Revisiting PINNs: Generative Adversarial Physics-informed Neural Networks and Point-weighting Method

Wensheng Li[1], Chao Zhang[1]*, Chuncheng Wang[2], Hanting Guan[1], Dacheng Tao[3]

[1]School of Mathematical Sciences, Dalian University of Technology, China

`Lws@mail.dlut.edu.cn; chao.zhang@dlut.edu.cn; miracie@mail.dlut.edu.cn`

[2]School of Mathematics, Harbin Institute of Technology, China

`wangchuncheng@hit.edu.cn`

[3]JD Explore Academy, China

`dacheng.tao@gmail.com`

May 19, 2022

Abstract

Physics-informed neural networks (PINNs) provide a deep learning framework for numerically solving partial differential equations (PDEs), and have been widely used in a variety of PDE problems. However, there still remain some challenges in the application of PINNs: 1) the mechanism of PINNs is unsuitable (at least cannot be directly applied) to exploiting a small size of (usually very few) extra informative samples to refine the networks; and 2) the efficiency of training PINNs often becomes low for some complicated PDEs. In this paper, we propose the generative adversarial physics-informed neural network (GA-PINN), which integrates the generative adversarial (GA) mechanism with the structure of PINNs, to improve the performance of PINNs by exploiting only a small size of exact solutions to the PDEs. Inspired from the weighting strategy of the Adaboost method, we then introduce a point-weighting (PW) method to improve the training efficiency of PINNs, where the weight of each sample point is adaptively updated at each training iteration. The numerical experiments show that GA-PINNs outperform PINNs in many well-known PDEs and the PW method also improves the efficiency of training PINNs and GA-PINNs.

1 Introduction

Partial differential equations (PDEs) are the core mathematical tools for studying the evolution of physical systems, and play an essential role in various engineering applications, *e.g.,* Navier-Stokes equations for hydrodynamics [3] and Laplace's equation for celestial problems [15]. Consider a general form of partial differential equations (PDEs):

---

*Corresponding author.

$$\begin{cases} \mathcal{L}[u](\mathbf{x}) = F(\mathbf{x}), & \mathbf{x} \in \Omega; \\ \mathcal{B}_i[u](\mathbf{x}) = G_i(\mathbf{x}), & \mathbf{x} \in \partial_i\Omega, \ \ 1 \le i \le I, \end{cases} \tag{1}$$

where $u(\mathbf{x}) : \Omega \to \mathbb{R}^d$ is the solution to the PDE problem; $\mathcal{L}$ is the nonlinear partial differential operator; $\mathcal{B}_i$ is the boundary condition operator; $\Omega \subset \mathbb{R}^D$ is the domain of PDE problem; and $\partial_i\Omega$ are the boundaries of the domain Ω ($1 \le i \le I$). Since the operators $\mathcal{L}$ and $\mathcal{B}_i$ usually have complicated forms, it is difficult to obtain the analytical solutions to PDEs. Instead, the numerical methods become the main manners for solving PDEs, *e.g.,* finite element method (FEM) [25], finite difference method (FDM) [20] and finite volume method (FVM) [10]. However, especially for complex PDEs, these methods often need high computational cost and extra expert knowledge, *e.g.,* the mesh generation in FEM, the setting of complicated boundaries in FDM, and the treatment of irregular geometries in FVM.

1.1 Physics-informed Neural Networks

Physics-informed neural networks (PINNs) provide a deep learning framework for numerically solving PDEs based on the view that if a neural network $\widehat{u}(\mathbf{x})$ holds the equation and the boundary conditions of the PDE problem (1), it will be an accurate approximation to the exact solution $u(\mathbf{x})$ [2, 17]. To guarantee the differentiability of $\widehat{u}(\mathbf{x})$, all activation functions in the network are set to be $\tanh(\cdot)$. The equation loss L_f and the i-th boundary-condition loss L_{b_i} are defined as follows:

$$\begin{array}{l} \mathrm{L}_f := \frac{1}{N} \sum_{n=1}^{N} \left\| \mathcal{L}[\widehat{u}](\mathbf{x}^{(n)}) - f(\mathbf{x}^{(n)}) \right\|^2, \\ \mathrm{L}_{b_i} := \frac{1}{M_i} \sum_{m=1}^{M_i} \left\| \mathcal{B}_i[\widehat{u}](\mathbf{x}_i^{(m)}) - g_i(\mathbf{x}_i^{(m)}) \right\|^2. \end{array}$$

Then, the weights of $\widehat{u}(\mathbf{x})$ can be empirically obtained by minimizing the physics-informed (PI) loss:

$$\mathrm{L}_{\mathrm{PINN}} := \mathrm{L}_f + \lambda_1 \mathrm{L}_b \ \ \text{with} \ \ \mathrm{L}_b := \sum_{i=1}^{I} \mathrm{L}_{b_i}, \tag{2}$$

where L_b is called the boundary-condition loss; $\lambda_1 > 0$ is a constant to adjust the proportion between the L_f and L_b; $\{\mathbf{x}^{(n)}\}_{n=1}^N$ are the points taken from the domain Ω; and $\{\mathbf{x}_i^{(m)}\}_{m=1}^{M_i}$ are the points taken from the boundaries $\partial_i\Omega$ of domain ($1 \le i \le I$). Take Poisson equation as an example:

$$\begin{cases} \Delta u = -\sin(\pi x)\sin(\pi y), & (x,y) \in \Omega; \\ u(x,0) = u(x,1) = 0, & (x,y) \in \partial_1\Omega; \\ u(0,y) = u(1,y) = 0, & (x,y) \in \partial_2\Omega, \end{cases} \tag{3}$$

where $\Omega = (0,1) \times (0,1)$, $\partial_1\Omega = (0,1) \times \{0,1\}$ and $\partial_2\Omega = \{0,1\} \times (0,1)$. Let $\widehat{u}(\mathbf{x})$ be the network providing numerical solutions to the PDE and its weights are obtained by minimizing

the PI loss $L_{PINN} := L_f + \lambda_1 (L_{b_1} + L_{b_2})$ with

$$L_f = \frac{1}{N} \sum_{n=1}^{N} \left[\Delta \widehat{u}(x^{(n)}, y^{(n)}) + \sin(\pi x^{(n)}) \sin(\pi y^{(n)}) \right]^2;$$

$$L_{b_1} = \frac{1}{M_1} \sum_{m=1}^{M_1} \left[\widehat{u}(x_1^{(m)}, 0) \right]^2 + \left[\widehat{u}(x_1^{(m)}, 1) \right]^2;$$

$$L_{b_2} = \frac{1}{M_2} \sum_{m=1}^{M_2} \left[\widehat{u}(0, y_2^{(m)}) \right]^2 + \left[\widehat{u}(1, y_2^{(m)}) \right]^2,$$

where $(x^{(n)}, y^{(n)})$ are taken from the domain Ω; $(x_1^{(m)}, 0)$ and $(x_1^{(m)}, 1)$ (resp. $(0, y_2^{(m)})$ and $(1, y_2^{(m)})$) are taken from the boundary $\partial_1 \Omega$ (resp. $\partial_2 \Omega$).

Different from the traditional PDE numerical solution methods, PINNs only need to build a network with an appropriate structure and then to determine the network weights by minimizing L_{PINN} rather than to consider the mathematical properties of the PDEs. Because of the high applicability, PINNs have been widely used to solve a variety of PDEs, *e.g.,* the Euler equations for high-speed aerodynamic flows simulation [12] and the Fokker-Planck equation for inverse stochastic problems from discrete particle observations [1]. Moreover, Wang et al. [21] adopted the neural tangent kernel method to study the reason why PINNs sometimes cannot be trained. Shin et al. [18] studied the convergence of generalization error of PINNs for linear second-order elliptic and parabolic PDEs. Mishra & Molinaro [14] estimated the generalization error of PINNs based on the training error and the size of training samples. In addition, there are also some variants of PINNs, such as MPINN for multi-fidelity surrogate modeling [13] and B-PINNs for quantifying the aleatoric uncertainty arising from the noisy data [22].

However, there are some caveats in the applications of PINNs:

(1) To maintain a strong nonlinear mapping capability, the network $\widehat{u}(\mathbf{x})$ usually has multiple hidden layers and a large amount of nodes in each hidden layer. Such a complicated structure significantly increases the training difficulty.

(2) Different from the general supervised learning setting, the sample points used to train PINNs are only taken from the domain Ω and its boundaries $\partial_i \Omega$. There is no labeled sample of the form $(\mathbf{x}^{(n)}, u(\mathbf{x}^{(n)}))$ $(\mathbf{x}^{(n)} \in \Omega)$ to correct the network outputs.[1] Therefore, the process of training PINNs often becomes unstable and hard to control.

(3) Let $\{(\mathbf{x}_T^{(j)}, u_T^{(j)})\}_{j=1}^{J}$ be a set of labeled samples, which are the exact solutions $u_T^{(j)}$ to the PDE at the points $\mathbf{x}_T^{(j)}$. This situation is common in practice. For example, some sensors are equipped on the surface of a physical objective to capture its real-time physical responses, or some numerical methods can provide high-accuracy approximations of the exact solutions at some points in the domain. Following the mechanism of PINNs, a natural way of exploiting these labeled samples is to minimize the objective function

$$\overline{L}_{PINN} := L_{PINN} + \lambda_2 L_T, \tag{4}$$

---

[1]As addressed by Raissi et al. [17], in some cases, the boundary-condition loss L_b in PINNs can be replaced with the mean squared loss computed on a small size of labeled boundary samples $\{(\mathbf{x}_i^{(m)}, u(\mathbf{x}_i^{(m)}))\}_{m=1}^{M_i}$ $(\mathbf{x}_i^{(m)} \in \partial_i \Omega)$.

where $L_T = \frac{1}{J} \sum_{j=1}^{J} \|\widehat{u}(\mathbf{x}_T^{(j)}) - u_T^{(j)}\|^2$. Unfortunately, when the labeled sample size J is small, the minimization of L_T is likely to bring a negative effect to the training performance provided by the minimization of L_{PINN}.

1.2 Generative Adversarial Networks

Generative adversarial networks (GANs), proposed by Goodfellow et al. [5], are referred to a class of neural networks that are composed of two sub-networks: the generator and the discriminator, and are trained in the manner of minimax game between them. Taking advantage of the specific network structure with the minimax game training, many empirical evidences have shown that GANs can approximate the data distribution accurately with a lower demand on the sample size than the traditional machine-learning models. This characteristic provides an applicable way of exploiting a limited size of labeled samples to refine the networks. For example, some variants of GANs have been successfully used to deal with multi-fidelity surrogate modeling [11] and the TBM tunnel geological prediction [24], where only a small size of labeled samples are available. Such a characteristic motivates this paper as well.

1.3 Overview of Main Results

In this paper, we mainly concern with two issues on PINNs: one is how to improve the training efficiency and the other is how to exploit a small size of (usually very few) labeled samples to refine the networks.

Inspired from the weighting strategy of the Adaboost method [4], we introduce a point-weighting (PW) method to improve the training performance of PINNs, where the weight of every sample point is adaptively updated at each training iteration. This method splits the process of training PINNs into two stages. In the first stage, since the network is relatively fragile, the weights of points that provide small (resp. big) training errors are increased (resp. decreased) to stabilize the network as quickly as possible. In the second stage, the weights of the points that have big (resp. small) training errors are increased (resp. decreased) to make the network fit all sample points accurately. The numerical experiments show that the proposed PW method not only speeds up the training process but also improves the network performance.

We then consider the extension of PINNs in the situation that there are a small size of (usually very few) labeled samples, *i.e.,* the exact solutions or their high-accuracy approximations at some discrete points in the domain. As stated above, it is difficult for the mechanism of PINNs to handle this situation. Instead, by integrating the generative adversarial (GA) mechanism with the structure of PINNs, we propose the generative adversarial physics-informed neural networks (GA-PINNs) to numerically solve PDEs in this situation.

The GA-PINN is composed of two sub-networks: a generator and a discriminator. The generator is a PINN, *i.e.,* a network that, given an input $\mathbf{x} \in \Omega$, produces the approximation $\widehat{u}(\mathbf{x})$ of the corresponding exact solution $u(\mathbf{x})$ to the PDE at the point $\mathbf{x}$. The discriminator aims to identify whether the pair $(\mathbf{x}, \widehat{u}(\mathbf{x}))$ is a real exact solution to the PDE. Similar to the classical GANs, the weights of GA-PINNs are determined by using the minimax game training between the generator and the discriminator. When they reach the Nash equilibrium, *i.e.,* the discriminator is trained to be incapable of identifying whether $(\mathbf{x}, \widehat{u}(\mathbf{x}))$ is a real exact solution, the generator will approximate the exact solution $u(\mathbf{x})$ accurately. However, since the labeled samples are insufficient, the generator outputs could be far away from the exact solutions to the

PDE, even though the discriminator has be trained to be capable of identifying the fake samples. To overcome this shortcoming, at each training iteration, we further minimize the PI loss L_{PINN} (or the weighted PI loss L_{PINN}^{PW} given in (9)) to fine-tune the generator weights after the minimax game training. The experimental results support the effectiveness of the proposed GA-PINNs and show that GA-PINNs outperform PINNs in many well-known PDEs.

1.4 Related Works

The universal approximation theory, given by Hornik et al. [7], provides a theoretical guarantee for using the neural networks to numerically solve PDEs. Lagaris et al. [9] expressed the solution to a PDE as a summation of two parts. The first part can be directly obtained to satisfy the boundary conditions in some simple situations. The second part is the output of a neural network that is trained by only minimizing the equation loss L_f. In PINNs, the boundary conditions are also encoded into the network and the network weights are obtained by minimizing the PI loss L_{PINN} [2, 17].

Yu et al. [23] proposed the deep ritz method for using neural networks to solve the variational problem

$$\min_{u \in H} I(u) := \int_\Omega \left(\frac{1}{2} |\nabla u(x)|^2 - f(x)u(x) \right) \mathrm{d}x,$$

where the integral $I(u)$ will be discretized by selecting some points from Ω. Since some PDEs can be converted into the equivalent variational forms, the deep ritz method can also be used to solve these PDEs. However, it is still challenging to treat some specific boundary conditions of PDEs.

As addressed by Dissanayake & Phan-Thien [2], Raissi et al. [17], the weights of PINNs are obtained by using the batch gradient descent (BDG) method, where the weights are updated by taking the derivative of the loss function computed on all samples. Instead, Sirignano & Spiliopoulos [19] proposed the deep Galerkin method (DGM) that adopts the stochastic gradient descent (SGD) to update the weights of PINNs. In each iteration, the objective function is computed on a mini-batch of points taken from Ω and $\partial_i\Omega$ randomly. Taking advantage of the randomness brought from the SGD method, DGM could be more suitable to numerically solving the high-dimensional PDEs than the classical PINNs. In this paper, PINNs are still referred to the PINNs with the BDG method if no confusion arises.

The rest of this paper is organized as follows. In Section 2, we introduce the PW method to improve the training performance of PINNs, In Section 3, we show the structure of GA-PINNs and the minimax game training. The numerical experiments are arranged in Section 4 and the last section concludes the paper.

2 Point-weighting Method

As shown in Tab. 5, the main difficulty of training PINNs lies in the sharp fluctuation appearing in the minimization of the equation loss L_f and the boundary-condition losses L_{b_i}. To improve the training performance of PINNs, we introduce a point-weighting (PW) method to speed up the process of minimizing L_f and L_{b_i}.

Consider the i-th boundary condition $\mathcal{B}_i[u](\mathbf{x}) = g_i(\mathbf{x})$ ($\mathbf{x} \in \partial_i\Omega$). Let $\{\mathbf{x}_i^{(m)}\}_{m=1}^{M_i}$ be a set of points taken from the boundary $\partial_i\Omega$, and let $e_i > 0$ be the desired level of the boundary-condition

error. Define a binary surrogate function

$$\beta(\mathbf{x}_i^{(m)}; e_i) = \begin{cases} +1, & \left\|\widehat{u}\big(\mathbf{x}_i^{(m)}\big) - u\big(\mathbf{x}_i^{(m)}\big)\right\|^2 \le e_i; \\ -1, & \left\|\widehat{u}\big(\mathbf{x}_i^{(m)}\big) - u\big(\mathbf{x}_i^{(m)}\big)\right\|^2 > e_i, \end{cases}$$

which signifies whether the point $\mathbf{x}_i^{(m)}$ satisfies the accuracy requirement of the i-th boundary condition. For convenience of presentation, the points leading to $\beta(\mathbf{x}_i^{(m)}; e_i) = 1$ are called easy-to-learn (EL) points, and otherwise called hard-to-learn (HL) points

The PW method splits the process of minimizing the boundary-condition loss L_{b_i} into two stages: at the first stage, the weights of EL points are increased to stabilize the network as quickly as possible; and at the second stage, the weights of HL points are increased to make the network fit these HL points and then to improve the network performance.

Accordingly, the i-th weighted boundary-condition loss at the k-th ($k \in \mathbb{N}$) iteration is defined as:

$$\mathrm{L}_{b_i,k}^{\mathrm{PW}} = \sum_{m=1}^{M_i} \omega_{i,k}^{(m)} \left\| \mathcal{B}_i[\widehat{u}]\big(\mathbf{x}_i^{(m)}\big) - g_i\big(\mathbf{x}_i^{(m)}\big) \right\|^2, \tag{5}$$

where $\omega_{i,k}^{(m)}$ is the point weight of $\mathbf{x}_i^{(m)}$ at the k-th iteration with $\sum_{m=1}^{M_i} \omega_{i,k}^{(m)} = 1$. When $k = 0$, it is clear that the initial weights are $\omega_{i,0}^{(m)} = \frac{1}{M_i}$.

Borrowing the idea from the weighting strategy of the Adaboost method [4], after each iteration of training the network, the point weight $\omega_{i,k}^{(m)}$ is updated in the following way:

$$\begin{cases} \rho_{i,k} := \displaystyle\sum_{\{m:\beta(\mathbf{x}_i^{(m)};e_i)=-1\}} \omega_{i,k}^{(m)}; \\[2ex] \alpha_{i,k} := q_i \log \frac{1-\rho_{i,k}}{\rho_{i,k}}; \\[2ex] \omega_{i,k+1}^{(m)} := \frac{\omega_{i,k}^{(m)} \cdot \exp(-\alpha_{i,k} \cdot \beta(\mathbf{x}_i^{(m)};e_i))}{\sum_{m=1}^{M_i} \omega_{i,k}^{(m)} \cdot \exp(-\alpha_{i,k} \cdot \beta(\mathbf{x}_i^{(m)};e_i))}, \end{cases} \tag{6}$$

where $\rho_{i,k}$ is the summation of the HL-point weights; the hyperparameter q controls the magnitude of updating the weight $\omega_{i,k}^{(m)}$. It is obvious that $\rho_{i,k} = 0.5$ is the borderline between the afore-mentioned two stages. Moreover, we also introduce an extra hyperparameter ϵ for the terminal condition: when $1 - \rho_{i,k} \le \epsilon$ is achieved, we deem that there has been no HL point left and then terminate the training.

Similarly, given a point set $\{\mathbf{x}^{(n)}\}_{n=1}^N \subset \Omega$, define the weighted equation loss at the k-th iteration ($k \in \mathbb{N}$) as

$$\mathrm{L}_{f,k}^{\mathrm{PW}} = \sum_{n=1}^N \omega_k^{(n)} \big[\mathcal{L}[\widehat{u}]\big(\mathbf{x}^{(n)}\big) - f\big(\mathbf{x}^{(n)}\big) \big]^2 \tag{7}$$

with $\sum_{n=1}^N \omega_k^{(n)} = 1$. Then, given a desired level of equation error e, the point weight $\omega_k^{(n)}$ of $\mathbf{x}^{(n)}$ is updated as follows:

$$\begin{cases} \rho_k := \displaystyle\sum_{\{n:\beta(\mathbf{x}^{(n)};e)=-1\}} \omega_k^{(n)}; \\[2ex] \alpha_k := q \log \frac{1-\rho_k}{\rho_k}; \\[2ex] \omega_{k+1}^{(n)} := \frac{\omega_k^{(n)} \cdot \exp(-\alpha_k \cdot \beta(\mathbf{x}^{(n)};e))}{\sum_{n=1}^N \omega_k^{(n)} \cdot \exp(-\alpha_k \cdot \beta(\mathbf{x}^{(n)};e))}. \end{cases} \tag{8}$$

Interestingly, as shown in Tab. 1, the PW method cannot influence the smooth minimization process. Therefore, the PW method has a good applicability in practice, and can be directly used to train PINNs. Accordingly, the loss function for PINN training with the PW method (called PINN+PW) is expressed as follows:

$$L_{PINN}^{PW} := L_{f,k}^{PW} + \lambda \sum_{i=1}^{I} L_{b_i,k}^{PW}. \tag{9}$$

3 Generative Adversarial Physics-informed Neural Networks

Let $\mathcal{S} := \{\mathbf{x}^{(n)}\}_{n=1}^{N}$ and $\mathcal{S}_i := \{\mathbf{x}_i^{(m)}\}_{m=1}^{M_i}$ $(1 \leq i \leq I)$ be the sets of points taken from the domain and its boundary $\partial_i\Omega$, respectively. We also consider a small size of labeled samples $\mathcal{S}_T := \{(\mathbf{x}_T^{(j)}, u_T^{(j)})\}_{j=1}^{J}$, where $\mathbf{x}_T^{(j)}$ are the points taken from the domain Ω and $u_T^{(j)}$ are the exact solutions to the PDE at $\mathbf{x}_T^{(j)}$ with $J \ll N, M$.

As stated above, since the labeled samples are insufficient, the minimization of (4) not only fails to refine PINNs but also influences the training performance provided by the minimization of L_{PINN}. To overcome this shortcoming, we integrate the GA mechanism with the structure of PINNs to form the proposed GA-PINNs. Moreover, in view of the difficulty of training GA-PINNs (it is still challenging to efficiently train individual PINNs or GANs in the literature), we also adopt the PW method to improve the training performance of GA-PINNs.
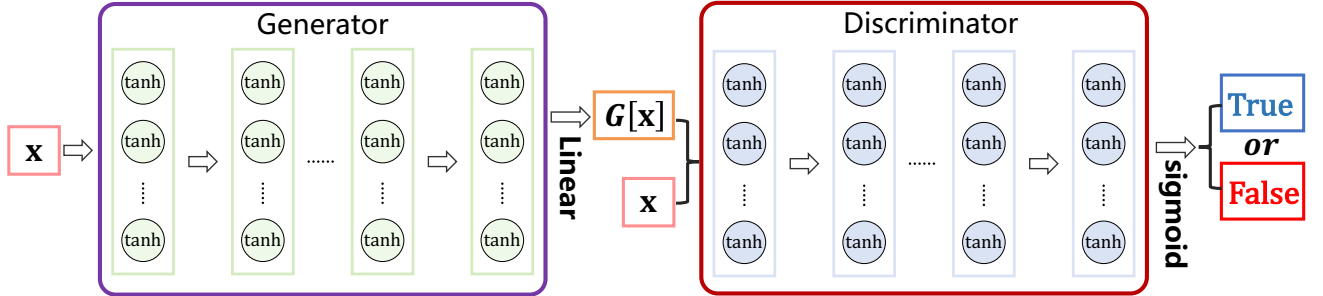

Figure 1: The structure of GA-PINNs.

As shown in Fig. 1, the generator of GA-PINNs inherits the structure of PINNs, where all hidden nodes are activated by using the $\tanh(\cdot)$ functions and the output nodes are activated by using the linear function $y = x$. Its input $\mathbf{x}$ is a point in the domain Ω and its output $G[\mathbf{x}]$ is the approximation of the exact solution $u(\mathbf{x})$ to the PDE. The input of the discriminator is either the pair $(\mathbf{x}_T^{(j)}, G[\mathbf{x}_T^{(j)}])$ or the labeled sample $(\mathbf{x}_T^{(j)}, u_T^{(j)})$. The discriminator output $D[(\mathbf{x}, G[\mathbf{x}])]$, activated by using the sigmoid function, is set as a 0-1 node to identify whether the input pair $(\mathbf{x}_T^{(j)}, G[\mathbf{x}_T^{(j)}])$ is an exact solution to the PDE (labeled as "1") or not (labeled as "0").

The following are the generative loss and the discriminative loss without the logarithmic

operation, respectively:

$$L_D = \frac{1}{J} \sum_{j=1}^{J} \left(1 - D[(\mathbf{x}_T^{(j)}, u_T^{(j)})]\right) + D[(x_L^{(j)}, G[x_L^{(j)}])],$$

$$L_G = L_T + \frac{1}{J} \sum_{j=1}^{J} \left(1 - D[(x_T^{(j)}, G[x_T^{(j)}])]\right).$$

The generator weights $\mathbf{W}_G$ and the discriminator weights $\mathbf{W}_D$ are updated by minimizing L_G and L_D, respectively. Since the labeled samples $(\mathbf{x}_T^{(j)}, u_T^{(j)})$ are insufficient, even if the discriminator is trained to be capable of identifying that the pair $(x_T^{(j)}, G[x_T^{(j)}])$ is a fake labeled sample, the generator could not provide satisfactory approximations of the exact solutions to the PDE. Therefore, at each training iteration, we will minimize L_{PINN} (or $L_{\text{PINN}}^{\text{PW}}$) to fine-tune the generator weights $\mathbf{W}_G$ so as to correct the generator outputs after the minimax game between the generator and the discriminator.

Moreover, since the discriminator is responsible to guide the behavior of generator, we first update the discriminator weights and then update the generator weights in order. Another important thing is the ratio of the learning rates η_G and η_D for minimizing the generative loss L_G and the discriminator loss L_D, respectively. Here, we set $\eta_G : \eta_D = 1 : 5$ to raise the degree of training the discriminator so as to maintain a powerful guidance to the generator. The workflow of training GA-PINN is given in Alg. 1, and it also illustrates the workflow of training GA-PINN with PW method (called GA-PINN+PW) by replacing $L_{\text{PINN}}^{\text{PW}}$ with L_{PINN} in Line 5:

Algorithm 1 The process of training GA-PINN (resp. GA-PINN+PW)

Input: $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$, $\{\mathbf{x}_i^{(m)}\}_{m=1}^{M_i}$, $\{(\mathbf{x}_T^{(j)}, u_T^{(j)})\}_{j=1}^{J}$, e, e_i, η_D, η_G, K, q, q_i, ϵ;
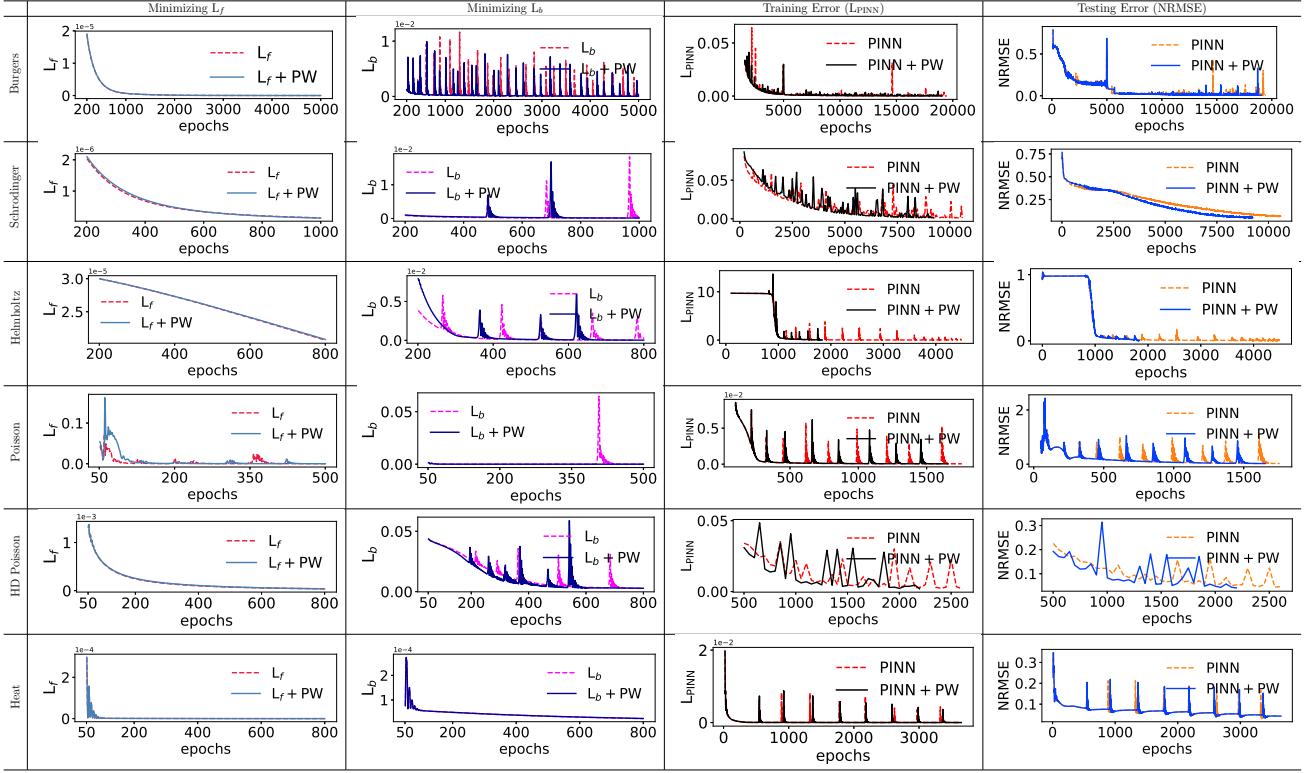**Output:** $\mathbf{W}_G$, $\mathbf{W}_D$;
1: Initialize $\mathbf{W}_G^{(0)}$, $\mathbf{W}_D^{(0)}$; let $\omega_{i,0}^{(m)} = \frac{1}{M_i}$ and $\omega_0^{(n)} = \frac{1}{N}$;
2: **for all** $k = 1, 2, \cdots, K$; **do**
3:    Update $\mathbf{W}_D$ by minimizing the discriminative loss L_D: $\mathbf{W}_D^{(k+1)} = \mathbf{W}_D^{(k)} - \eta_D \cdot \frac{\partial L_D}{\partial \mathbf{W}_D}$;
4:    Update $\mathbf{W}_G$ by minimizing the generative loss L_G: $\mathbf{W}_G^{(k,1)} = \mathbf{W}_G^{(k)} - \eta_G \cdot \frac{\partial L_G}{\partial \mathbf{W}_G}$;
5:    Update $\mathbf{W}_G^{(k,1)}$ by minimizing L_{PINN}: $\mathbf{W}_G^{(k+1)} = \mathbf{W}_G^{(k,1)} - \eta_G \cdot \frac{\partial L_{\text{PINN}}}{\partial \mathbf{W}_G}$ (resp. by minimizing $L_{\text{PINN}}^{\text{PW}}$: $\mathbf{W}_G^{(k+1)} = \mathbf{W}_G^{(k,1)} - \eta_G \cdot \frac{\partial L_{\text{PINN}}^{\text{PW}}}{\partial \mathbf{W}_G}$).
6: **end for**

---

4 Numerical Experiments

In this section, we conduct the numerical experiments to verify the effectiveness of the proposed PW method and GA-PINNs for numerically solve PDEs in the situation that a small size of labeled samples are available. We mainly concern with the following issues: 1) whether the PW method can speed up the process of training a PINN and improve its performance; and 2) whether the proposed GA-PINNs outperform PINNs in such a situation.

All experiments are processed in the DELL PowerEdge T640 Tower Server with two Intel Xeon 20-core processors, 128 GB RAM and a NVIDIA Tesla V100 16GB GPU. We select six well-known PDEs as the testing problems, including Burgers equation, Poisson equation, Helmholtz

Table 1: Examining the effect of the PW method for different testing PDEs

equation, Schrodinger equation, High-dimensional (HD) Poisson equation and Heat equation (*cf.* Tab. 2). We consider the following models for numerically solving these PDEs, including PINN, PINN+PW, GA-PINN and GA-PINN+PW. Given a testing data $\{(\mathbf{x}^{(l)}, u^{(l)})\}_{l=1}^L$, we adopt the normalized root mean square error (NRMSE) as the criterion of testing performance:

$$\text{NRMSE} := \sqrt{\frac{\sum_{l=1}^L \left\| \widehat{u}(\mathbf{x}^{(l)}) - u^{(l)} \right\|^2}{\sum_{l=1}^L \left\| u^{(l)} \right\|^2}}.$$

For the sake of fairness, in each testing PDE problem, PINN, PINN+PW and the generators of GA-PINN and GA-PINN+PW share the same structure and initial weights that are obtained by using the Xavier initialization [6]. Their objective functions are minimized by using Adam optimization algorithm [8]. Following the setting of PINNs given by Raissi et al. [17], the Adam method is of the batch form, *i.e.,* all samples are used to compute the objective functions at each iteration. Different from the ordinary supervised learning tasks, where the labeled samples have contained the relationship between the input and the output, the training of PINNs is achieved by minimizing the PI loss L_{PINN} that is computed on the points taken from the domain and its boundaries. Therefore, when using the Adam method with a small mini-batch, the incomplete physics information provided by the PI loss L_{PINN} computed on the insufficient points could misguide the updating direction of network weights. Moreover, since the size of labeled samples is small, the mini-batch Adam method is not be used to optimize the generative loss L_G and the discriminative loss L_D as well.

The termination condition of training these models is that the PI loss L_{PINN} computed on

Table 2: Six testing PDE problems

PDE	Main Equation	Boundary Condition
Burgers	$u_t + uu_x - (0.01/\pi)u_{xx} = 0,\ (x,t) \in [-1,1] \times [0,1]$	$\begin{cases} u(0,x) = -\sin(\pi x); \\ u(t,-1) = u(t,1) = 0. \end{cases}$
Poisson	$\Delta u = -\sin(\pi x)\sin(\pi y),\ (x,y) \in (0,1) \times (0,1)$	$\begin{cases} u(x,y) = 0,\ (x,y) \in \{0,1\} \times (0,1); \\ u(x,y) = 0,\ (x,y) \in (0,1) \times \{0,1\}. \end{cases}$
Helmholtz	$\Delta u + k^2 u = 0,\ (x,y) \in (0,1) \times (0,1)$	$\begin{cases} u(x,y) = \sin(kx),\ (x,y) \in \{0,1\} \times (0,1); \\ u(x,y) = \sin(kx),\ (x,y) \in (0,1) \times \{0,1\}. \end{cases}$
Schrodinger	$ih_t + 0.5h_{xx} + |h|^2 h = 0,\ (x,t) \in [-5,5] \times [0,\pi/2]$	$\begin{cases} h(0,x) = 2\,\text{sech}(x), \\ h(t,-5) = h(t,5), \\ h_x(t,-5) = h_x(t,5), \end{cases}$
HD Poisson	$-\Delta u = 0,\ \boldsymbol{x} \in (0,1)^{10}$	$u(\boldsymbol{x}) = x_1^2 - x_2^2 + x_3^2 - x_4^2 + x_5 x_6 + x_7 x_8 x_9 x_{10},$ $\boldsymbol{x} \in \partial_i(0,1)^{10},\ 1 \le i \le 10;$
Heat	$u_t - \Delta u = 0,\ (x,y,t) \in \mathbb{R}^3$	$u(x,y,0) = x - y,\ (x,y) \in \mathbb{R}^2.$

[1] In HD Poisson equation, $\boldsymbol{x} \in \partial_i(0,1)^{10}$ means that $\boldsymbol{x} \in \underbrace{(0,1) \times (0,1)}_{i-1} \times \{0,1\} \times \underbrace{(0,1) \times (0,1)}_{10-i}$.

[2] Helmholtz equation, Poisson equation, HD Poisson equation and heat equation have the analytic solutions: $u(x,y) = \sin(kx)$, $u(x,y) = \frac{1}{2\pi^2}\sin(\pi x)\sin(\pi y)$, $u(\boldsymbol{x}) = x_1^2 - x_2^2 + x_3^2 - x_4^2 + x_5 x_6 + x_7 x_8 x_9 x_{10}$ and $u(x,y,t) = \frac{1}{(4\pi t)}\int_{\mathbb{R}^2} e^{-\frac{(x-z_1)^2+(y-z_2)^2}{4t}}(z_1 - z_2)dz_1 dz_2$, respectively. The testing datasets of the three PDEs are obtained by using Latin hypercube sampling on their domains, respectively.

[3] The testing datasets of Burgers equation and Schrodinger equation are given in the PINN software package [16].

Table 3: The choices of hyperparameters in the numerical experiments

PDE	q_1	e_1	q_2	e_2	TC (L_{PINN})	N	M	J	η_G	η_P	η_D	HL_G	HL_D
Burgers	1×10^{-4}	0.02	1×10^{-4}	5×10^{-4}	1×10^{-4}	10000	100	10	0.001	0.001	0.005	L: 7, N: 20	L: 8, N: 20
Schrodinger	0.005	5×10^{-4}	0.005	1×10^{-4}	0.001	20000	100	10	0.001	0.001	0.005	L: 4, N: 100	L: 3, N: 100
Helmholtz	6×10^{-5}	5×10^{-4}	6×10^{-5}	5×10^{-4}	0.01	20000	200	3	0.001	1×10^{-5}	5×10^{-5}	L: 4, N: 100	L: 1, N: 100
Poisson	5×10^{-5}	5×10^{-6}	5×10^{-5}	5×10^{-6}	5×10^{-5}	5000	100	5	0.001	1×10^{-6}	5×10^{-6}	L: 4, N: 100	L: 1, N: 100
HD Poisson	0.001	0.05	$\begin{array}{c} q_1 = q_2 = \\ \cdots = q_{10} \end{array}$	$\begin{array}{c} e_1 = e_2 = \\ \cdots = e_{10} \end{array}$	0.002	10000	500	100	0.001	0.001	0.005	L: 4, N: 100	L: 1, N: 100
Heat	5×10^{-5}	5×10^{-6}	5×10^{-5}	5×10^{-6}	5×10^{-6}	5000	100	10	0.001	0.001	0.005	L: 4, N: 100	L: 1, N: 100

[1] q_1, q_2 and e_1, e_2 are the hyperparameters used in the PW method (cf. (6)). In HD Poisson, the hyperparameters q_i (resp. e_i) ($1 \le i \le 10$) share the same value $q_1 = \cdots = q_{10} = 0.001$ (resp. $e_1 = \cdots = e_{10} = 0.05$). Especially, we set $q = 5 \times 10^{-5}$ and $e = 0.001$ in (8) for Poisson equation.

[2] TC (L_{PINN}) stands for the termination condition that is the loss L_{PINN} computed on the training data reaches the predefined level.

[3] N and M are the sizes of training data taken from a domain and its boundaries, respectively. Following the suggestions provided by Raissi et al. [17], we adopt a relatively size number N_b to maintain the stability of training process. Moreover, J is the size of labeled samples.

[4] η_G, η_D and η_P are the learning rates for minimizing the generative loss L_G, the discriminator loss L_D an the loss L_{PINN}, respectively.

[5] HL_G (resp. HL_D) stands for the hidden layers of the generator (resp. discriminator), where "L: 7, N: 20" means that there are 7 hidden layers and each hidden layer has 20 nodes.

Table 4: The epoch numbers and testing NRMSEs of the four models when the training process satisfies the termination conditions.

PDE Method	Burgers		Schrodinger		Helmholtz		Poisson		HD Poisson		Heat	
	epoch	NRMSE	epoch	NRMSE	epoch	NRMSE	epoch	NRMSE	epoch	NRMSE	epoch	NRMSE
PINN	19475	6.39×10^{-3}	10593	0.0669	4494	4.87×10^{-3}	1758	0.0242	2608	0.0435	3640	0.0432
PINN+PW	19068	5.89×10^{-3}	9235	0.0523	1834	$\underline{3.62 \times 10^{-3}}$	**1654**	$\underline{0.0230}$	**2195**	0.0424	3649	0.0428
GA-PINN	**17585**	$\underline{4.69 \times 10^{-3}}$	6080	$\underline{0.0122}$	2433	4.36×10^{-3}	3704	0.0255	4354	0.0469	**1847**	0.0099
GA-PINN+PW	18000	4.81×10^{-3}	**5798**	0.0156	**1800**	4.74×10^{-3}	3475	$\underline{0.0238}$	2909	0.0399	2047	$\underline{0.0089}$
DGM	50000	0.0265	20000	0.2266	8000	6.28×10^{-3}	6000	0.1025	6000	$\underline{0.0317}$	6000	0.0181

Table 5: Heat maps of the discrepancies between the exact solutions and the numerical solutions provided by different models
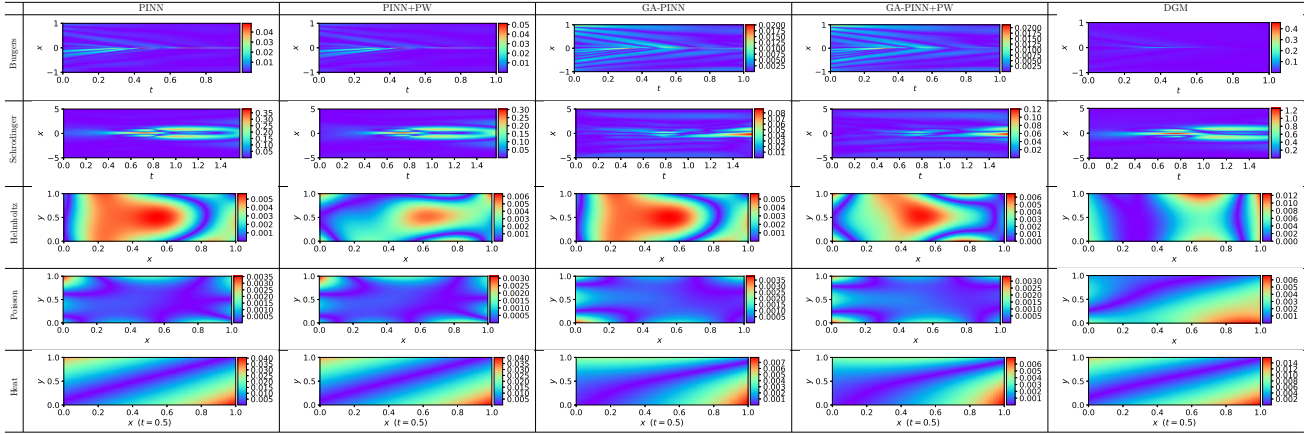
the sample points reaches the pre-defined level, and meanwhile we use L_{PINN} as the criterion of training performance as well. The reason why to set such a termination condition is because a well-trained network should be able to recover the complete physics information, and thus L_{PINN} actually acts as a criterion to evaluate the completeness of physics information. The choices of the hyperparameters are listed in Tab. 3.

As a comparison, we also adopt the deep Galerkin method (DGM) [19] to solve these testing PDEs. As addressed in Section 1.4, since DGM actually is a PINN with the SGD method, DGM share the same learning rate η_P and network structure with the PINN for each testing PDE problem. In DGM, the sizes of the mini-batches for L_f and L_b are both set to be 256 so as to maintain the relatively complete physics information and the points are randomly drawn from the domain and its boundaries at each iteration, respectively. The termination condition of training is set to reach the pre-defined iteration epoch (*cf.* Tab. 4).
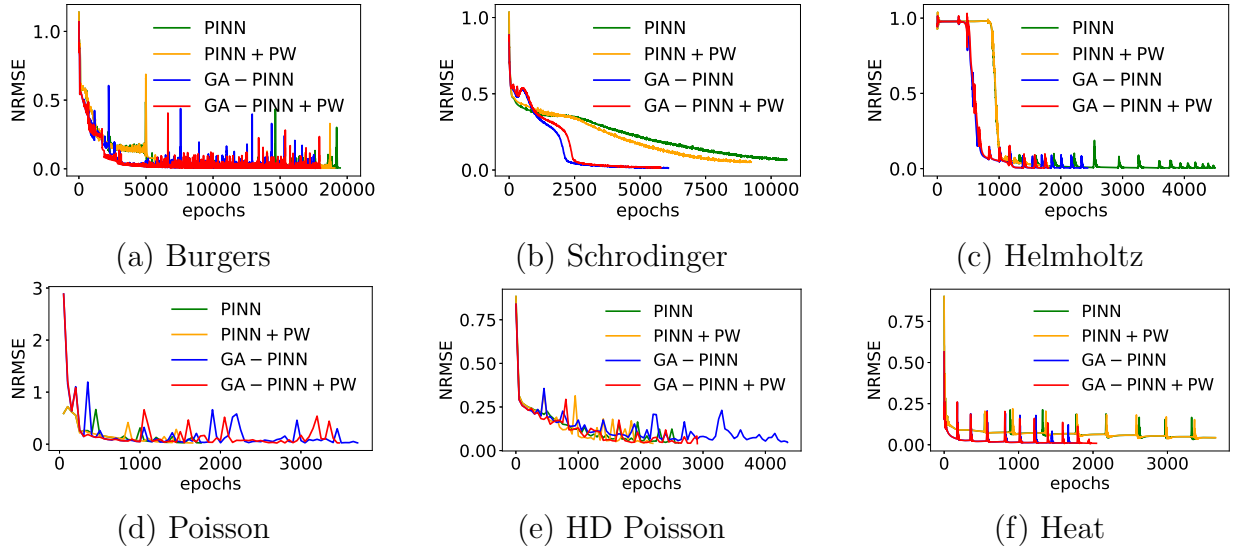

Figure 2: The curves of testing error (NRMSE) for different testing PDEs.

11

4.1 Experimental Results for PW Method

Here, we consider the effect of the PW method in the training process of PINNs. For each testing PDE problem, we first study the minimization process of L_f and L_b with or without the PW method, and then analyze the training process of PINN and PINN+PW. As shown in the first two columns of Tab. 1, the PW method can speed up the minimization process of L_f and L_b when they have sharp fluctuation, but do not evidently influence the smooth minimization process of the objective functions, *e.g.*, the L_b of Heat and the L_fs of Burgers, Helmholtz, Schrodinger, HD Poisson and Heat. Therefore, in the following experiments, we only apply the PW method to speed up the minimization process of L_b in the five equations, and the PW method is used to minimize both of L_f and L_b in the process of training PINN+PW for Poisson equation.

As shown in the last two columns of Tab. 1 and the first two rows of Tab. 4, PINN+PW not only reach the training termination condition earlier than PINN, but also provides a lower testing performance (NRMSE) in most cases. It implies that the introduction of the PW method can evidently improve the performance of PINNs. Interestingly, we also find that the effect of the PW method is not evident to the minimization of L_f in Poisson equation, but the training process of PINN+PW is much faster than that of PINN with a lower testing NRMSE.

4.2 Experimental Results for GA-PINNs

GA-PINNs aim to use a small size of (usually very few) labeled samples, which are the exact solutions (or their high-accuracy approximations) to a PDE at some discrete points in the domain Ω, to refine the numerical solutions to the PDE provided by PINNs. At each iteration of training GA-PINN, after minimizing the discriminative loss L_D and the generative loss L_G, the PI loss L_{PINN} (or the weighted PI loss L_{PINN}^{PW}) is minimized to fine-tune the generator weights that could be misguided by the discriminator trained based on insufficient labeled samples. To further examine the effectiveness of the PW method, we also compare the performance of GA-PINN with that of GA-PINN+PW (*cf.* Alg. 1).

Figure 2 and Tables 4∼5 show that the introduction of the GA mechanism can effectively exploit the small size of labeled samples to speed up the training process and improve the accuracy of the numerical solutions. However, the PW method sometimes will bring a negative effect to the process of training GA-PINNs in some complicated PDEs such as Burgers equation and Schrodinger equation. The reason is that the introduction of the PW method will influence the minimax game training, which could become fragile in the case that the PDEs are complicated but the labeled samples are not sufficient. In addition, PINN+PW performs comparably with GA-PINN (or GA-PINN+PW) in the relatively simple problems, such as Helmholtz, Poisson and HD Poisson. This finding implies that PINN+PW is a good candidate for numerically solving simple PDEs. Moreover, benefiting from the randomness of the SGD method, DGM has a good performance in HD Poisson but needs a relatively large epoch number and mini-batch size.

5   Conclusion

In this paper, we consider the numerical solutions to PDEs in the situation that the exact solutions (or their high-accuracy approximations) to PDEs are available at a small amount of (usually very few) discrete points in the domain. This situation is common in practice, but the mechanism of PINNs is unsuitable (at least cannot be directly applied) to this situation (*cf.* Section 1.1). To

overcome this limitation, we integrate the GA mechanism with the structure of PINNs to form the proposed GA-PINNs.

The GA-PINN is composed of two sub-networks: a generator and a discriminator. The generator, with the same structure as PINNs, produces the numerical solution $\widehat{u}(\mathbf{x})$ to a PDE associated with the input point $\mathbf{x} \in \Omega$. The discriminator aims to identify whether the pair $(\mathbf{x}, \widehat{u}(\mathbf{x}))$ is a real exact solution to the PDE. The training process of GA-PINNs is achieved by using a small size of labeled samples $\{(\mathbf{x}_T^{(j)}, u_T^{(j)})\}_{j=1}^J$. Each iteration of training GA-PINNs contains two stages: one is the minimax game between the generator and the discriminator and the other is the minimization of $\mathrm{L_{PINN}}$ (or $\mathrm{L_{PINN}^{PW}}$) to refine the generator after the minimax game training. The goal of the second stage is to fine-tune the generator weights influenced by the insufficient labeled samples. Taking advantage of the GA mechanism, GA-PINNs are able to effectively exploit the insufficient labeled samples to improve the accuracy of numerical solutions to PDEs. The experimental results support the effectiveness of GA-PINNs and show that GA-PINNs outperform PINNs in these testing PDE problems.

Since the efficiency of training PINNs often becomes low especially for some complicated PDEs, we also propose the PW method to speed up the process of training PINNs. This method splits the training process into two stages: one is to promote the network to reach a relatively stable status by increasing the EL-point weights; and the other is to improve the network performance by increasing the HL-point weights. The numerical experiments show that the PW method can evidently improve the efficiency of training PINNs with a lower testing performance. Interestingly, the PW method cannot influence the smooth minimization process, but only speeds up the minimization process that has the sharp fluctuation. Therefore, the PW method has high applicability in practice. Moreover, we also introduce the PW method into the process of training GA-PINNs (called GA-PINN+PW accordingly) (*cf.* Alg. 1), and the experimental results demonstrate that GA-PINN+PW performs better than GA-PINN in most cases. In the further works, we will consider the feasibility of the PW method in training other deep learning models such as AlexNet, VGGNet and ResNet, and use GA-PINNs to numerically solve some important PDEs such as Navier-Stokes equations.

References

[1] Chen, X., Yang, L., Duan, J., and Karniadakis, G. E. Solving inverse stochastic problems from discrete particle observations using the fokker–planck equation and physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(3):B811–B830, 2021.

[2] Dissanayake, M. and Phan-Thien, N. Neural-network-based approximations for solving partial differential equations. *communications in Numerical Methods in Engineering*, 10(3): 195–201, 1994.

[3] Ershkov, S. V., Prosviryakov, E. Y., Burmasheva, N. V., and Christianto, V. Towards understanding the algorithms for solving the navier–stokes equations. *Fluid Dynamics Research*, 53(4):044501, 2021.

[4] Freund, Y. and Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[5] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[6] He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[7] Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[8] Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Lagaris, I. E., Likas, A., and Fotiadis, D. I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[10] LeVeque, R. J. et al. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

[11] Liu, L., Zhang, C., and Tao, D. Gan-mdf: A method for multi-fidelity data fusion in digital twins. *arXiv preprint arXiv:2106.14655*, 2021.

[12] Mao, Z., Jagtap, A. D., and Karniadakis, G. E. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

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

[14] Mishra, S. and Molinaro, R. Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for pdes. *IMA Journal of Numerical Analysis*, 2021.

[15] Ortí, J. A. L., Gumbau, M. F., and Rochera, M. B. An alternative method to construct a consistent second-order theory on the equilibrium figures of rotating celestial bodies. *Journal of Computational and Applied Mathematics*, 404:113305, 2022.

[16] Raissi, M. Pinns. `https://github.com/maziarraissi/PINNs`, 2020.

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

[18] Shin, Y., Darbon, J., and Karniadakis, G. E. On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes. *arXiv preprint arXiv:2004.01806*, 2020.

[19] Sirignano, J. and Spiliopoulos, K. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[20] Smith, G. D., Smith, G. D., and Smith, G. D. S. *Numerical solution of partial differential equations: finite difference methods.* Oxford university press, 1985.

[21] Wang, S., Yu, X., and Perdikaris, P. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

[22] Yang, L., Meng, X., and Karniadakis, G. E. B-pinns: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data. *Journal of Computational Physics*, 425:109913, 2021.

[23] Yu, B. et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *arXiv preprint arXiv:1710.00211*, 2017.

[24] Zhang, C., Liang, M., Song, X., Liu, L., Wang, H., Li, W., and Shi, M. Generative adversarial network for geological prediction based on tbm operational data. *Mechanical Systems and Signal Processing*, 162:108035, 2022.

[25] Zienkiewicz, O. C., Taylor, R. L., Nithiarasu, P., and Zhu, J. *The finite element method*, volume 3. McGraw-hill London, 1977.