf{x}. \quad (20)$$

Because  $|x|^p$  is a convex function for  $p \geq 1$ , it holds that

$$(t|\mathcal{N}(u_1)| + (1-t)|\mathcal{N}(u_2)|)^p \leq t|\mathcal{N}(u_1)|^p + (1-t)|\mathcal{N}(u_2)|^p. \quad (21)$$

Thus, it follows that

$$\begin{aligned} \hat{L}_I(v(t)) &\leq \int_{\Omega} (t|\mathcal{N}(u_1)|^p + (1-t)|\mathcal{N}(u_2)|^p) d\mathbf{x} \leq t \int_{\Omega} |\mathcal{N}(u_1)|^p d\mathbf{x} + (1-t) \int_{\Omega} |\mathcal{N}(u_2)|^p d\mathbf{x} \\ &= t\hat{L}_I(u_1) + (1-t)\hat{L}_I(u_2). \end{aligned} \quad (22)$$

The same reasoning can be applied to show that when the boundary operator  $B$  is linear,

$$\hat{L}_B(v(t)) \leq t\hat{L}_B(u_1) + (1-t)\hat{L}_B(u_2). \quad (23)$$

Eqs. (22) and (23) can then be combined to complete the proof.  $\square$

Thus, for any linear PDE, the loss functional defined in Eq. (5) is convex. A direct consequence is that the loss functional has no local minima. Therefore, any local minima that one encounters while training must necessarily be a product of the configuration of the employed neural network. Although it is well known that neural networks convert convex loss functionals into non-convex problems in parameter space, many recent studies indicate that this non-convexity can be overcome when dealing with classification problems. For instance, the study of [23] empirically shows that local minima can be connected with paths through parameter space for which the loss stays low. Other work has also shown that in the limit of the number of nodes tending to infinity, convexity can be assumed [24]. Even though classification problems cannot be directly compared to solving PDEs, one would expect these problems to share some characteristics; after all, in function space, classification problems are convex, just like solving linear PDEs.

It is important to note that [Theorem 2](#) required the assumption of linearity; for nonlinear PDEs, it is not clear whether the loss functional is convex. For some nonlinear PDEs the loss functional may even have local minima. In these cases, one might consider using global optimization algorithms instead of local algorithms.

### 3. Loss functional modification

The previous section has provided a motivation for the use of the methods introduced in the study of [10]. This was done by reframing the process of solving a PDE as a minimization problem.

In this section, the methods are considered from a different perspective to highlight an important choice to be made. In this section a proposed modification to the loss functionals is discussed. This modification results in a more general formulation that contains an additional hyperparameter. This parameter may be chosen in a way that is tailored to the specific PDE one aims to solve, resulting in a method that is more generally applicable.

### 3.1. Multi-objective optimization

Recall that solving a PDE is equivalent to finding a function satisfying certain constraints. So far, these constraints have been included in the loss functionals, such as in Eq. (5). Because there is more than a single constraint in this functional, minimizing it can be viewed as a scalarization of the multi-objective optimization problem given by

$$\arg \min_{\hat{u}} \left( \hat{L}_I(\hat{u}), \hat{L}_B(\hat{u}) \right). \quad (24)$$

In other words, multiple objectives were merged into a single objective function. This method only works under the assumption that a very low loss can be reached. When low losses are reached, the exact value of the individual losses is not important, as both are necessarily small. However, given the limitations of the capacity of finite neural networks, it may not always be possible to reach such low values. If the different constraints that must be satisfied are not of similar difficulty, then the neural network might not optimize all of them equally. In some cases the network might even sacrifice one objective to better optimize the others. To avoid such problems, the multi-objective optimization problem should be treated more carefully.

Viewed from this perspective, it makes sense to apply a weighted scalarization. The question how these weights should be chosen will be addressed later. To perform this weighted scalarization, we introduce the scaling parameter  $\lambda \in (0, 1)$ , such that the modified loss functional is given by

$$\begin{aligned} \hat{L}(\hat{u}) &:= \lambda \hat{L}_I(\hat{u}) + (1 - \lambda) \hat{L}_B(\hat{u}) \\ &= \lambda \int_{\Omega} |\mathcal{N}(\mathbf{x}, \hat{u})|^p d\mathbf{x} + (1 - \lambda) \int_{\partial\Omega} |\mathcal{B}(\mathbf{x}, \hat{u})|^p d\mathbf{x}_\Gamma. \end{aligned} \quad (25)$$

This hyperparameter  $\lambda$  will be referred to as the loss weight. Note that Theorem 1 holds for the redefined loss functional of Eq. (25) for any constant  $\lambda$ . With this modification, a scaled version of the original loss function can be recovered by setting

$$\lambda = \frac{|\partial\Omega|}{|\partial\Omega| + |\Omega|}, \quad (26)$$

and applying a Monte-Carlo approximation, though, there is little motivation for this particular choice of  $\lambda$ . This family of loss functions can be used to solve a wider range of problems. This is a valuable property, since the original methods were designed to be general. To exploit this, the choice of  $\lambda$  is investigated in Section 3.2 and Section 4. In the first of these sections,  $\lambda$  is treated as a constant. In Section 4, it is treated as a function instead, allowing the hyperparameter to have a deeper effect when training the neural networks.

### 3.2. Optimized loss weights

This section investigates the previously introduced hyperparameter  $\lambda$  in more detail, under the assumption that it is kept constant. This includes the derivation of an optimal choice for this loss weight with respect to a specific error measure.

Optimizing  $\lambda$  is not a straightforward process, mainly because it is unclear with respect to which quantity this parameter should be optimized. Since the described methods all work by minimizing a loss function to solve a PDE, one may consider choosing  $\lambda$  to optimize the relation between the loss functional and the solution of the PDE; a weak relation between the accuracy and the loss functional might result in low losses, but also in inaccurate approximations. On the other hand, a strong relation between these variables would imply that the neural network spends its capacity as effectively as possible.

The strength of this relation can be probed by comparing the loss of a function to the error of this function. The aim is to find the loss functional of which the minimizer is as close to the solution  $u$  as possible. Without explicit knowledge about the network configuration, it is a-priori not clear which functions can and cannot be reached by the neural network, and therefore it is also not clear which loss values may be achieved. Here, a helpful assumption can be made. Under the current setting, the total loss functional interpolates two different functionals with weight  $\lambda$ . If the value of  $\lambda$  is changed such that the interior loss becomes more important, then the neural network might be able to reduce the interior loss at the expense of some accuracy at the boundary. As a result, the total loss value may not change significantly if  $\lambda$  is changed. This provides an important tool to perform the optimization: the optimal loss value can be held constant.

One could consider minimizing the absolute error of the solution for a given, fixed loss value. Although this would yield the tightest upper bound on the error for any given loss value, practical results may not benefit much from this: because neural networks cannot approximate every function equally easily, certain error distributions may not arise in practice. In particular, the results of Section 6 suggest that the error of the derivatives tends to be highly correlated with the derivatives of the true solution. Depending on the shape of the true solution of a PDE, such error distributions often lead to absolute errors that are significantly smaller than the worst-case scenario that would be assumed to derive absolute error bounds.

For this reason, instead of considering the absolute error, one could consider minimizing some form of relative error, both of the solution itself and of its derivatives. More formally, this results in the following definition.



**Definition 2.** A candidate solution  $\hat{u}$  is called  $\epsilon$ -close to the true solution  $u$  if it satisfies

$$|\partial_{\mathbf{x}}^{\gamma} u(\mathbf{x}) - \partial_{\mathbf{x}}^{\gamma} \hat{u}(\mathbf{x})| \leq \epsilon |\partial_{\mathbf{x}}^{\gamma} u(\mathbf{x})| \quad (27)$$

for all  $\mathbf{x} \in \mathbb{R}^d$  and for any vector  $\gamma \in \mathbb{R}^d$  with positive elements  $\gamma_i \geq 0$ . Here,  $\partial_{\mathbf{x}}^{\gamma}$  denotes the parametrized partial derivative given by

$$\partial_{\mathbf{x}}^{\gamma} u := \left( \prod_{i=1}^d \frac{\partial^{\gamma_i}}{\partial x_i^{\gamma_i}} \right) u. \quad (28)$$

This property represents the earlier mentioned observation that the error of solutions found by neural networks seems to be highly correlated with the true solution. Intuitively,  $\epsilon$ -closeness allows approximations to have large errors in the derivatives where the true solution itself has large derivatives. In order to be  $\epsilon$ -close, approximations must be very accurate at the flatter regions of the true solution.

Given this definition, it is desirable that  $\epsilon$  be small. Thus,  $\epsilon$  can be used to guide the choice of  $\lambda$ . To this end, consider a general, linear PDE given by

$$\begin{cases} \sum_{i=1}^{k_I} \alpha_i^I(\mathbf{x}) \partial_{\mathbf{x}}^{\beta_i} u(\mathbf{x}) = F(\mathbf{x}), & \text{in } \Omega, \\ \sum_{i=1}^{k_B} \alpha_i^B(\mathbf{x}) \partial_{\mathbf{x}}^{\gamma_i} u(\mathbf{x}) = G(\mathbf{x}), & \text{on } \partial\Omega. \end{cases} \quad (29)$$

The aim is to derive an upper bound for the losses under the assumption that the approximated solution  $\hat{u}$  is  $\epsilon$ -close to  $u$ . First, consider the interior loss functional. Applying Eq. (6) to the PDE results in the interior loss functional

$$\hat{L}_I(u) = \int_{\Omega} |\mathcal{N}(\mathbf{x}, u)|^p d\mathbf{x} = \int_{\Omega} \left| \sum_{i=1}^{k_N} \alpha_i^I(\mathbf{x}) \partial_{\mathbf{x}}^{\beta_i} u(\mathbf{x}) - F(\mathbf{x}) \right|^p d\mathbf{x}. \quad (30)$$

Note that the solution  $u$  satisfies

$$\mathcal{N}(\mathbf{x}, u) = \sum_{i=1}^{k_N} \alpha_i^I(\mathbf{x}) \partial_{\mathbf{x}}^{\beta_i} u(\mathbf{x}) - F(\mathbf{x}) = 0 \quad (31)$$

for all  $\mathbf{x} \in \Omega$ . Therefore, it follows that

$$\mathcal{N}(\mathbf{x}, \hat{u}) = \sum_{i=1}^{k_N} \alpha_i^I(\mathbf{x}) \left[ \partial_{\mathbf{x}}^{\beta_i} \hat{u}(\mathbf{x}) - \partial_{\mathbf{x}}^{\beta_i} u(\mathbf{x}) \right]. \quad (32)$$

Substituting the approximation  $\hat{u}$  into Eq. (30) allows the integral to be written as

$$\hat{L}_I(\hat{u}) = \int_{\Omega} \left| \sum_{i=1}^{k_N} \alpha_i^I(\mathbf{x}) \left[ \partial_{\mathbf{x}}^{\beta_i} \hat{u}(\mathbf{x}) - \partial_{\mathbf{x}}^{\beta_i} u(\mathbf{x}) \right] \right|^p d\mathbf{x}. \quad (33)$$

Because it was assumed that  $\hat{u}$  satisfies Eq. (27), the integrand, which will be labeled  $l_I(\mathbf{x}, \hat{u})$ , can be bounded by

$$\begin{aligned} l_I(\mathbf{x}, \hat{u}) &= \left| \sum_{i=1}^{k_N} \alpha_i^I(\mathbf{x}) \left[ \partial_{\mathbf{x}}^{\beta_i} \hat{u}(\mathbf{x}) - \partial_{\mathbf{x}}^{\beta_i} u(\mathbf{x}) \right] \right|^p \leq \left[ \sum_{i=1}^{k_N} |\alpha_i^I(\mathbf{x})| \left| \partial_{\mathbf{x}}^{\beta_i} \hat{u}(\mathbf{x}) - \partial_{\mathbf{x}}^{\beta_i} u(\mathbf{x}) \right| \right]^p \\ &\leq \left[ \sum_{i=1}^{k_N} |\alpha_i^I(\mathbf{x})| \epsilon \left| \partial_{\mathbf{x}}^{\beta_i} u(\mathbf{x}) \right| \right]^p = \epsilon^p \left[ \sum_{i=1}^{k_N} |\alpha_i^I(\mathbf{x}) \partial_{\mathbf{x}}^{\beta_i} u(\mathbf{x})| \right]^p. \end{aligned} \quad (34)$$

The resulting upper bound for the interior loss is thus given by

$$\hat{L}_I(\hat{u}) = \int_{\Omega} l_I(\mathbf{x}, \hat{u}) d\mathbf{x} \leq \epsilon^p \int_{\Omega} \left[ \sum_{i=1}^{k_N} |\alpha_i^I(\mathbf{x}) \partial_{\mathbf{x}}^{\beta_i} u(\mathbf{x})| \right]^p d\mathbf{x} =: \epsilon^p M_I(u). \quad (35)$$

One can derive an upper bound for the boundary loss functional in a similar fashion. For the PDE defined in Eq. (29), the resulting upper bound for the boundary loss is given by

$$\hat{L}_B(\hat{u}) \leq \epsilon^p \int_{\partial\Omega} \left[ \sum_{i=1}^{k_B} |\alpha_i^B(\mathbf{x}) \partial_{\mathbf{x}}^{\gamma_i} u(\mathbf{x})| \right]^p d\mathbf{x}_\Gamma =: \epsilon^p M_B(u). \quad (36)$$

These inequalities define necessary, but not sufficient, conditions for a solution to be  $\epsilon$ -close to the true solution. Crucially, these bounds are extremely lenient. For a given  $\epsilon$ , the upper bound on the loss functional is much larger than a sufficient bound would need to be to guarantee the same absolute error. This discrepancy arises mainly from the assumptions on the distributions of the derivatives. Note that these assumptions result in loss bounds that depend on the true solution of the PDE. In contrast, sufficient conditions to guarantee a certain absolute error would only depend on the PDE itself. This ties in with the same reasoning as used before, i.e. solutions with large derivatives tend to have favorable error distributions, and can therefore have larger errors in their derivatives to achieve the same accuracy.

From a different perspective, Eqs. (35), (36) can be used to obtain lower bounds on  $\epsilon$ . Recall that the aim of this section is to find the optimal value of  $\lambda$  that would result in the most accurate approximation, i.e. the smallest  $\epsilon$  such that the approximation is  $\epsilon$ -close. Since training algorithms are only aware of the total loss, it makes sense to perform this optimization with respect to some arbitrary, fixed value  $\hat{L}(\hat{u})$ .

Because  $\lambda$  and  $\epsilon$  have no direct relation, the lower bounds on  $\epsilon$  given in Eqs. (35), (36) should be minimized instead. Rewriting these bounds into a single expression yields

$$\epsilon \geq \hat{\epsilon} := \left[ \max \left\{ \frac{\hat{L}_I(\hat{u})}{M_I(u)}, \frac{\hat{L}_B(\hat{u})}{M_B(u)} \right\} \right]^{\frac{1}{p}}. \quad (37)$$

While  $\hat{\epsilon}$  is not dependent on  $\lambda$ , it can be bounded from above using the total loss functional. In particular, for the interior loss one finds that

$$\hat{L}_I(\hat{u}) = \frac{1}{\lambda} \left[ \hat{L}(\hat{u}) - (1 - \lambda)\hat{L}_B(\hat{u}) \right] \leq \frac{1}{\lambda} \hat{L}(\hat{u}). \quad (38)$$

Similarly, one finds for the boundary loss that

$$\hat{L}_B(\hat{u}) = \frac{1}{1 - \lambda} \left[ \hat{L}(\hat{u}) - \lambda \hat{L}_I(\hat{u}) \right] \leq \frac{1}{1 - \lambda} \hat{L}(\hat{u}). \quad (39)$$

Thus,  $\hat{\epsilon}$  is bounded from above by

$$\hat{\epsilon} = \left[ \max \left\{ \frac{\hat{L}_I(\hat{u})}{M_I(u)}, \frac{\hat{L}_B(\hat{u})}{M_B(u)} \right\} \right]^{\frac{1}{p}} \leq \left[ \max \left\{ \frac{\hat{L}(\hat{u})}{\lambda M_I(u)}, \frac{\hat{L}(\hat{u})}{(1 - \lambda)M_B(u)} \right\} \right]^{\frac{1}{p}}. \quad (40)$$

Since  $\hat{L}(\hat{u})$  is assumed to be constant, the optimal choice for  $\lambda$  then becomes

$$\lambda = \arg \min_{\lambda} \hat{\epsilon} = \arg \min_{\lambda} \left[ \max \left\{ \frac{1}{\lambda M_I(u)}, \frac{1}{(1 - \lambda)M_B(u)} \right\} \right]. \quad (41)$$

The smallest maximum occurs when both terms are equal, i.e. when

$$\lambda M_I(u) = (1 - \lambda)M_B(u), \quad (42)$$

of which the solution is given by

$$\lambda = \frac{M_B(u)}{M_I(u) + M_B(u)}. \quad (43)$$

Notice that this choice of  $\lambda$  may be very different from the original choice as given in Eq. (26).

Eq. (43) gives a choice of  $\lambda$  that is in some sense optimal: for a given loss value  $\hat{L}(\hat{u})$ , this choice of  $\lambda$  results in the smallest  $\epsilon$  for which  $\hat{u}$  may be  $\epsilon$ -close to  $u$ . This choice gives the loss functionals some additional useful properties. Most notably, the relative importance of the interior and boundary losses become scale-invariant. Specifically, when one rewrites a PDE of the form given in Eq. (1) as

$$\begin{cases} c_1 \mathcal{N}(\mathbf{x}, u) = 0 & \text{in } \Omega, \\ c_2 \mathcal{B}(\mathbf{x}, u) = 0 & \text{on } \partial\Omega, \end{cases} \quad (44)$$

then the ratio

$$\frac{\lambda \hat{L}_I(\hat{u})}{(1 - \lambda) \hat{L}_B(\hat{u})} \quad (45)$$

remains constant for all  $c_1, c_2 \neq 0$ . This is a very important property. All PDEs come with some inherent scale factors, and even the default choices  $c_1 = c_2 = 1$  are difficult to justify. This choice of  $\lambda$  renders these scales almost completely irrelevant.

However, it is important to realize that  $\epsilon$ -closeness is contingent on some very strong assumptions, some which are easily shown to not hold true in practice. In particular, functions that zero out anywhere in the domain are problematic for this definition, since  $\epsilon$ -closeness implies zero error for such functions at specific regions. It is likely that the true dynamics of neural networks depend on the behavior of the target function in a small area, rather than at a particular point.



This does not immediately render  $\epsilon$ -closeness useless. However, one should carefully examine the PDE before using  $\epsilon$ -closeness to estimate the difficulty of optimizing the involved objectives. For instance, problems with homogeneous boundary conditions require different methods to properly estimate the difficulties; applying  $\epsilon$ -closeness to such problems results in zero expected boundary error, leading to an optimal loss weight of 0. This definition is likely only useful if the behavior of the true solution on the boundary is comparable to its behavior in the interior of the domain. Similarly, adding offsets to linear PDEs can have a major impact on the theoretical optimal loss weight, even though neural networks can compensate for such offsets by simply tuning the bias of the output layer. Therefore, the definition is best suited for problems with solutions with zero mean.

#### 4. Magnitude normalization

The particular choice of  $\lambda$  derived in the previous section resulted in a relation between  $\lambda$  and the true solution  $u$ . In many practical applications, the required information about  $u$  is unavailable. This section addresses this problem by introducing a heuristic method that approximates the optimal choice as  $\hat{u}$  approaches  $u$ . This heuristic method has one crucial difference compared to the methods discussed so far;  $\lambda$  is no longer kept constant. Instead,  $\lambda$  is treated as an additional functional to be optimized by the neural network.

The starting point of deriving this heuristic method is Eq. (43). This optimal choice of  $\lambda$  depends solely on the true solution of the PDE, leading to a total loss functional given by

$$\hat{L}(\hat{u}) = \frac{M_B(u)\hat{L}_I(\hat{u})}{M_I(u) + M_B(u)} + \frac{M_I(u)\hat{L}_B(\hat{u})}{M_I(u) + M_B(u)}. \quad (46)$$

Since  $u$  is unavailable, and  $\hat{u}$  is meant to approximate  $u$ , one could consider approximating this loss functional by

$$\hat{L}(\hat{u}) = \frac{M_B(\hat{u})\hat{L}_I(\hat{u})}{M_I(\hat{u}) + M_B(\hat{u})} + \frac{M_I(\hat{u})\hat{L}_B(\hat{u})}{M_I(\hat{u}) + M_B(\hat{u})}. \quad (47)$$

When  $\hat{u} \approx u$ , this redefined loss functional behaves similarly to the loss functional of Eq. (25) with the optimal choice of  $\lambda$ , since the bounds  $M_I$  and  $M_B$  will be relatively constant. However, when  $\hat{u}$  differs from  $u$ , the behavior may no longer be similar, as the approximated bounds  $M_I(\hat{u})$  and  $M_B(\hat{u})$  cannot be treated as constants.

This behavior is difficult to analyze without specific information about the PDE one aims to solve. However, some quirks may be identified by considering examples. Many PDEs admit trivial solutions when the boundary conditions are homogeneous. Even though such PDEs are generally only of interest with inhomogeneous boundary conditions, the mere existence of these trivial solutions is problematic. To see why, consider the linear, well-posed PDE given in Eq. (48).

$$\begin{cases} \mathcal{N}(\mathbf{x}, u) = 0, & \text{in } \Omega, \\ u(\mathbf{x}) = G(\mathbf{x}), & \text{on } \partial\Omega, \end{cases} \quad (48)$$

with  $G(\mathbf{x}) \neq 0$  and  $\mathcal{N}(\mathbf{x}, 0) = 0$ . Clearly, the trivial solution would solve this problem if and only if  $G(\mathbf{x}) = 0$ . To show why this is problematic, we consider behavior of the loss functional given in Eq. (47) for this function. For the interior and boundary losses, it holds that

$$\hat{L}_I(0) = 0, \quad \hat{L}_B(0) = \int_{\partial\Omega} |G(\mathbf{x})|^p d\mathbf{x}_\gamma > 0. \quad (49)$$

Similarly, the bounds given in Eq. (36)–(35) satisfy

$$M_I(0) = 0, \quad M_B(0) = \int_{\partial\Omega} |G(\mathbf{x})|^p d\mathbf{x}_\gamma = \hat{L}_B(0). \quad (50)$$

Therefore, the total loss thus becomes

$$\hat{L}(0) = \frac{M_B(0)\hat{L}_I(0) + M_I(0)\hat{L}_B(0)}{M_I(0) + M_B(0)} = \frac{\hat{L}_B(0) \cdot 0 + 0 \cdot \hat{L}_B(0)}{\hat{L}_B(0)} = 0. \quad (51)$$

Thus, a function that violates the boundary conditions is able to bring the loss functional of Eq. (47) down to zero. Note that this problem can also arise if the boundary magnitude and losses can be zeroed out simultaneously; in that case, the PDE does not need to be satisfied to find a solution with zero loss. Thus, Eq. (47) is not a viable loss functional.

It turns out that this issue can be easily addressed while maintaining the relative sizes of the terms  $M_B\hat{L}_I$  and  $M_I\hat{L}_B$ . This can be achieved by multiplying equation (47) with a single scale factor, such that the total loss functional is given by

$$\begin{aligned} \hat{L}(\hat{u}) &= \frac{M_I(\hat{u}) + M_B(\hat{u})}{M_I(\hat{u})M_B(\hat{u})} \left[ \frac{M_B(\hat{u})\hat{L}_I(\hat{u})}{M_I(\hat{u}) + M_B(\hat{u})} + \frac{M_I(\hat{u})\hat{L}_B(\hat{u})}{M_I(\hat{u}) + M_B(\hat{u})} \right] \\ &= \frac{\hat{L}_I(\hat{u})}{M_I(\hat{u})} + \frac{\hat{L}_B(\hat{u})}{M_B(\hat{u})}. \end{aligned} \quad (52)$$

The resulting loss functional has a very intuitive interpretation. Each loss functional is normalized by the magnitude of the terms that comprise it. The resulting terms  $\frac{\hat{L}_I(\hat{u})}{M_I(\hat{u})}$  and  $\frac{\hat{L}_B(\hat{u})}{M_B(\hat{u})}$  can be viewed as relative losses. The resulting method is called *magnitude normalization*.

This loss functional, unlike the functional defined in Eq. (47), again possesses the property that the unique global minimizer is the true solution of the PDE. However, it is not clear whether this loss functional possesses additional local minima, which might compromise convergence to the true solution. Local minima thus pose a threat to the stability of the method.

However, such local minima are characterized by strictly positive loss. In many cases, stability problems can be therefore avoided by pre-training the network; if the neural network is brought to a state with loss lower than one would have at a local minimum, then this local minimum should never be reached.

To further manage stability problems, some additional modifications are introduced. Many problems come with Dirichlet or Neumann boundary conditions, i.e. known operator values along the boundary. This allows one to compute the value  $M_B(u)$  in advance. Using this value instead of approximating it with  $M_B(\hat{u})$  can significantly improve stability. Another stability concern arises from source functions. Since  $\epsilon$ -closeness does not depend on source functions, neither do  $M_I$  and  $M_B$ . However, source functions can significantly inflate the initial value of the loss functional, allowing the network to converge to local minima with higher losses. To prevent this from happening, source functions are included in  $M_I$ . Close to the true solution, this generally has little effect, as the effective loss weight may only change by up to a factor two. Far away from the true solution, however, the behavior is much more stable. For a PDE of the form given in Eq. (29) with Dirichlet or Neumann boundary conditions, the resulting magnitude normalized loss functional is given by

$$\hat{L}(\hat{u}) = \frac{\int_{\Omega} \left| \left( \sum_{j=1}^{k_I} \alpha_j^I(\mathbf{x}) \partial_{\mathbf{x}}^{\beta_j} \hat{u}(\mathbf{x}, \theta) \right) - F(\mathbf{x}) \right|^p d\mathbf{x}}{\int_{\Omega} \left[ \sum_{j=1}^{k_I} \left| \alpha_j^I(\mathbf{x}) \partial_{\mathbf{x}}^{\beta_j} \hat{u}(\mathbf{x}, \theta) \right| + |F(\mathbf{x})| \right]^p d\mathbf{x}} + \frac{\int_{\partial\Omega} [\partial_{\mathbf{x}}^{\gamma} \hat{u}(\mathbf{x}, \theta) - G(\mathbf{x})]^p d\mathbf{x}_\Gamma}{\int_{\partial\Omega} |G(\mathbf{x})|^p d\mathbf{x}_\Gamma}. \quad (53)$$

**Remark 2.** Magnitude normalization can favor functions with large derivatives, since those functions lead to a large denominator in the loss functional. While this can cause instabilities, it can also aid in solving more difficult problems, as is shown in Section 6.

## 5. Method setup

As stated, this section aims to assess the methods by utilizing small neural networks in conjunction with powerful training algorithms. In this section, the network configuration, the evaluation of the loss functionals, and the specifics of the training algorithms are discussed.

For the sake of generality, only fully connected feedforward neural networks are considered here. The networks have four hidden layers with twenty neurons each unless specified otherwise. This yields  $1301 + 20d$  degrees of freedom, where  $d$  is the dimension of the PDE. This network size is kept constant throughout this section. The hyperbolic tangent is used as the activation function, though alternative choices like sinusoids seem to yield comparable results. Glorot initialization [25] is used to generate the initial weights of the neural networks. Initially, the biases are set to zero.

There are many different algorithms with which these networks can be trained. First-order methods such as Adam are among the most popular. Another prominent training algorithm is the limited-memory version of the Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS) [26]. This is a quasi-Newton method that is able to achieve exceptionally accurate results, at the cost of one major drawback: this algorithm is incompatible with batching, and must be provided with a dataset that is representative of the full solution. The problems that are solved in this work are simple enough for this limitation to not be prohibitive, and therefore this algorithm is the training method of choice in this work. We also tested first-order methods such as Adam [27] and stochastic gradient descent, but these methods were always outperformed by L-BFGS. For problems that require datasets that are too large to be processed at once, we recommend using Adam.

### 5.1. Loss functions

Thus, we aim to use L-BFGS to optimize the vector  $\theta$  containing all parameters of the neural network with respect to some objective function. As stated before, the integrals present in the loss functionals in Sections 2–4 can be approximated using Monte-Carlo integration. We explicate this for linear PDEs of the form of Eq. (29). Recall that the general form of the interior and boundary loss functionals  $\hat{L}_I$  and  $\hat{L}_B$  is given in Eq. (30). Let the output of the neural network at position  $\mathbf{x}$  be given by  $\hat{u}(\mathbf{x}, \theta)$ . Then, after applying a Monte-Carlo approximation, we obtain the interior and boundary loss functions for the parameter vector  $\theta$  given by

$$L_I(\theta) = \frac{1}{n_I} \sum_{i=1}^{n_I} \left| \left( \sum_{j=1}^{k_I} \alpha_j^I(\mathbf{x}_i^I) \partial_{\mathbf{x}}^{\beta_j} \hat{u}(\mathbf{x}_i^I, \theta) \right) - F(\mathbf{x}_i^I) \right|^p, \quad (54)$$

$$L_B(\theta) = \frac{1}{n_B} \sum_{i=1}^{n_B} \left| \left( \sum_{j=1}^{k_B} \alpha_j^B(\mathbf{x}_i^B) \partial_{\mathbf{x}}^{\gamma_j} \hat{u}(\mathbf{x}_i^B, \theta) \right) - G(\mathbf{x}_i^B) \right|^p. \quad (55)$$

Here, the collocation points  $\{\mathbf{x}_i^I\}_{i=1}^{n_I}$  and  $\{\mathbf{x}_i^B\}_{i=1}^{n_B}$  are distributed uniformly over  $\Omega$  and  $\partial\Omega$ , respectively. Using Eqs. (54), (55), Monte-Carlo approximations of Eq. (4) with  $p = 2$  and Eq. (25) are given by

$$L(\theta) = L_I(\theta) + L_B(\theta), \quad (56)$$

$$L(\theta) = \Omega \lambda L_I(\theta) + \partial\Omega (1 - \lambda) L_B(\theta), \quad (57)$$

respectively. The Monte-Carlo approximation of Eq. (53) is given by

$$L(\theta) = \frac{\sum_{i=1}^{n_I} \left[ \left( \sum_{j=1}^{k_I} \alpha_j^I(\mathbf{x}_i^I) \partial_{\mathbf{x}}^{\beta_j} \hat{u}(\mathbf{x}_i^I, \theta) \right) - F(\mathbf{x}_i^I) \right]^p}{\sum_{i=1}^{n_I} \left[ \sum_{j=1}^{k_I} \left| \alpha_j^I(\mathbf{x}_i^I) \partial_{\mathbf{x}}^{\beta_j} \hat{u}(\mathbf{x}_i^I, \theta) \right| + |F(\mathbf{x}_i^I)| \right]^p} + \frac{\sum_{i=1}^{n_B} \left[ \partial_{\mathbf{x}}^{\gamma} \hat{u}(\mathbf{x}_i^B) - G(\mathbf{x}_i^B) \right]^p}{\sum_{i=1}^{n_B} |G(\mathbf{x}_i^B)|^p}. \quad (58)$$

These loss functions define three methods of interest that will be considered in our experiments: the original method as introduced in [10], our proposed modification to this method with an optimal loss weight, and the heuristic approximation of this optimal weight, respectively. To perform the experiments, the norm  $p = 2$  is used. Note that instead of minimizing some function  $f(\theta)$ , one could equivalently minimize the function  $g(f(\theta))$  for some monotonically increasing  $g$ . Because the loss functions are supposed to become small during training, the gradients of these loss functions with respect to  $\theta$  will likely become very small as well. Therefore, instead of minimizing the loss functions directly, the logarithm of the loss functions is minimized. In many cases, this significantly improves the results.

## 5.2. Adaptive collocation points

The loss functions depend on the hyperparameters  $n_I$  and  $n_B$ . Determining the required number of collocation points, both on the boundary and in the interior of the domain, is generally challenging, as it depends on the variance of the loss functions, which in turn depends on the state of the neural network. We propose an alternative solution: choosing the point counts adaptively, by comparing the training loss to some validation loss. First choose initial values for the point counts  $n_I$  and  $n_B$  and then generate two different sets of collocation points, each consisting of  $n_I$  interior and  $n_B$  boundary points. The first set is the training set, and the second set is the validation set. L-BFGS is used to minimize the loss function evaluated on the training set. During training, the loss function is also evaluated on the validation set. If at any point during training the interior or boundary validation loss is more than a factor  $q$  larger than either the interior or boundary training loss, then the variance of the respective loss function is too large to be accurately approximated with the number of collocation points used. In this case, the corresponding point count  $n_I$  or  $n_B$  is doubled, new collocation point sets are generated, and training is resumed. The initial point counts  $n_I$  and  $n_B$  are generally set to 512. The precise value of the hyperparameter  $q$  seems to have little effect, and  $q = 5$  seems to be a reasonable trade-off between accuracy and speed.

## 5.3. GPU acceleration

Neural networks are well suited for parallelization, as many of the computations that must be performed to compute a training iteration are independent. In this work, we use Tensorflow version 1.15.0. to perform the computations in parallel. Python source code that implements the listed methods to solve a model problem can be found on GitHub.<sup>1</sup> The numerical experiments presented in this work were computed on a Tesla K80 GPU provided by Google's Colaboratory project.

## 6. Experimental results

This section aims to experimentally compare the methods developed in this work to the original method introduced in the study of [10] by applying all methods to solve several different PDEs.

Two new methods were developed in work. Section 3 introduced loss weights, with the aim to balance the priorities of the neural network, and derived an optimal choice for this loss weight. Section 4 introduced a heuristic method for approximating this optimum, which may be of value when limited information about the analytical solution is available. We label the original method *Original*, and our new methods *Optimal Loss Weight* and *Magnitude Normalization*, respectively.

To probe the limits of these three methods, we use them to solve several different model PDEs, which are chosen such that they can easily be made more difficult. Section 6.1 covers the first two PDEs that we consider: the Laplace and Poisson equations. Because these PDEs are well understood and easy to solve with traditional numerical methods, they are also easy to analyze. Since we aim to probe the limits of the three methods, more challenging variants of these PDEs are also considered here, including higher-dimensional problems and problems with peaks.

Section 6.3 treats the convection–diffusion equation. This PDE becomes extremely challenging to solve with traditional numerical methods if the diffusivity becomes small. As the diffusion rate approaches zero, the solutions start to exhibit boundary layers, which often require prohibitively fine grids and small time steps to solve. Decreasing the diffusion rate thus serves as the main tool to increase the difficulty of these problems.

<sup>1</sup> <https://github.com/remcovandermeer/Optimally-Weighted-PINNs>

**Table 1**

Relative  $L^2$  and  $L^\infty$  errors of the approximations that were obtained by the three methods for various frequencies. Problems with frequencies higher than  $4\pi$  were not solved accurately by the original method. Magnitude normalization and Optimal loss weights both resulted in significantly more accurate approximations.

$\omega$	Original		Optimal loss weight		Magnitude normalization	
	$L^2$	$L^\infty$	$L^2$	$L^\infty$	$L^2$	$L^\infty$
$1\pi$	9.07e-5	1.34e-4	2.76e-5	1.10e-4	2.38e-5	6.79e-5
$2\pi$	1.12e-3	2.61e-3	6.36e-5	1.91e-4	2.38e-4	5.07e-4
$4\pi$	2.11e-2	6.44e-2	7.27e-4	4.63e-4	1.75e-3	1.31e-3
$6\pi$	7.50e-1	9.77e-1	1.01e-3	3.39e-4	3.77e-3	1.50e-3
$8\pi$	8.25e-1	1.21	1.84e-2	8.71e-3	5.95e-3	2.08e-3
$10\pi$	2.44	1.65	1.53e-2	9.26e-3	7.05e-3	2.30e-3

### 6.1. Laplace equation

This section covers the Laplace equation, which describes the stationary heat equation and forms the basis of many model PDEs. As mentioned, this PDE is well understood and therefore serves as the starting point of the experimental analysis. Here, our focus is on analyzing behavior that is specific to neural networks. Traditional numerical methods are generally only dependent on the PDE and the grid size. However, since neural networks have limited learning capacity based on their size, one would expect that the difficulty of solving a PDE depends, among other things, on the complexity of the true solution. Therefore, choosing boundary conditions that lead to increasingly complex solutions forms the basis of probing the limits of the original and proposed methods. For a  $d$ -dimensional system, boundary value problems corresponding to the Laplace equation assume the form

$$\begin{cases} \nabla^2 u(\mathbf{x}) = 0 & \text{in } \Omega, \\ u(\mathbf{x}) = G(\mathbf{x}) & \text{on } \partial\Omega. \end{cases} \quad (59)$$

To keep the arithmetic simple, problems are considered on the  $d$ -dimensional unit hypercube with boundary conditions that correspond to the eigenfunctions of the Laplace equation,  $u(\mathbf{x}) = \prod_{i=2}^d \sin \omega_i x_i e^{-\omega_1 x_1}$ , with  $\omega_i$  multiples of  $\pi$  and  $\omega_1 = \sqrt{\sum_{i=2}^d \omega_i^2}$ . This gives rise to inhomogeneous boundary conditions at  $x_1 = 0$  and  $x_1 = 1$ , and homogeneous boundary conditions on the remaining  $2d - 2$  boundary hyperplanes. High frequencies are expected to be challenging to learn compared to lower frequencies. Section 6.1.1 covers the problem in two dimensions. Section 6.1.2 covers the Laplace equation in up to six dimensions.

#### 6.1.1. Two-dimensional problems

In two dimensions, eigenfunctions are characterized by a single eigenfrequency  $\omega$ . The three methods are compared for various frequencies ranging from  $\omega = \pi$  to  $\omega = 10\pi$ . For these problems, the optimal loss weight follows from the magnitude bounds of Eq. (36)–(35), which are given by

$$\begin{aligned} M_I(u) &= \int_{\Omega} [2\omega^2 e^{-\omega x_1} \sin \omega x_2]^2 d\mathbf{x} = \omega^3 (1 - e^{-2\omega}), \\ M_B(u) &= \int_{\partial\Omega} [e^{-\omega x_1} \sin \omega x_2]^2 d\mathbf{x}_\Gamma = \frac{1}{2} (1 + e^{-2\omega}). \end{aligned} \quad (60)$$

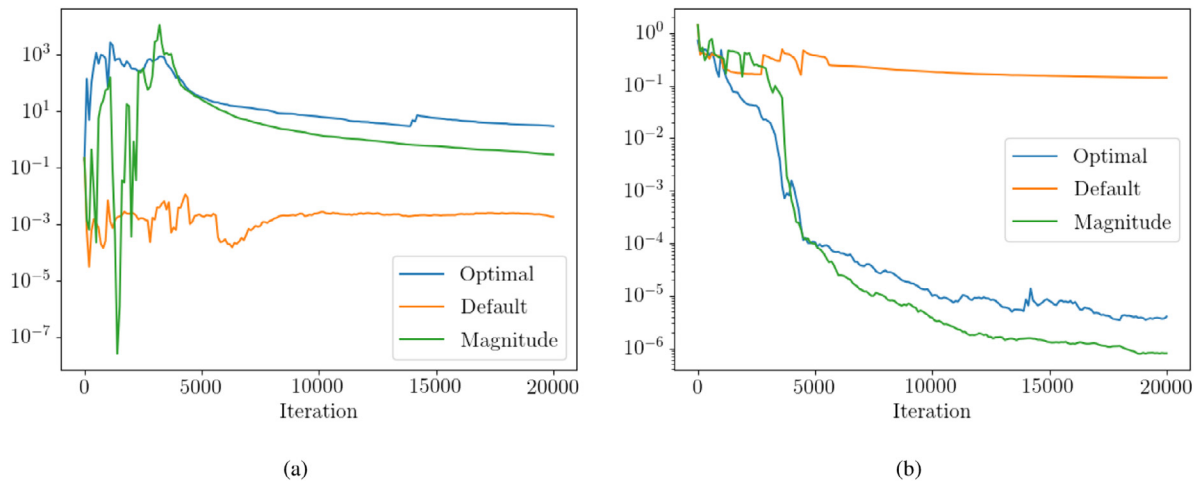
The optimal loss weight is thus approximately given by  $\lambda = \frac{M_B(u)}{M_B(u) + M_I(u)} \approx \frac{1}{1 + 2\omega^3}$ . As a result, optimal loss weights range from approximately  $1.58e-2$  for  $\omega = \pi$  to  $1.61e-5$  for  $\omega = 10\pi$ . This forecasts significant differences between the original method given in [10] and the proposed ones, especially for higher frequencies. Recall that the loss weight used by the original method is given in Eq. (26). For this particular problem, that evaluates to  $\lambda = \frac{4}{5}$ .

To perform the training we use adaptive collocation point counts, starting with 2 interior and boundary collocation points. L-BFGS ran for 20,000 iterations. The relative  $L^2$  and  $L^\infty$  errors obtained by the three methods are given in Table 1.

Table 1 shows that there is a significant difference in accuracy between the original method and the methods proposed in this work, which becomes even more pronounced for higher frequencies. In most cases, the proposed methods were at least one order of magnitude more accurate than the original.

To better understand the differences between the three methods, the values of the functions  $L_I(\hat{u})$  and  $L_B(\hat{u})$  during training are plotted in Fig. 1. This highlights that our proposed methods do not yield solutions with lower losses than the original method; instead, the interior loss is sacrificed to better approximate the boundary.

To gain some insight in the adaptive point counts that were used to obtain the results, the number of collocation points used for the extreme frequencies are given in Table 2. This table highlights the strong dependency of the required point counts on the complexity of the true solution, as well as the differences between the three methods. In particular, observe how magnitude normalization tends to use more collocation points. This is caused by its tendency to overfit, as stated earlier.



**Fig. 1.** Loss profiles of the different methods during training for the problem with frequency  $\omega = 10\pi$ . Fig. 1(a) shows the interior loss, and Fig. 1(b) shows the boundary loss. Even though the original method resulted in a very inaccurate approximation, the interior loss was much smaller, showing that it is important to assign the right weight to the different loss terms. Note that the neural networks were trained to minimize the logarithm of these losses.

**Table 2**

Adaptive point counts used by the three methods for the extreme problem frequencies. The higher frequency problem required significantly more collocation points, as expected.

$\omega$	Original		Optimal loss weight		Magnitude normalization	
	Interior	Boundary	Interior	Boundary	Interior	Boundary
$1\pi$	512	64	128	128	512	64
$10\pi$	4096	128	4096	512	8192	512

**Table 3**

The problem with frequency  $\omega = 10\pi$  solved by the three different methods, using a neural network with five layers of 50 neurons. L-BFGS was run for 50,000 iterations with adaptive point counts, starting with 2 interior and boundary collocation points. The proposed methods achieved excellent accuracy, exceeding the accuracy obtained by the original method when solving the problem with frequency  $\omega = 2\pi$  using a smaller network.

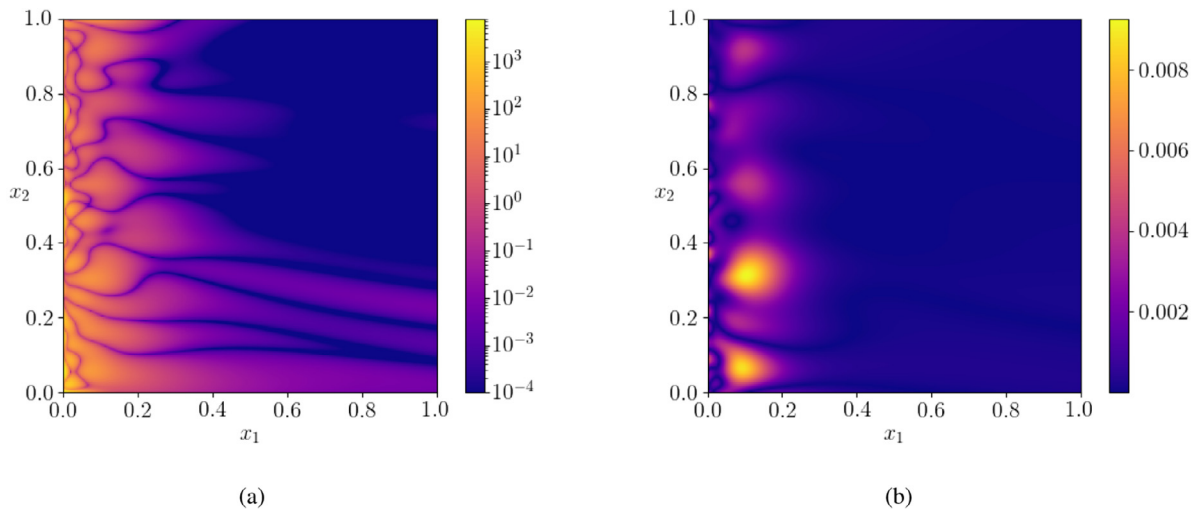
	Original	Optimal loss weight	Magnitude normalization
Relative $L^2$ error	$9.61e-1$	$1.04e-3$	$1.03e-3$
Relative $L^\infty$ error	1.13	$6.43e-4$	$6.81e-4$
Interior point count	16,384	16,384	32,768
Boundary point count	64	512	512

As observed in Table 1, increasing the frequency leads to a drop in accuracy. To confirm that accuracy improvements can be achieved by increasing the network size or the training iterations, the problem with frequency  $\omega = 10\pi$  is solved with the three methods using a neural network with five layers of 50 neurons. Here, L-BFGS with adaptive points, starting with 2 interior and boundary collocation points, was performed for 50,000 iterations. The resulting errors and final collocation point counts are given in Table 3. The collocation point counts generally exceeded the number of points used with the smaller network; this likely resulted from the bigger network being slightly more prone to overfitting.

To end this section, we briefly consider  $\epsilon$ -closeness, which forms the foundation on which the proposed methods are built. As mentioned,  $\epsilon$ -closeness likely only holds approximately. To examine this, the loss and error of the solution obtained by the unscaled method using optimal loss weights are depicted in Fig. 2. This figure highlights that while  $\epsilon$ -closeness is not satisfied exactly, as errors are nonzero where the true solution zeros out, the overall distributions of the errors are shaped similarly to the true solution. Both the loss function, which measures the errors of the second derivatives, and the absolute errors decay significantly in  $x_1$ , just like the true solution, showing that  $\epsilon$ -closeness can be a useful concept even in practice.

### 6.1.2. Higher-dimensional problems

With basic results established, let us turn to higher-dimensional problems, as they form one of the core motivations for the development of neural network based solvers. As in the previous section, the eigenfunctions are used to generate boundary conditions with easy-to-analyze solutions. Using the Laplace eigenfunctions,  $d - 1$  frequencies can be chosen freely, while  $\omega_1$  is determined by the other frequencies. To prevent the solutions from becoming exponentially complex,



**Fig. 2.** Loss (Fig. 2(a)) and error (Fig. 2(b)) distributions of the solution of the problem with frequency  $\omega = 10\pi$ , obtained by using optimal loss weights. Notice how both the loss and the error vanish as the solution becomes flat, highlighting that although  $\epsilon$ -closeness does not hold exactly, it can be used to predict the overall behavior of the loss function.

**Table 4**

Relative  $L^2$  and  $L^\infty$  errors of the approximations that were found by the three methods for different dimensionalities. The original method has trouble solving even the three-dimensional problem, while the proposed methods gradually become less accurate as the dimensionality grows.

$d$	Original		Optimal loss weight		Magnitude normalization	
	$L^2$	$L^\infty$	$L^2$	$L^\infty$	$L^2$	$L^\infty$
3	2.00e-1	4.35e-1	2.02e-3	4.22e-3	2.66e-3	4.21e-3
4	7.29e-1	9.68e-1	1.00e-2	2.49e-2	8.66e-3	3.54e-2
5	9.84e-1	1.01	1.22e-2	2.31e-2	1.47e-2	2.22e-2
6	1.06	9.84e-1	4.31e-2	4.13e-2	4.32e-2	4.38e-2

$d - 2$  frequencies are held constant at  $\omega_i = \pi$ , and the frequency  $\omega_2$  is set to  $4\pi$ . The focus of this section is on examining the effects of the dimensionality. The results for different dimensionalities are given in Table 4. Here, L-BFGS with adaptive point counts, starting with 2 interior and boundary collocation points, was used.

Adaptive point counts never resulted in more than 16,384 interior or boundary collocation points, even for six-dimensional problems, indicating that high-dimensional problems do not necessarily require massive amounts of data to be solved accurately; instead, this likely only depends on the complexity of the solution. This highlights a big advantage of neural network based methods over traditional numerical methods, which would suffer from an exponential increase in computational cost.

Like in the previous section, results degraded as the problem difficulty was increased. We again attribute this to the constant network size and number of iterations, which in practice ought to be increased for more difficult problems. To verify this, the six-dimensional problem was solved with a neural network with five layers of 50 neurons over 100,000 iterations, using magnitude normalization with adaptive point counts, starting with 2 interior and boundary collocation points. The obtained solution had a relative  $L^2$  error of  $5.99\text{e-}3$  and a relative  $L^\infty$  error of  $9.88\text{e-}3$ . This again shows that the accuracy can be improved by increasing the available computational resources.

## 6.2. Poisson equation

This section covers the Poisson equation, which is the extension of the Laplace equation with a source function. This PDE is analyzed in order to assess the effects of source functions on the proposed methods. The main aim of this section is to show that  $\epsilon$ -closeness, which by construction does not depend on source functions, is a useful concept even for inhomogeneous PDEs. The Poisson equation has also been studied in other works on neural network based PDE solvers, including the studies of [8,11,28]. However, the problems that are solved accurately in these studies are generally characterized by very smooth solutions, whereas our interest lies in solving more complicated problems, as we did in Section 6.1.1. Only the study of [28] considers a problem with less regular behavior, but the authors did not manage to solve it accurately.

As in the previous section, we aim to solve increasingly complex problems in order to probe the limits of the proposed methods. We restrict ourselves to the 2-dimensional case. In the first part of this section, we consider source functions



**Table 5**

Relative  $L^2$  and  $L^\infty$  errors of the approximations obtained by the three methods for different  $\omega$ . The original method failed to solve problems with frequencies of  $4\pi$  or higher, while the proposed methods could solve all considered problems. The decline in accuracy for higher frequency problems can likely be explained by the increased complexity of the solutions of these problems.

$\omega$	Original		Optimal loss weight		Magnitude normalization	
	$L^2$	$L^\infty$	$L^2$	$L^\infty$	$L^2$	$L^\infty$
$1\pi$	6.66e-4	3.51e-3	3.47e-5	1.86e-4	2.82e-5	1.16e-4
$2\pi$	3.01e-2	1.14e-1	1.99e-4	5.28e-4	2.47e-4	5.35e-4
$4\pi$	4.56e-1	8.57e-1	2.94e-2	7.36e-2	7.74e-3	1.21e-2
$6\pi$	9.82	1.27e1	2.43e-2	1.08e-1	3.71e-2	5.03e-2

corresponding to the eigenfunctions of this PDE. These problems can be made more difficult by increasing the frequency. In the second part of this section we solve the problem with a peak source function that was studied but not solved in [28].

### 6.2.1. Oscillating solutions

For a 2-dimensional system, boundary value problems corresponding to the Poisson equation can be defined by

$$\begin{cases} \nabla^2 u(x, y) = F(x, y) & \text{in } \Omega, \\ u(x, y) = G(x, y) & \text{on } \partial\Omega. \end{cases} \quad (61)$$

We consider problems on the unit square  $\Omega = [0, 1] \times [0, 1]$ . We first consider the eigenfunction problems, with solutions given by  $u(x, y) = \cos(\omega\pi x) \sin(\omega\pi y)$ . These solutions are prescribed as Dirichlet boundary conditions on the entire boundary of the domain. The source functions that correspond to these eigenfunctions are given by  $F(x, y) = -2\omega^2 \cos(\omega\pi x) \sin(\omega\pi y)$ . This set of problems is parametrized by the frequency  $\omega$ , which can be utilized to control the difficulty. Similar to the Laplace equation, high frequency problems are expected to be more challenging to solve with these methods than low frequency problems. In addition to this, the optimal loss weights vary based on the frequency. For these problems, the optimal weights follow from

$$M_I(u) = \int_{\Omega} [2\omega^2 \cos(\omega x_1) \sin(\omega x_2)]^2 d\mathbf{x} = \omega^4, \quad (62)$$

$$M_B(u) = \int_{\partial\Omega} [\cos(\omega x_1) \sin(\omega x_2)]^2 d\mathbf{x}_\Gamma = 1, \quad (63)$$

such that

$$\lambda = \frac{M_B(u)}{M_B(u) + M_I(u)} = \frac{1}{1 + \omega^4}. \quad (64)$$

This dependency of  $\lambda$  on  $\omega$  is even stronger than it was for the Laplace equation, which is why we consider a smaller range of frequencies, given by  $\omega \in [\pi, 6\pi]$ . For this range of frequencies, the optimal loss weights vary between  $\lambda \approx 1.02\text{e-}2$  for  $\omega = \pi$  to  $\lambda \approx 7.90\text{e-}6$  for  $\omega = 6\pi$ , whereas the original method uses  $\lambda = \frac{4}{5}$ . Based on these values, one would again expect significant differences between the three methods for higher frequencies. The problems were solved using the three methods using adaptive collocation point counts, starting with 2 interior and boundary collocation points. 20,000 iterations were performed. Table 5 contains the relative errors of the obtained approximations.

These results are similar in nature to our results of the Laplace equation; as expected, the accuracy of the original method drops off much faster than the accuracy of the proposed methods. We furthermore observed that the large error of the original method again arises mainly from the boundary conditions not being satisfied, while the two proposed methods were able to fit the boundary conditions, even for the highest frequency considered here.

To conclude this section, our results thus show that the concept of  $\epsilon$ -closeness remains valid even in the presence of source functions. Both optimally weighted method and magnitude normalization make it possible to solve problems with much greater derivatives.

### 6.2.2. Source functions with peaks

Next, we consider the problem mentioned earlier that has been studied in [28], however without a convincing solution so far. As mentioned, this problem is characterized by the peak in the source function, which causes a peak in the solution. The problem is defined by its solution, which is chosen as

$$u(x, y) = \sin(\pi x) + e^{-1000((x-\frac{1}{2})^2 + (y-\frac{1}{2})^2)} - \frac{1}{2}. \quad (65)$$

The optimal loss weight of this problem is approximately given by  $\lambda \approx 4.45\text{e-}5$ , which suggests that the proposed methods might be better suited to solving this problem than the original method, which uses  $\lambda = \frac{4}{5}$ . To perform the experiments, we again use adaptive collocation point counts, starting with 2 interior and boundary collocation points. For each method



**Table 6**

Relative  $L^2$  errors of the approximations obtained for the problem with a peak. The original method is outperformed by the two proposed methods. Note that Magnitude normalization used a large number of boundary collocation points. This was caused by an instability that occurred during early training, but was eventually overcome by the method.

	Original	Optimal loss weight	Magnitude normalization
Relative $L^2$ error	1.24e-1	4.01e-3	2.08e-3
Relative $L^\infty$ error	7.56e-2	2.49e-3	2.94e-3
Interior point count	8192	8192	8192
Boundary point Count	8	64	512

we performed 20,000 L-BFGS iterations. The results of the three methods, as well as the final numbers of collocation points, are given in Table 6.

Overall, these results line up with the expectations. However, the difference between the three methods is fairly small, despite the very small optimal value of the loss weight. We suspect that the reason that this difference is so small originates from the different characteristics present in the solution; during the early training, the few collocation points present are likely only able to capture the low frequency part of the solution, leading to early approximations that already satisfy the boundary conditions. When more collocation points are added, the peak starts affecting the loss function, which causes the original method to neglect the boundary conditions from then on. However, at this stage, the neural network already fits the boundary conditions rather well. All three methods yielded significantly better results than were obtained in [28], showing that our approach is promising.

### 6.3. Convection-dominated convection–diffusion equation

So far only nicely elliptic PDEs have been discussed. The experimental results suggest that for hyperbolic PDEs, choosing the loss weight according to the theoretical optimum can have a significant positive effect on the accuracy of these methods. Magnitude normalization, which has qualitatively different behavior, generally results in similar accuracy improvements, but can also be unstable. These instabilities reveal an underlying property of this method: it acts as a driving force towards larger derivatives.

It turns out that this quirk can be exploited to solve a certain class of problems: singularly perturbed problems. These problems are typically challenging to solve with traditional numerical methods because of extreme differences in the behavior of the solutions. One instance of this class of problems is the stationary convection-dominated convection–diffusion equation. This PDE gives rise to solutions with boundary layers, which are small regions where the solution has dramatically different behavior compared to the rest of the solution. For this particular PDE, boundary layers are characterized by extremely large gradients. In this section we briefly show that the tendency of magnitude normalization to find solutions with large gradients makes this method a promising candidate for solving problems with boundary layers.

In one dimension, the stationary convection–diffusion equation is given by

$$\begin{cases} v \frac{du}{dx} + \alpha \frac{d^2u}{dx^2} = 0 & \text{in } \Omega, \\ u(x) = G(x) & \text{on } \partial\Omega. \end{cases}$$

We examine this PDE on the domain  $x \in [0, 1]$  with  $v = 1$ , subject to the boundary conditions  $u(0) = \frac{1}{2}$ ,  $u(1) = -\frac{1}{2}$ . The analytical solution of these problems takes the form

$$u(x) = \frac{e^{-\frac{vx}{\alpha}}}{1 - e^{-\frac{v}{\alpha}}} - \frac{1}{2}. \quad (66)$$

Eq. (66) implies that the size of the boundary layer is proportional to the diffusivity  $\alpha$ . Therefore, one would expect that these problems more are challenging to solve as  $\alpha \rightarrow 0$ . Thus, reducing the diffusivity provides an excellent tool to tune the difficulty of the problem. The three methods are compared for various  $\alpha$ , ranging from  $\alpha = 10^{-1}$  down to  $\alpha = 10^{-4}$ . Here, we use adaptive point counts, starting with 2 interior points. The 2 boundary collocation points fully cover the boundary domain, and are hence kept constant. We performed 20,000 iterations for each problem. The results are given in Table 7.

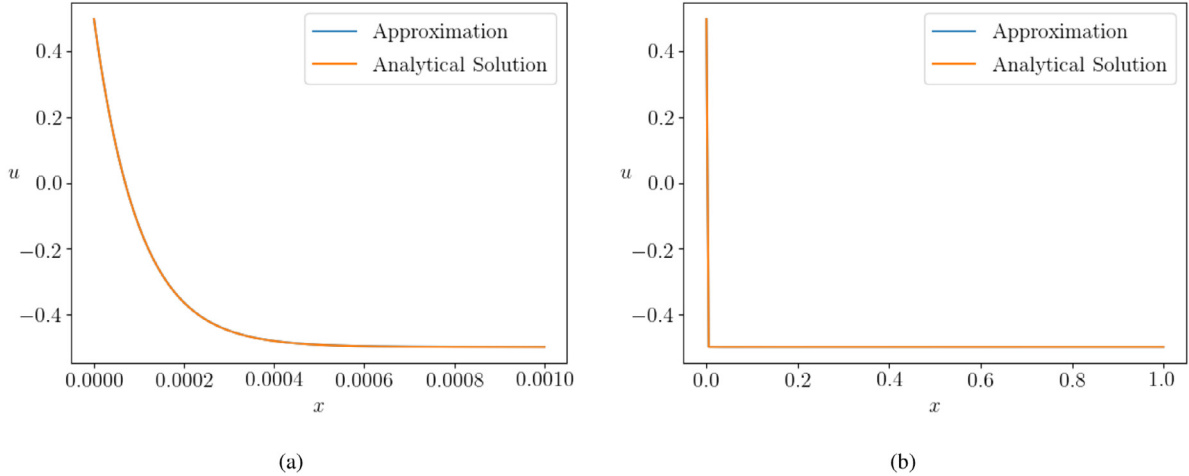
The results given in Table 7 show that for  $\alpha \geq 10^{-2}$ , all three methods yielded very accurate results. Remarkably, the original method outperformed the proposed methods. We suspect that due to the extreme variance of the true solution, a single loss weight is not sufficient to simultaneously deal with the different behaviors inside and outside of the boundary layer. If this is indeed the issue, a local loss weight  $\lambda(\mathbf{x})$  would likely improve the results. However, that is beyond the scope of this work. For smaller  $\alpha$ , none of the three methods were able to obtain accurate results.

We will show that it only takes a small modification for magnitude normalization to solve these problems. In particular we observed that magnitude normalization with adaptive point counts resulted in a very large number of collocation points. This was caused by magnitude normalization being unstable for this problem. Eliminating instabilities is not

**Table 7**

Relative  $L^2$  and  $L^\infty$  errors of the approximations that were obtained by the three methods for various  $\alpha$ . For  $\alpha \leq 10^{-3}$ , no good approximations were found. For larger diffusivity, all three methods yielded very accurate approximations.

$\alpha$	Original		Optimal loss weight		Magnitude normalization	
	$L^2$	$L^\infty$	$L^2$	$L^\infty$	$L^2$	$L^\infty$
$10^{-1}$	1.25e-8	2.28e-8	1.48e-6	3.02e-6	4.45e-8	8.87e-8
$10^{-2}$	1.50e-7	4.11e-7	2.34e-6	4.63e-6	2.87e-7	5.80e-7
$10^{-3}$	1.02	1.14	1.14	1.98	9.25	4.56e1
$10^{-4}$	1.01	1.20	1.15	2.00	1.91	3.51e1



**Fig. 3.** The approximation obtained using magnitude normalization with 256 initial interior collocation points. Even in the boundary layer, no deviations from the true solution are visible.

straightforward, and this process typically depends on the nature of the instability. We already mentioned that pre-training is a valid way of overcoming instabilities. However, it may also be helpful to increase the initial number of collocation points, as this prevents the method from introducing spikes into the solution between collocation points. For the particular problem with  $\alpha = 10^{-4}$ , it seems to be sufficient: starting the algorithm with 256 interior collocation points resulted in a relative  $L^2$  error of  $1.41\text{e-}4$ , and a relative  $L^\infty$  error of  $8.78\text{e-}4$ . Here, adaptive point counts ultimately resulted in 65,536 interior collocation points, which is a fairly small number considering the size of the boundary layer. This suggests that the poor accuracy of magnitude normalization as shown in Table 7 was indeed caused by instabilities. The approximated solution when starting with 256 interior collocation points is depicted in Fig. 3.

We remark that increasing the initial number of collocation points does not enable the other two methods to solve the problems with small  $\alpha$ . It only slightly improved the obtained accuracy for the problems with large  $\alpha$ , at significant computational cost.

## 7. Discussion and conclusion

The study of [10] introduces an unsupervised learning method that solves PDEs by simultaneously minimizing two objective functions: one encoding the boundary conditions, and one encoding the PDE itself. This method lends itself to solving certain types of problems more efficiently than traditional numerical solvers. For example, neural networks do not suffer from the curse of dimensionality, as they are able to recognize low-dimensional features in high-dimensional space. Furthermore, this method is likely well suited to solving problems for which the underlying geometry is challenging to discretize.

In this work, we generalized this method by introducing a loss weight which compensates for potential imbalances in these two objective functions, which the original method simply adds together with equal weight. To derive an optimal value for this loss weight, the notion of  $\epsilon$ -closeness was introduced in Section 3. This concept was used to predict the deviations of a neural network approximation and its derivatives from the target function, and allowed us to express an optimal value for the loss weight in terms of the true solution of a PDE. We also derived a heuristic method to approximate this loss weight in terms of the approximated solution, which we coined *magnitude normalization*. Furthermore, we introduced a method of adaptively updating the number of collocation points based on the training loss and a test loss.

The significance of  $\epsilon$ -closeness and the accompanying optimal loss weights was showcased in Section 6, which contains several model problems that were specifically constructed to have imbalanced objective functions. Using our proposed

methods, much better accuracy was obtained for most problems we considered. This shows that neural network based PDE solvers have inherently different behavior compared to traditional numerical methods; their accuracy does not only depend on the PDE one aims to solve, but also on the complexity of the solution. Magnitude normalization was also shown to have useful properties for solving singularly perturbed problems.

We must, however, mention that magnitude normalization in its current form can be sensitive to the initialization of the neural network, and may be unstable for more difficult problems. These instabilities pose challenges that remain to be solved in future work. The singularly perturbed problems that we considered also indicate that using constant loss weights may not always be ideal, in particular when the solution is characterized by changing behavior. In these cases, a local loss weight  $\lambda(\mathbf{x})$  might prove useful. Such a local loss weight can likely be derived from  $\epsilon$ -closeness, since this is a local concept. Local loss weights might even be the key to stabilizing magnitude normalization.

Other open problems that remain have already been identified by the authors of the original study of [10]: error bounds and convergence guarantees are currently non-existent. Given the computational cost of training a neural network, we agree that these methods should not yet be used to solve a single problem instance of a PDE for which traditional methods are available. However, these methods can likely be used to solve parametrized families of problems. While the computational costs involved in training a neural network are large, neural networks are typically cheap to evaluate. Being able to solve many PDEs at once would hence provide a computational reason to prefer neural network based methods over traditional solvers.

The progress made in this work should bring us one step closer to being able to solve parametrized problems: the theoretical results presented here concern the simultaneous optimization of two general objective functions, and should extend beyond the setting of solving PDEs. A natural next step would be to combine more than two objectives, e.g. by solving multiple PDEs at once.

## References

- [1] J. Schmidhuber, Deep learning in neural networks: An overview, 2014, ArXiv E-Prints [arXiv:1404.7828](https://arxiv.org/abs/1404.7828).
- [2] H. Owhadi, Bayesian numerical homogenization, *Multiscale Model. Simul.* 13 (3) (2015) 812–828.
- [3] M. Raissi, P. Perdikaris, G.E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, *J. Comput. Phys.* 348 (2017) 683–693.
- [4] H. Owhadi, C. Scovel, T. Sullivan, et al., Brittleness of Bayesian inference under finite information in a continuous world, *Electron. J. Stat.* 9 (1) (2015) 1–79.
- [5] M. Raissi, P. Perdikaris, G.E. Karniadakis, Numerical Gaussian processes for time-dependent and nonlinear partial differential equations, *SIAM J. Sci. Comput.* 40 (1) (2018) A172–A198.
- [6] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, *Neural Netw.* 2 (5) (1989) 359–366.
- [7] K. Hornik, M. Stinchcombe, H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, *Neural Netw.* 3 (5) (1990) 551–560.
- [8] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, 1997, ArXiv E-Prints [arXiv:physics/9705023](https://arxiv.org/abs/physics/9705023).
- [9] J. Sirignano, K. Spiliopoulos, DGM: A Deep learning algorithm for solving partial differential equations, *J. Comput. Phys.* 375 (2018) 1339–1364.
- [10] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (Part I): Data-driven solutions of nonlinear partial differential equations, 2017, ArXiv E-Prints [arXiv:1711.10561](https://arxiv.org/abs/1711.10561).
- [11] T. Dockhorn, A discussion on solving partial differential equations using neural networks, 2019, ArXiv E-Prints [arXiv:1904.07200](https://arxiv.org/abs/1904.07200).
- [12] J. He, L. Li, J. Xu, C. Zheng, ReLU deep neural networks and linear finite elements, 2018, ArXiv E-Prints [arXiv:1807.03973](https://arxiv.org/abs/1807.03973).
- [13] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, *Proc. Natl. Acad. Sci.* 115 (34) (2018) 8505–8510.
- [14] Q. Chan-Wai-Nam, J. Mikael, X. Warin, Machine learning for semi linear PDEs, *J. Sci. Comput.* 79 (3) (2019) 1667–1712.
- [15] P. Grohs, F. Hornung, A. Jentzen, P. Von Wurstemberger, A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations, 2018, arXiv preprint [arXiv:1809.02362](https://arxiv.org/abs/1809.02362).
- [16] J. Berner, P. Grohs, A. Jentzen, Analysis of the generalization error: Empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of black-scholes partial differential equations, *SIAM J. Math. Data Sci.* 2 (3) (2020) 631–657.
- [17] A. Jentzen, D. Salimova, T. Welti, A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of Kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients, 2018, arXiv preprint [arXiv:1809.07321](https://arxiv.org/abs/1809.07321).
- [18] J. Darbon, G.P. Langlois, T. Meng, Overcoming the curse of dimensionality for some Hamilton–Jacobi partial differential equations via neural network architectures, *Res. Math. Sci.* 7 (3) (2020) 1–50.
- [19] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient pathologies in physics-informed neural networks, 2020, arXiv preprint [arXiv:2001.04536](https://arxiv.org/abs/2001.04536).
- [20] S. Chakraborty, Transfer learning based multi-fidelity physics informed deep neural network, *J. Comput. Phys.* 426 (2021) 109942.
- [21] P. Wesseling, *Principles of Computational Fluid Dynamics*, Springer-Verlag, Berlin, Heidelberg, 2000.
- [22] O. Hölder, Ueber einen Mittelwertsatz, in: *Nachrichten Von Der Königl. Gesellschaft Der Wissenschaften Und Der Georg-Augusts-Universität Zu Göttingen*, 1889.
- [23] F. Draxler, K. Veschgini, M. Salmhofer, F.A. Hamprecht, Essentially no barriers in neural network energy landscape, 2018, ArXiv E-Prints [arXiv:1803.00885](https://arxiv.org/abs/1803.00885).
- [24] F. Bach, Breaking the curse of dimensionality with convex neural networks, *J. Mach. Learn. Res.* 18 (1) (2017) 629–681.
- [25] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Y.W. Teh, M. Titterton (Eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, in: *Proceedings of Machine Learning Research*, vol. 9 (2010) 249–256, URL <http://proceedings.mlr.press/v9/glorot10a.html>.
- [26] R.H. Byrd, P. Lu, J. Nocedal, C. Zhu, A limited memory algorithm for bound constrained optimization, *SIAM J. Sci. Comput.* 16 (5) (1995) 1190–1208.
- [27] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, ArXiv E-Prints [arXiv:1412.6980](https://arxiv.org/abs/1412.6980).
- [28] X. Kailai, Deep learning for partial differential equations PDEs, 2018.