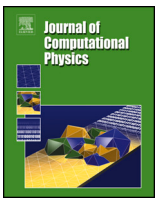
Journal of Computational Physics 501 (2024) 112781

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp



Loss-attentional physics-informed neural networks

Yanjie Song [a], He Wang [b], He Yang [a], Maria Luisa Taccari [a], Xiaohui Chen [a,*]

[a] Geomodelling and AI Centre, School of Civil Engineering, University of Leeds, Leeds, LS2 9JT, United Kingdom
[b] Department of Computer Science, University College London, Gower Street, London, WC1E 6BT, United Kingdom

ARTICLE INFO

Keywords:
Physics-informed neural network
Point error-based weighting method
Loss attention
Novel architecture
Neural network

ABSTRACT

Physics-informed neural networks (PINNs) have emerged as a significant endeavour in recent years to utilize artificial intelligence technology for solving various partial differential equations (PDEs). Nevertheless, the vanilla PINN model structure encounters challenges in accurately approximating solutions at hard-to-fit regions with, for instance, "stiffness" points characterized by fast-paced alterations in timescale. To this end, we introduce a novel model architecture based on PINN, named loss-attentional physics-informed neural networks (LA-PINN), which equips each loss component with an independent loss-attentional network (LAN). Feeding the squared errors (SE) on every training point into LAN as the input, the attentional function is then built by each LAN and provides different weights to diverse point SEs. A point error-based weighting approach that utilizes the adversarial training between multiple networks in the LA-PINN model is proposed to dynamically update weights of SE during every training epoch. Additionally, the weighting mechanism of LA-PINN is analysed and also be validated by performing several numerical experiments. The experimental results indicate that the proposed method displays superior predictive performance compared to the vanilla PINN and holds a swift convergence characteristic. Moreover, it can advance the convergence of those hard-to-fit points by progressively increasing the growth rates of both the weight and the update gradient for point error.

1. Introduction

Recent robust advancements in computational capabilities [1] have enabled physics-informed neural networks (PINNs) to emerge as a promising alternative for solving partial differential equations (PDEs) [2–4]. This development represents an important integration of artificial intelligence technology with scientific computation. Classic data-driven deep learning approaches often fail to achieve ideal predictive accuracy when faced with insufficient training data, and they encounter significant challenges in handling incomplete equation information while solving PDEs [5]. Leveraging the progress in automatic differentiation technology [6], PINN can incorporate physical understanding into model training by embedding the equation information into loss functions [7,8]. Based on this, PINN is well-equipped to effectively solve the PDE problems with the help of a limited amount of experimental data, particularly in scenarios where comprehensive condition information is lacking or when dealing with uncertain forms of the PDEs themselves [9–11]. This, in a way, alleviates the limitations of purely data-driven approaches.

* Corresponding author.
E-mail address: x.chen@leeds.ac.uk (X. Chen).

https://doi.org/10.1016/j.jcp.2024.112781
Received 4 August 2023; Received in revised form 14 January 2024; Accepted 14 January 2024

Available online 19 January 2024

The operation of PINN can be characterized as initially "scattering" points (referred to as estimation points) within the solution space, then computing the loss error concerning these points. As the point errors progressively approach zero, the network parameters suitable for this approaching process are identified, continuously updating the network that acts as a solution approximator [12]. The entire solution space is ultimately estimated using those estimation points, reflecting the integrality and continuity of the solution derived via PINN [13,14]. However, in light of the principle of this operation, it is apparent that the level of loss error minimization at each estimation point directly impacts the predictive accuracy of PINN [15]. Consequently, if some points that are hard to fit, such as stiffness points with fast-paced spatiotemporal transitions [16], appear in the loss component of initial conditions, boundary conditions, or governing equation residuals [17], it becomes difficult for this loss component to approach zero through training [18]. Also, the predicted solution of PINN cannot reach high accuracy.

Numerous studies take steps to minimize loss error by directly applying weights to each component of the loss function [19,20]. For example, Wang et al. [21] utilized the ratio of gradients for different loss components to update the weight, which is multiplied before the independent loss component. Xiang et al. [22] presented an adaptive method to update the weight before multiple loss terms grounded in the Gaussian likelihood estimation. Wang et al. [23] employed Neural Tangent Kernel (NTK) to interpret the evolution of gradients throughout the training of PINN and applied the knowledge from NTK to strategically adjust weights associated with each loss component.

However, the weights required to effectively fit diverse points within a loss component are different. Excessively small or large weights may result in slow convergence or even failure to converge at certain points. Based on this, the point-weighting method becomes the focus in many investigations. Self-adaptive PINN (SA-PINN) [24] is one of the typical models among them. It used a mask function to weight every training point selected from the initial and boundary conditions as well as the residuals, with experimental results indicating that SA-PINN can assign higher weights to "stiffness" regions during training. Li et al. [25] implemented a two-stage weighting process: the weights of points with small errors are first increased to enhance model stability, and those with big errors are then increased to improve the predictive performance. Zhang et al. [26] employed a sub-network that inputs and outputs the weights themselves, to find appropriate weight values for each point error within the residual component. Similar to [26], Anagnostopoulos et al. [27] also concentrated on the weighting of residual points and introduced a residual-based attention strategy to update point weights during training.

Inspired by these point-weighting methods, we also recognize the importance of minimizing point errors for the predictive results of PINN, which is informative for dynamically weighting corresponding point squared error (SE) in the loss function. However, deviating from existing works, we have two observations that have led us to more fine-grained modelling of the mapping from the point errors to the weighted loss terms. First, while learnable weighting has been used in all point-weighting methods, we find that it is intrinsically harder to minimize the errors for some points than others, e.g. those in stiffness regions. That is to say there might be a natural bias in the error at these points which should be taken into consideration during learning. This motivates us to propose a new model that not only learns the weight to scale point error as in previous works [24–27], but also incorporates the learnable bias for each point to account for the general "hardness" of minimizing the error. Considering both the scalar and bias makes our model a generalization of existing point-weighting methods. Second, we observe that the gradient information in backpropagation across the three loss terms (i.e., the residual, initial and boundary conditions) interferes with each other [21,28,29]. Additionally, the hard-to-fit points might appear in all three loss terms not just the residual. Thus, we further apply separate linear networks to learn the weight mapping for alleviating the interference between different terms, as opposed to using one network to learn point weights across all loss terms or only in residual term as the work [26]. Theoretically, we analyse the weighting mechanism of our model and show it has an enhancing effect on the gradient update of "stiffness" points during learning. Empirically, we select the SA-PINN, which is a typical point-weighting method and also focuses on the weighting behaviour of hard-to-fit points, as a key comparative reference to demonstrate the good predictive performance of our model.

The rest of this paper is organized as follows: Section 2 presents an overview of vanilla PINN at first, focusing on the main net that serves as the solution approximator. This main net is also a vital part of the proposed framework. The concept of the Loss-Attentional PINN (LA-PINN) is then introduced, followed by a point error-based weighting method of the framework. Section 3 shows some numerical experiments results to demonstrate the predictive efficacy and discusses the weighting distribution pattern using LA-PINN. In section 4, we conclude this work and illustrate the prospective direction for extending proposed model in the future.

2. Methods

2.1. Physics-informed neural networks

Take into account time-dependent and nonlinear PDEs, which have the form

$$\begin{aligned} &\mathcal{N}_{\boldsymbol{x},t}[u] = f(\boldsymbol{x},t), \quad \boldsymbol{x}\in\Omega,\ t\in[0,T],\\ &u(\boldsymbol{x},0) = h(\boldsymbol{x},0), \quad \boldsymbol{x}\in\Omega,\\ &\mathcal{B}_{\boldsymbol{x},t}[u] = g(\boldsymbol{x},t), \quad \boldsymbol{x}\in\partial\Omega,\ t\in[0,T]. \end{aligned} \tag{1}$$

The first equation is recognized as the governing equation for this initial-boundary value problem. The second and third equations are designated as the initial and boundary conditions, respectively. Here, f, h, and g sequentially correspond to the functions of governing equation, initial and boundary conditions.

We name the fully connected neural network in vanilla PINN as "main net". It is this main net that is responsible for providing the predicted outcome ($\hat{u}$) of PINN model. In order to allow $\hat{u}$ gradually approximate the actual solution, the PINN model employs a strategy of embedding three equations from Eq. (1) into loss functions during training [2]. Namely, the loss function of PINN comprises three components:

$$\begin{aligned}
\mathcal{L}_r(\boldsymbol{\theta}) &= \frac{1}{N_r}\sum_{i=1}^{N_r}\left|\mathcal{N}_{\boldsymbol{x},t}[\hat{u}(\boldsymbol{x}_r^i, t_r^i, \boldsymbol{\theta})] - f(\boldsymbol{x}_r^i, t_r^i)\right|^2 \\
\mathcal{L}_0(\boldsymbol{\theta}) &= \frac{1}{N_0}\sum_{i=1}^{N_0}\left|\hat{u}(\boldsymbol{x}_0^i, 0, \boldsymbol{\theta}) - h(\boldsymbol{x}_0^i, 0)\right|^2 \\
\mathcal{L}_b(\boldsymbol{\theta}) &= \frac{1}{N_b}\sum_{i=1}^{N_b}\left|\mathcal{B}_{\boldsymbol{x},t}[\hat{u}(\boldsymbol{x}_b^i, t_b^i, \boldsymbol{\theta})] - g(\boldsymbol{x}_b^i, t_b^i)\right|^2
\end{aligned} \tag{2}$$

Where $\boldsymbol{\theta}$ is the parameters of main net including weights ($\boldsymbol{W}$) and biases ($\boldsymbol{b}$). $\mathcal{L}_0(\boldsymbol{\theta})$ and $\mathcal{L}_b(\boldsymbol{\theta})$ respectively denote the loss components for initial and boundary conditions. $\mathcal{L}_r(\boldsymbol{\theta})$ represents the residual loss component associated with governing equation. By leveraging the automatic differential tool of deep learning frameworks (such as TensorFlow and PyTorch), the differential form for $\hat{u}$ (i.e., $\mathcal{N}_{\boldsymbol{x},t}$ and $\mathcal{B}_{\boldsymbol{x},t}$) can be executed to facilitate the integration of Eq. (1) into the diverse components of loss function. $\{\boldsymbol{x}_r^i, t_r^i, f^i\}_{i=1}^{N_r}$ signifies N_r collocation points randomly chosen from $\boldsymbol{x}$ and t domains. The superscript i denotes the i-th point. $\{\boldsymbol{x}_0^i, 0, h^i\}_{i=1}^{N_0}$ and $\{\boldsymbol{x}_b^i, t_b^i, g^i\}_{i=1}^{N_b}$ represent N_0 and N_b points selected from initial and boundary conditions, respectively.

It is precisely Eq. (2) that PINN model employs to compute the squared error (SE) of each training point, thereby calculating the mean of SE and deriving the total loss function. The total loss is subsequently utilized in a gradient descent approach [30] to identify the parameters of main net that yields the minimum loss, and this set of parameters is then applied to update the main net. The aforementioned encapsulates the procedure for minimizing the loss value in PINN model. Evidently, finding the minimal loss value mirrors the process of $\hat{u}$ approximating the actual u.

2.2. Loss-attentional physics-informed neural network (LA-PINN)

For the sake of simplification, we write out the SE of each point in the following manner:

$$SE_r(\boldsymbol{\theta}) = \left|\mathcal{N}_{\boldsymbol{x},t}[\hat{u}(\boldsymbol{x}_r^i, t_r^i, \boldsymbol{\theta})] - f(\boldsymbol{x}_r^i, t_r^i)\right|^2 \tag{3}$$

$$SE_0(\boldsymbol{\theta}) = \left|\hat{u}(\boldsymbol{x}_0^i, 0, \boldsymbol{\theta}) - h(\boldsymbol{x}_0^i, 0)\right|^2 \tag{4}$$

$$SE_b(\boldsymbol{\theta}) = \left|\mathcal{B}_{\boldsymbol{x},t}[\hat{u}(\boldsymbol{x}_b^i, t_b^i, \boldsymbol{\theta})] - g(\boldsymbol{x}_b^i, t_b^i)\right|^2 \tag{5}$$

The individual loss components could be written as

$$\mathcal{L}_j(\boldsymbol{\theta}) = \frac{1}{N_j}\sum_{i=1}^{N_j} SE_j^i(\boldsymbol{\theta}), \quad j = r, 0, b \tag{6}$$

From the discussion in the previous section, it can be deduced that reducing the total loss function to nearly zero during the PINN training is indicative of a good predictive performance (i.e., $\hat{u}$ closely approximates the actual u). Additionally, the process of loss function approaching zero can be equivalently viewed as SE converging to zero at each training point, which is expressed as:

$$\mathcal{L}(\boldsymbol{\theta}) \to 0 \iff \begin{cases} \mathcal{L}_r(\boldsymbol{\theta}) \to 0 \\ \mathcal{L}_0(\boldsymbol{\theta}) \to 0 \\ \mathcal{L}_b(\boldsymbol{\theta}) \to 0 \end{cases} \iff \frac{1}{N_j}\sum_{i=1}^{N_j}\begin{bmatrix} SE_j^1(\boldsymbol{\theta}) \to 0, \\ \cdots, \\ SE_j^i(\boldsymbol{\theta}) \to 0, \\ \cdots, \\ SE_j^{N_j}(\boldsymbol{\theta}) \to 0 \end{bmatrix}, \quad j = r, 0, b \tag{7}$$

Namely, without considering model overfitting, minimizing the SE towards zero at each training point ensures good predictive results for PINN.

A prevalent method for driving the loss function towards zero in many studies [31–33,21] is to iteratively assign individual weights to different loss components throughout every training epoch, as follows

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{j=r,0,b}\lambda_j\mathcal{L}_j(\boldsymbol{\theta}) = \sum_{j=r,0,b}\frac{\lambda_j}{N_j}\sum_{i=1}^{N_j} SE_j^i(\boldsymbol{\theta}) \tag{8}$$

The essence of this method lies in the updating process of network parameters using gradient descent, and it could be illustrated as

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \eta \cdot \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$$

$$= \theta^k - \sum_{j=r,0,b} \eta\lambda_j \cdot \nabla_\theta \mathcal{L}_j(\theta)$$

$$= \theta^k - \sum_{j=r,0,b} \eta\lambda_j \cdot \frac{1}{N_j} \sum_{i=1}^{N_j} \nabla_\theta SE_j^i(\theta) \tag{9}$$

This implies that an identical weight is assigned to the gradient of SE at each point in an independent loss component. As the parameter updating step-length is determined by the gradient when the learning rate (η) remains constant, this method is primarily to assign the same weights to the parameter updating step-length at different points. However, significant differences exist between the gradient or value of point SE in the stiffness region and the non-stiffness area during the fitting process. The required weighting values for the updating step-length that can aid convergence are different. Therefore, the assignment of the same weights might not significantly mitigate the challenge involved in minimizing point SEs in hard-to-fit regions. From this observation, the notion of providing diverse weights for each point SE spontaneously arises, that is

$$\mathcal{L}_j(\theta) = \frac{1}{N_j} \sum_{i=1}^{N_j} \lambda_j^i SE_j^i(\theta) = \text{MEAN}(\begin{bmatrix} \lambda_j^1 SE_j^1(\theta), \\ \cdots, \\ \lambda_j^i SE_j^i(\theta), \\ \cdots, \\ \lambda_j^{N_j} SE_j^{N_j}(\theta) \end{bmatrix}), \quad j = r, 0, b \tag{10}$$

MEAN(.) here refers to calculating the average of errors for N_j points within the error vector.

To this end, we present a novel PINN model architecture that pays attention to the $SE_j^i(\theta)$ of every training point named loss-attentional physics-informed neural network (LA-PINN). With the complexity of assigning diverse weights to each point SE in mind, we identify this as the process of finding the "attentional function" of loss error and define a loss function where the attentional function has already been established as $\mathcal{L}^\star$. The proposed LA-PINN uses the following loss function:

$$\mathcal{L}^\star(\theta, \xi_j) = \mathcal{L}_r^\star(\theta, \xi_r) + \mathcal{L}_0^\star(\theta, \xi_0) + \mathcal{L}_b^\star(\theta, \xi_b) \tag{11}$$

$$= \text{LAN}_\text{r}\left(SE_r^i(\theta), \xi_r\right) + \text{LAN}_0\left(SE_0^i(\theta), \xi_0\right) + \text{LAN}_\text{b}\left(SE_b^i(\theta), \xi_b\right) \tag{12}$$

Three neural networks known collectively as loss-attentional nets (LANs) with the respective parameters (ξ_r, ξ_0, and ξ_b) are employed for the construction of attentional functions at different loss components, denoted as LAN_r, LAN_0, and LAN_b, respectively. Notice that aiming to optimize the proficiency of LANs in "learning" the weight distribution for various point errors, we strategically utilize three LANs. Every LAN is paired with a specific loss component to mitigate complications for appropriate weights assignment arising from overlapping training points (certain initial-boundary condition points might concurrently appear in different loss components for calculating errors [2]) or pronounced deviations in error variation patterns between different components. In practice, we observed that this one-to-one pairing, or even splitting loss components to pair with more LANs, indeed outperforms merging components to align with fewer LANs.

$SE_j^i(\theta)$ is input into three LANs as illustrated in Eq. (12), and its specific form is:

$$\mathcal{L}_r^\star(\theta, \xi_r) = \text{MEAN}\left(\boldsymbol{W}_r^{\star(l+1)}\left(\left(\boldsymbol{W}_r^{\star(1)}\left[\cdots, SE_r^i(\theta), \cdots\right]^T + \boldsymbol{b}_r^{\star(1)}\right)\cdots\right) + \boldsymbol{b}_r^{\star(l+1)}\right) \tag{13}$$

$$\mathcal{L}_0^\star(\theta, \xi_0) = \text{MEAN}\left(\boldsymbol{W}_0^{\star(l+1)}\left(\left(\boldsymbol{W}_0^{\star(1)}\left[\cdots, SE_0^i(\theta), \cdots\right]^T + \boldsymbol{b}_0^{\star(1)}\right)\cdots\right) + \boldsymbol{b}_0^{\star(l+1)}\right) \tag{14}$$

$$\mathcal{L}_b^\star(\theta, \xi_b) = \text{MEAN}\left(\boldsymbol{W}_b^{\star(l+1)}\left(\left(\boldsymbol{W}_b^{\star(1)}\left[\cdots, SE_b^i(\theta), \cdots\right]^T + \boldsymbol{b}_b^{\star(1)}\right)\cdots\right) + \boldsymbol{b}_b^{\star(l+1)}\right) \tag{15}$$

where $(\boldsymbol{W}_j^\star, \boldsymbol{b}_j^\star) \subseteq \xi_j$; $j = r, 0, b$ are the parameters of LAN, and $(\boldsymbol{W}_j^{\star(l+1)}, \boldsymbol{b}_j^{\star(l+1)})$ correspond to the parameters at output layer. It can be observed that the learnable bias $\boldsymbol{b}_j^\star$ is considered during the weighting process. A column vector, constituted by each point SE, serves as the input for the LAN. The feature dimension number of this input vector is one, while its length reflects the quantity of training points. Rather than implementing an activation function for nonlinear transformations, LANs only use parameters ξ to find an appropriate direction in high-dimensional space and follow this direction to weight the SE with attention. Under the influence of this "attentional function", each SE undergoes a purely linear transformation. Hence, defining the squared error that has passed through the LAN as $SE^\star$, there exists the weight $\lambda(\xi)$ that satisfies the following equation in a linear pattern.

$$SE_j^{\star i}(\theta, \xi_j) = \boldsymbol{W}_j^{\star(l+1)}\left(\left(\boldsymbol{W}_j^{\star(1)} SE_j^i(\theta) + b_j^{\star(1)}\right)\cdots\right) + b_j^{\star(l+1)} \tag{16}$$

$$= \lambda_j^i(\xi_j) SE_j^i(\theta) \tag{17}$$

We refer to $\lambda_j^i(\xi_j)$ as the point error-based weight (including the scale and bias) in LA-PINN model and use it to represent the purely linear relationship composed of parameters $\boldsymbol{W}^\star$ and $\boldsymbol{b}^\star$ in LANs without applying activation functions.

Then, the parameters updating gradient of LAN_j about the corresponding loss component is

$$\nabla_{\xi_j} \mathcal{L}_j^\star(\theta, \xi_j) = \frac{1}{2} \sum_{i=1}^{N_j} \nabla_{\xi_j} SE_j^{\star i}(\theta, \xi_j) = \frac{1}{2} \sum_{i=1}^{N_j} \nabla_{\xi_j} \lambda_j^i(\xi_j) SE_j^i(\theta) \tag{18}$$

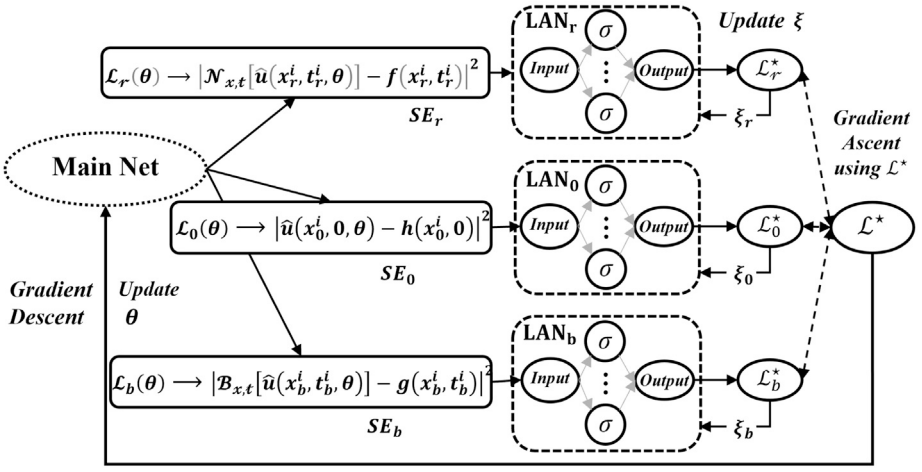


Fig. 1. The architecture schematic of LA-PINN model.

$$= \frac{1}{2}\begin{bmatrix} \nabla_{\xi_j}\lambda_j^1(\xi_j), \\ \cdots, \\ \nabla_{\xi_j}\lambda_j^i(\xi_j), \\ \cdots, \\ \nabla_{\xi_j}\lambda_j^{N_j}(\xi_j) \end{bmatrix}^T \begin{bmatrix} SE_j^1(\theta), \\ \cdots, \\ SE_j^i(\theta), \\ \cdots, \\ SE_j^{N_j}(\theta) \end{bmatrix} = \frac{1}{2}\nabla_{\xi_j}\lambda_j^T(\xi_j)\boldsymbol{SE}_j(\theta) \tag{19}$$

Here, since the mean coefficient $1/N_j$ does not impact the optimization process, it is simplified to $1/2$ for ease of representation. The vectorized expressions of the SE and its associated weights are presented at the end of the formula above. It can be stated that the LANs impose the attentional function on the $\boldsymbol{SE}_j(\theta)$ by manipulating the distribution of the weight vector $\lambda_j^T(\xi_j)$.

Note that the LANs are parameterized by ξ, whereas the parameter of main net is θ. Therefore, there is a distinct form of the gradient for main net with respect to the loss function

$$\nabla_\theta \mathcal{L}^\star(\theta, \xi) = \sum_{j=r,0,b} \frac{1}{2} \sum_{i=1}^{N_j} \lambda_j^i(\xi) \nabla_\theta SE_j^i(\theta) \tag{20}$$

$$= \frac{1}{2}\begin{bmatrix} \cdots, \\ \lambda_r^{N_r}(\xi), \\ \cdots, \\ \lambda_0^{N_0}(\xi), \\ \cdots, \\ \lambda_b^{N_b}(\xi), \end{bmatrix}^T \begin{bmatrix} \cdots, \\ \nabla_\theta SE_r^{N_r}(\theta), \\ \cdots, \\ \nabla_\theta SE_0^{N_0}(\theta), \\ \cdots, \\ \nabla_\theta SE_b^{N_b}(\theta), \end{bmatrix} = \frac{1}{2}\nabla_\theta \lambda^T(\xi)\boldsymbol{SE}(\theta) \tag{21}$$

where $\boldsymbol{SE}(\theta)$ denotes the point error vector composed of SEs from all training points. $\lambda^T(\xi)$ could be known as a point error-based weight vector; it conveys the weighted values of the updating step-length for each point SE. The vector length, similar to $\boldsymbol{SE}(\theta)$, is equal to $N_r + N_0 + N_b$.

2.3. The point error-based dynamic weighting method in LA-PINN

The architecture schematic of LA-PINN can be found in Fig. 1; notice that the main net is simplified in the depiction and still undertakes the role of providing predictive result $\hat{u}$. Contrary to vanilla PINN, the LA-PINN pays more attention to the evolution of loss error. It allocates a LAN to each loss component to facilitate the minimization of SE at every training point.

The specific workflow of LA-PINN is as follows. We initiate a new architecture segment after acquiring SE from main net. This segment, which is different from vanilla PINN, feeds point errors SE_r, SE_0, and SE_b as inputs into their respective LAN, and then the attentional function of SE is generated. The overall attentional loss value $\mathcal{L}^\star$ is obtained through the calculations of Formulas (11)-(15).

During the training phase, the LANs and main net simultaneously undertake adversarial training pertaining to $\mathcal{L}^\star$ below

$$\mathcal{L}^\star(\theta, \xi) = \sum_{j=r,0,b} \mathcal{L}_j^\star(\theta, \xi_j) \tag{22}$$

$$= \sum_{j=r,0,b} \frac{1}{2} \sum_{i=1}^{N_j} \lambda_j^i(\xi_j) SE_j^i(\theta) \tag{23}$$

$$= \frac{1}{2} \lambda^T(\xi) \boldsymbol{SE}(\theta) \tag{24}$$

Their adversarial operation is reflected in that the main net employing gradient descent method about $\mathcal{L}^\star$ to update its parameters θ as an approximator of u, while the LANs update its parameters ξ as a point error-based weight distributor via gradient ascent about $\mathcal{L}^\star$. The operation has the following specific expression

$$\text{Main Net:} \quad \theta^{k+1} = \theta^k - \eta \nabla_\theta \mathcal{L}^\star = \theta^k - \frac{1}{2}\eta \cdot \nabla_\theta \lambda^T(\xi^k) \boldsymbol{SE}(\theta^k) \tag{25}$$

$$\text{LAN}_\text{r}: \quad \xi_r^{k+1} = \xi_r^k + \rho_r \nabla_{\xi_r} \mathcal{L}_r^\star = \xi_r^k + \frac{1}{2}\rho_r \cdot \nabla_{\xi_r} \lambda_r^T(\xi_r^k) \boldsymbol{SE}_r(\theta^k) \tag{26}$$

$$\text{LAN}_0: \quad \xi_0^{k+1} = \xi_0^k + \rho_0 \nabla_{\xi_0} \mathcal{L}_0^\star = \xi_0^k + \frac{1}{2}\rho_0 \cdot \nabla_{\xi_0} \lambda_0^T(\xi_0^k) \boldsymbol{SE}_0(\theta^k) \tag{27}$$

$$\text{LAN}_\text{b}: \quad \xi_b^{k+1} = \xi_b^k + \rho_b \nabla_{\xi_b} \mathcal{L}_b^\star = \xi_b^k + \frac{1}{2}\rho_b \cdot \nabla_{\xi_b} \lambda_b^T(\xi_b^k) \boldsymbol{SE}_b(\theta^k) \tag{28}$$

where $\rho_j, j = r, 0, b$ is the learning rate of LAN. Despite LAN_j relies on $\mathcal{L}^\star$ for gradient computation, it fundamentally updates parameters ξ_j, $j = r, 0, b$ based on the SE_j inherent to the corresponding loss component $\mathcal{L}_j^\star$ within $\mathcal{L}^\star$; the point errors from other components remain unrelated and do not contribute to its gradient computations or parameters update process. This accounts for the arrows pointing from right to left depicted in the right segment of Fig. 1 and the reason for using the gradient form in Formulas (26)-(28).

From the perspective of Formula (24), that is, $\mathcal{L}^\star = \frac{1}{2}\lambda^T(\xi)\boldsymbol{SE}(\theta)$, the point error-based weights distributor LAN adjusts the distribution of the vector $\lambda^T(\xi)$ in search of a direction that maximizes $\mathcal{L}^\star$. Conversely, the main net, serving as the u approximator, modifies θ so that the vector $\boldsymbol{SE}(\theta)$ continuously moves towards minimizing $\mathcal{L}^\star$. Given this scenario, LANs should employ a similar network size in complexity (the configuration of the hidden layers should remain consistent while the input and output layers may not be identical) to effectively compete in this "tug-of-war" with main net and to handle the complex process of identifying optimal weighting directions. Our extensive trial-and-error experimentations in practice also demonstrate that the size of LANs should closely resemble that of main net for the best performance of LA-PINN.

Moving forward, we further explore the behaviour of weights and errors at specific stiffness points in the vector $\lambda^T(\xi)$ and $\boldsymbol{SE}(\theta)$ to investigate the weighting mechanism of LA-PINN. Accordingly, the focus here is on point-based weights or errors, instead of on the entire vector. Assume that the model training has not yet reached convergence, and there exists a stiffness point p with point error $SE^p(\theta)_n$ at the nth epoch and a non-stiffness point $p+1$ with point error $SE^{p+1}(\theta)_n$. We discuss the alterations occurring at these two points over m iterations, i.e., from the nth to the $(n+m)$th epoch.

The hard-to-fit characteristics of stiffness points indicate that they tend to present significant point errors and comparably smaller update gradients of parameters forcing toward a zero error:

$$SE^p(\theta)_n \gg SE^{p+1}(\theta)_n \tag{29}$$

$$\|\nabla_\theta SE^p(\theta)_n\| \ll \|\nabla_\theta SE^{p+1}(\theta)_n\| \tag{30}$$

Here, we employ Euclidean norm $\|.\|$ to describe the magnitude of parameter update gradient. Given the relatively small update gradients, the value of $SE^p(\theta)_n$ would struggle to show notable reduction even after m epochs of literation. This is represented as:

$$SE^p(\theta)_{n+1} \gg SE^{p+1}(\theta)_{n+1} \tag{31}$$

$$\vdots$$

$$SE^p(\theta)_{n+m} \gg SE^{p+1}(\theta)_{n+m} \tag{32}$$

It is clear that $SE^p(\theta)$ consistently surpasses $SE^{p+1}(\theta)$ throughout the continuous m epochs. Shifting our focus on the weight update step-length $\left\| \frac{1}{2} \nabla_\xi \lambda^i(\xi) SE^i(\theta) \right\|$ as indicated by Formula (26) without considering the learning rate ρ, $SE^i(\theta)$ here can be regarded as a term using for weighting the update gradient $\nabla_\xi \lambda^i(\xi)$, namely, weighting the growth rate of $\lambda^i(\xi)$ in the direction of the maximum $\mathcal{L}^\star$. As a result of the significant numerical difference between $SE^p(\theta)$ and $SE^{p+1}(\theta)$, when they weight the growth rate of λ across m epochs, we have

$$\frac{\lambda^p(\xi)_{n+m} - \lambda^p(\xi)_n}{\lambda^p(\xi)_n} > \frac{\lambda^{p+1}(\xi)_{n+m} - \lambda^{p+1}(\xi)_n}{\lambda^{p+1}(\xi)_n} \tag{33}$$

$$\Longrightarrow \frac{\lambda^p(\xi)_{n+m}}{\lambda^p(\xi)_n} > \frac{\lambda^{p+1}(\xi)_{n+m}}{\lambda^{p+1}(\xi)_n} \tag{34}$$

It can be stated that when dealing with stiffness points, the LA-PINN accordingly enhances the growth rate of the weight imposed on such points. In this context, it is naturally considered whether LA-PINN also has an enhancing impact on the growth rate of the

update gradients for SE at stiffness points. Considering the parameter update step-length about SE, i.e., $\left\|\frac{1}{2}\lambda^i(\xi)\nabla_\theta SE^i(\theta)\right\|$ without regarding the learning rate η, the growth rate of update step-length about $SE^i(\theta)$ at stiffness point is

$$\frac{\left\|\frac{1}{2}\lambda^p(\xi)_{n+m}\nabla_\theta SE^p(\theta)_{n+m}\right\|}{\left\|\frac{1}{2}\lambda^p(\xi)_n\nabla_\theta SE^p(\theta)_n\right\|} = \frac{\left\|\nabla_\theta SE^p(\theta)_{n+m}\right\|}{\left\|\nabla_\theta SE^p(\theta)_n\right\|}\cdot\frac{\lambda^p(\xi)_{n+m}}{\lambda^p(\xi)_n} \tag{35}$$

The growth rate of update step-length about $SE^i(\theta)$ at non-stiffness point is

$$\frac{\left\|\frac{1}{2}\lambda^{p+1}(\xi)_{n+m}\nabla_\theta SE^{p+1}(\theta)_{n+m}\right\|}{\left\|\frac{1}{2}\lambda^{p+1}(\xi)_n\nabla_\theta SE^{p+1}(\theta)_n\right\|} = \frac{\left\|\nabla_\theta SE^{p+1}(\theta)_{n+m}\right\|}{\left\|\nabla_\theta SE^{p+1}(\theta)_n\right\|}\cdot\frac{\lambda^{p+1}(\xi)_{n+m}}{\lambda^{p+1}(\xi)_n} \tag{36}$$

If we view $\frac{\lambda^i(\xi)_{n+m}}{\lambda^i(\xi)_n}$ as a weighting term, in the light of Formulas (34)-(36), it can be derived that LA-PINN applies a greater weighting to the growth rate of the gradient about SE for stiffness points during m training epochs.

Based on the mentioned above, when encountering stiffness points, the LA-PINN leverages the point error-based weights distributor LANs to increase the growth rate of weight λ^i, and by using the λ^i growth rate as a scale, the growth rate of the update gradient about point error is weighted. This is the weighting mechanism of LA-PINN, progressively assigning increasing weights on stiffness points and gradually enhancing the update gradient (i.e., step-length) for SE to facilitate the error reduction towards zero.

3. Numerical examples

In this section, we apply various PDE examples to demonstrate the performance of LA-PINN in predicting solutions with the hard-to-fit regions, which is typically marked by rapidly changing in spatial scale (i.e., sharp size) or fast-paced alteration in temporal scale (i.e., stiffness area) [34]. Accordingly, the Navier-Stokes equation, 2-D Poisson equation, and Helmholtz equation are utilized to test the convergence performance of LA-PINN at regions with sharp size, while the 1-D Burger equation, 2-D Burger equation, Allen-Cahn equation, and Diffusion-reaction equation are employed to examine the fitting performance at areas with stiffness points.

All codes of LA-PINN model for solving these PDEs are programmed under TensorFlow 2.0 framework and are operated in a laptop that configures NVIDIA RTX 3070 Ti for GPU and has i7-12700H for CPU. We set initial parameters of main net through Xavier initialization method [35] and take the hyperbolic tangent (tanh) [36] as activation function. All training processes utilize the Adam algorithm [37] for global optimization, followed by the L-BFGS algorithm [38] for fine-tuning (except for Navier-Stokes equation that only uses Adam algorithm), and the LANs update weights only in Adam optimizing procedure. Both optimization algorithms optimize concerning the total loss function ($\mathcal{L}^\star$) that has already established the attentional function for each point SE.

In order to discuss the predictive performance of LA-PINN, we introduce the concept of L_2 error:

$$L_2\,\text{error} = \frac{\sqrt{\sum_{i=1}^{N}|\hat{u}(x_i,t_i)-u(x_i,t_i)|^2}}{\sqrt{\sum_{i=1}^{N}|u(x_i,t_i)|^2}} \tag{37}$$

where N denotes the number of high-fidelity data points selected in the solution domain of the PDE example. The assessment of the LA-PINN performance is implemented by calculating the Euclidean distance (namely, L_2 error) at these points between the predicted and exact solution. It should be pointed out that all L_2 errors shown in examples are obtained by restarting and running the code 10 times, then calculating the mean of these outcomes.

In practice, it is found that there is a significant disparity in the weight values of each training point. For a more intuitive display of the distribution pattern, the weight values in the following experimental examples are visualized as a logarithmic form:

$$\lambda^T(\xi)_{log} = \ln(\lambda^T(\xi) - \min(\lambda^T(\xi)) + 1) \tag{38}$$

Here, we add one to each weight to prevent them from becoming zero after taking the logarithm.

3.1. Navier-Stokes equation

In the first example of numerical experiments, we evaluate the performance of our proposed approach by solving the lid-driven cavity flow problem governed by the Navier-Stokes (NS) equations, a classic benchmark in the field of computational fluid dynamics. We initially present the results for the NS equation. Subsequently, we use the NS equation as a case study to explore the impact of various parameter initialization methods for LANs on the computational results.

3.1.1. The experimental results for Navier-Stokes equation

The NS equation manifests itself in the subsequent format:

$$u_x + v_y = 0, \tag{39}$$

$$uu_x + vu_y + \frac{1}{\rho}\cdot p_x - \nu(u_{xx}+u_{yy}) = 0, \tag{40}$$

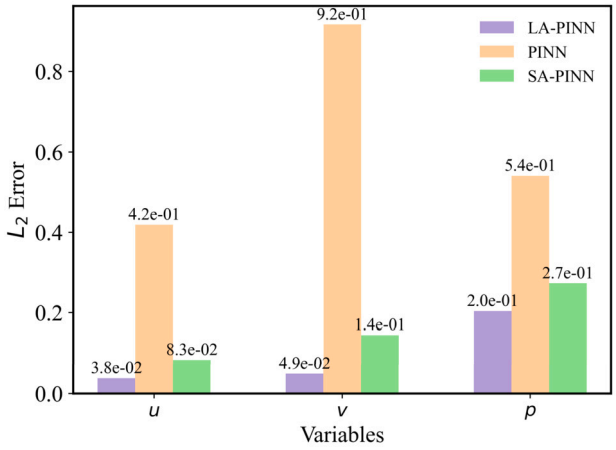


Fig. 2. Comparison of L_2 errors across different models at 100k training epochs.

$$uv_x + vv_y + \frac{1}{\rho} \cdot p_y - \nu(v_{xx} + v_{yy}) = 0. \tag{41}$$

Here, $\rho = 1$ represents the density, and $\nu = 0.01$ denotes the viscosity, with x, y ranging within $[0, 1]$. Regarding the boundary conditions, we set $u = 1, v = 0$ at the top boundary, and $u = 0, v = 0$ on all other boundaries. The LA-PINN model comprises a fully connected architecture, including an input layer with two neurons (to input x, y), four hidden layers each containing 110 neurons, and an output layer with three neurons, corresponding to the outputs $\hat{u}, \hat{v}, \hat{p}$. The layer structure of the LANs is similar to the main net, with variations only in the input and output layers as previously described. Four LANs are configured in total. Specifically, two of these LANs are dedicated to the loss component of boundary conditions, each corresponding to u and v (i.e., $\mathrm{LAN}_{\mathrm{bu}}$ and $\mathrm{LAN}_{\mathrm{bv}}$). For the residuals of governing equations, $\mathrm{LAN}_{\mathrm{rc}}$ is designated to control the point SE arising from the continuity equation (Eq. (39)), while $\mathrm{LAN}_{\mathrm{rm}}$ is assigned to weight the residuals in the two momentum equations (Eq. (40) and (41)). The loss function for NS equation can be expressed as:

$$\mathcal{L}^\star = \mathcal{L}^\star_{bu} + \mathcal{L}^\star_{bv} + \mathcal{L}^\star_{rc} + \mathcal{L}^\star_{rm} \tag{42}$$

where four loss components are obtained by allowing point errors to pass through the four aforementioned LANs. In this NS equation case, we select 100 training points individually from each of four boundaries and choose 10000 training points from the residuals of governing equations. The initial learning rate of Adam optimizer is set to 1e-4 for main net and 2e-4 for LANs. We exclusively adopt the Adam optimizer to optimize the LA-PINN model [32], as the combined "Adam + L-BFGS" strategy has not shown superior performance in practice. It should be noted that using only the Adam optimizer may lead to fluctuating predictions in solving this NS problem [39]. The adaptive learning rate mechanism of Adam algorithm might struggle to rapidly adapt to the complex loss surface associated with NS equation during training. It typically requires several epochs to adequately adjust and find the local minimum loss. To address this, we monitor three predictive L_2 errors for u, v, and p at every 10-epochs interval. The model parameters corresponding to the minimum sum of these three L_2 errors are then saved as the best model to effectively mitigate the issue of fluctuating predictions.

After running the code 10 times to calculate the average, the results at 100k training epochs, as shown in Fig. 2, demonstrate that LA-PINN significantly outperforms PINN and SA-PINN in terms of the predictive L_2 error for u, v, p. Vanilla PINN fails to accurately predict NS equation even after 100k epochs. Fig. 3 shows a good predicted result of LA-PINN in solving NS equations compared with the exact solutions. In addition, the L_2 errors of LA-PINN here are smaller than those reported in the investigation by Wang et al. [21].

We focus on the solution of LA-PINN in velocity field. Predictions for v largely agree with the exact solution at $y = 0.25$ and $y = 0.75$, whereas a comparatively notable discrepancy for u is observed within the x range of [0.2, 0.8], as evidenced in Fig. 4a and 4b. Accordingly, LA-PINN "intelligently" identifies this situation and progressively increases weight values during training process to facilitate convergence of training points in this area, as clearly visualized in Fig. 5. The gradually brightening colours of the points within [0.2, 0.8] from Fig. 5a to 5d indicate the emphasis LANs place on weighting this area. This demonstrates the focus of LA-PINN on fitting hard-to-fit regions with sharp variations.

3.1.2. The impact of different parameter initialization methods for LANs

In this subsection, we assess the impact of various LANs' parameter initialization methods on the training outcomes. For the comparative analysis, we employ six methods that are feasible for linear networks. These methods, which include Gaussian, LeCun Normal, He Normal, Small Random Numbers (SRN), Xavier, and Constant Initialization, are each applied across 10 runs of the code to generate average predictions. Considering the fluctuating predictions over training epochs when solving the NS equations, we select two specific epochs for observation: an early, non-converged stage at 20k epochs, and a later stage where training has reached convergence at 100k epochs. As observed in Fig. 6, the LA-PINN models initialized using Gaussian and LeCun methods

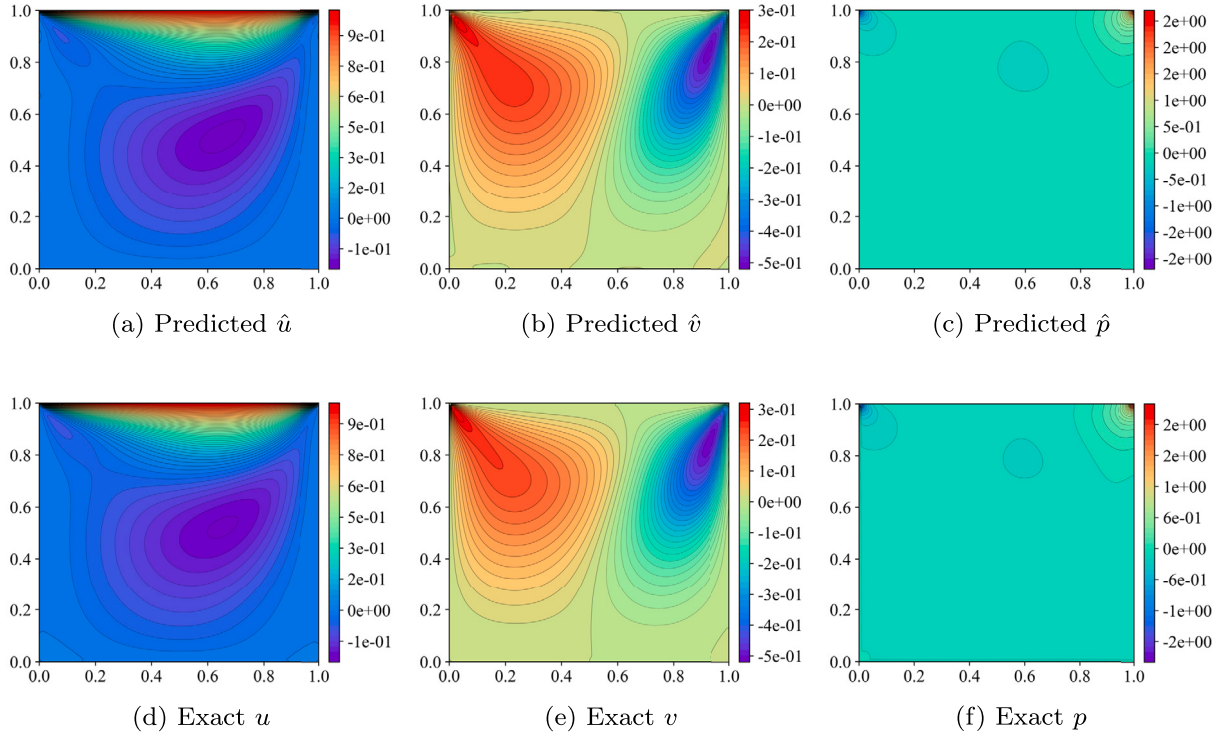


Fig. 3. The comparison between the predicted and exact solution of Navier-Stokes equation. Specifically, L_2 error for u is 3.84e-02, for v is 4.92e-02, and for p is 2.04e-01.

have comparatively better predictive performance at 20k epochs. However, at 100k epochs, they exhibit a slight reduction in L_2 errors, indicating challenges in achieving model convergence with these two initialization approaches. In contrast, the Constant method (where the constant is set to 1), while not the best performer at 20k epochs, shows a significant error decrease at 100k epochs, eventually becoming the most effective among the six methods. This error reduction between 20k and 100k epochs under the Constant method highlights its positive impact on model convergence.

The Gauss and LeCun initialization methods introduce greater randomness into the initial weight distribution, which may help LA-PINN model rapidly reduce errors during the early training stage. However, this initial randomness might lead to issues like gradient vanishing or exploding in the later stages, resulting in the instability of model performance and difficulty of model convergence. Conversely, the Constant initialization approach assigns the same weight to each point error in LA-PINN at the beginning and honours the original error distribution to ensure the stability of initial training. It then modifies the weight distribution via changing LAN parameters throughout the later training process, thereby aiding the error minimization and steadily guiding the model towards convergence.

We also experimented with different initial constants for different LANs. For example, since the boundary condition loss of NS equation is larger than the residual during training, we set the initial parameters of $\mathrm{LAN_b}$ to 2 and $\mathrm{LAN_r}$ to 1. However, the prediction results were much worse compared to the case where both were initialized at 1, owing to such initial bias can easily induce instability of the model training.

In subsequent experiments of other PDEs, we found that the constant 1 initialization for all LANs consistently outperformed other initialization methods. Therefore, the results presented in the following experiments all employ this initialization strategy.

3.2. 2-D Poisson equation

We consider a 2-D Poisson equation that is widely applied in the electromagnetism field.

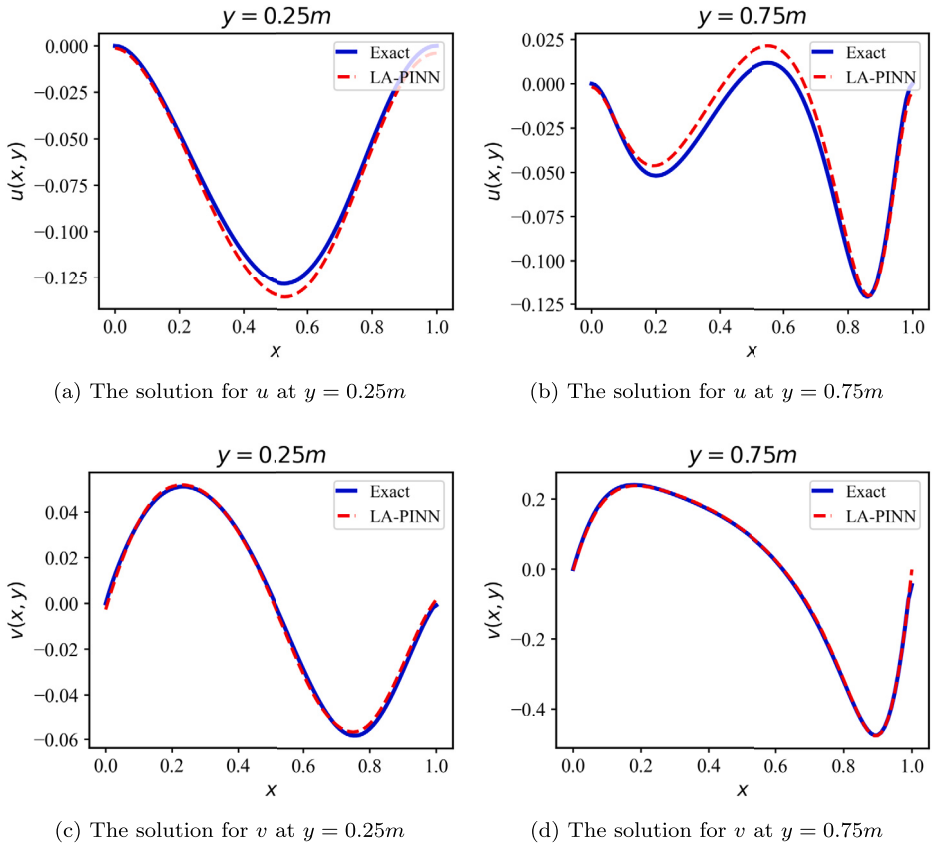
$$-\Delta u(x,y) = 2\pi^2 \sin(\pi x)\sin(\pi y), \quad x,y \in \Omega \tag{43}$$

$$u(x,y) = \sin(\pi x)\sin(\pi y), \quad x,y \in \partial\Omega \tag{44}$$

The analytical solution of this equation is:

$$u(x,y) = \sin(\pi x)\sin(\pi y) \tag{45}$$

The experimental results for the 2-D Poisson equation are all generated by a main net with [2, 50, 50, 50, 50, 1] fully-connected structure. The main net has 2 input neurons corresponding to x and y, 4 hidden layers of 50 neurons each, and an output layer with



(a) The solution for u at $y = 0.25m$ (b) The solution for u at $y = 0.75m$

(c) The solution for v at $y = 0.25m$ (d) The solution for v at $y = 0.75m$

Fig. 4. The solution of LA-PINN for NS equation when $y = 0.25m$ and $y = 0.75m$.

one outcome corresponding to the predicted solution $\hat{u}(x, y)$. As for the configuration of LANs, two LANs are respectively used to control $\mathcal{L}_r$ and $\mathcal{L}_b$ in this no initial condition problem. We set the layer sizes of LANs to [1, 50, 50, 50, 50, 1], closely matching the main net. 10k iterations of Adam are done for global optimization, followed by 10k L-BFGS operated for fine-tuning. Additionally, the number of points chosen from the boundary condition and governing equation residual is set to $N_b = 200$ and $N_r = 8000$, respectively. On account of there being four boundaries in the x-y domain, only 50 points are allocated to each boundary. The Adam optimizer for main net and LANs has the same initial learning rate of 1e-3.

As shown in Fig. 7, the LA-PINN model has good accuracy in solving Poisson equation. We achieve 5.83e-5 for L_2 error, an order of magnitude smaller than 2.21e-4 of SA-PINN proposed in [24]. Remarkably, for the sake of comparing the performance of LA-PINN with other models, all settings are configured to be the same.

The A-10k and L-10k in Fig. 8 denote using Adam and L-BFGS optimizers to individually run 10k iterations. It could be observed that the LA-PINN model can reach an evidently higher accuracy than PINN and SA-PINN that have trained 10k epochs, even only training 1k iterations of Adam and L-BFGS. The shaded area around the curve in Fig. 8 represents the variation of standard deviation over training epochs, which calculated after running the code ten times. LA-PINN has relatively small standard deviations at various epochs, indicating its stable characteristics during training.

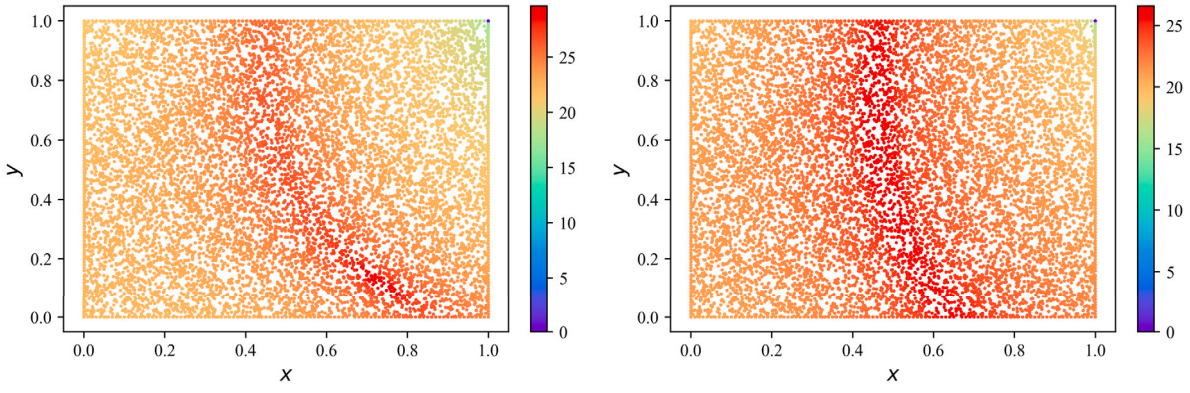
From Figs. 9 and 10, the point colour on boundaries is all bright while the point colour in the inner area has a partly blue colour; it can be described that points are assigned higher weight values at the boundary than points at residual when boundary points have lower mean loss gradients. Observing from Fig. 10a to Fig. 10c, the area of blue points gradually diminishes across 85 epochs. This shows that as the mean gradient difference between the boundary and residual lessens, the discrepancy in weight distributions also correspondingly decreases. Notice that the LA-PINN likewise allocates high weights on the regions where u values change rapidly in spatial scale, as depicted in Figs. 7 and 10a. This implies that the LA-PINN can give the spatially fast-alteration region relatively higher weights to promote its convergence when this region has low update gradients at the initial stage of training.

3.3. Burger equation

We discuss the burger equation in this part, a benchmark fluid mechanics equation. It has the form as follows:

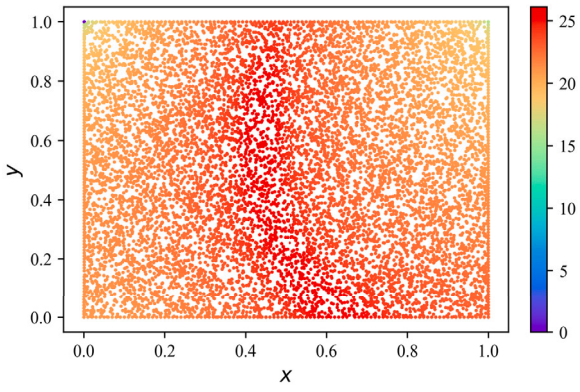
$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1, 1], t \in [0, 1], \tag{46}$$

$$u(0, x) = -\sin(\pi x), \tag{47}$$

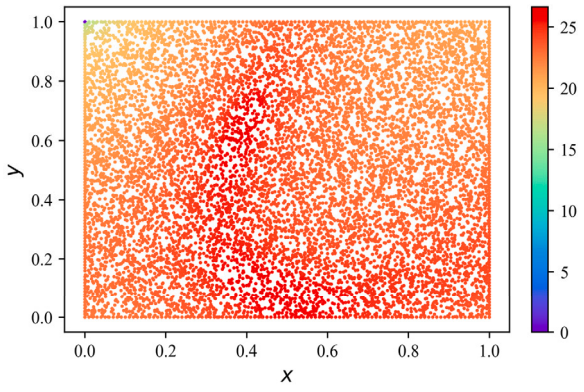


(a) Weights distribution at 200th epoch

(b) Weights distribution at 300th epoch



(c) Weights distribution at 400th epoch



(d) Weights distribution at 500th epoch

Fig. 5. The evolution of weights distribution for NS equation. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

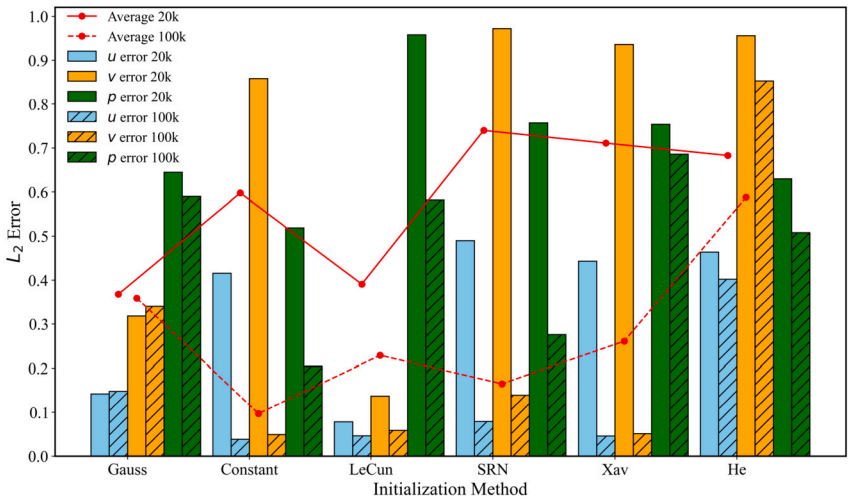


Fig. 6. Comparisons of L_2 errors across different initialization methods at 20k and 100k training epochs. Here, "Average 20k" and "Average 100k" respectively represent the average L_2 error of u, v, and p at 20k and 100k epochs, serving as a comprehensive evaluation for the prediction results.

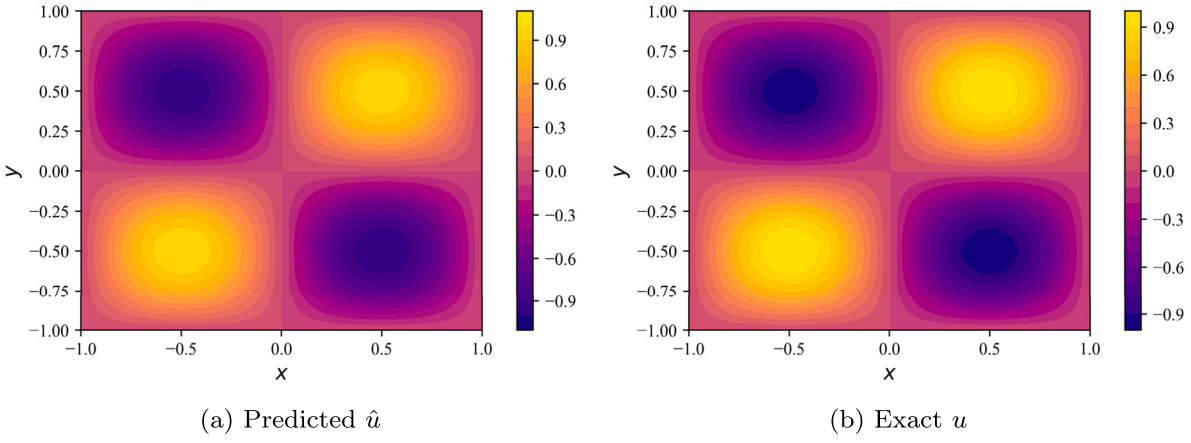


Fig. 7. The comparison between the predicted and exact solution of Poisson equation.

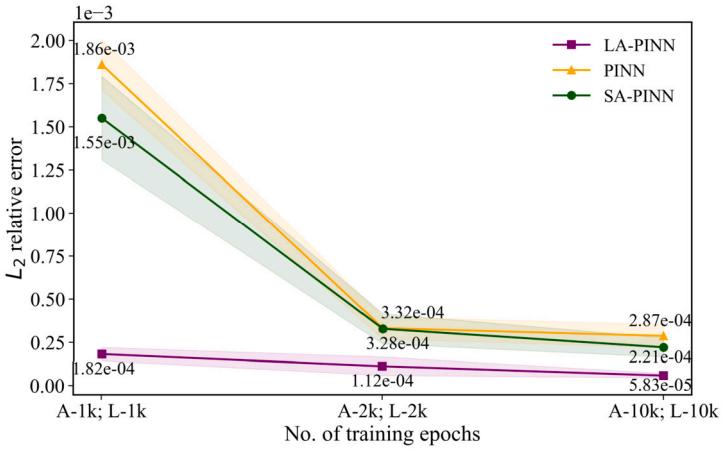


Fig. 8. The L_2 error of Poisson equation at different training epochs.

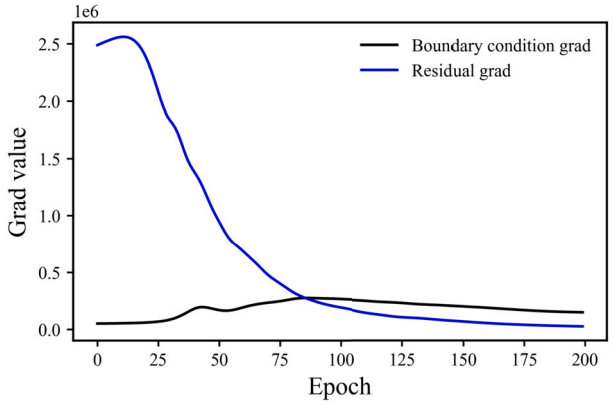
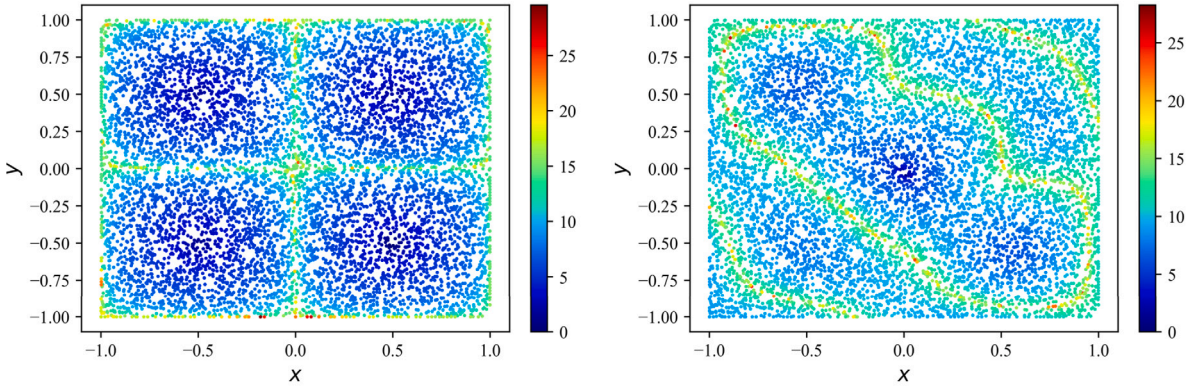


Fig. 9. The mean gradient of individual loss components throughout 200 epochs.

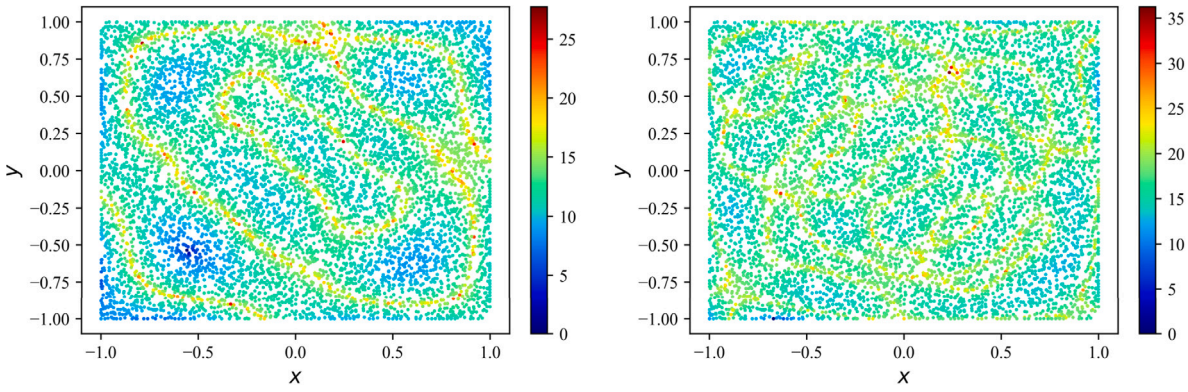
$$u(t,-1) = u(t,1) = 0. \tag{48}$$

The main net of LA-PINN is fully connected with a layer structure including 1 input layer of two neurons, 8 hidden layers with 20 neurons each, and 1 output layer with one neuron. LANs have the same structure except for the input layer with only one neuron. We select 100, 200, and 10000 points from the initial condition, boundary condition, and residual, respectively. All initial learning rates for Adam optimizer in main net and LANs are set to 1e-3.



(a) Weights distribution at 10th epoch

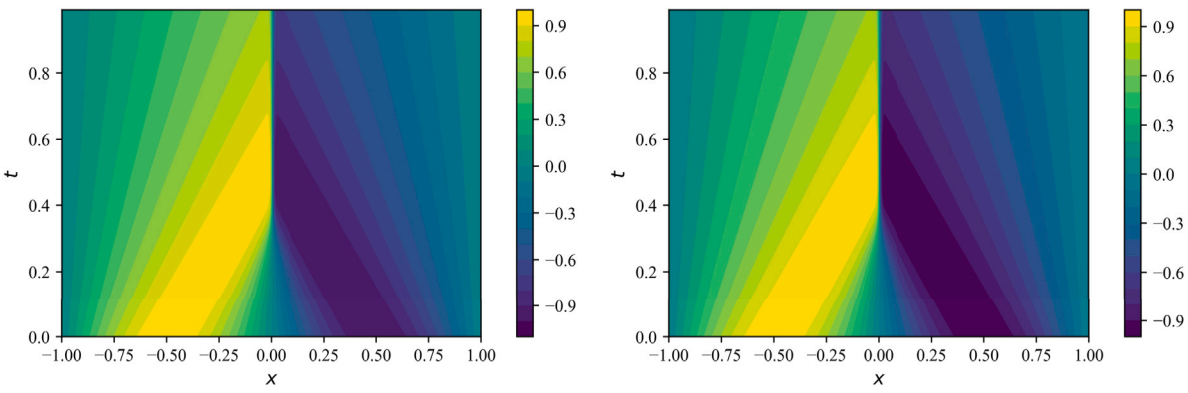
(b) Weights distribution at 50th epoch



(c) Weights distribution at 85th epoch

(d) Weights distribution at 200th epoch

Fig. 10. The evolution of weights distribution for Poisson equation at each training point. The points weight distributions represent the 10th, 50th, 85th, and 200th epochs, respectively, as we navigate from left to right and from top to bottom.



(a) Predicted $\hat{u}$

(b) Exact u

Fig. 11. The comparison between the predicted and exact solution of Burger equation.

Fig. 11 shows a good result of LA-PINN, compared with the exact solution. It is capable of accurately solving this burger equation under the observation of different times, as demonstrated in Fig. 12.

To compare the performance of LA-PINN, PINN, and SA-PINN, we allow all the settings, including net architecture, points number, learning rate, and so forth, to keep consistent. It can be found in Fig. 13, a 2.83e-4 mean L_2 error is obtained from LA-PINN after 10 runs, which is better than 4.80e-4 achieved by SA-PINN. More importantly, the LA-PINN model demonstrates considerable

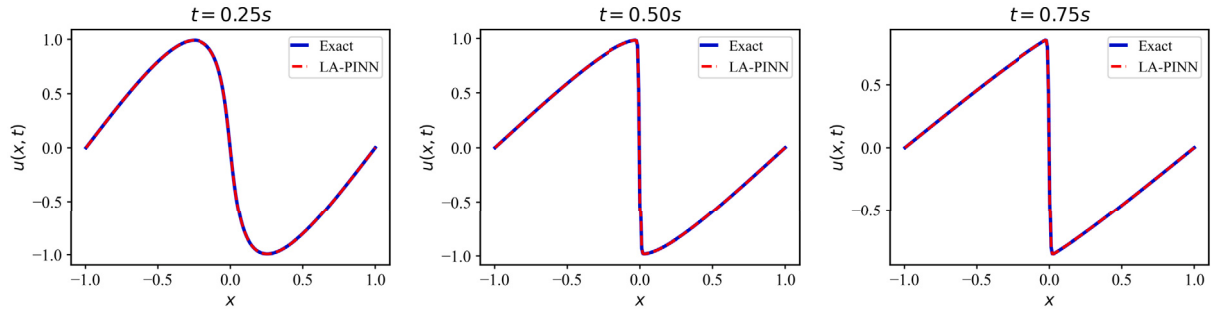


Fig. 12. The solution of Burger equation at different times using LA-PINN.

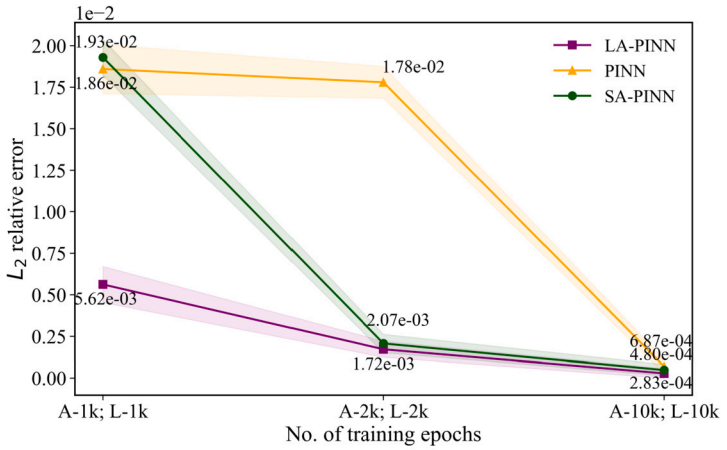


Fig. 13. The L_2 error of 1-D Burger equation at different training epochs.

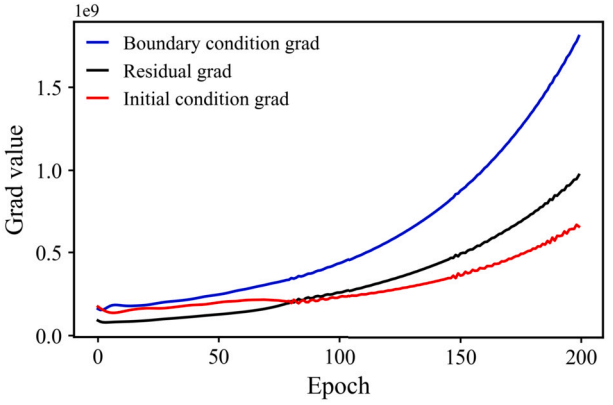
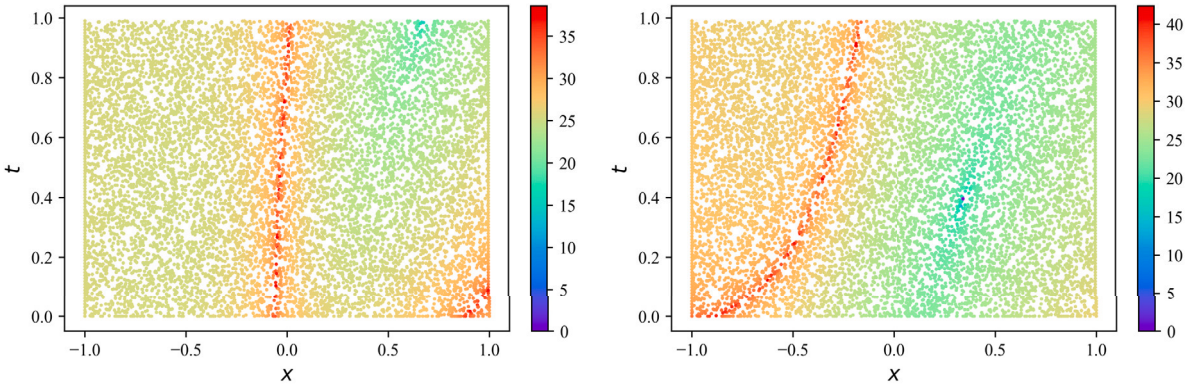


Fig. 14. The mean gradient of individual loss components throughout 200 epochs.

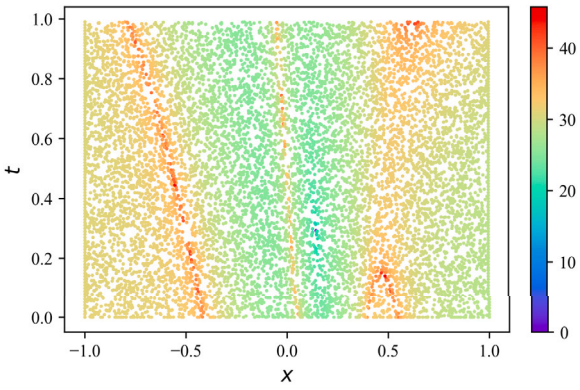
predictive capability with 5.62e-3 L_2 error after merely 1k training epochs, reflecting that the proposed novel architecture has a rapid convergence attribute.

The mean gradient of the residual is consistently lower than the boundary gradient, with the difference between them gradually widening in Fig. 14. This means that the weights of the points in the residual should progressively increase to assist points with smaller gradients in accelerating the procedure of minimizing point SE. Fig. 15 illustrates this phenomenon: as the number of epochs rises in Figures from 15a to 15d, the area of the red lines (representing larger values) within the x-t domain regularly expands. In addition, we would add that the points in the stiffness region with rapid changes in timescale, which are located at the area with $x = 0$, are assigned relatively higher weight values during training epochs, as depicted in Fig. 15a, 15c, and 15d; noticeable red lines appear at the area with $x = 0$.

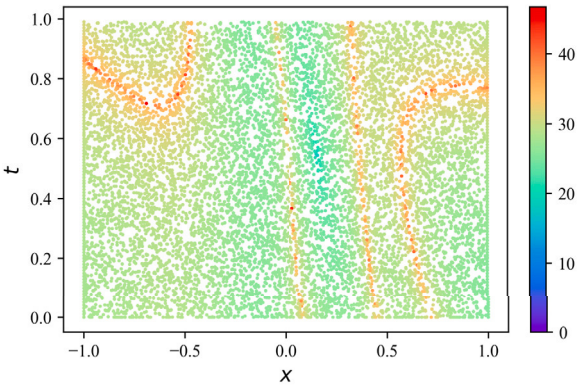


(a) Weights distribution at 20th epoch

(b) Weights distribution at 50th epoch



(c) Weights distribution at 100th epoch



(d) Weights distribution at 150th epoch

Fig. 15. The evolution of weights distribution for 1-D Burger equation.

3.4. 2D Burger equation

In this section, the solving results of a 2D burger equation are reported.

$$u_t + uu_x + vu_y = v(u_{xx} + u_{yy}), \tag{49}$$

$$v_t + uv_x + vv_y = v(v_{xx} + v_{yy}). \tag{50}$$

The equation could be analytically solved as

$$u(x,y,t) = \frac{3}{4} - \frac{1}{4}[1 + exp(\frac{-4x + 4y - t}{32v})]^{-1} \tag{51}$$

$$v(x,y,t) = \frac{3}{4} + \frac{1}{4}[1 + exp(\frac{-4x + 4y - t}{32v})]^{-1} \tag{52}$$

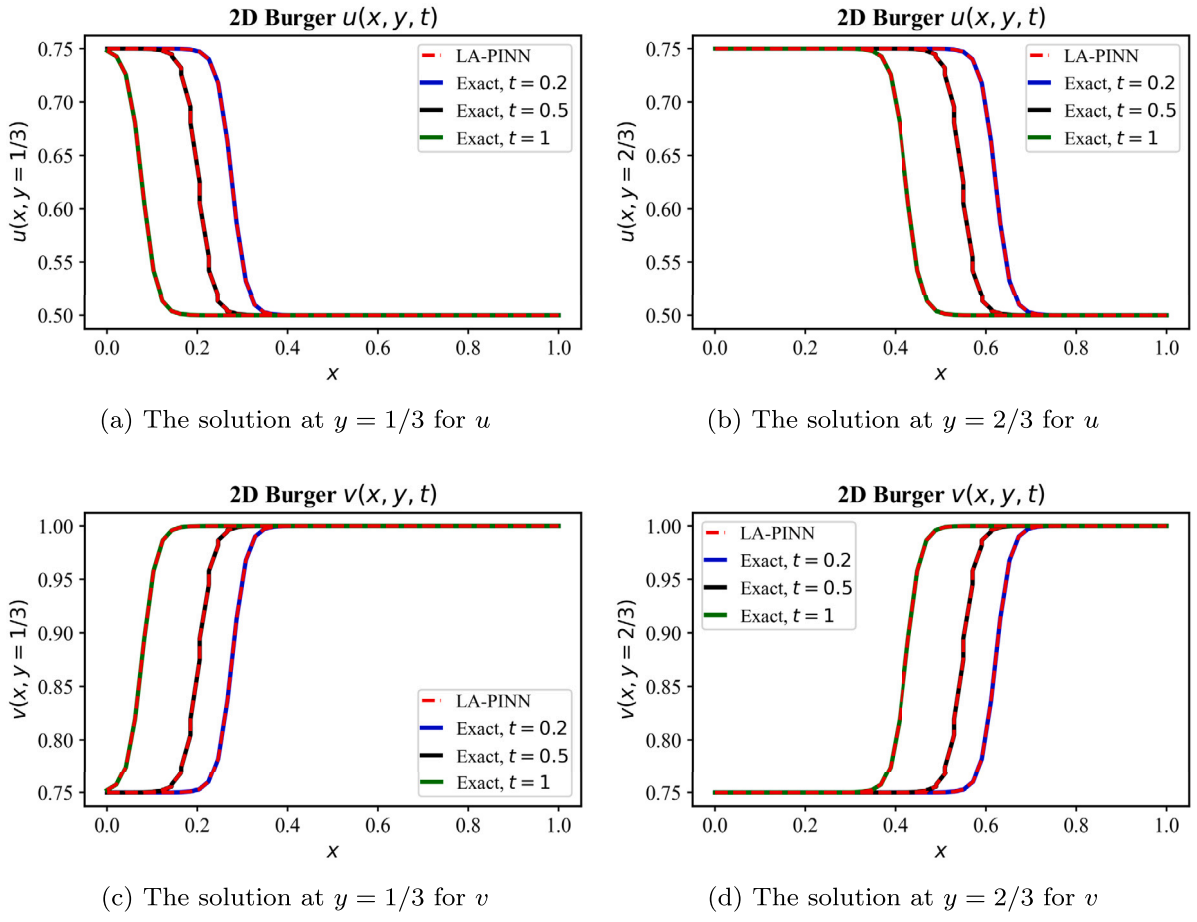
Where x, y, and t all belong to the interval $[0,1]$. According to the governing equations and analytical solution, the training points are produced from initial and boundary conditions (with corresponding u, v values) along with the residual (without exact u, v values) by randomly selecting $N_0 = 100$, $N_b = 200$, and $N_r = 10000$ points from respective datasets. For this two outputs problem, two corresponding terms for u and v exist in each loss component, collectively constituting the total loss, namely

$$\mathcal{L}^\star = \mathcal{L}^\star_{0u} + \mathcal{L}^\star_{0v} + \mathcal{L}^\star_{bu} + \mathcal{L}^\star_{bv} + \mathcal{L}^\star_{ru} + \mathcal{L}^\star_{rv} \tag{53}$$

It should be pointed out that three LANs remain to be used to build attentional function, and the loss sub-components of u and v are included in one LAN to be weighted.

Three neurons input layer, 8 hidden layers of 20 neurons, and 2 neurons output layer are net configurations for this 2-D equation. The initial learning rates set for all optimizers are 1e-3 in the main net and LANs.

We adopt the same manner of figure showing as in paper [24] for comparison. The LA-PINN could achieve good accuracy in predicting the solution u and v, regardless of observing at different times or values of y, as shown in Fig. 16. Moreover, for the



(a) The solution at $y = 1/3$ for u

(b) The solution at $y = 2/3$ for u

(c) The solution at $y = 1/3$ for v

(d) The solution at $y = 2/3$ for v

Fig. 16. The comparison between the predicted and exact solution for the 2-D Burger equation. The solutions at $t = 0.2, 0.5, 1$ are displayed within the same figure. From top to bottom, the first row presents the solution for u, while the second row shows the solution for v. From left to right, the first column depicts the solution at $y = 1/3$, and the second column illustrates the solution at $y = 2/3$.

Table 1
L_2 error of 2-D Burger equation.

L_2 error at various epochs	LA-PINN	PINN	SA-PINN
A-1k; L-1k	u: 1.54e-4 v: 1.38e-4	u: 9.0e-4 v: 8.59e-4	u: 4.64e-4 v: 3.45e-4
A-10k; L-10k	u: 1.15e-05 v: 9.32e-06	u: 6.82e-4 v: 6.24e-4	u: 3.64e-4 v: 2.91e-4

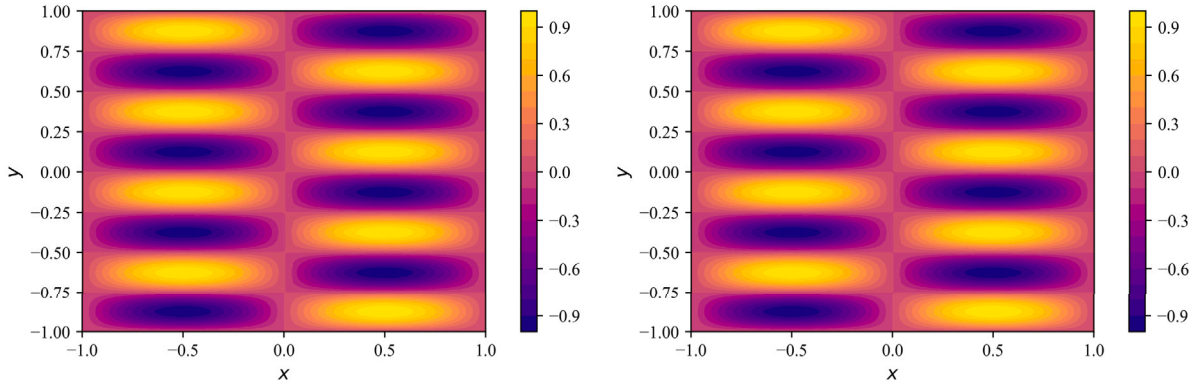
hard-to-fit area from about $x = 0.1$ to $x = 0.7$, the predictive efficacy of LA-PINN is markedly more advanced than those reported in [24].

Table 1 illustrates that the LA-PINN has the best performance in solving the 2-D burger equation either at 1k iterations or at 10k iterations using Adam and L-BFGS, compared with PINN and SA-PINN. With only 1k epochs, its predictive performance surpasses that of other models at 10k epochs, and its L_2 error still has a significant reduction when the epoch reaches 10k. In contrast, this reduction is not manifested apparently in PINN and SA-PINN, indicating that the proposed LA-PINN retains a notable capacity for converging to the optimal solution when navigating the challenges of 2-D burger equation that is prone to local minimization [40].

3.5. Helmholtz equation

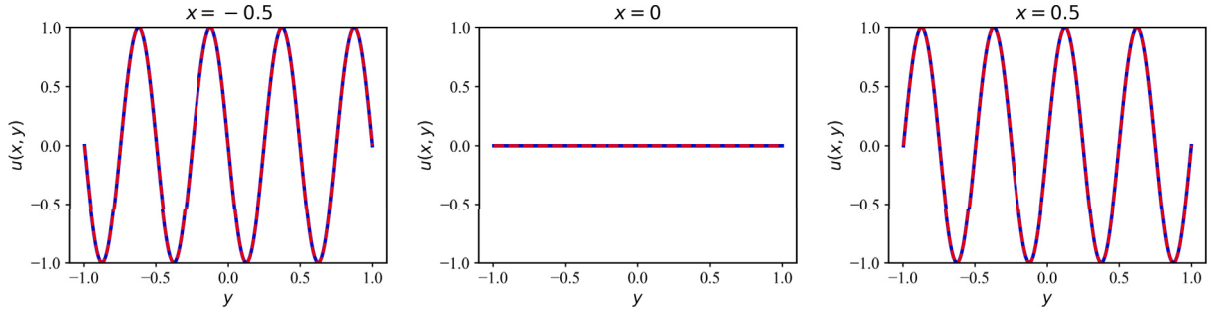
Here we illustrate the efficacy of LA-PINN in solving the Helmholtz equation.

$$u_{xx} + u_{yy} + k^2 u - q(x, y) = 0, \quad x \in [-1, 1], y \in [-1, 1] \tag{54}$$



(a) Predicted $\hat{u}$ (b) Exact u

Fig. 17. The comparison between the predicted and exact solution of Helmholtz equation.



(a) The solution at $x = -0.5$ (b) The solution at $x = 0$ (c) The solution at $x = 0.5$

Fig. 18. The solution of LA-PINN for Helmholtz equation.

$$u(-1, y) = u(1, y) = u(x, -1) = u(x, 1) = 0 \tag{55}$$

The forcing term is

$$q(x, y) = -(\pi)^2 \sin(\pi x)\sin(4\pi y) \tag{56}$$

$$-(4\pi)^2 \sin(\pi x)\sin(4\pi y) \tag{57}$$

$$+k^2 \sin(\pi x)\sin(4\pi y)w \tag{58}$$

which leads to an analytical solution:

$$u(x, y) = \sin(\pi x)\sin(4\pi y) \tag{59}$$

In this problem, we use [2, 50, 50, 50, 50, 1] layer sizes to configure main net and LANs (except for the input layer). $N_b = 400$, $N_r = 20000$, and learning rate equals to 1e-3 are basic settings. The results can be observed from Figs. 17 and 18; LA-PINN realizes an appreciable level of predictive accuracy when solving the Helmholtz equation.

By observing the L_2 errors of various models at different epochs in Fig. 19, we can find that LA-PINN showcases a significantly lower error compared to the other two models at only 2k training epochs. When the epoch count reaches 10k, the error contracts to 2.29e-4, considerably less than PINN, SA-PINN, and the error reported in [21].

Figs. 20 and 21 demonstrate that at the initial stage of training epochs, a significant gap exists between the mean gradients of boundary condition and residuals—the gradient of boundary condition is extremely small, approximating zero, while that of the residual is obviously larger. This gap influences the weights distribution, leading to a marked contrast between the boundary and internal regions, as shown in Fig. 21a. The LA-PINN allocates relatively minor weights for residual points with a more substantial gradient, lying many deep blue colour points within the internal space. Conversely, it assigns comparatively larger weights for points with lesser gradients on the boundary condition, allocating nearly all bright colour points on the boundary. The gap in the gradients begins to diminish after 100 epochs, along with changes happening in weights distribution, as portrayed in Figs. 21b, 21c, and 21d, there are gradually more high weight points (bright colour points) appearing at the inner region, and the difference in weight distribution between boundary condition and residual becomes less noticeable. Furthermore, from Figs. 21a and 21b, we can see that the high-weight points with bright colours are distributed in a grid pattern within the internal region. This means that the LA-PINN

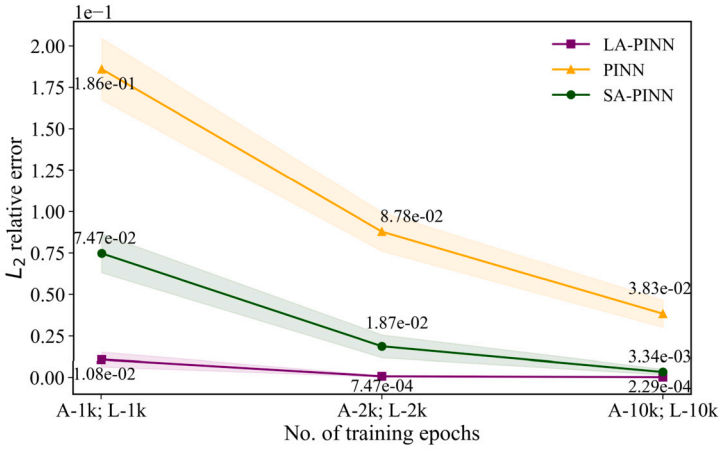


Fig. 19. The L_2 error of Helmholtz equation at different training epochs.

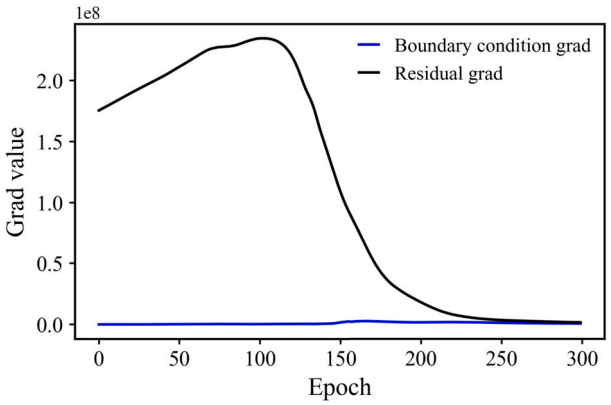


Fig. 20. The mean gradient of individual loss components throughout 300 epochs.

also holds the capability to "identify" points undergoing rapid changes in the Helmholtz solution space and to accord them higher weights.

3.6. Allen-Cahn equation

The Allen-Cahn (AC) equation, characterized by possessing stiffness region, has drawn broad interest in the investigations of solving PDEs by PINN model [20,41,42]. In this context, we consider the subsequent AC equation form:

$$u_t - 0.0001u_{xx} + 5u^3 - 5u = 0, \quad x \in [-1, 1],\ t \in [0, 1], \tag{60}$$

$$u(x, 0) = x^2 \cos(\pi x), \tag{61}$$

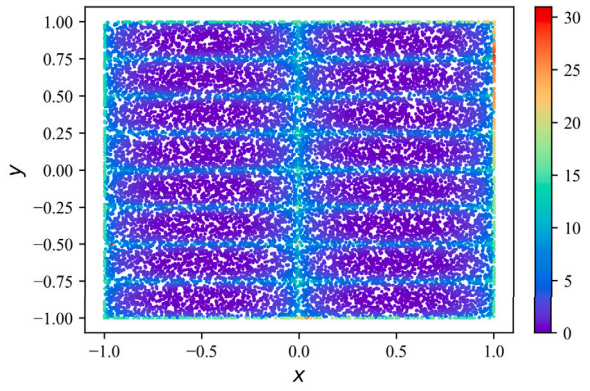
$$u(t, -1) = u(t, 1), \tag{62}$$

$$u_x(t, -1) = u_x(t, 1). \tag{63}$$

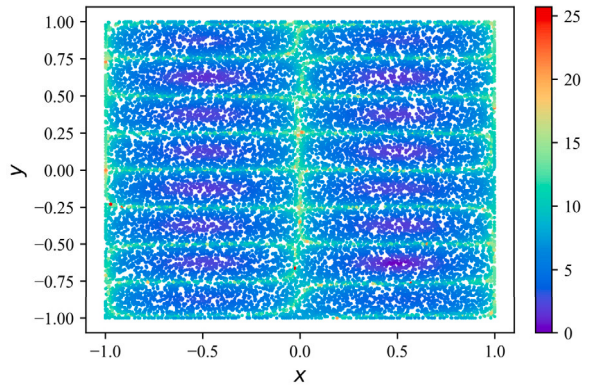
The main net of LA-PINN is trained using layer sizes with [2, 128, 128, 128, 128, 1], and we set $N_0 = 100$, $N_b = 100$, and $N_r = 20000$ as well as the learning rate of 1e-3 as basic configurations. Given that the boundary conditions of AC equation exhibit periodic alteration, which significantly hinders the model convergence, two LANs are deployed on the loss component of boundary conditions corresponding to Equations (62) and (63), respectively. That is to say, we have four LANs, and they are subjected to simultaneous training and parameter updating along with the main net.

After training 100k iterations for Adam and 100k for L-BFGS, we obtain an 8.22e-3 L_2 error much better than both PINN and SA-PINN models (Table 2).

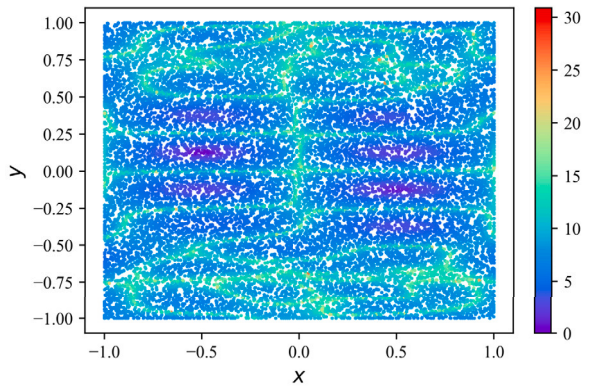
The results can be found in Fig. 22, revealing that LA-PINN performs well in addressing the AC problem. The stiffness region of AC equation appears in the area with $x = 0$, as shown in Fig. 23c and 23d. Accordingly, LA-PINN autonomously identifies $x = 0$ as a "challenging" region through the distribution of weights within the space, a detail visible in Fig. 24. From the initiation of epochs, the LANs promptly perceive this area and progressively increase the weights during the following training iterations, as demonstrated



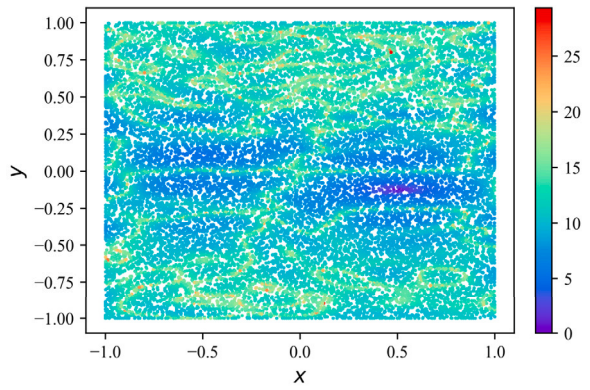
(a) Weights distribution at 5th epoch



(b) Weights distribution at 100th epoch



(c) Weights distribution at 150th epoch

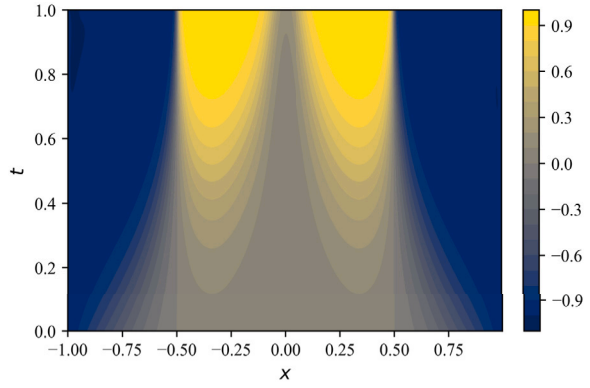


(d) Weights distribution at 300th epoch

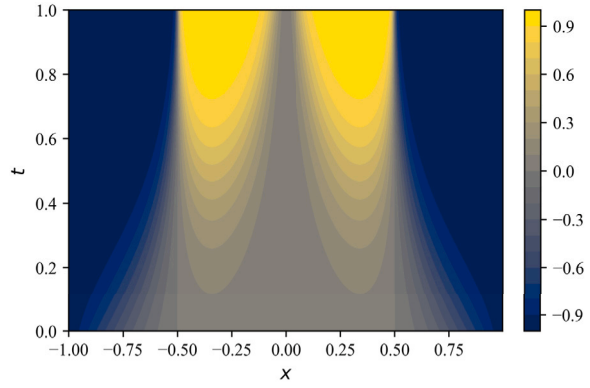
Fig. 21. The evolution of weights distribution for Helmholtz equation.

Table 2
L_2 error of Allen-Cahn equation.

L_2 error at various epochs	LA-PINN	PINN	SA-PINN
A-100k; L-100k	8.22e-3	9.53e-1	3.34e-2

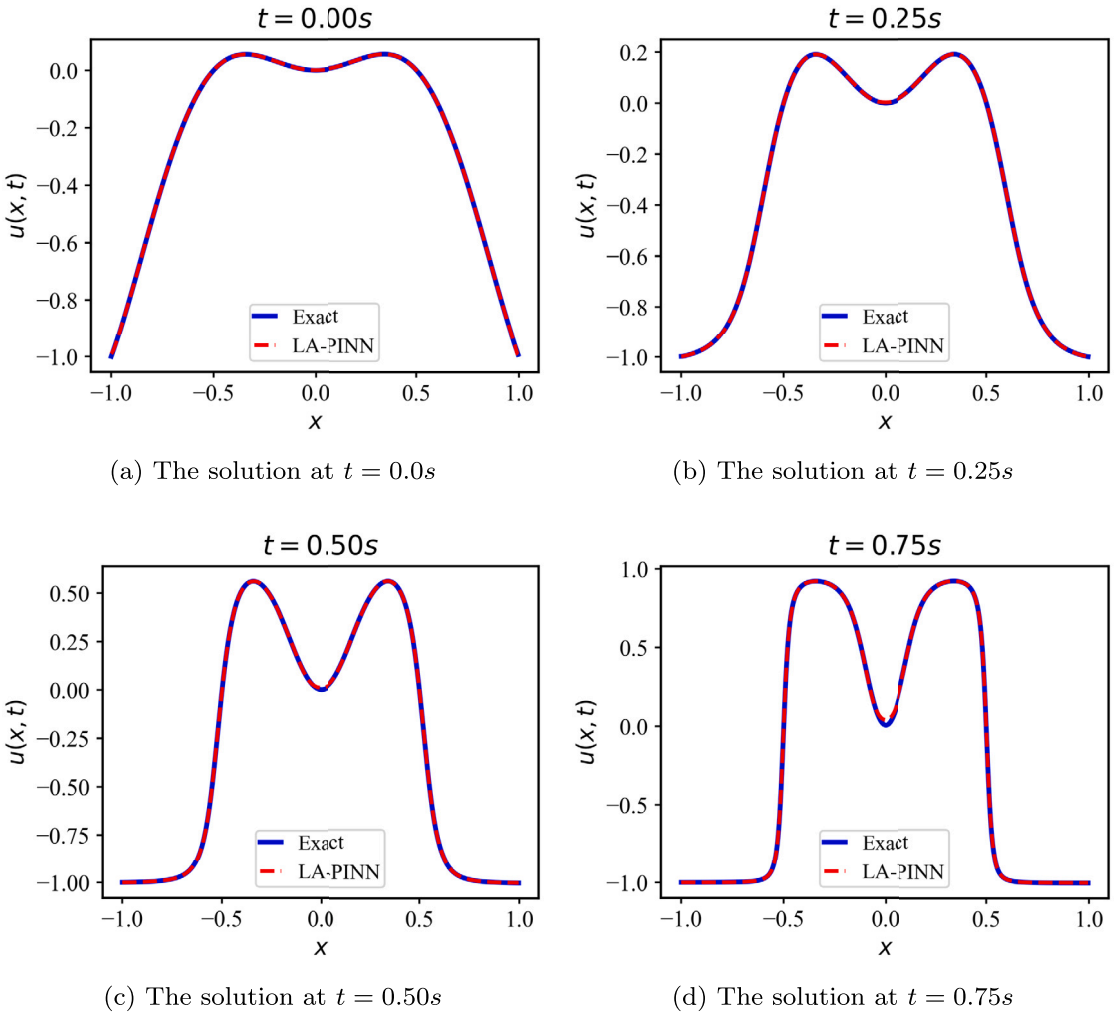


(a) Predicted $\hat{u}$



(b) Exact u

Fig. 22. The comparison between the predicted and exact solution of AC equation.



(a) The solution at $t = 0.0s$ (b) The solution at $t = 0.25s$

(c) The solution at $t = 0.50s$ (d) The solution at $t = 0.75s$

Fig. 23. The solution of LA-PINN for AC equation at different times.

by the gradually deepening red colour of the line at $x = 0$ in Fig. 24a-24d. It is found that LA-PINN can autonomously identify the stiffness region and gradually increase weight values for the points in that region during training process. This validates what was described in subsection 2.3.

3.7. Diffusion-reaction equation

We consider the diffusion-reaction equation in this section, which has extensive application scenarios in chemistry and biological fields.

The governing equation is:

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + R(x,t), \quad x \in [-\pi, \pi],\ t \in [0,1], \tag{64}$$

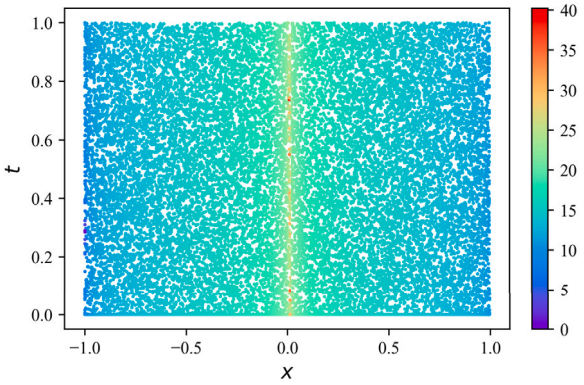
here $D = 1$ denotes the diffusion coefficient and u is defined as the measure of solute concentration. R is the chemical reaction with the expression:

$$R(x,t) = e^{-t}\left[\frac{3}{2}\sin(2x) + \frac{8}{3}\sin(3x) + \frac{15}{4}\sin(4x) + \frac{63}{8}\sin(8x)\right]. \tag{65}$$

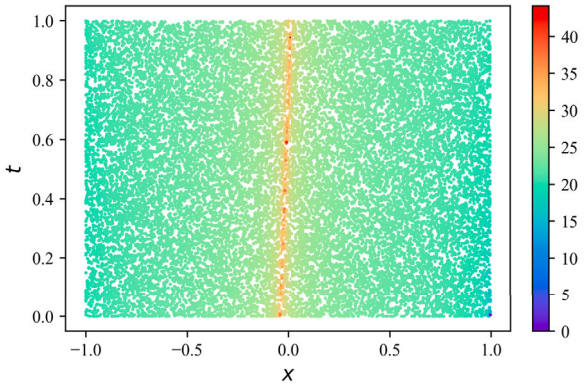
The initial and boundary conditions are as follows:

$$u(x,0) = \sum_{i=1}^{4}\frac{\sin(ix)}{i} + \frac{\sin(8x)}{8}, \tag{66}$$

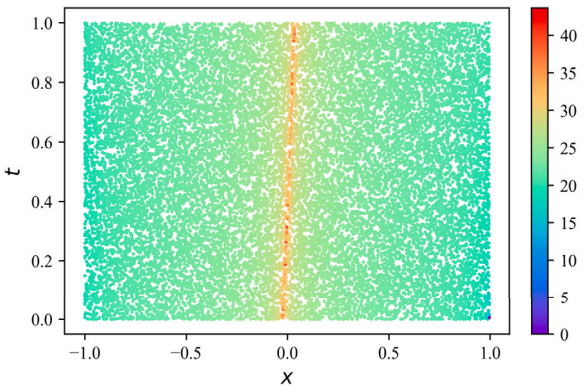
$$u(-\pi,t) = u(\pi,t) = 0 \tag{67}$$



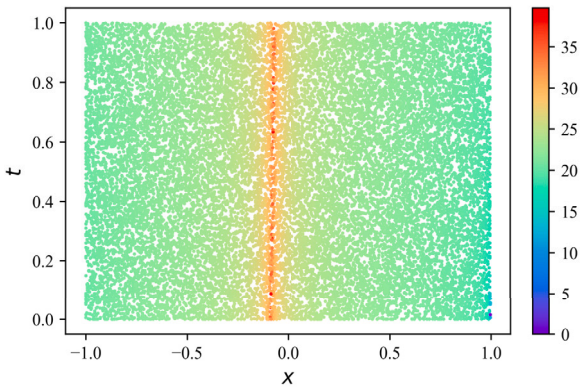
(a) Weights distribution at 5th epoch



(b) Weights distribution at 50th epoch

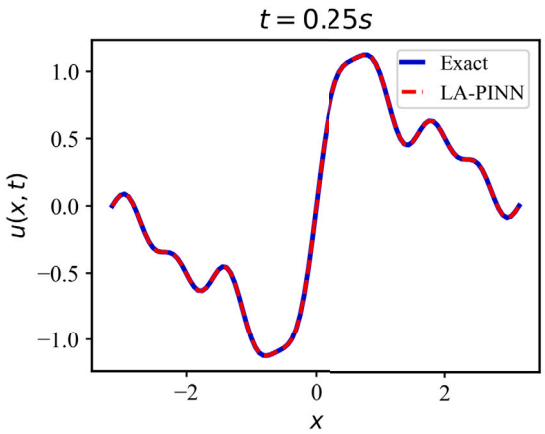


(c) Weights distribution at 51th epoch

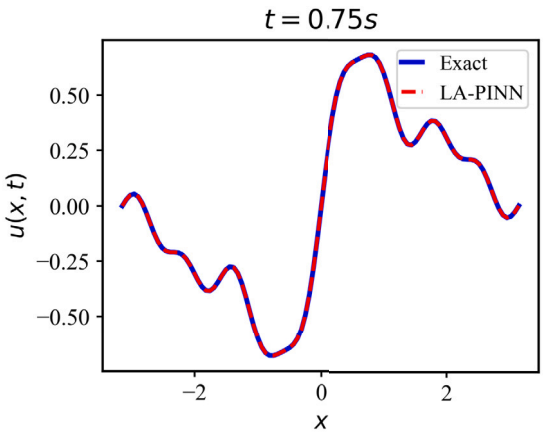


(d) Weights distribution at 52th epoch

Fig. 24. The evolution of weights distribution for AC equation.



(a) The solution at $t = 0.25s$



(b) The solution at $t = 0.75s$

Fig. 25. The solution of LA-PINN for Diffusion-reaction equation at different times.

The analytical solution is described as:

$$u(x,t) = e^{-t}\left[\sum_{i=1}^{4}\frac{\sin(ix)}{i} + \frac{\sin(8x)}{8}\right]. \tag{68}$$

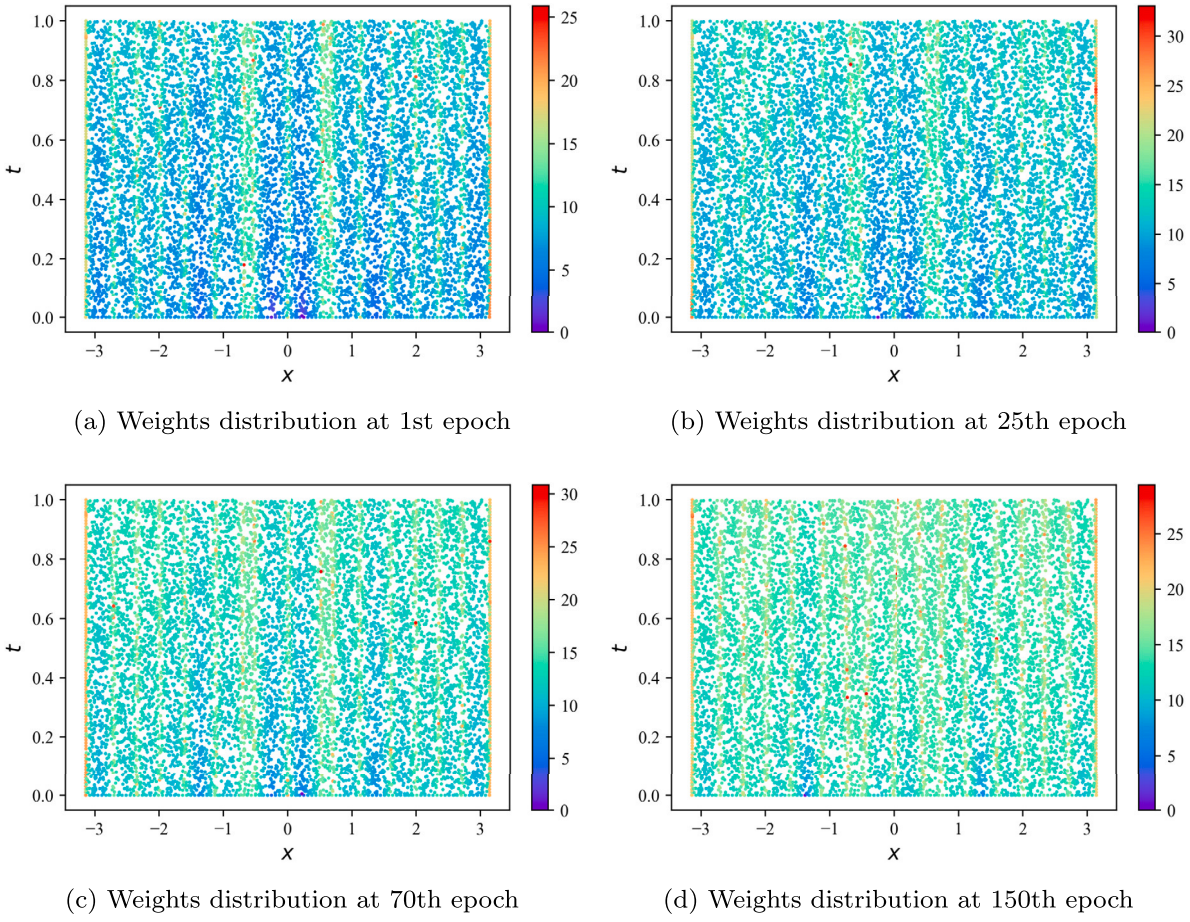


Fig. 26. The evolution of weights distribution for Diffusion-reaction equation.

We utilize a layer structure of [2, 50, 50, 50, 50, 50, 1] as the main net for predicting $\hat{u}$. Three loss components are associated with three different LANs, corresponding respectively to initial condition, boundary condition, and the residual of governing equations.

Fig. 25 provides a visual comparison between the predicted and exact results at $t = 0.25$ and $t = 0.75$. It is particularly noteworthy that the LA-PINN sustains a high level of predictive precision in regions of rapidly spatial change over time, exemplified within the [-1, 1] domain. To better fit these challenging areas, LANs increase the weights assigned to them throughout training process. This is evident in Fig. 26, where the colours in these difficult regions become progressively brighter as the number of training epochs increases.

In a comparative analysis involving PINN and SA-PINN, Fig. 27 shows that LA-PINN consistently exhibits a lower L_2 error across various training epochs (with only 4.21e-05 error after 20k epochs). The more gradual slope of the error curve associated with LA-PINN indicates an enhanced ability for quicker convergence compared to the other two models.

4. Conclusion

In this paper, to tackle the issue that the vanilla PINN model suffers from slow convergence speed and poor accuracy in approximating the solution at the hard-to-fit region, we introduced a novel architecture named loss-attentional physics-informed neural network (LA-PINN). This architecture pays attention to the control of loss error at each training point by allocating every independent loss component with a loss-attentional net (LAN). Proceeding by feeding point squared error (SE) into LAN, the "attentional function" could be built to accomplish the task of distributing different weights to diverse point errors. The point error-based dynamic weighting approach employed in the LA-PINN architecture draws inspiration from the adversarial training process of the generator and discriminator in the Generative Adversarial Networks (GAN), with one implementing gradient ascent and the other gradient descent. In LA-PINN, the main net, which delivers the predictive solution, shoulders the responsibility of minimizing loss function via gradient descent, while the LANs take on the role of assigning different weights to each point SE using gradient ascent. When encountering stiffness points, LA-PINN model can increase the growth rate of weights at such points and use this growth rate as a scale to multiply the growth rate of the update gradient for point SE, thereby aiding the convergence by this weighting mechanism.

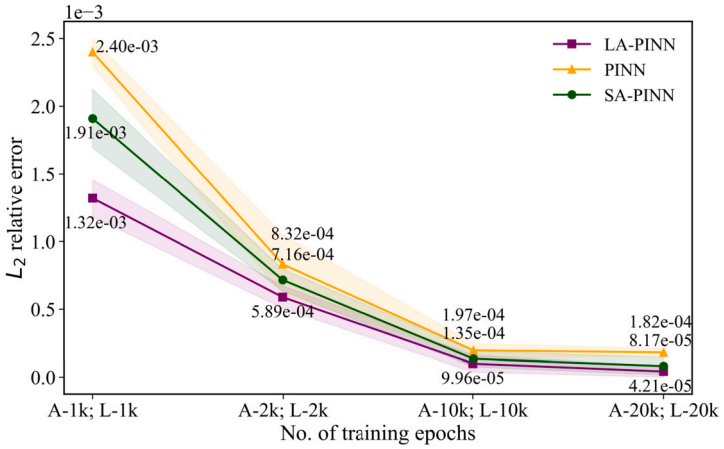


Fig. 27. The L_2 error of Diffusion-reaction equation at different training epochs.

Numerical experiments of several benchmark PDEs indicate that LA-PINN outperforms both PINN and SA-PINN in predictive accuracy, given the same basic settings and number of training epochs. The predicted L_2 error is lower by 1-2 orders of magnitude than other methods. More importantly, it showcases an impressively fast convergence capability, with high accuracy at merely 1k Adam and 1k L-BFGS epochs. Additionally, LA-PINN exhibits "intelligence" in its assignment of weights. Not only can LA-PINN distributes higher weights to point errors with smaller update gradients and relatively lower weights to those with larger gradients, but it can also identify points in regions with sharp size or stiffness feature, providing them with higher weights, which in turn facilitates the minimization of their errors.

Despite the LA-PINN architecture demonstrating remarkable performance in prediction accuracy and convergence speed, we only analysed the weighting mechanism, lacking a deeper theoretical interpretation for this algorithm. Currently, NTK can yield an initial theoretical analysis for PINN model, which is a part of our future endeavour concerning the LA-PINN. It is projected that the LA-PINN framework holds immense potential, notably in tasks with multiple loss components. Collaborating multiple networks in LA-PINN architecture may significantly boost predictive performance when dealing with complex equation problems. In addition, investigating the update algorithms for each LAN and exploring the alternatives for main net will also be essential research directions for us in the future.

CRediT authorship contribution statement

Yanjie Song: Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis. **He Wang:** Writing – review & editing, Supervision, Conceptualization. **He Yang:** Writing – review & editing. **Maria Luisa Taccari:** Writing – review & editing, Conceptualization. **Xiaohui Chen:** Writing – review & editing, Supervision, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–444.
[2] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[3] M. Raissi, Forward-backward stochastic neural networks: deep learning of high-dimensional partial differential equations, arXiv preprint, arXiv:1804.07010, 2018.
[4] M. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, Commun. Numer. Methods Eng. 10 (1994) 195–201.
[5] D. Zhang, L. Lu, L. Guo, G.E. Karniadakis, Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems, J. Comput. Phys. 397 (2019) 108850.
[6] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (2018) 1–43.
[7] S. Cai, Z. Mao, Z. Wang, M. Yin, G.E. Karniadakis, Physics-informed neural networks (PINNs) for fluid mechanics: a review, Acta Mech. Sin. 37 (2021) 1727–1738.
[8] J. Willard, X. Jia, S. Xu, M. Steinbach, V. Kumar, Integrating scientific knowledge with machine learning for engineering and environmental systems, ACM Comput. Surv. 55 (2022) 1–37.

[9] T. Praditia, Physics-informed neural networks for learning dynamic, distributed and uncertain systems, https://doi.org/10.18419/opus-13229, 2023.
[10] X. Meng, Z. Li, D. Zhang, G.E. Karniadakis, PPINN: parareal physics-informed neural network for time-dependent PDEs, Comput. Methods Appl. Mech. Eng. 370 (2020) 113250.
[11] L. Yang, X. Meng, G.E. Karniadakis, B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data, J. Comput. Phys. 425 (2021) 109913.
[12] S. Cai, Z. Wang, S. Wang, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks for heat transfer problems, J. Heat Transf. 143 (2021) 060801.
[13] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (2021) 422–440.
[14] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, DeepXDE: a deep learning library for solving differential equations, SIAM Rev. 63 (2021) 208–228.
[15] A.F. Psaros, K. Kawaguchi, G.E. Karniadakis, Meta-learning PINN loss functions, J. Comput. Phys. 458 (2022) 111121.
[16] R.L. Burden, J.D. Faires, A.M. Burden, Numerical Analysis, Cengage learning, 2015.
[17] E. Kharazmi, Z. Zhang, G.E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations, arXiv preprint, arXiv:1912.00873, 2019.
[18] J. Yu, L. Lu, X. Meng, G.E. Karniadakis, Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems, Comput. Methods Appl. Mech. Eng. 393 (2022) 114823.
[19] D. Liu, Y. Wang, A dual-dimer method for training physics-constrained neural networks with minimax architecture, Neural Netw. 136 (2021) 112–125.
[20] C.L. Wight, J. Zhao, Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks, arXiv preprint, arXiv:2007.04542, 2020.
[21] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM J. Sci. Comput. 43 (2021) A3055–A3081.
[22] Z. Xiang, W. Peng, X. Liu, W. Yao, Self-adaptive loss balanced physics-informed neural networks, Neurocomputing 496 (2022) 11–34.
[23] S. Wang, X. Yu, P. Perdikaris, When and why PINNs fail to train: a neural tangent kernel perspective, J. Comput. Phys. 449 (2022) 110768.
[24] L.D. McClenny, U.M. Braga-Neto, Self-adaptive physics-informed neural networks, J. Comput. Phys. 474 (2023) 111722.
[25] W. Li, C. Zhang, C. Wang, H. Guan, D. Tao, Revisiting PINNs: generative adversarial physics-informed neural networks and point-weighting method, arXiv preprint, arXiv:2205.08754, 2022.
[26] G. Zhang, H. Yang, F. Zhu, Y. Chen, x. zheng, Dasa-Pinns: Differentiable Adversarial Self-Adaptive Pointwise Weighting Scheme for Physics-Informed Neural Networks. Available at SSRN: https://ssrn.com/abstract=4376049, https://doi.org/10.2139/ssrn.4376049.
[27] S.J. Anagnostopoulos, J.D. Toscano, N. Stergiopulos, G.E. Karniadakis, Residual-based attention and connection to information bottleneck theory in PINNs, arXiv preprint, arXiv:2307.00379, 2023.
[28] P.-H. Chiu, J.C. Wong, C. Ooi, M.H. Dao, Y.-S. Ong Can-pinn, A fast physics-informed neural network based on coupled-automatic–numerical differentiation method, Comput. Methods Appl. Mech. Eng. 395 (2022) 114909.
[29] J. Li, J. Chen, B. Li, Gradient-optimized physics-informed neural networks (GOPINNS): a deep learning method for solving the complex modified KdV equation, Nonlinear Dyn. 107 (2022) 781–792.
[30] S. Ruder, An overview of gradient descent optimization algorithms, arXiv preprint, arXiv:1609.04747, 2016.
[31] R. Bischof, M. Kraus, Multi-objective loss balancing for physics-informed deep learning, arXiv preprint, arXiv:2110.09813, 2021.
[32] X. Jin, S. Cai, H. Li, G.E. Karniadakis, NSFnets (Navier-Stokes flow nets): physics-informed neural networks for the incompressible Navier-Stokes equations, J. Comput. Phys. 426 (2021) 109951.
[33] Y. Liu, L. Cai, Y. Chen, B. Wang, Physics-informed neural networks based on adaptive weighted loss functions for Hamilton-Jacobi equations, Math. Biosci. Eng. 19 (2022) 12866–12896.
[34] G.-M. Gie, Y. Hong, C.-Y. Jung, Semi-analytic PINN methods for singularly perturbed boundary value problems, arXiv preprint, arXiv:2208.09145, 2022.
[35] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.
[36] E. Fan, Extended tanh-function method and its applications to nonlinear equations, Phys. Lett. A 277 (2000) 212–218.
[37] D.P. Kingma, J. Ba Adam, A method for stochastic optimization, arXiv preprint, arXiv:1412.6980, 2014.
[38] D.C. Liu, J. Nocedal, On the limited memory BFGS method for large scale optimization, Math. Program. 45 (1989) 503–528.
[39] S. Hu, M. Liu, S. Zhang, S. Dong, R. Zheng, Physics-informed neural network combined with characteristic-based split for solving Navier–Stokes equations, Eng. Appl. Artif. Intell. 128 (2024) 107453.
[40] C. Bajaj, L. McLennan, T. Andeen, A. Roy, Recipes for when physics fails: recovering robust learning of physics informed neural networks, Mach. Learn.: Sci. Technol. 4 (2023) 015013.
[41] H. Xu, J. Chen, F. Ma, Adaptive deep learning approximation for Allen-Cahn equation, in: International Conference on Computational Science, Springer, 2022, pp. 271–283.
[42] R. Mattey, S. Ghosh, A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations, Comput. Methods Appl. Mech. Eng. 390 (2022) 114474.