

Journal of Physics Communications



PAPER

OPEN ACCESS

RECEIVED
17 January 2023REVISED
15 June 2023ACCEPTED FOR PUBLICATION
4 July 2023PUBLISHED
31 July 2023

Dynamic & norm-based weights to normalize imbalance in back-propagated gradients of physics-informed neural networks

Shota Deguchi and Mitsuteru Asai*

Department of Civil Engineering, Kyushu University, 744 Motooka, Nishi-ku, Fukuoka 819-0395, Japan

* Author to whom any correspondence should be addressed.

E-mail: deguchi@doc.kyushu-u.ac.jp and asai@doc.kyushu-u.ac.jp**Keywords:** physics-informed neural networks, partial differential equations, multi-objective optimization

Original content from this work may be used under the terms of the [Creative Commons Attribution 4.0 licence](#).

Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.



Abstract

Physics-Informed Neural Networks (PINNs) have been a promising machine learning model for evaluating various physical problems. Despite their success in solving many types of partial differential equations (PDEs), some problems have been found to be difficult to learn, implying that the baseline PINNs is biased towards learning the governing PDEs while relatively neglecting given initial or boundary conditions. In this work, we propose Dynamically Normalized Physics-Informed Neural Networks (DN-PINNs), a method to train PINNs while evenly distributing multiple back-propagated gradient components. DN-PINNs determine the relative weights assigned to initial or boundary condition losses based on gradient norms, and the weights are updated dynamically during training. Through several numerical experiments, we demonstrate that DN-PINNs effectively avoids the imbalance in multiple gradients and improves the inference accuracy while keeping the additional computational cost within a reasonable range. Furthermore, we compare DN-PINNs with other PINNs variants and empirically show that DN-PINNs is competitive with or outperforms them. In addition, since DN-PINN uses exponential decay to update the relative weight, the weights obtained are biased toward the initial values. We study this initialization bias and show that a simple bias correction technique can alleviate this problem.

1. Introduction

Data-driven science has provided powerful applications in a wide range of disciplines [1, 2]. With the success of neural networks and variants, machine learning models have been influencing a variety of scientific fields, including image recognition [3, 4], natural language processing [5, 6], strategy learning [7, 8], data generation [9], and engineering problems [10, 11]. Many of these successes are made in data-rich domains, where a tremendous amount of training examples are available; however, there are still many scenarios where purely data-driven methods can plateau or even fail. The leading reason for such unsatisfactory results is the insufficient amount of data, which is crucial from the machine learning perspective, especially when the model is over-parameterized. An approach to dealing with insufficient data is to incorporate prior knowledge into the learning process, which is known to improve the performance and data efficiency of machine learning models [12, 13]. In fact, a trend in scientific machine learning is currently oriented to the development of methods integrating data and prior knowledge rather than purely data-driven approaches [14, 15].

Lately, there has been a surge in attempts to evaluate physical phenomena using machine learning, with many of them seeking to couple known physical laws (governing equations) with machine learning models [15, 16]. Among them, Physics-Informed Neural Networks (PINNs) [17] has been one of the most successful approaches to tackle a wide range of problems [18–22]. PINNs incorporates constraints given by the governing partial differential equations (PDEs), initial conditions (ICs), and boundary conditions (BCs) into the loss function via automatic differentiation [23]. This allows PINNs to seamlessly integrate data with physical laws during training. Despite their success in a wide range of fields, their learning process remains unclear in terms of their convergence performance as a class of multi-objective

optimization problems. Some works report fatal failure modes in which the original formulation of PINNs (loss function composed as an unweighted linear combination of multiple components) performs poorly, mainly due to converging to trivial solutions [24–26]. Although several works have been made to improve their convergence performance (e.g. adaptive activation [27, 28], self-attention [29], gradient enhancement [30]), they do not fundamentally address learning process issues. PINNs has multiple loss terms corresponding to PDEs, ICs, and BCs, therefore, it is critical to determine relative weights to balance the convergence speed of each objective [26]. However, it can be laborious to find the optimal weights. In addition, a slight modification in network architecture (e.g. depth, width, activation function) could induce different optimal weights. Several countermeasures to such failures have been proposed in the literature (e.g. [31, 32]); however, the community has not fully explored their possibilities.

In this work, we propose Dynamically Normalized PINNs (DN-PINNs), a method to train PINNs that evenly distributes back-propagated parameter gradients to avoid the above-mentioned issues. DN-PINNs determines the relative weights of individual loss terms dynamically during the training by using gradient norms. Thus, relative weights are tuned automatically, eliminating the need for manual/brute-force parameter search. The implementation is simple, as one only needs to insert a few lines to update the weights within the training loop, but the result is compelling. In several numerical experiments, we demonstrate the effectiveness of the proposed method. We also study the initialization bias of the relative weights, which is due to exponential decay we employ. We show that the bias correction technique [33] can effectively mitigate this issue, making our proposed method robust and versatile in a wide range of problems.

2. Background and related work

2.1. PINNs: physics-informed neural networks

Consider the following initial and boundary value problem:

$$\frac{\partial}{\partial t} \mathbf{u}(t, \mathbf{x}) = \mathcal{F}[\mathbf{u}(t, \mathbf{x}); \boldsymbol{\mu}] \quad (t, \mathbf{x}) \in (0, T) \times \Omega \quad (1)$$

$$\mathbf{u}(0, \mathbf{x}) = \mathbf{f}(\mathbf{x}) \quad \mathbf{x} \in \Omega \quad (2)$$

$$\mathbf{u}(t, \mathbf{x}) = \mathbf{g}(t, \mathbf{x}) \quad (t, \mathbf{x}) \in (0, T) \times \partial\Omega_D \quad (3)$$

where Ω is the target domain ($\Omega \subset \mathbb{R}^d$, d is the dimension), $\partial\Omega_D$ is the Dirichlet boundary, and $\mathcal{F}[\cdot; \boldsymbol{\mu}]$ is a non-linear differential operator parameterized by $\boldsymbol{\mu}$ (for instance, diffusion rate in diffusion equation, or viscosity in Navier–Stokes equation). Here we remark equation (3) corresponds to Dirichlet boundary condition, and other boundary conditions can be imposed as well (e.g. for Neumann boundary condition, $\frac{\partial}{\partial n} \mathbf{u} = \mathbf{h}$, with unit normal vector \mathbf{n} on Neumann boundary, $\partial\Omega_N$). PINN [17] first approximates the solution \mathbf{u} by a neural network $\hat{\mathbf{u}}(t, \mathbf{x}; \boldsymbol{\theta})$ ($\boldsymbol{\theta}$ is a parameter vector) [34]. For a neural network whose input is $\mathbf{x} (= (t, \mathbf{x}) \in \mathbb{R}^{f_{\text{in}}})$ and output is $\hat{\mathbf{y}} (= \hat{\mathbf{u}} \in \mathbb{R}^{f_{\text{out}}})$, its forward propagation $\mathbf{z}^{(l)} (\in \mathbb{R}^{f_{\text{hidden}}^{(l)}})$ ($l = 1, 2, \dots, L$) can be written as:

$$\mathbf{z}^{(l)} = \sigma^{(l)}(\mathbf{W}^{(l)} \mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}) \quad (4)$$

where $\mathbf{z}^{(0)} = \mathbf{x}$, $\mathbf{z}^{(L)} = \hat{\mathbf{y}}$, and $\sigma^{(l)}(\cdot)$ is the activation function (element-wise non-linearity). $\mathbf{W}^{(l)} (\in \mathbb{R}^{f_{\text{hidden}}^{(l)} \times f_{\text{hidden}}^{(l-1)}})$ and $\mathbf{b}^{(l)} (\in \mathbb{R}^{f_{\text{hidden}}^{(l)}})$ are weights and biases that form $\boldsymbol{\theta} (= \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^L)$. The network parameter $\boldsymbol{\theta}$ is then trained by minimizing the following loss function:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}) + \mathcal{L}_{\text{IC}}(\boldsymbol{\theta}) + \mathcal{L}_{\text{BC}}(\boldsymbol{\theta}) (+ \mathcal{L}_{\text{Data}}(\boldsymbol{\theta})) \quad (5)$$

where \mathcal{L}_{PDE} represents a loss term to penalize PDE residual, \mathcal{L}_{IC} and \mathcal{L}_{BC} are associated with IC/BC, respectively. $\mathcal{L}_{\text{Data}}$ could be added to equation (5) if there are available data within Ω . For typical problems, each term takes the following form:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_{\text{PDE}}} \sum_i^{N_{\text{PDE}}} \left| \frac{\partial}{\partial t} \hat{\mathbf{u}}_i - \mathcal{F}[\hat{\mathbf{u}}_i; \boldsymbol{\mu}] \right|^2 \quad (6)$$

$$\mathcal{L}_{\text{IC}} = \frac{1}{N_{\text{IC}}} \sum_i^{N_{\text{IC}}} |\hat{\mathbf{u}}_i - \mathbf{f}(\mathbf{x}_i)|^2 \quad (7)$$

$$\mathcal{L}_{\text{BC}} = \frac{1}{N_{\text{BC}}} \sum_i^{N_{\text{BC}}} |\hat{\mathbf{u}}_i - \mathbf{g}(t_i, \mathbf{x}_i)|^2 \quad (8)$$

$$\mathcal{L}_{\text{Data}} = \frac{1}{N_{\text{Data}}} \sum_i^{N_{\text{Data}}} |\hat{\mathbf{u}}_i - \mathbf{u}(t_i, \mathbf{x}_i)|^2 \quad (9)$$

where $\hat{\mathbf{u}}_i = \hat{\mathbf{u}}(t_i, \mathbf{x}_i; \boldsymbol{\theta})$. In the present work, we assume that the physical parameter $\boldsymbol{\mu}$ is known; hence, we focus on forward problems. If $\boldsymbol{\mu}$ is unknown and any data are available to define equation (9), one can extend PINNs to inverse problems as well [17, 18, 35]. One can rewrite equation (5) in the general form as follows.

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_j \mathcal{L}_j(\boldsymbol{\theta}) \quad (10)$$

where \mathcal{L}_j represents the loss component associated with PDE residual, IC, BC, etc. In general, $\boldsymbol{\theta}$ is learned with a gradient decent method or one of its variants, such as [33, 36].

$$\begin{aligned} \boldsymbol{\theta}^{(n+1)} &= \boldsymbol{\theta}^{(n)} - \eta^{(n)} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(n)}) \\ &= \boldsymbol{\theta}^{(n)} - \eta^{(n)} \sum_j \nabla_{\boldsymbol{\theta}} \mathcal{L}_j(\boldsymbol{\theta}^{(n)}) \end{aligned} \quad (11)$$

where $\boldsymbol{\theta}^{(n)}$ and $\eta^{(n)}$ are the parameter and the learning rate at the n -th epoch, respectively.

2.2. Weight tuning to train PINNs

2.2.1. Static weight tuning

It is natural to extend the equation (10) by introducing relative weights to each component.

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_j \lambda_j \mathcal{L}_j(\boldsymbol{\theta}) \quad (12)$$

where $\lambda_j (\in \mathbb{R}^+)$ is a weight assigned to \mathcal{L}_j , controlling the relative importance of \mathcal{L}_j to \mathcal{L}_k (for $j \neq k$). Accordingly, gradient descent is rewritten as follows.

$$\boldsymbol{\theta}^{(n+1)} = \boldsymbol{\theta}^{(n)} - \eta^{(n)} \sum_j \lambda_j \nabla_{\boldsymbol{\theta}} \mathcal{L}_j(\boldsymbol{\theta}^{(n)}) \quad (13)$$

It is critical to choose an appropriate λ_j as it drastically affects the learning procedure and the performance of PINNs without any change in the architecture [24, 26, 37]. Many works (e.g. [19, 24, 35, 37]) often treat λ_j as hyper-parameters, which are fixed prior to training, hence, we refer to such weight tuning as ‘static weight tuning’. In many cases, the choice of λ_j is problem specific, making the cost to find the optimal λ_j prohibitively expensive, although several techniques can exploit these difficulties [38, 39].

2.2.2. Dynamic weight tuning

Back-propagated gradients often provide beneficial information to understand the training process of neural networks [40, 41]. Wang *et al* [31] have shown that regular training with equation (10) often causes an imbalance in multiple parameter gradients, $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{PDE}}$, $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{IC}}$, $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{BC}}$, etc. Observing that $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{PDE}}$ often dominates other gradients, they consider some positive weight $\lambda_j^{(n)} (\in \mathbb{R}^+)$ which approximately satisfies the following.

$$\max \{ |\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}^{(n)})| \} = \lambda_j^{(n)} |\nabla_{\boldsymbol{\theta}} \mathcal{L}_j(\boldsymbol{\theta}^{(n)})| \quad (14)$$

Equation (14) then yields the following dynamic weight.

$$\hat{\lambda}_j^{(n)} = \frac{\max \{ |\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}^{(n)})| \}}{\lambda_j^{(n-1)} \overline{|\nabla_{\boldsymbol{\theta}} \mathcal{L}_j(\boldsymbol{\theta}^{(n)})|}} \quad (15)$$

$$\lambda_j^{(n)} = \beta \lambda_j^{(n-1)} + (1 - \beta) \hat{\lambda}_j^{(n)} \quad (16)$$

where $\lambda_j^{(n)}$ is the relative weight assigned to \mathcal{L}_j at n -th epoch ($\lambda_j^{(n)}$ replaces λ_j in equation (13)). $|\cdot|$ is the element-wise absolute value, $\overline{(\cdot)}$ is the arithmetic mean, $\beta (\in [0, 1])$ is the exponential decay rate, and $\lambda_j^{(0)}$ is chosen as $\lambda_j^{(0)} = 1$. Although equations (15) is the weight directly obtained from (14), they apply exponential decay to smooth out the noisy behavior of gradient descent methods (equation (16)). In this method, the weights are determined by the ratio of the maximum value of $|\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{PDE}}|$ to the average value of $|\nabla_{\boldsymbol{\theta}} \mathcal{L}_j|$, hence we refer to it as ‘Maximum Average PINNs (MA-PINNs)’. The update interval for these relative weights is arbitrary, but it is recommended that it be large enough to avoid an additional computational cost increase, especially when the network is deep and wide.

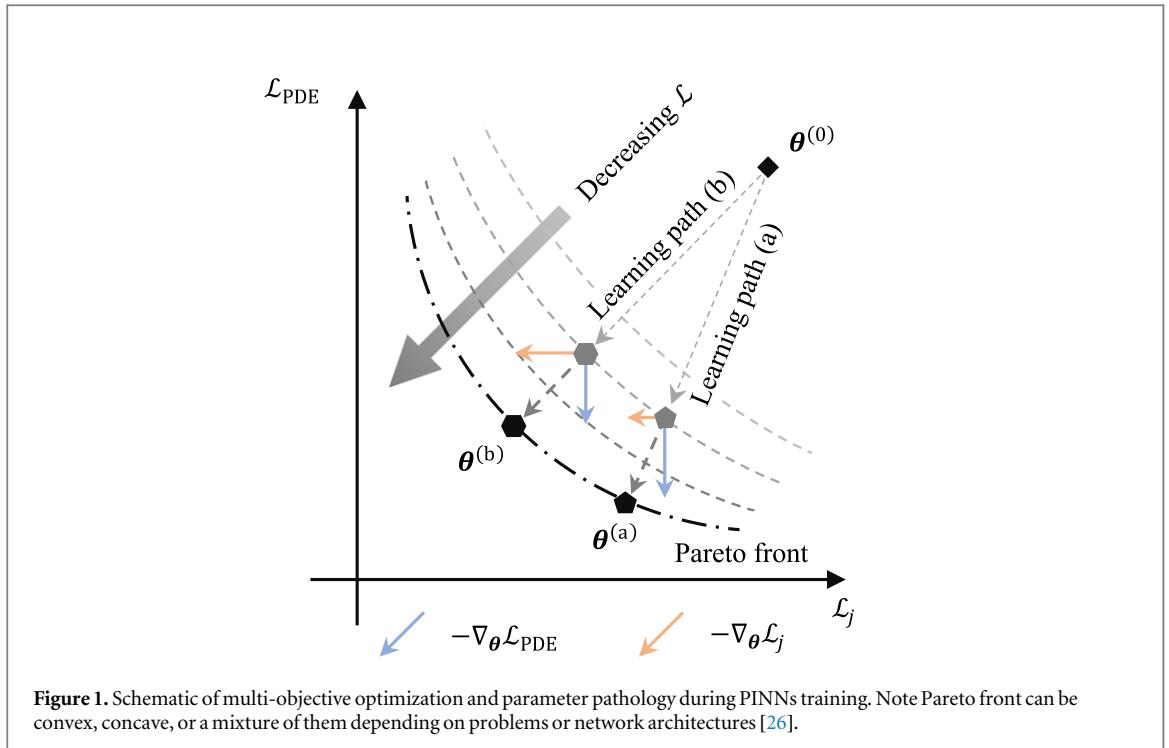
Similarly, Maddu *et al* [32] considered a positive scalar to equalize the variance of each gradient. Assuming that gradients are normally distributed ($\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{PDE}} \sim \mathcal{N}(m_{\text{PDE}}, v_{\text{PDE}}^2)$ and $\nabla_{\boldsymbol{\theta}} \mathcal{L}_j \sim \mathcal{N}(m_j, v_j^2)$, where $\mathcal{N}(m, v^2)$ is a normal distribution with mean m and variance v^2), we have $\lambda_j \nabla_{\boldsymbol{\theta}} \mathcal{L}_j \sim \mathcal{N}(\lambda_j m_j, \lambda_j^2 v_j^2)$. For the variance of $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{PDE}}$ and that of $\nabla_{\boldsymbol{\theta}} \mathcal{L}_j$ be (approximately) equal at n -th epoch, we have:

$$v[\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}^{(n)})] = (\lambda_j^{(n)})^2 v[\nabla_{\boldsymbol{\theta}} \mathcal{L}_j(\boldsymbol{\theta}^{(n)})] \quad (17)$$

which gives the following dynamic weight.

$$\hat{\lambda}_j^{(n)} = \frac{s[\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}^{(n)})]}{s[\nabla_{\boldsymbol{\theta}} \mathcal{L}_j(\boldsymbol{\theta}^{(n)})]} \quad (18)$$

where $s[\cdot]$ is the sample standard deviation (equation (18) is valid when we have only one set of PDEs as \mathcal{L}_{PDE} has the highest order of derivatives; see [32] for the generalized form with multiple sets of governing PDEs). Applying



exponential decay, equation (16) is performed after equation (18). This weight tuning scheme keeps the variance of each gradient component approximately equal, and such weights are proportional to the inverse of the square root of the Dirichlet energy of each loss component. Therefore, we refer to this method as ‘Inverse Dirichlet PINNs (ID-PINNs)’.

3. Dynamic and norm-based weights to normalize imbalance in back-propagated gradients

To obtain the solution of the PDEs, one needs to properly impose initial and boundary conditions. For instance, the trivial solution $\mathbf{u} = \mathbf{0}$ could exactly satisfy equation (1), but is of no interest from a physical perspective, which means we need to avoid converging to the parameter θ that yields $\hat{\mathbf{u}}(\cdot; \theta) \approx \mathbf{0}$. Unfortunately, the training of PINNs is often biased towards learning such trivial solutions, which is characterized as one of the main failure modes [25, 26, 42]. This behavior may imply that the effect of \mathcal{L}_{IC} , \mathcal{L}_{BC} , etc is relatively neglected. In fact, the back-propagated gradient of each loss term is often found to be imbalanced; in particular, it is commonly reported that the gradient of the PDE residual ($\nabla_{\theta} \mathcal{L}_{\text{PDE}}$) is dominant over that of other terms ($\nabla_{\theta} \mathcal{L}_j$) [31, 32]. This indicates that regular PINNs training is biased to satisfy the governing equations (PDEs), while neglecting the initial and boundary conditions (see figure 1 for a schematic description).

MA-PINNs and ID-PINNs (equation (15), (18), and exponential decay in (16)) have been found to be effective training methods to avoid the aforementioned problem. However, instead of defining weights based on gradient statistics, it is more natural to use their norms for normalization, respecting that they are vectors instead of samples from a probability distribution. Moreover, gradient norms emerge naturally from Taylor expansion of the loss function.

$$\begin{aligned} \mathcal{L}(\theta^{(n+1)}) &= \sum_{k=0}^{m-1} \frac{1}{k!} (\Delta \theta^{(n)} \top \nabla_{\theta})^k \mathcal{L}(\theta^{(n)}) + \mathcal{R}_m \\ \mathcal{R}_m &= \frac{1}{m!} (\Delta \theta^{(n)} \top \nabla_{\theta})^m \mathcal{L}(\theta^{(n)}) + c \Delta \theta^{(n)} \end{aligned} \quad (19)$$

where \mathcal{R}_m is the Lagrange form of the remainder of order m ($c \in (0, 1)$). Here, we consider the first-order approximation ($m = 2$) to proceed, as gradient descent methods are widely applied in neural network training. Denoting $\mathcal{L}^{(n)} = \mathcal{L}(\theta^{(n)})$ and $\mathbf{g}^{(n)} = \nabla_{\theta} \mathcal{L}(\theta^{(n)})$, we have the following.

$$\begin{aligned}\mathcal{L}^{(n+1)} - \mathcal{L}^{(n)} &= \Delta\boldsymbol{\theta}^{(n)\top} \mathbf{g}^{(n)} \\ \sum_j \Delta\mathcal{L}_j^{(n)} &= -\eta^{(n)} \mathbf{g}^{(n)\top} \mathbf{g}^{(n)} \\ &= -\eta^{(n)} \left\| \sum_j \lambda_j^{(n)} \mathbf{g}_j^{(n)} \right\|_2^2\end{aligned}\quad (20)$$

where $\|\cdot\|_2$ is L^2 norm. Assuming the orthogonality of each gradient and that the j -th objective $\mathcal{L}_j^{(n)}$ is reduced only by the corresponding gradient $\mathbf{g}_j^{(n)}$, we obtain $\Delta\mathcal{L}_j^{(n)} = -\eta^{(n)} \|\lambda_j^{(n)} \mathbf{g}_j^{(n)}\|_2^2$. In order to reduce multiple loss components at similar speed ($\Delta\mathcal{L}_j^{(n)} \approx \Delta\mathcal{L}_k^{(n)}$), we have:

$$\lambda_j^{(n)} = \frac{\|\mathbf{g}_k^{(n)}\|_2}{\|\mathbf{g}_j^{(n)}\|_2} \quad (21)$$

where we set $\lambda_k^{(n)} = 1.0$ to only consider the relative importance of the objective $\mathcal{L}_j^{(n)}$ to $\mathcal{L}_k^{(n)}$. In practice, we take $\mathbf{g}_k = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{PDE}}$. We found that direct use of equation (21) is unstable, hence, apply exponential decay similar to [31, 32]:

$$\hat{\lambda}_j^{(n)} = \frac{\|\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}^{(n)})\|_2}{\|\nabla_{\boldsymbol{\theta}} \mathcal{L}_j(\boldsymbol{\theta}^{(n)})\|_2} \quad (22)$$

$$\lambda_j^{(n)} = \beta \lambda_j^{(n-1)} + (1 - \beta) \hat{\lambda}_j^{(n)} \quad (23)$$

where $\beta (\in [0, 1])$ is the decay rate. We have observed empirically that a large $\beta (\beta \rightarrow 1)$ stabilizes the training well. Similar to MA-PINNs and ID-PINNs, the proposed method requires the computation of each gradient; consequently, the computational cost increases for deep/wide neural network architectures. Therefore, λ_j should be updated every $\tau (\geq 1)$ epochs. Note that we only define the weights for \mathcal{L}_j and leave the weights for λ_{PDE} to $\lambda_{\text{PDE}} = 1.0$, because we should consider the relative importance of each component. If there are multiple sets of governing PDEs (e.g., fluid motion governed by continuity and Navier–Stokes equations), we can take the loss term with the highest order of derivative for the numerator of equation (22). This reflects the analyses given in [31, 32], which state that the gradient flow of higher-order derivative terms dominates that of others, and components with lower derivatives often tend to vanish. The proposed method determines λ_j based on gradient norms. We therefore refer to this method as ‘Dynamically Normalized PINNs (DN-PINNs)’. Since the proposed method (DN-PINNs) is classified as one of the dynamic weight tuning methods, we claim here its difference from MA-PINNs or ID-PINNs.

- MA-PINNs (proposed in [31]) sets relative weights using statistics with different characteristics, i.e., the maximum value of the gradients of PDE residual ($\max\{|\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}^{(n)})|\}$), and the arithmetic mean of other gradients ($\overline{|\nabla_{\boldsymbol{\theta}} \mathcal{L}_j(\boldsymbol{\theta}^{(n)})|}$). This does not guarantee to eliminate the imbalance in multiple gradient terms, and could underestimate or overestimate the weights as we see in section 4. In contrast, the proposed DN-PINNs sets weights by L^2 norm of each gradient, and we find it successful in evenly distributing multiple gradients in a wide range of problems.
- ID-PINNs (proposed in [32]) also determines relative weights based on statistics; in particular, it uses the sample standard deviation of the gradients instead. This avoids the underestimation or overestimation of relative weights seen in MA-PINN, as we confirm in section 4. The proposed DN-PINNs is very similar to ID-PINNs but only differs in that DN-PINNs uses the norm of gradients instead of statistics. Therefore, the performance difference is very small, as our numerical experiments show.

3.1. Bias correction

Equation (23) smooths the weight defined in equation (22). As one can see, the choice of β plays a crucial role; small $\beta (\beta \rightarrow 0)$ gives a wild change in $\lambda_j^{(n)}$, while large $\beta (\beta \rightarrow 1)$ makes a gradual change. Although our preliminary numerical experiments find that large β provides stable training of PINNs, such β causes a strong bias toward its initial values. In fact, $\beta = 1$ does not update $\lambda_j^{(n)}$, hence there is no difference from the standard PINN. Here, motivated by [33], we consider the initialization bias of $\lambda_j^{(n)}$. Recursive evaluation of equation (23) gives the following.

$$x\lambda_j^{(n)} = \beta^n \lambda_j^{(0)} + (1 - \beta) \sum_{k=1}^n \beta^{n-k} \hat{\lambda}_j^{(k)} \quad (24)$$

with its initial value $\lambda_j^{(0)}$. Taking expectation of both hand side, we have:

$$\begin{aligned}\mathbb{E}[\lambda_j^{(n)}] &= \beta^n \lambda_j^{(0)} + (1 - \beta) \mathbb{E} \left[\sum_{k=1}^n \beta^{n-k} \hat{\lambda}_j^{(k)} \right] \\ &\approx \beta^n \lambda_j^{(0)} + (1 - \beta) \sum_{k=1}^n \beta^{n-k} \mathbb{E}[\hat{\lambda}_j^{(n)}] \\ &= \beta^n \lambda_j^{(0)} + (1 - \beta^n) \mathbb{E}[\hat{\lambda}_j^{(n)}]\end{aligned}\quad (25)$$

where we used the approximation under the assumption that the change of $\hat{\lambda}_j^{(k)}$ is relatively small over different epochs, and the property of geometric progression. With the decay rate β set to $0 \leq \beta < 1$, $\beta^n \rightarrow 0$ with increasing n (training epoch). However, in the early stage of the training, $\lambda_j^{(n)}$ is considered to be biased toward its initial value $\lambda_j^{(0)}$. To remove it, one can initialize $\lambda_j^{(0)} = 0$ and introduce bias-corrected weight $\tilde{\lambda}_j^{(n)}$ [33] as:

$$\tilde{\lambda}_j^{(n)} = \frac{\lambda_j^{(n)}}{1 - \beta^n} \quad (26)$$

where $\tilde{\lambda}_j^{(n)}$ is applied to \mathcal{L}_j in place of $\lambda_j^{(n)}$. The effect of this bias correction is discussed in section 4.4.

4. Numerical experiments

In this section, we demonstrate the performance of the proposed DN-PINNs through several initial and boundary value problems and compare them with baseline PINNs [17], MA-PINNs [31], and ID-PINNs [32], as described in sections 2 and 3. Throughout all experiments presented, we employ fully-connected feed-forward neural network with $\tanh(\cdot)$ activation with Glorot initialization [40] (network depth L and width $f_{\text{hidden}}^{(l)}$ are specified in each problem). We employ Adam [33] with a constant learning rate $\eta = 1 \times 10^{-3}$, decay rates for the first and second moment estimates being $\beta_1 = 0.9$ and $\beta_2 = 0.999$, respectively. The exponential decay rate for $\lambda_j^{(n)}$ is set to $\beta = 0.9$ and $\beta = 0.99$, and $\lambda_j^{(n)}$ is updated every 10 epochs (i.e. $\tau = 10$). Note that several works in the literature apply L-BFGS optimization [43] after gradient descent (such as [17, 24, 29]), however, these second-order optimization methods are generally expensive and may not be feasible for deep and wide neural network architectures. Moreover, since the proposed method as well as [31] and [32] focuses on gradient descent (first-order optimization), we only applied Adam for training. For the performance evaluation, we compare the relative L^2 error, ϵ , from the reference solution.

$$\epsilon = \frac{\|\hat{\mathbf{u}} - \mathbf{u}\|_2}{\|\mathbf{u}\|_2} \quad (27)$$

where \mathbf{u} is the reference, and $\hat{\mathbf{u}}$ is the inference solution. Here, we report the mean and standard error of the relative L^2 error for 10 independent runs (i.i.d. network initializations). All methods are implemented in TensorFlow [44] and the experiments are carried out with NVIDIA RTX A6000 (with 48 GiB memory).

4.1. 1D Burgers' equation

We first consider the following 1D Burgers' equation, which has been widely studied as a benchmark problem for PINNs [17, 27, 29, 45]. Burgers' equation is one of the fundamental convection–diffusion systems and can be considered as a simplified Navier–Stokes equation with a sharp solution transition.

$$\frac{\partial u}{\partial t} + (u \cdot \nabla) u = \nu \nabla^2 u \quad (28)$$

$$u(0, x) = -\sin(\pi x) \quad (29)$$

$$u(t, \pm 1) = 0 \quad (30)$$

where $u = u(t, x)$, $t \in [0, 1]$, $x \in [-1, 1]$, and ν is the viscosity set to $\nu = 0.01/\pi$. For training, we set $L = 9$, $f_{\text{hidden}}^{(l)} = 20$, and the number of training points as $N_{\text{PDE}} = 10,000$, $N_{\text{IC}} = 100$, $N_{\text{BC}} = 200$, which are consistent with [17, 29]. We performed full-batch training for 30,000 epochs.

Table 1 summarizes the results obtained by the baseline PINN, MA-PINN, ID-PINN, and the proposed DN-PINN. For $\beta = 0.9$, we observe MA-PINN achieves the best performance among variants, while for $\beta = 0.99$, ID-PINN appears to be the best. Figure 2 shows the inference result and the absolute error of the baseline PINN and variants. Figure 3 shows the back-propagated gradient distribution of each method after training. In figure 2, we observe that the baseline PINN is able to learn the overall solution but still struggles to properly capture the initial condition, especially in $t \in [0, 0.4]$. This may be explained in figure 3, which shows that for PINN, $\nabla_\theta \mathcal{L}_{\text{IC}}$ and $\nabla_\theta \mathcal{L}_{\text{BC}}$ have relatively vanished compared to $\nabla_\theta \mathcal{L}_{\text{PDE}}$. On the contrary, again in figure 2, we see a small discrepancy between the reference and the inference solution for MA-PINN, ID-PINN,

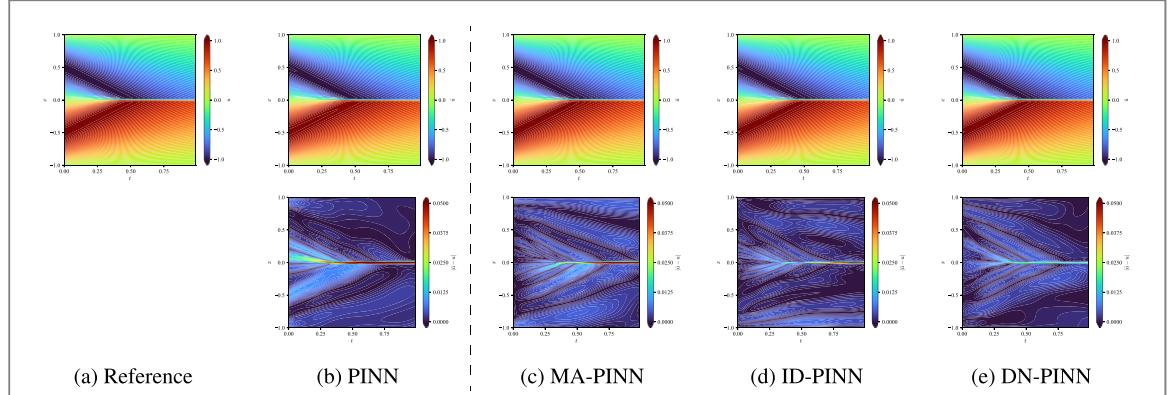


Figure 2. 1D Burgers equation. Reference and inference solution (top) and point-wise absolute error (bottom) by the baseline PINN and variants. For the presented results, $\beta = 0.99$ is chosen for MA-PINN, ID-PINN, and DN-PINN.

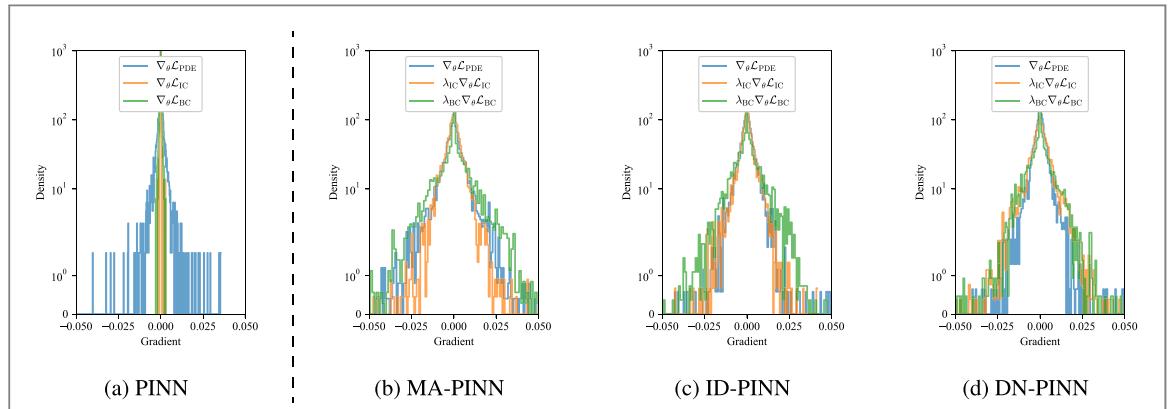


Figure 3. 1D Burgers' equation. Distribution of back-propagated gradients of the baseline PINN and variants after 30,000 epochs of full-batch training. For the presented results, $\beta = 0.99$ is chosen for MA-PINN, ID-PINN, and DN-PINN.

Table 1. 1D Burgers' equation. Mean of relative L^2 error (\pm standard error) for 10 runs and training time for 30,000 epochs of full-batch training (+ increase from the baseline PINN in percentile).

Method	β	Rel. L^2 Err. $\epsilon (\times 10^{-3})$	Training time (s)
PINN [17]	...	9.26 (± 1.36)	526.55
MA-PINN [31]	0.9	7.68 (± 0.74)	701.31 (+33%)
ID-PINN [32]	0.9	10.26 (± 2.62)	705.08 (+34%)
DN-PINN (Ours)	0.9	12.70 (± 2.25)	703.61 (+34%)
MA-PINN [31]	0.99	8.58 (± 1.79)	704.95 (+34%)
ID-PINN [32]	0.99	7.12 (± 1.28)	704.98 (+34%)
DN-PINN (Ours)	0.99	7.49 (± 1.12)	703.45 (+34%)

and DN-PINN, including $t \in [0, 0.4]$ region. In addition, figure 3 illustrates that each gradient is evenly distributed for the dynamic weighting schemes presented, which explains a small error near the $t = 0$. One drawback of dynamic weighting schemes is their additional computational cost. Table 1 shows that all dynamic weighting schemes increase the training time from the baseline PINN. However, an increase of 30%~40% is reasonable considering the elimination of manual parameter search and the improvement obtained by introducing dynamic weights.

To further investigate the learning process of PINNs, in figure 4, we plot the individual loss \mathcal{L}_{PDE} , \mathcal{L}_{IC} , \mathcal{L}_{BC} , relative L^2 error, ϵ , and relationship between loss components (loss values are normalized by its initial value to realize the convergence rate). Here, the relative weights of MA-PINN, ID-PINN, and DN-PINN are removed in figure 4 for comparison with PINN. For $\mathcal{L}_{\text{IC}}-\mathcal{L}_{\text{PDE}}$ relationship and $\mathcal{L}_{\text{BC}}-\mathcal{L}_{\text{PDE}}$ relationship (figures 4(e) and 4(f)), we plot the lowest loss achieved by 10 runs with pentagons and plot the training trajectories of PINN and DN-PINN (lighter color, earlier stage of training; darker color, later stage, only the trajectories of PINN and

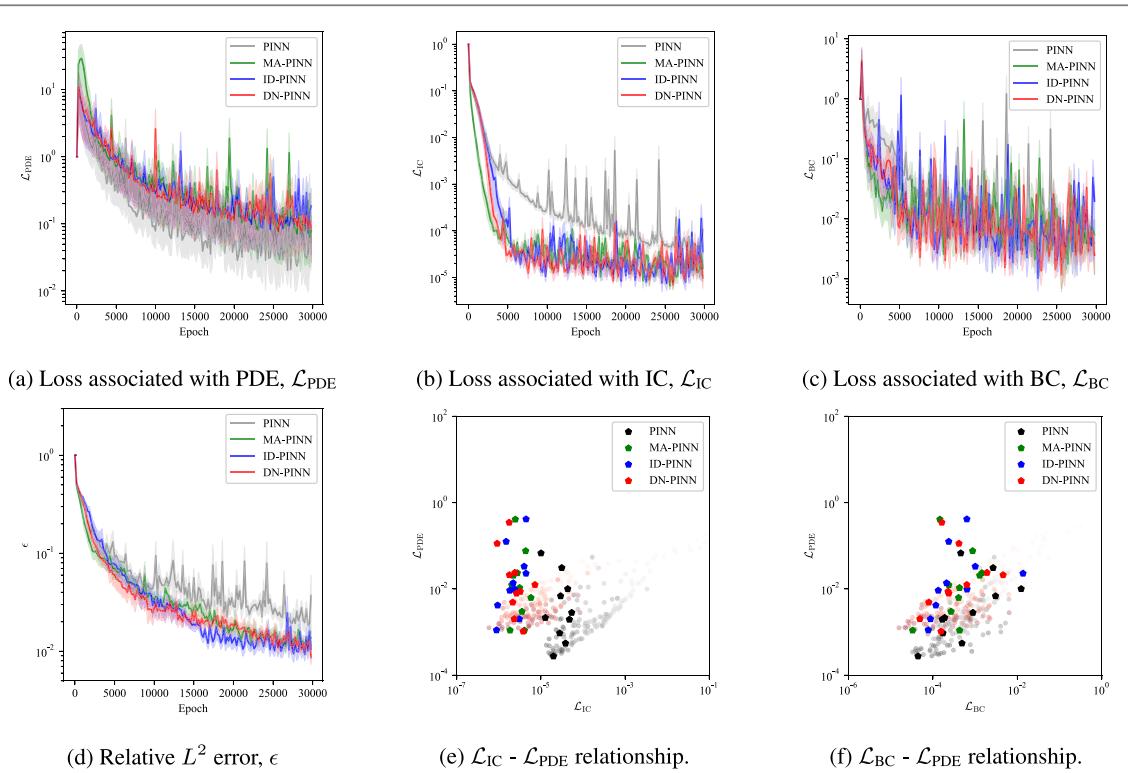


Figure 4. 1D Burgers' equation. (a), (b), (c) Loss components. (d) Relative L^2 error, ϵ . (e), (f) Relationship of multiple loss components (pentagons indicate the lowest loss achieved by 10 independent runs). For (a)–(d), solid lines indicate mean and shaded areas indicate standard error for 10 runs. All figures are plotted every 200 epochs, and $\beta = 0.99$ is chosen for MA-PINN, ID-PINN, and DN-PINN for the presented results.

DN-PINN are shown for visualization purpose, but we find that those of MA-PINN and ID-PINN are similar to DN-PINN). Figure 4 (e) shows that MA-PINN, ID-PINN, and DN-PINN, compared to the baseline PINN, are possible to reduce \mathcal{L}_{IC} even better, while being unable to lower \mathcal{L}_{PDE} any further. This result is fairly natural; as is well known on the Pareto front, PINNs (or many multi-objective optimizations) cannot further reduce one loss component without increasing another [26]. Again, from figures 3, it is evident that PINN tends to greatly respect \mathcal{L}_{PDE} and decrease it rapidly while paying little attention to \mathcal{L}_{IC} , resulting in a large accumulation of errors in $t \in [0, 0.4]$. MA-PINN, ID-PINN, and DN-PINN, on the other hand, successfully avoid the Pareto optimum where the standard PINN is trapped due to the effect of relative weights λ_{IC} . Though they cannot reduce \mathcal{L}_{PDE} as much as PINN does, they are able to lower \mathcal{L}_{IC} in return. As a consequence, they yield a lower relative L^2 error compared to the baseline PINN. For the $\mathcal{L}_{\text{PDE}} - \mathcal{L}_{\text{BC}}$ relationship, we find very little difference among them. This indicates that the loss component associated with the boundary condition is of relatively small importance, as \mathcal{L}_{BC} plateaus at almost the same level for all methods (see figure 4(f)).

From this comparative study, although we did not find a significant improvement over the baseline PINN for this problem, we stress that all dynamic weighting schemes were successful in evenly distributing multiple loss gradients, as illustrated in figure 3, and changed the learning dynamics of the PINNs as shown in figure 4. Besides, they only require a reasonable computational cost for the improvement, although it varies depending on the choice of τ (we chose $\tau = 10$ in this study).

4.2. 2D Poisson equation

As another evaluation and comparison of PINN and variants, we next consider the 2D Poisson equation. The Poisson equation is closely related to many natural science and engineering problems, such as electromagnetics and fluid mechanics.

$$\nabla^2 u = q(x, y) \quad (31)$$

$$u(x, 0) = u(x, 1) = 0 \quad (32)$$

$$u(0, y) = u(1, y) = \sin(\omega y) \quad (33)$$

where $u = u(x, y)$, $x \in [0, 1]$, $y \in [-1, 1]$. The analytical solution for equation (31) can be fabricated as $u = \cos(\omega x)\sin(\omega y)$ with the forcing term $q = -2\omega^2 \cos(\omega x)\sin(\omega y)$. ω is a parameter to control the oscillation frequency of the solution, set as $\omega = 4\pi$. For this problem, we used a network with $L = 5$, $f_{\text{hidden}}^{(l)} = 50$, $N_{\text{PDE}} = 2,500$, $N_{\text{BC}} = 400$, and performed full-batch training for 30,000 epochs [32].

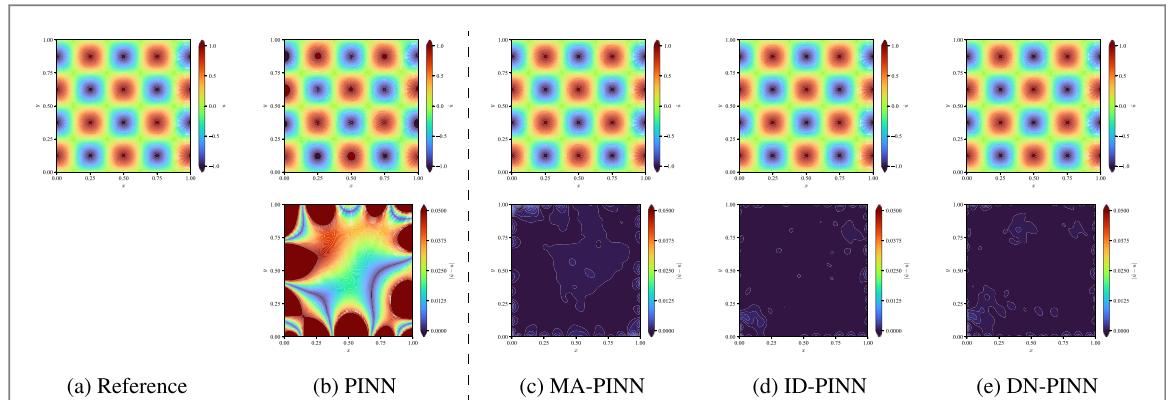


Figure 5. 2D Poisson equation. Reference and inference solution (top) and point-wise absolute error (bottom) by the baseline PINN and variants. For the presented results, $\beta = 0.99$ is chosen for MA-PINN, ID-PINN, and DN-PINN.

Table 2. 2D Poisson equation. Mean of relative L^2 error (\pm standard error) for 10 runs and training time for 30,000 epochs of full-batch training (+ increase from the baseline PINN in percentile).

Method	β	Rel. L^2 Err. $\epsilon (\times 10^{-3})$	Training time (s)
PINN [17]	...	232.29 (± 19.18)	245.30
MA-PINN [31]	0.9	2.05 (± 0.11)	297.25 (+21%)
ID-PINN [32]	0.9	1.47 (± 0.11)	300.04 (+22%)
DN-PINN (Ours)	0.9	1.41 (± 0.11)	298.10 (+22%)
MA-PINN [31]	0.99	2.06 (± 0.19)	299.77 (+22%)
ID-PINN [32]	0.99	1.13 (± 0.11)	298.66 (+22%)
DN-PINN (Ours)	0.99	1.11 (± 0.11)	297.92 (+21%)

Table 2 summarizes the results by PINN, MA-PINN, ID-PINN, and DN-PINN. We find that DN-PINN slightly outperforms other variants for both $\beta = 0.9$ and 0.99 , followed by ID-PINN, MA-PINN, and PINN, but the performance of DN-PINN and that of ID-PINN are almost identical. Clearly, the baseline PINN works poorly even on this simple problem. In particular, figure 5 shows that PINN accumulates a large error near the boundaries, while reducing the error near the geometric center of the domain. Similar to the previous example, this behavior can be interpreted by looking at the back-propagated gradient distribution and the convergence property of the loss function. In figure 6(a), for PINN, it can be seen that the parameter gradient of the PDE residual $\nabla_{\theta} \mathcal{L}_{\text{PDE}}$ is widely distributed, helping to reduce \mathcal{L}_{PDE} . However, the gradient of the boundary condition $\nabla_{\theta} \mathcal{L}_{\text{BC}}$ is concentrated near zero and therefore the decrease of \mathcal{L}_{BC} is slowed down (see figure 7(d)). Note that in figure 7(d), we plot the lowest loss achieved by 10 runs with pentagons and plot the training trajectories of PINN and DN-PINN (lighter color, earlier stage of training; darker color, later stage, only the trajectories of PINN and DN-PINN are shown). As a result, the baseline PINN is driven to provide a general solution rather than a specific solution to the given boundary value problem. On the other hand, the effect of dynamic weight tuning methods is evident in this problem. In particular, MA-PINN greatly reduces the error near the boundaries, resulting in a relative L^2 error that is nearly 100 times smaller than that of PINN. From figure 7, we can see that MA-PINN has successfully decreased \mathcal{L}_{BC} as well as \mathcal{L}_{PDE} . However, in figure 6, we observe that MA-PINN still struggles to alleviate the gradient imbalance. On the contrary, ID-PINN and DN-PINN successfully reduce the error throughout the domain as shown in figure 5, while distributing $\nabla_{\theta} \mathcal{L}_{\text{PDE}}$ and $\nabla_{\theta} \mathcal{L}_{\text{BC}}$ equally (figures 6(c) and (d)). Similarly to the previous 1D Burgers problem, the additional computational cost of MA-PINN, ID-PINN, and DN-PINN remains small ($\sim 30\%$), while the gain in accuracy is $\sim 100\times$.

To examine the learning process of each method, figure 7 illustrates the relationship between \mathcal{L}_{PDE} and \mathcal{L}_{BC} , loss values of \mathcal{L}_{PDE} and \mathcal{L}_{BC} during training (normalized by their initial values to realize the convergence rate). Evidently, PINN can reduce \mathcal{L}_{PDE} , but it fails to deal with \mathcal{L}_{BC} even with a long training time. In particular, the loss associated with the boundary condition \mathcal{L}_{BC} remains 4~5 orders of magnitude greater than the loss of PDE residual \mathcal{L}_{PDE} . These observations imply that regular PINN training is biased to satisfy the governing PDE with little attention is paid to the imposed boundary condition. Figure 7 shows that MA-PINN effectively reduces both \mathcal{L}_{PDE} and \mathcal{L}_{BC} . As a result, it achieves an improvement of almost 2 orders of magnitude over the baseline PINN in relative L^2 error, as summarized in table 2. Moreover, ID-PINN and DN-PINN achieve even lower

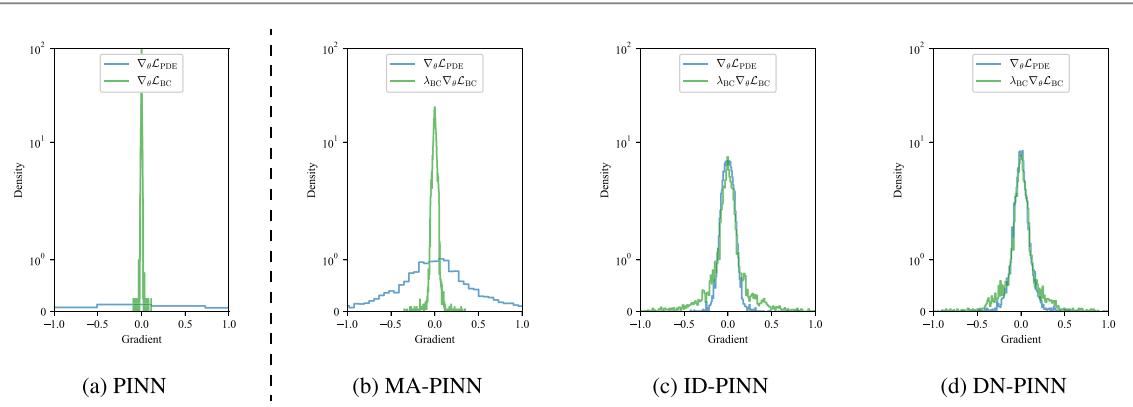


Figure 6. 2D Poisson equation. Distribution of back-propagated gradients of the baseline PINN and variants after 30,000 epochs of full-batch training. For the presented results, $\beta = 0.99$ is chosen for MA-PINN, ID-PINN, and DN-PINN.

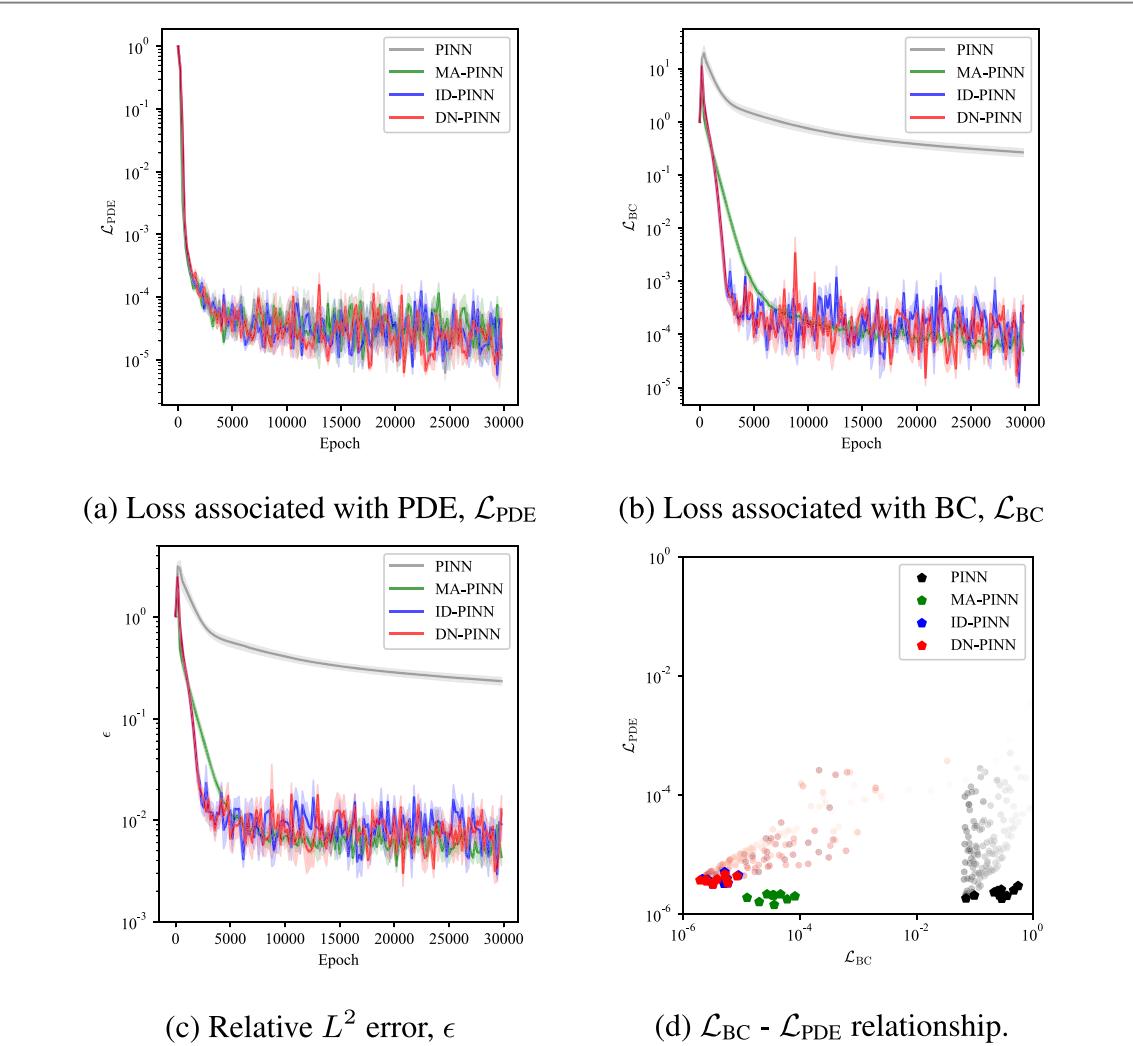


Figure 7. 2D Poisson equation. (a), (b) Loss components. (c) Relative L^2 error, ϵ . (d) Relationship of multiple loss components (pentagons indicate the lowest loss achieved by 10 independent runs). For (a)-(c), solid lines indicate mean and shaded areas indicate standard error for 10 runs. All figures are plotted every 200 epochs, and $\beta = 0.99$ is chosen for MA-PINN, ID-PINN, and DN-PINN for the presented results.

\mathcal{L}_{BC} than MA-PINN. We consider this to be due to the aforementioned incomplete alleviation of the gradient imbalance found in figure 6, while the gradients of \mathcal{L}_{PDE} and \mathcal{L}_{BC} in ID-PINN and DN-PINN are equally distributed. Consequently, their inference results are more accurate.

Table 3. 1D Allen-Cahn equation. Mean of relative L^2 error (\pm standard error) for 10 runs and training time for 50,000 epochs with a batch size of 4,096 (+ increase from the baseline PINN in percentile).

Method	β	Rel. L^2 err. $\epsilon (\times 10^{-2})$	Training time (s)
PINN [17]	—	22.79 (± 7.35)	8,121.97
MA-PINN [31]	0.9	3.30 (± 0.19)	8,245.85 (+2%)
ID-PINN [32]	0.9	2.08 (± 0.23)	8,246.31 (+2%)
DN-PINN (Ours)	0.9	1.60 (± 0.19)	8,245.83 (+2%)
MA-PINN [31]	0.99	2.60 (± 0.11)	8,246.16 (+2%)
ID-PINN [32]	0.99	1.50 (± 0.17)	8,245.36 (+2%)
DN-PINN (Ours)	0.99	1.58 (± 0.17)	8,246.36 (+2%)

A comparative study on this problem suggests that, in some cases, the baseline PINN may rely heavily on the PDE residual for its training while little attention is drawn to the given boundary conditions, as reported in [26, 31, 32]. These symptoms are considered due to an unequal gradient distribution over multiple objectives, as illustrated in figure 6. We observe that MA-PINN is very effective in avoiding this problem, but it still suffers from the imbalance. We also find that ID-PINN and DN-PINN could be powerful weight tuning strategies in evenly distributing multiple gradient components without being harmed by the gradient imbalance.

4.3. 1D Allen-Cahn equation

As the last example, we consider the following 1D Allen-Cahn equation, which is reported to be challenging for the baseline PINN [24]. The Allen-Cahn equation is a class of reaction-diffusion systems of which application spans from material science to biology [46, 47].

$$\frac{\partial u}{\partial t} = a_1 \nabla^2 u + a_2(u - u^3) \quad (34)$$

$$u(0, x) = x^2 \cos(\pi x) \quad (35)$$

$$u(t, -1) = u(t, 1) \quad (36)$$

$$\nabla u(t, -1) = \nabla u(t, 1) \quad (37)$$

where $u = u(t, x)$, $t \in [0, 1]$, $x \in [-1, 1]$. a_1 and a_2 are diffusion and reaction rates, respectively, and are chosen as $a_1 = 0.0001$ and $a_2 = 5$. As for the neural network architecture and problem setup, we chose $L = 5$, $f_{\text{hidden}}^{(l)} = 128$, $N_{\text{PDE}} = 20,000$, $N_{\text{IC}} = 512$, $N_{\text{BC}} = 200$ in accordance with [24]. We performed mini-batch training for 50,000 epochs with batch size 4,096. Here, mini-batching is only applied to PDE residual computation since $N_{\text{PDE}} \gg N_{\text{IC}}, N_{\text{BC}}$. Furthermore, we applied the aforementioned dynamic weights (MA-PINN, ID-PINN, and DN-PINN) only for the loss component of the initial condition, since the weights for the periodic boundary conditions have diverged for all methods.

Table 3 summarizes the results. For this problem, we again find ID-PINN and DN-PINN perform well better than MA-PINN or the baseline PINN. Furthermore, figure 8 shows the reference and inference solutions obtained by each method. For the result presented in 8, the baseline PINN is unable to capture the solution correctly, having a large discrepancy from the reference solution, especially in the sharp transition region, $x \in [-0.5, 0.5]$, whereas dynamic weighting schemes are able to produce accurate solutions. To understand the cause of PINN failure, we again look at the back-propagated gradient distribution after training, as shown in figure 9. For the baseline PINN, the gradient of IC loss ($\nabla_{\theta} \mathcal{L}_{\text{IC}}$) has a sharp distribution concentrated near zero compared to that of PDE loss ($\nabla_{\theta} \mathcal{L}_{\text{PDE}}$), implying that IC training is drastically slowed down. On the contrary, dynamic weighting schemes distribute the gradient of IC loss more widely than PINN, indicating that IC training is enhanced. These results are consistent with [24, 29], where it has been found that high attention needs to be paid to the initial condition for learning this Allen-Cahn system. In figure 9, we also find MA-PINN behaves slightly differently from ID-PINN and DN-PINN. Particularly, MA-PINN distributes $\nabla_{\theta} \mathcal{L}_{\text{IC}}$ wider than $\nabla_{\theta} \mathcal{L}_{\text{PDE}}$, which is the opposite behavior from the previous example in 4.2. On the other hand, ID-PINN and DN-PINN achieve an even distribution of both gradient components. Combined with the observation in the previous example, MA-PINNs can be potentially unstable, underestimating or overestimating relative weights depending on the problems. Note that in figure 8, one can see that even dynamic weighting schemes cannot still accurately capture the sharp transition in $t \in [0.5, 1.0]$. It is well known that PINNs struggle to capture a sharp and discontinuous spatial change or a fast temporal evolution [48]. Especially for reaction-diffusion equations, sequential learning could potentially be effective for further improvement [24, 25]. As for training time, the extra cost introduced by dynamic weights was almost negligible for this problem, because we chose mini-batch training with long epochs.

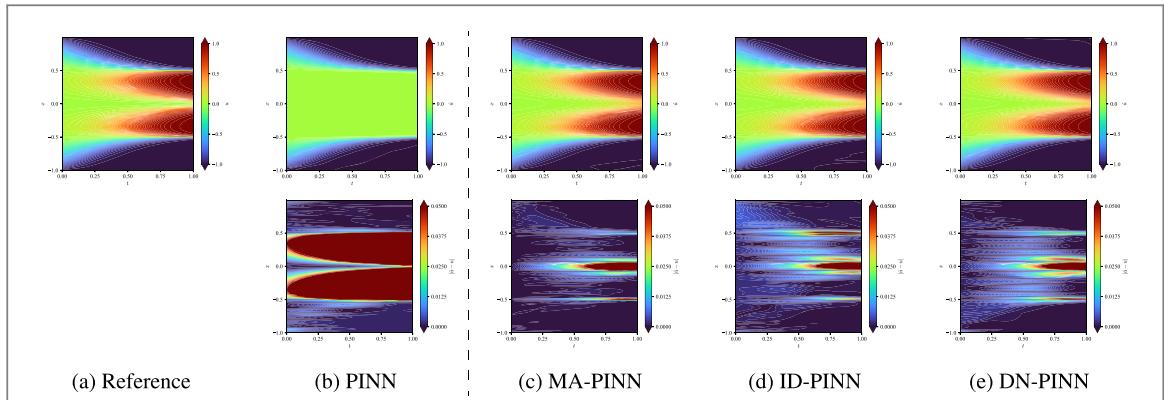


Figure 8. 1D Allen-Cahn equation. Reference and inference solution (top) and point-wise absolute error (bottom) by the baseline PINN and variants. For the presented results, $\beta = 0.99$ is chosen for MA-PINN, ID-PINN, and DN-PINN.

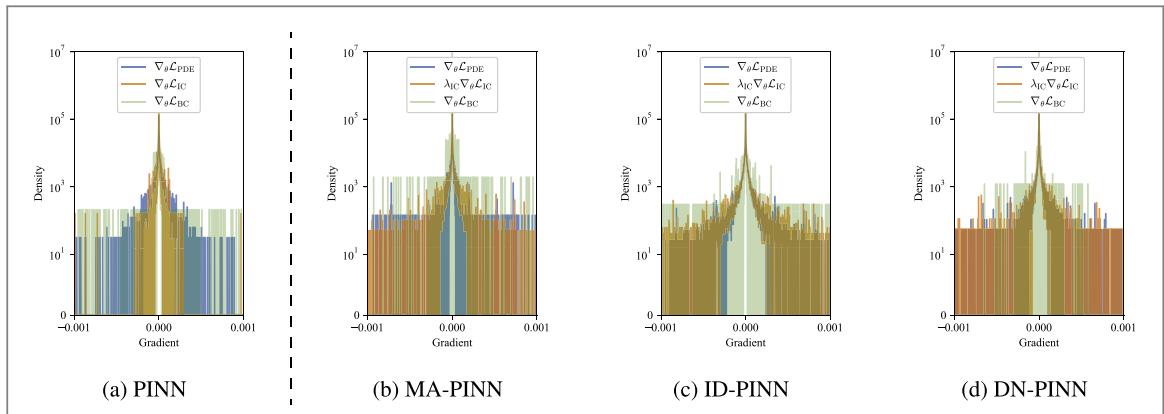


Figure 9. 1D Allen-Cahn equation. Distribution of back-propagated gradients of the baseline PINN and variants after 50,000 epochs of training with batch-size of 4,096. Note that no weight is assigned to $\nabla_\theta \mathcal{L}_{BC}$. For the presented results, $\beta = 0.99$ is chosen for MA-PINN, ID-PINN, and DN-PINN.

To better understand the failure of PINN for this problem, figure 10 shows the loss components as a function of training epochs, relative L^2 error against the reference solution and the relationship between \mathcal{L}_{PDE} and \mathcal{L}_{IC} , \mathcal{L}_{PDE} and \mathcal{L}_{BC} . Again, in figures 10(e) and 10(f), the loss values are normalized by their initial values and the relative weights are removed so we can compare with PINN. Pentagons indicate the lowest loss achieved by 10 independent runs, and only the trajectory of DN-PINN is shown here (the trajectories of MA-PINN and ID-PINN showed similar behavior). From figure 10(e), we find two clusters of PINN, one trapped around $(\mathcal{L}_{IC}, \mathcal{L}_{PDE}) = (10^{-2}, 10^{-3})$ and the other near $(\mathcal{L}_{IC}, \mathcal{L}_{PDE}) = (10^{-4}, 10^{-4})$. With the presence of the latter cluster, the former is considered to be local convergence solutions, and longer training could drive them closer to the latter cluster. Figure 11(a) shows 2 representatives from each cluster and (b) and (c) are the corresponding inference solutions. We can see that when PINN successfully reaches $(\mathcal{L}_{IC}, \mathcal{L}_{PDE}) = (10^{-4}, 10^{-4})$ then its inference solution is almost identical to those of dynamic weighting schemes (figure 11(b)), but it is also possible to reach $(\mathcal{L}_{IC}, \mathcal{L}_{PDE}) = (10^{-2}, 10^{-3})$ and yield very poor solution (figure 11(c)). We stress that they are independent runs with the exactly the same setup (e.g. depth, width, activation, learning rate, etc), but differ only in the random initialization (although taken from the same distribution), and therefore it is noteworthy that the baseline PINN may be able to capture the accurate solution without any weight tuning, but also has a high risk of failure.

While some of the PINN samples get stuck near $(\mathcal{L}_{IC}, \mathcal{L}_{PDE}) = (10^{-2}, 10^{-3})$ and have difficulty reducing \mathcal{L}_{IC} after 10,000 epochs, MA-PINN, ID-PINN, and DN-PINN constantly lower both \mathcal{L}_{PDE} and \mathcal{L}_{IC} during training, being almost invariant to initialization randomness. This indicates that proper weight tuning allows for reliable training of PINNs. As a result, they successfully avoid the Pareto optimum in which PINN is trapped, as shown in figure 10(e). Although MA-PINN seems to be the fastest to start decreasing the loss of initial condition as can be seen in figure 10(b), it fails to completely resolve gradient imbalance as observed in figure 9(b). ID-PINN and DN-PINN achieve an almost equal distribution of $\nabla_\theta \mathcal{L}_{IC}$ and $\nabla_\theta \mathcal{L}_{PDE}$, providing an even better approximation than MA-PINN. As for the boundary condition loss \mathcal{L}_{BC} , we do not observe a significant difference among PINN and variants since we did not apply weights as stated above. Similarly to Burgers'

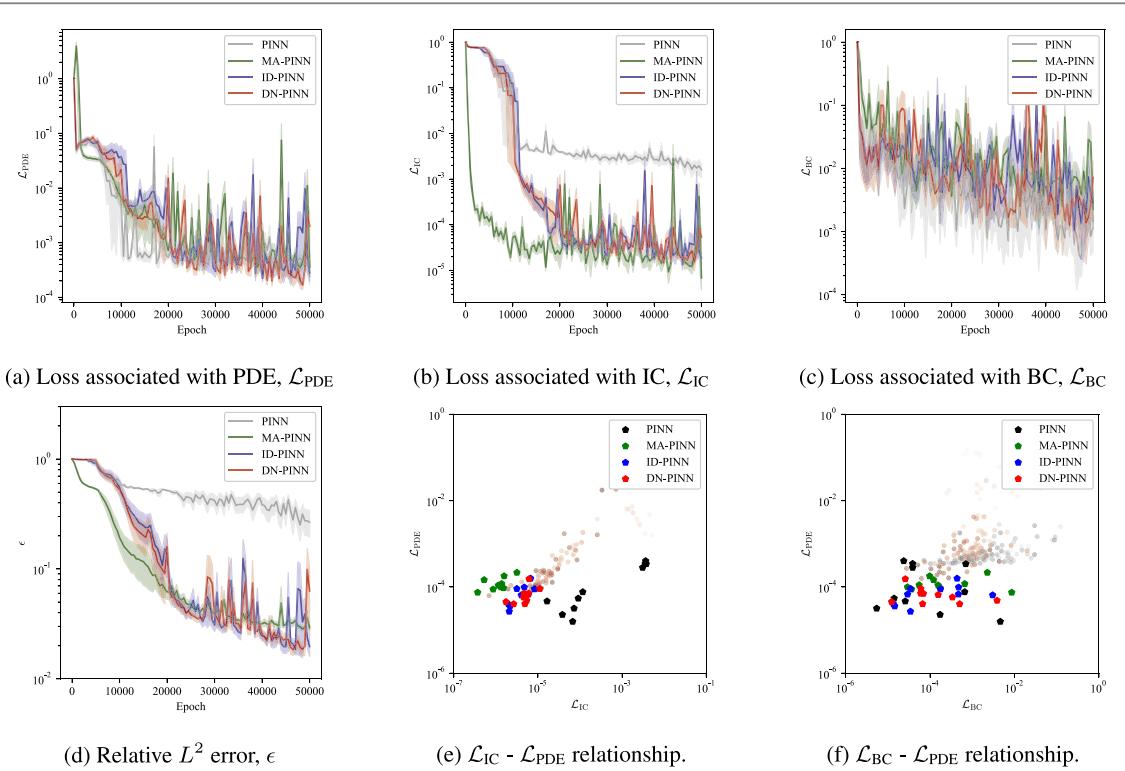


Figure 10. 1D Allen-Cahn equation. (a), (b), (c) Loss components. (d) Relative L^2 error, ϵ . (e), (f) Relationship of multiple loss components (pentagons indicate the lowest loss achieved by 10 independent runs). For (a)-(d), solid lines indicate mean and shaded areas indicate standard error for 10 runs. All figures are plotted every 500 epochs, and $\beta = 0.99$ is chosen for MA-PINN, ID-PINN, and DN-PINN for the presented results.

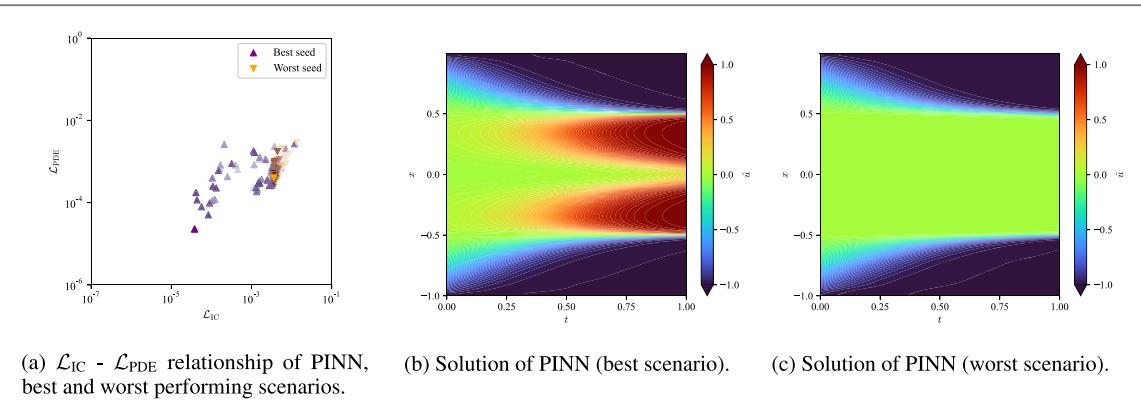


Figure 11. 1D Allen-Cahn equation. Result variation of PINN, depending on random initialization of the network. (a) $\mathcal{L}_{\text{IC}} - \mathcal{L}_{\text{PDE}}$ relationship, plotted every 500 epochs. (b) and (c) Inference result of PINN, obtained from best and worst performing scenarios, respectively.

problem, we consider this to mean that the loss of the loss associated with the initial condition \mathcal{L}_{IC} is of great importance compared to \mathcal{L}_{BC} . In fact, applying the relative weight λ_{IC} barely changes the convergence trend of \mathcal{L}_{BC} , as illustrated in figures 10(c) and (f).

4.4. Effect of bias correction

As discussed in 3.1, the choice of the exponential decay rate β is critical, but the bias correction in equation (26) can be a strong candidate to alleviate the initialization bias caused by the exponential decay. Here, we compare the results of DN-PINN without bias correction (equation (23)) and with bias correction (equation (26)). These additional experiments are performed as follows. For 1D Burgers and 1D Allen-Cahn problems, all settings (network architecture, optimizer, epochs, and batch-size) are unchanged from sections 4.1 and 4.3. For 2D Poisson problem, the neural network is trained for only 5,000 epochs (other settings are unchanged from section 4.2). This is because we have observed in section 4.2 that it quickly converges and begins to oscillate after

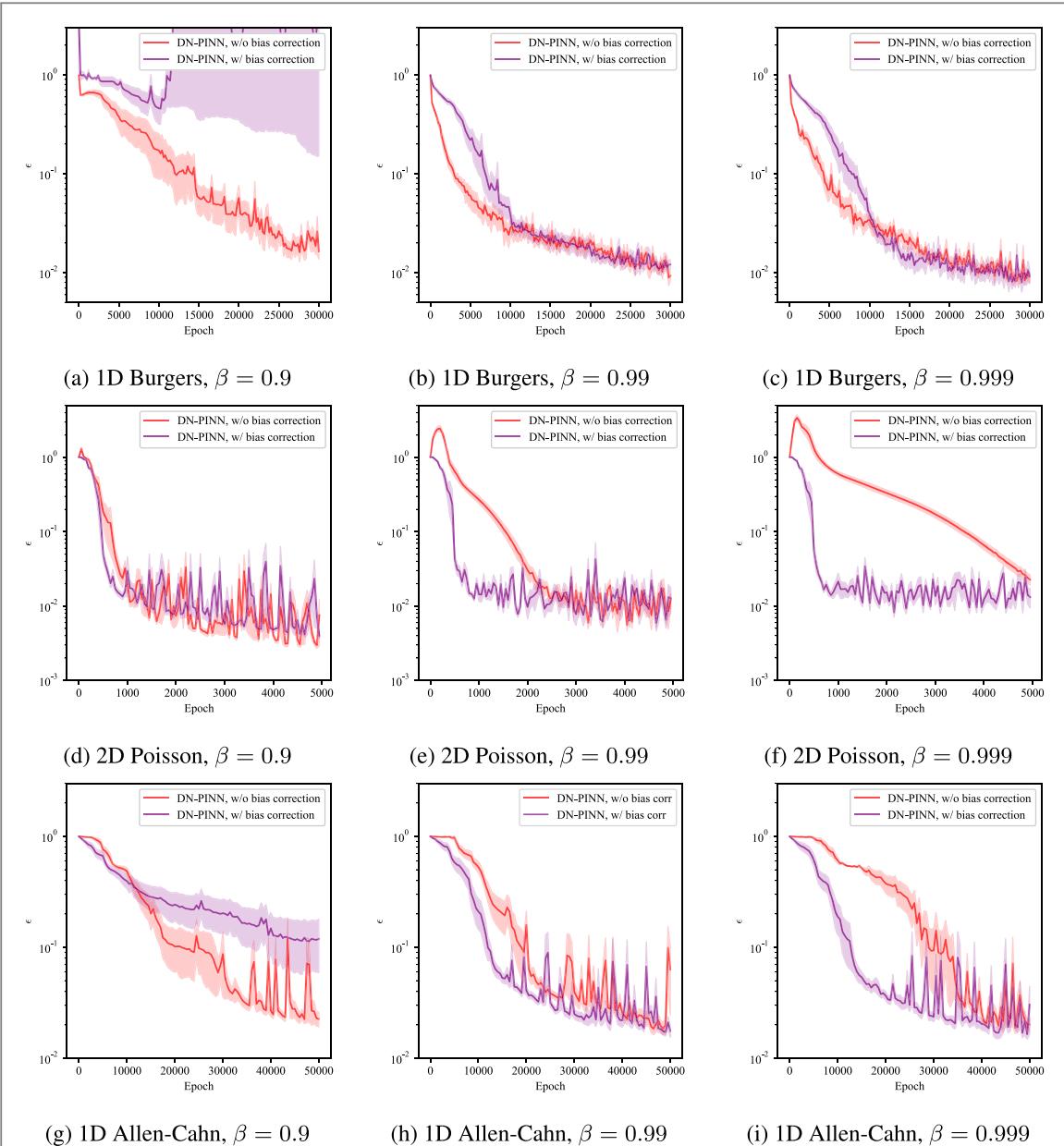


Figure 12. Effect of bias correction. Comparison of DN-PINN without and with bias correction. 1D Burgers (top row), 2D Poisson (middle), 1D Allen-Cahn problem (bottom) for different exponential decay rates β . Solid lines indicate mean and shaded areas indicate standard error for 10 independent runs.

the first few thousand epochs. We also vary the exponential decay rate as $\beta = 0.9, 0.99, 0.999$ while fixing the update interval to $\tau = 10$.

Figure 12 shows the relative L^2 error, ϵ , of DN-PINN without and with bias correction, as a function of the training epochs. For 1D Burgers problem (figures 12(a)–(c)), increasing β appears to have a subtle effect on the DN-PINN training procedure without bias correction, as the convergence speed of the relative L^2 error, ϵ , are almost unchanged for $\beta = 0.99$ and $\beta = 0.999$. In contrast, for 2D Poisson (figures 12(d)–(f)) and 1D Allen-Cahn problems (figures 12(g)–(i)), DN-PINN without bias correction tends to require more training epochs to converge as β increases. This indicates that the weight update slows down and the effect of initialization bias becomes dominant when β is close to 1. On the other hand, DN-PINN with bias correction shows a nearly identical convergence speed for $\beta = 0.99$ and $\beta = 0.999$, suggesting that they are robust to the initialization bias caused by large β . Although it can be seen that, when β is relatively small ($\beta = 0.9$), the bias correction can sometimes make training unstable (see figures 12(a) and (g)), as our numerical experiments in sections 4.1, 4.2, 4.3 suggest, large β often provides better inference accuracy, and thus the combination of large β and bias correction can be a robust and versatile improvement. We also find that the training cost increment of introducing bias correction is negligible.

5. Conclusion

In the context of physics-informed neural networks (PINNs), there has been discussion on the importance of loss weights to balance the effect of multiple loss components. In this work, we have presented DN-PINNs, a method to train PINNs while distributing the multiple gradient components evenly. The motivation of DN-PINNs comes from the failure modes of PINNs [25, 26], where the constraints of the governing PDEs are dominant over others, so the effect of initial or boundary conditions is relatively neglected. In our numerical experiments, we found that PINNs quickly reduces the loss component associated with PDEs (\mathcal{L}_{PDE}), while the reduction of loss with initial or boundary conditions (\mathcal{L}_{IC} , \mathcal{L}_{BC}) is relatively slow, again indicating that regular PINNs training prioritizes constraints associated with given PDEs. As PINNs training can be interpreted as a class of multi-objective optimization problems, the relative weights of each objective play a crucial role during model training [49]. The proposed DN-PINNs automatically tunes the relative weights for each component, reducing the cost of searching for the optimal weights.

In our experiments, although DN-PINNs assumes orthogonality of each gradient, which may not always be true, we have empirically found the improved accuracy of the DN-PINNs compared to that of the baseline PINNs. We also performed a series of comparisons with other variants in the literature, which we referred to as MA-PINNs (Max Average weighting scheme proposed in [31]) and ID-PINNs (Inverse Dirichlet weighting scheme [32]). In summary, we have shown that DN-PINNs and ID-PINNs have almost comparable performance because their definitions are very similar. However, in many cases, DN-PINNs and ID-PINNs outperformed MA-PINNs, showing their robustness to a wide range of problems. Although we found that in some cases DN-PINNs could negatively affect the results (see table 1), we did not see a significant degradation and choosing an appropriate decay rate β can avoid this problem. In addition, since the proposed DN-PINNs uses exponential decay and its performance depends on the choice of decay rate β , we have also considered the initialization bias of the relative weights, induced by β . This is of great importance since we have empirically observed that DN-PINNs prefers large β for stability, but such decay rate introduces strong initialization bias. To mitigate this, we applied bias correction motivated by Adam [33], and found that it can effectively remove initialization bias even when large β is chosen. Although we did not perform an application to inverse problems, we believe that DN-PINNs (with bias correction) can be easily extended to them, and it is one of the future scopes of this work. Here, we stress that the proposed DN-PINNs is designed to train PINNs, not to generalize to other multi-objective optimization problems (for example, the assumption of having \mathcal{L}_{PDE} as the numerator in equation (22) may not be appropriate, or an objective corresponding to \mathcal{L}_{PDE} may not exist in general multi-objective optimization).

PINNs in general still has several limitations. First, PINNs cannot properly infer solutions when the domain shape, initial or boundary conditions change. This is because it takes a spatio-temporal feature as input to learn the solution of PDEs. Therefore, out-of-domain inputs severely degrade the model performance. Second, PINNs often employs so-called ‘strong-form’ PDE loss, where derivative terms in the governing PDEs are computed by differentiating the network output. This may drastically slow down the training of the network, especially when there are higher-order derivatives and certain vanishing gradients. Operator learning methods [50, 51] and variational PDE loss formulations [52, 53] are strong candidates to address these issues. Furthermore, integration with attention mechanisms [6, 29] or approaches compatible with numerical methods [53, 54] can further develop the related research area.

Acknowledgments

This work was supported by Japan Society for the Promotion of Science (JSPS) KAKENHI Grant Number 23KJ1685, 20H02418 and 22H03601, Japan Science and Technology Agency (JST) Support for Pioneering Research Initiated by the Next Generation (SPRING) Grant Number JPMJSP2136, and the Education and Research Center for Mathematical and Data Science, Kyushu University, Japan.

Data availability statement

The data that support the findings of this study will be openly available following an embargo at the following URL/DOI: https://github.com/ShotaDeguchi/DN_PINN. Data will be available from 12 June 2023.

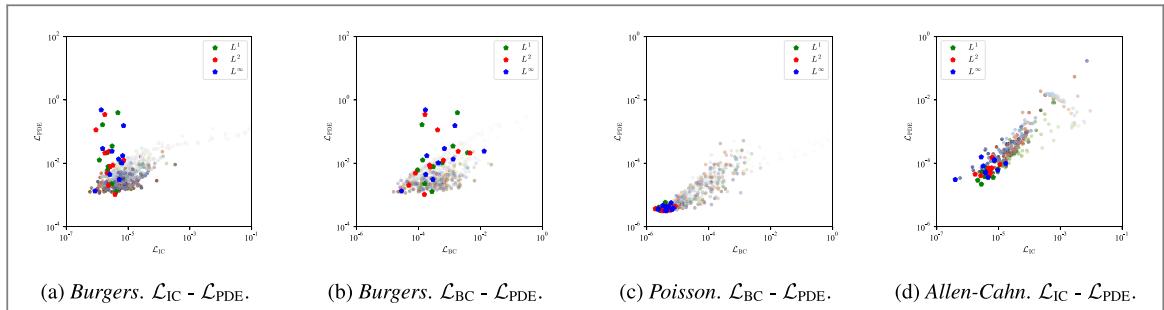


Figure 13. Extension to L^p norm. Relationship of multiple loss components using L^1, L^2, L^∞ norm to determine weights. Pentagons indicate the lowest loss achieved by 10 independent runs, and $\beta = 0.99$ is chosen for the presented results.

Table 4. Extension to L^p norm. Mean of relative L^2 error (\pm standard error) for 10 independent runs on each problem while varying the norm type, p .

Method	β	p	1D Burgers ($\times 10^{-2}$)	2D Poisson ($\times 10^{-3}$)	1D Allen-Cahn ($\times 10^{-2}$)
DN-PINN	0.9	1	17.58 (± 7.83)	1.17 (± 0.13)	1.55 (± 0.08)
	0.9	2	12.70 (± 2.25)	1.41 (± 0.11)	1.60 (± 0.19)
	0.9	∞	11.78 (± 2.72)	1.16 (± 0.06)	1.70 (± 0.19)
	0.99	1	7.65 (± 0.90)	1.09 (± 0.08)	1.28 (± 0.08)
	0.99	2	7.49 (± 1.12)	1.11 (± 0.11)	1.58 (± 0.17)
	0.99	∞	8.60 (± 1.30)	1.19 (± 0.13)	1.78 (± 0.15)

Appendix. Extension to L^p norm

One may consider extending the proposed method to use L^p norm for normalization instead of L^2 .

$$\hat{\lambda}_j^{(n)} = \frac{\|\nabla_{\theta} \mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}^{(n)})\|_p}{\|\nabla_{\theta} \mathcal{L}_j(\boldsymbol{\theta}^{(n)})\|_p} \quad (38)$$

$$\lambda_j^{(n)} = \beta \lambda_j^{(n-1)} + (1 - \beta) \hat{\lambda}_j^{(n)} \quad (39)$$

Here, we test the performance of DN-PINN while varying p for $p = 1, 2, \infty$, with three problems presented in section 4. Hyperparameters, such as network architecture, optimizer, epochs, and batch-size, are unchanged from section 4.

Table 4 summarizes the relative L^2 error from the reference solution in each problem. We skip the comparison of training costs because the choice of norm type has little impact. From table 4, we find slight variations in the results depending on the norm types, but overall there are no significant differences. Furthermore, figure 13 illustrates the relationship of individual loss components in each example. As in section 4, lighter colors indicate earlier stages of training and darker colors indicate later stages, and pentagons are the lowest loss values achieved in each run. Figure 13 shows that the choice of norm type makes insignificant changes in the individual trajectories, and their behavior is mostly identical. Overall, we observe that the choice of norm type has a subtle effect on the training and inference results, but we consider that the use of L^2 is natural and robust in many problems (at least second best performance in most problems, as reported in table 4).

ORCID iDs

Shota Deguchi <https://orcid.org/0000-0002-9538-8663>
Mitsuteru Asai <https://orcid.org/0000-0002-1124-2895>

References

- [1] My Ha D, Tkalich P and Chan E S 2008 Tsunami forecasting using proper orthogonal decomposition method *Journal of Geophysical Research: Oceans* **113** C06019
- [2] Brunton S L, Proctor J L and Kutz J N 2016 Discovering governing equations from data by sparse identification of nonlinear dynamical systems *Proceedings of the National Academy of Sciences* **113** 3932–7
- [3] Krizhevsky A, Sutskever I and Hinton G E 2012 Imagenet classification with deep convolutional neural networks *Advances in Neural Information Processing Systems* ed F Pereira et al (Lake Tahoe, Nevada: Curran Associates, Inc.) vol. 25

- [4] He K, Zhang X, Ren S and Sun J 2016 Deep residual learning for image recognition *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, IEEE Computer Society (Las Vegas, NV, USA, June 27–30, 2016)* 770–8
- [5] Cho K, van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H and Bengio Y 2014 Learning phrase representations using RNN encoder–decoder for statistical machine translation *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* 1724–34
- [6] Bahdanau D, Cho K and Bengio Y 2015 Neural machine translation by jointly learning to align and translate *3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings (San Diego, CA, USA, May 7–9, 2015)* ed Y Bengio and Y LeCun
- [7] Silver D *et al* 2016 Mastering the game of Go with deep neural networks and tree search *Nature* **529** 484–9
- [8] Fawzi A *et al* 2022 Discovering faster matrix multiplication algorithms with reinforcement learning *Nature* **610** 47–53
- [9] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A and Bengio Y 2014 Generative adversarial nets *Advances in Neural Information Processing Systems* vol. 27 ed Z Ghahramani *et al* (Curran Associates Inc.)
- [10] Dung C V and Anh L D 2019 Autonomous concrete crack detection using deep fully convolutional neural network *Autom. Constr.* **99** 52–8
- [11] Li S and Zhao X 2019 Image-based concrete crack detection using convolutional neural network and exhaustive search technique *Advances in Civil Engineering* **2019** 6520620
- [12] Daw A, Karpatne A, Watkins W, Read J and Kumar V 2017 Physics-guided neural networks (pgnn): an application in lake temperature modeling arXiv:1710.11431
- [13] Czarnecki W M, Osindero S, Jaderberg M, Swirszcz G and Pascanu R 2017 Sobolev training for neural networks *Advances in Neural Information Processing Systems* vol. 30 ed I Guyon *et al* (Curran Associates, Inc.)
- [14] von Rueden L *et al* 2021 Informed machine learning—a taxonomy and survey of integrating prior knowledge into learning systems *IEEE Trans. Knowl. Data Eng.* **35** 614–33
- [15] Karniadakis G E, Kevrekidis I G, Lu L, Perdikaris P, Wang S and Yang L 2021 Physics-informed machine learning *Nature Reviews Physics* **3** 422–40
- [16] Brunton S L, Noack B R and Koumoutsakos P 2020 Machine learning for fluid mechanics *Annual Review of Fluid Mechanics* **52** 477–508
- [17] Raissi M, Perdikaris P and Karniadakis G E 2019 Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations *J. Comput. Phys.* **378** 686–707
- [18] Haghhighat E, Raissi M, Moure A, Gomez H and Juanes R 2021 A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics *Comput. Meth. Appl. Mech. Eng.* **379** 113741
- [19] Buhendwa A B, Adami S and Adams N A 2021 Inferring incompressible two-phase flow fields from the interface motion using physics-informed neural networks *Machine Learning with Applications* **4** 100029
- [20] Cai S, Wang Z, Fuest F, Jeon Y J, Gray C and Karniadakis G E 2021 Flow over an espresso cup: inferring 3-d velocity and pressure fields from tomographic background oriented schlieren via physics-informed neural networks *J. Fluid Mech.* **915** A102
- [21] Yazdani A, Lu L, Raissi M and Karniadakis G E 2020 Systems biology informed deep learning for inferring parameters and hidden dynamics *PLoS Comput. Biol.* **16** 1–19
- [22] Arzani A, Wang J-X and D’Souza R M 2021 Uncovering near-wall blood flow from sparse data with physics-informed neural networks *Phys. Fluids* **33** 071905
- [23] Baydin A, Pearlmutter B A, Radul A A and Siskind J M 2017 Automatic differentiation in machine learning: a survey *J. Mach. Learn. Res.* **18** 5595–637
- [24] Wight C L and Zhao J 2021 Solving allen-cahn and cahn-hilliard equations using the adaptive physics informed neural networks *Communications in Computational Physics* **29** 930–54
- [25] Krishnapriyan A, Gholami A, Zhe S, Kirby R and Mahoney M W 2021 Characterizing possible failure modes in physics-informed neural networks *Advances in Neural Information Processing Systems* vol. 34 ed M Ranzato *et al* (Curran Associates, Inc.) 26548–60
- [26] Rohrhofer F M, Posch S and Geiger B C 2021 On the pareto front of physics-informed neural networks arXiv: 2105.00862
- [27] Jagtap A D, Kawaguchi K and Karniadakis G E 2020 Adaptive activation functions accelerate convergence in deep and physics-informed neural networks *J. Comput. Phys.* **404** 109136
- [28] Jagtap A D, Kawaguchi K and Karniadakis G E 2020 Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **476** 20200334
- [29] McClenney L and Braga-Neto U 2020 Self-adaptive physics-informed neural networks using a soft attention mechanism arXiv: 2009.04544
- [30] Yu J, Lu L, Meng X and Karniadakis G E 2022 Gradient-enhanced physics-informed neural networks for forward and inverse pde problems *Comput. Meth. Appl. Mech. Eng.* **393** 114823
- [31] Wang S, Teng Y and Perdikaris P 2021 Understanding and mitigating gradient flow pathologies in physics-informed neural networks *SIAM J. Sci. Comput.* **43** A3055–81
- [32] Maddu S, Sturm D, Müller C L and Sbalzarini I F 2022 Inverse dirichlet weighting enables reliable training of physics informed neural networks *Machine Learning: Science and Technology* **3** 015026
- [33] Kingma D P and Ba J 2015 Adam: a method for stochastic optimization *3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings (San Diego, CA, USA, May 7–9, 2015)* ed Y Bengio and Y LeCun
- [34] Hornik K, Stinchcombe M and White H 1989 Multilayer feedforward networks are universal approximators *Neural Netw.* **2** 359–66
- [35] Mao Z, Jagtap A D and Karniadakis G E 2020 Physics-informed neural networks for high-speed flows *Comput. Meth. Appl. Mech. Eng.* **360** 112789
- [36] Qian N 1999 On the momentum term in gradient descent learning algorithms *Neural Netw.* **12** 145–51
- [37] Deguchi S, Shibata Y and Asai M 2021 Unknown parameter estimation using physics-informed neural networks with noised observation data *Journal of Japan Society of Civil Engineers, Ser. A2 (Applied Mechanics (AM))* **77** I_35–I_45
- [38] Bergstra J and Bengio Y 2012 Random search for hyper-parameter optimization *Journal of Machine Learning Research* **13** 281–305
- [39] Snoek J, Larochelle H and Adams R P 2012 Practical bayesian optimization of machine learning algorithms *Advances in Neural Information Processing Systems* vol. 25 ed F Pereira *et al* (Curran Associates, Inc.)
- [40] Glorot X and Bengio Y 2010 Understanding the difficulty of training deep feedforward neural networks *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics 9 (Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010)* ed Y W Teh and M Titterington 249–56
- [41] Selvaraju R R, Cogswell M, Das A, Vedantam R, Parikh D and Batra D 2017 Grad-cam: visual explanations from deep networks via gradient-based localization *2017 IEEE International Conference on Computer Vision (ICCV)* 618–26

- [42] Daw A, Bu J, Wang S, Perdikaris P and Karpatne A 2022 Mitigating propagation failures in pinns using evolutionary sampling arXiv: [2207.02338](https://arxiv.org/abs/2207.02338)
- [43] Liu D C and Nocedal J 1989 On the limited memory bfgs method for large scale optimization *Math. Program.* **45** 503–28
- [44] Abadi M *et al* 2015 TensorFlow: large-scale machine learning on heterogeneous systems Software available from tensorflow.org
- [45] Xiang Z, Peng W, Liu X and Yao W 2022 Self-adaptive loss balanced physics-informed neural networks *Neurocomputing* **496** 11–34
- [46] Moelans N, Blanpain B and Wollants P 2008 An introduction to phase-field modeling of microstructure evolution *Calphad* **32** 268–94
- [47] Malvasi A, Tinelli A, Rahimi S, D’Agnese G, Rotoni C, Dell’Edera D, Tsin D A and Cavallotti C 2011 A three-dimensional morphological reconstruction of uterine leiomyoma pseudocapsule vasculature by the allen-cahn mathematical model *Biomed. Pharmacother.* **65** 359–63
- [48] Fuks O and Tchelepi H A 2020 Limitations of physics informed machine learning for nonlinear two-phase transport in porous media *Journal of Machine Learning for Modeling and Computing* **1** 19–37
- [49] Chen Z, Badrinarayanan V, Lee C-Y and Rabinovich A 2018 Gradnorm: gradient normalization for adaptive loss balancing in deep multitask networks *Proceedings of the 35th International Conference on Machine Learning* 80 794–803
- [50] Wang S, Wang H and Perdikaris P 2021 Learning the solution operator of parametric partial differential equations with physics-informed deeponets *Science Advances* **7** eabi8605
- [51] Li Z, Zheng H, Kovachki N, Jin D, Chen H, Liu B, Azizzadenesheli K and Anandkumar A 2021 Physics-informed neural operator for learning partial differential equations arXiv: [2111.03794](https://arxiv.org/abs/2111.03794)
- [52] Weinan E and Yu B 2018 The deep ritz method: a deep learning-based numerical algorithm for solving variational problems *Commun. Math. Stat.* **6** 1–12
- [53] Kharazmi E, Zhang Z and Karniadakis G E M 2021 hp-vpinns: variational physics-informed neural networks with domain decomposition *Comput. Meth. Appl. Mech. Eng.* **374** 113547
- [54] Berrone S, Canuto C and Pintore M 2022 Variational physics informed neural networks: the role of quadratures and test functions *J. Sci. Comput.* **92** 100