*Article*

# Dynamic Weight Strategy of Physics-Informed Neural Networks for the 2D Navier–Stokes Equations

**Shirong Li and Xinlong Feng ***

College of Mathematics and System Sciences, Xinjiang University, Urumqi 830046, China
* Correspondence: fxlmath@xju.edu.cn

**Abstract:** When PINNs solve the Navier–Stokes equations, the loss function has a gradient imbalance problem during training. It is one of the reasons why the efficiency of PINNs is limited. This paper proposes a novel method of adaptively adjusting the weights of loss terms, which can balance the gradients of each loss term during training. The weight is updated by the idea of the minmax algorithm. The neural network identifies which types of training data are harder to train and forces it to focus on those data before training the next step. Specifically, it adjusts the weight of the data that are difficult to train to maximize the objective function. On this basis, one can adjust the network parameters to minimize the objective function and do this alternately until the objective function converges. We demonstrate that the dynamic weights are monotonically non-decreasing and convergent during training. This method can not only accelerate the convergence of the loss, but also reduce the generalization error, and the computational efficiency outperformed other state-of-the-art PINNs algorithms. The validity of the method is verified by solving the forward and inverse problems of the Navier–Stokes equation.

**Keywords:** physics-informed neural networks; dynamic weight strategy; Navier–Stokes equations

**Citation:** Li, S.; Feng, X. Dynamic Weight Strategy of Physics-Informed Neural Networks for the 2D Navier–Stokes Equations. *Entropy* **2022**, *24*, 1254. https://doi.org/10.3390/e24091254

Academic Editor: Eun-jin Kim

Received: 8 July 2022
Accepted: 9 August 2022
Published: 6 September 2022

## 1. Introduction

Numerical simulation of fluid systems relies on solving partial differential equations using computational fluid dynamics (CFD) methods, including finite element, finite volume and finite difference methods [1–3]. However, it is usually expensive and time-consuming to generate meshes for solving equations in complex regions. Solving the inverse problem by CFD methods first requires tedious data assimilation and does not guarantee convergence [4]. Therefore, the use of CFD models in practical applications and real-time predictions is limited. It is important to develop an effective Navier–Stokes solver that can overcome these limitations.

In recent years, deep neural networks have received extensive attention in the field of scientific machine learning [5]. It can be used to construct new methods for solving partial differential equations, based on their well-known capability as universal function approximators [6]. The process of analyzing complex physical and engineering systems usually requires a large amount of data. The cost of data acquisition is often prohibitive, and we inevitably face the challenge of making decisions with partial information. When the governing equation is known, a neural network trained under the constraints of this equation can learn the solution of the partial differential equation with a small amount of data. The potential of using neural networks to solve partial differential equations has been recognized. With huge advances in computational power and training algorithms [7], and the invention of automatic differentiation methods [8], physics-informed neural networks(PINNs) were able to take this approach to a different level [9]. PINNs does not require mesh generation to solve fluid mechanics problems [10–13], so it has advantages for solving problems in complex regions. This is a good strategy to tackle the problem of the curse of dimensionality.

The baseline PINNs has provided new research ideas for solving Navier–Stokes equations, stochastic PDEs, and fractional PDEs [14–17]. However, it has been argued that the convergence and accuracy of PINNs are still of tremendous challenge, especially for Navier–Stokes with multi-scale characteristics. Wang [18] explored the problems existing in the training process of PINNs, and found that the gradient value of each residual term in the loss was greatly different in the process of backpropagation, and the training could not be balanced. Therefore, a method of adaptively adjusting the weights between different components in the loss is proposed to improve the training convergence. The problems of gradient vanishing and gradient explosion limit the application of this method. Zhao et al. also observed that the performance of PINNs is closely related to the appropriate combination of loss terms [10]. It introduces a non-adaptive weighting strategy and time adaptive strategy of loss function. However, using fixed weights is always time-consuming, labor-intensive, and prone to errors and omissions. If gradient descent optimizes multiple objectives consisting of fixed weights, there is a high probability of obtaining a local optimal solution [19]. Ref. [20] introduced adaptive weights for configuration and boundary losses, which are updated through the Neural Tangent Kernel (NTK). The distribution of the eigenvalues of NTK does not change, and it is more difficult to calculate the eigenvalues of NTK. The performance improvement is slight. A method that updates the adaptation weights of configuration points in the loss function concerning the network parameters was suggested [21]. It performs well for the AC equation. For other equations, the objective function has a convergence problem due to the existence of the Max system. A strategy for adaptive resampling of configuration points is proposed to improve the convergence and accuracy of some partial differential equations with steep gradient solutions [22]. After each resampling step, the number of residual points grows, increasing computational complexity.

This paper introduces a dynamic weight strategy for physics-informed neural networks(dwPINNs) to balance the contribution of each loss item to the network. The mechanism of weight update in this paper is completely different from other PINNs literature. It uses trainable weights as a soft multiplicative mask reminiscent of the attention mechanism used in computer vision [23]. Adaptive weights are trained simultaneously with network parameters, and the data are automatically weighted in the loss function, forcing the approximation of these data to improve. This is achieved by training the network to minimize loss and maximize weights. The weight update is based on the information of mean square error. This strategy is conducive to the convergence of loss and can reduce the generalization error.

This paper is organized as follows: Section 1 is the Introduction, and Section 2 gives a brief discussion of the effect of the loss terms on the network parameters in the optimization method. In Section 3 we discuss the construction of the loss function and some theoretical results on the convergence of weight sequence in the proposed method. Section 4 gives the results and detailed discussions on dwPINNs solving the Navier–Stokes. Finally, in Section 5, we summarize the conclusions of our work.

## 2. Preliminaries

### *2.1. Partial Differential Equations*

The incompressible Navier–Stokes equations is:

$$\frac{\partial \mathbf{u}}{\partial t} + \lambda_1(\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \lambda_2\nabla^2\mathbf{u} = \mathbf{f} \quad \text{in } \Omega\times[0,T], \tag{1a}$$

$$\nabla\cdot\mathbf{u} = 0 \quad \text{in } \Omega\times[0,T], \tag{1b}$$

$$\mathbf{u}(\mathbf{x},0) = \mathbf{u}_0 \quad \text{in } \Omega, \tag{1c}$$

$$\mathbf{u} = \mathbf{u}_b \quad \text{on } \Gamma_D\times[0,T]. \tag{1d}$$

where $\mathbf{u}(t,\mathbf{x}) = (u(t,\mathbf{x}), v(t,\mathbf{x}))$ is a velocity vector field, $p$ is a scalar pressure field. $\lambda_1, \lambda_2$ are the unknown parameters, and $\Omega$ is the solution domain. $\Gamma_D$ is the boundaries of the computational domain. Formula (1a) is the Momentum equation, Formula (1b) is the conservation of mass equation, Formula (1c) is Initial conditions, and Formula (1d) is

Dirichlet boundary conditions. In this paper, we shall solve both forward problems where solutions of partial differential equations are inferred given fixed model parameters $\lambda_1, \lambda_2$ as well as inverse problems, where the unknown parameters $\lambda_1, \lambda_2$ are learned from the observed data.

### 2.2. Fully Connected Neural Networks

Neural networks have well-known function approximation properties [24], so they can be used to approximate the solution of partial differential equations. The $(L-1)$ -hidden layer feed-forward neural network is defined by

$$\mathcal{Z}^k(\mathbf{x}) = \mathbf{W}^k \Phi\Big(\mathcal{Z}^{k-1}(\mathbf{x})\Big) + \mathbf{b}^k \in \mathbb{R}^{N_k}, \quad 2 \le k \le L,$$

and $\mathcal{Z}^1(\mathbf{x}) = \mathbf{W}^1\mathbf{x} + \mathbf{b}^1$, where in the last layer, the activation function is identity. By letting $\tilde{\mathbf{\Theta}} = (W^k, b^k)$ as the collection of all weights, biases. $\Phi$ is a nonlinear activation function. We can write the output of the neural network as

$$u_{\tilde{\Theta}}(\mathbf{x}) = \mathcal{Z}^L(\mathbf{x}; \tilde{\mathbf{\Theta}}),$$

where $\mathcal{Z}^L(\mathbf{x}; \tilde{\mathbf{\Theta}})$ emphasizes the dependence of the neural network output $\mathcal{Z}^L(\mathbf{x})$ on $\tilde{\mathbf{\Theta}}$. The loss function can be defined as

$$\mathcal{L} = \min_{\tilde{\mathbf{\Theta}}} \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u\Big(\mathbf{x}_u^i, \tilde{\mathbf{\Theta}}\Big) - u^i \right|^2.$$

Here, $\{\mathbf{x}_u^i, u^i\}_{i=1}^{N_u}$ denote the training data on $u(x)$. $N_u$ is the number of training data. The training process requires the loss function minimized with respect to the weights and biases in each network layer. Fitting in the sense of least squares requires the minimum mean square error between the fitting function and original data, and it is an approximation in the overall sense.

### 2.3. Optimization Method

We seek to find $\tilde{\mathbf{\Theta}}^*$ that minimizes the loss function $\mathcal{L}(\tilde{\mathbf{\Theta}})$. There are several optimization algorithms available to minimize the loss function. In general, a gradient-based optimization method is employed for the training of parameters [25]. Weights and biases are initialized from known probability distributions. In the basic form, given an initial value of parameters $\tilde{\mathbf{\Theta}}$ using Xavier initialization, the parameters are updated as

$$\tilde{\mathbf{\Theta}}^{m+1} = \tilde{\mathbf{\Theta}}^m - \eta_l \left.\frac{\partial \mathcal{L}(\tilde{\mathbf{\Theta}})}{\partial \tilde{\mathbf{\Theta}}}\right|_{\tilde{\mathbf{\Theta}} = \tilde{\mathbf{\Theta}}^m},$$

where $\eta_l$ is the learning rate. $\tilde{\mathbf{\Theta}}^m$ is the network parameter for step $m$.

Next, we illustrate the effect of the error on the gradient in backpropagation. The gradient of the loss function to the network parameters in the backpropagation algorithm [26] is as follows

$$\delta^{(L)} = -\Big(\boldsymbol{u} - \boldsymbol{a}^{(L)}\Big) \odot f'\Big(\boldsymbol{z}^{(L)}\Big), \tag{2a}$$

$$\delta^{(l)} = \bigg(\Big(W^{(l+1)}\Big)^\top \delta^{(l+1)}\bigg) \odot f'\Big(\boldsymbol{z}^{(l)}\Big), \tag{2b}$$

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \delta^{(l)} \Big(\boldsymbol{a}^{(l-1)}\Big)^\top, \tag{2c}$$

$$\frac{\partial \mathcal{L}}{\partial b^{(l)}} = \delta^{l}. \tag{2d}$$

$\odot$ represents the Hadamard product; $n_l$ represents the number of neurons in layer $l$; $f(\cdot)$ represents the activation function of the neuron; $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ represents the weight matrix from $l-1$ to $l$; $\boldsymbol{b}^{(l)} = \left(b_1^{(l)}, b_2^{(l)}, \cdots, b_{n_l}^{(l)}\right)^{\top} \in \mathbb{R}^{n_l}$ represents the bias from $l-1$ layer to $l$ layer; $\boldsymbol{z}^{(l)} = \left(z_1^{(l)}, z_2^{(l)}, z_{n_l}^{(l)}\right)^{\top} \in \mathbb{R}^{n_l}$ represents the output of neurons in the $l$ layer; $\boldsymbol{a}^{(l)} = \left(a_1^{(l)}, a_2^{(l)}, \cdots, a_{n_l}^{(l)}\right)^{\top} \in \mathbb{R}^{n_l}$ represents the activation value of neurons in the $l$ layer. Formula (2c) is the gradient of the loss function to the weight of the output layer and the hidden layer. Formula (2d) is the gradient of the loss function to the bias of the output layer and the hidden layer. We calculate the $\delta^{(l)}$ of the $l$ layer through the $\delta^{(l+1)}$ of the $l+1$ layer, and then obtain the gradient of the weight of the hidden layer by the Formula (2c).

The error has a great influence on the backpropagation gradient, which can be obtained from (2). When the objective function consists of many terms, it always tends to optimize the loss term with a larger error. Formulas (2b) and (2d)shows that the more hidden layers in the network, the easier the problem of gradient disappearance and explosion. Therefore, the network structure used in the experiment is a wide and shallow structure, and the number of hidden layers does not exceed five layers.

## 3. Methodology

The combination of multiple loss functions plays a significant role in the convergence of PINNs [10]. The most common way to combine losses of each constraint is the weighted summation. These are either nonadaptive or require training many times at an increased computational cost. Here, we propose a simple procedure using fully trainable weights. It is in line with the idea of neural network adaptation, that is, the dynamic weights in the loss function are updated together with network parameters through backpropagation.

### *3.1. Dynamic Weights Strategy for Physics-Informed Neural Networks*

We define residuals to be given by the left-hand-side of Equations (1a) and (1b); i.e.,

$$\mathbf{f}_1 := \frac{\partial \mathbf{u}}{\partial t} + \lambda_1(\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p - \lambda_2 \nabla^2 \mathbf{u} - \mathbf{f}, \mathbf{f}_2 := \nabla \cdot \mathbf{u}. \tag{3}$$

and proceed by approximating $\mathbf{u}(t, \mathbf{x})$ by neural networks. This assumption, along with Equation (2b) results in physical constraints $\mathbf{f}_1, \mathbf{f}_2$. $\mathbf{f}_1$ indicates that the numerical solution satisfies the conservation of momentum, and $\mathbf{f}_2$ indicates that the numerical solution satisfies the conservation of mass. The physical constraints of the network can be derived by applying the chain rule for differentiating compositions of functions using automatic differentiation. In order to balance the training of the residuals in each part of the loss, we multiply the trainable weights before each residual term of the PINNs loss function. The objective function is defined as follows

$$\mathcal{J} = w_u MSE_u + w_0 MSE_0 + w_b MSE_b + w_f MSE_f, \tag{4}$$

where

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| \mathbf{u}\left(t_u^i, \mathbf{x}_u^i, \tilde{\mathbf{\Theta}}\right) - \mathbf{u}_u^i \right|^2,$$

$$MSE_0 = \frac{1}{N_0} \sum_{i=1}^{N_0} \left| \mathbf{u}\left(t_0^i, \mathbf{x}_0^i, \tilde{\mathbf{\Theta}}\right) - \mathbf{u}_0^i \right|^2,$$

$$MSE_b = \frac{1}{N_b} \sum_{i=1}^{N_b} \left| \mathbf{u}\left(t_b^i, \mathbf{x}_b^i, \tilde{\mathbf{\Theta}}\right) - \mathbf{u}_b^i \right|^2,$$

and

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \left( \left| \mathbf{f}_1\left(t_f^i, \mathbf{x}_f^i, \tilde{\mathbf{\Theta}}\right) \right|^2 + \left| \mathbf{f}_2\left(t_f^i, \mathbf{x}_f^i, \tilde{\mathbf{\Theta}}\right) \right|^2 \right).$$

Here, $w_u, w_0, w_b, w_f$ are the newly introduced balance parameters. $\{t_u^i, \mathbf{x}_u^i, \mathbf{u}^i\}_{i=1}^{N_u}$ denote observed data (if any) $\{t_0^i, \mathbf{x}_0^i, \mathbf{u}_0^i\}_{i=1}^{N_0}$, $\{t_b^i, \mathbf{x}_b^i, \mathbf{u}_b^i\}_{i=1}^{N_b}$ denote the initial and boundary training data on $\mathbf{u}(t, \mathbf{x})$ and $\left\{t_f^i, \mathbf{x}_f^i\right\}_{i=1}^{N_f}$ specify the collocations points for $\mathbf{f}_1(t, \mathbf{x}), \mathbf{f}_2(t, \mathbf{x})$. $N_u, N_0, N_b, N_f$ is the number of corresponding data. The sampling method of initial and boundary training data is random selection at the corresponding boundary. The selection method of the collocations points is Latin hypercube sampling. We determine unknown parameters by performing the following tasks

$$\min_{\tilde{\mathbf{\Theta}}} \max_{w_u, w_0, w_b, w_f} \mathcal{J}\left(\tilde{\mathbf{\Theta}}, w_u, w_0, w_b, w_f\right). \tag{5}$$

This can be accomplished by a gradient descent/ascent procedure, with updates given by

$$\tilde{\mathbf{\Theta}}^{k+1} = \tilde{\mathbf{\Theta}}^k - \eta_k \nabla_{\tilde{\mathbf{\Theta}}} \mathcal{J}\left(\tilde{\mathbf{\Theta}}^k, w_u^k, w_f^k, w_b^k, w_0^k\right), \tag{6a}$$

$$w_u^{k+1} = w_u^k + \eta_w^k \nabla_{w_u} \mathcal{J}\left(\tilde{\mathbf{\Theta}}^k, w_u^k, w_f^k, w_b^k, w_0^k\right), \tag{6b}$$

$$w_0^{k+1} = w_0^k + \eta_w^k \nabla_{w_0} \mathcal{J}\left(\tilde{\mathbf{\Theta}}^k, w_u^k, w_f^k, w_b^k, w_0^k\right), \tag{6c}$$

$$w_b^{k+1} = w_b^k + \eta_w^k \nabla_{w_b} \mathcal{J}\left(\tilde{\mathbf{\Theta}}^k, w_u^k, w_f^k, w_b^k, w_0^k\right), \tag{6d}$$

$$w_f^{k+1} = w_f^k + \eta_w^k \nabla_{w_f} \mathcal{J}\left(\tilde{\mathbf{\Theta}}^k, w_u^k, w_f^k, w_b^k, w_0^k\right), \tag{6e}$$

where $\eta_k$ is the learning rate for the $k$th step in the process of updating the network parameters, $\eta_w^k$ is the learning rate for the $k$th step in the process of updating the balance parameters. Considering the dynamic weight $w_0$, to fix ideas, we see that

$$\nabla_{w_0} \mathcal{J}\left(\tilde{\mathbf{\Theta}}^k, w_u^k, w_0^k, w_b^k, w_f^k\right) = MSE_0^k \geq 0, \tag{7}$$

The sequence of weights $\left\{w_0^k; k = 1, 2 \ldots\right\}$ is monotonically nondecreasing, provided that $w_0^1$ is initialized to a non-negative value. Furthermore, (7) shows that the magnitude of the gradient, and therefore of the update, is larger when the mean squared error $MSE_0^k$ is large. This progressively penalizes the network more for not fitting the initial points closely. Notice that any of the weights can be set to fixed, non-trainable values if desired. For example, by setting $w_b^k \equiv 1$, only the weights of the initial and collocation points would be trained. If necessary, the weight can also be changed to other types of functions. The convergence of the weight sequence plays an important role in the stability of the Min system. Next, we prove the convergence of $w_0^k$. From (6), we can obtain

$$\left|w_0^{k+1} - w_0^k\right| = \eta_w^k \nabla_{w_0} \mathcal{J}\left(\tilde{\mathbf{\Theta}}^k, w_u^k, w_f^k, w_b^k, w_0^k\right),$$

$$\left|w_0^k - w_0^{k-1}\right| = \eta_w^{k-1} \nabla_{w_0} \mathcal{J}\left(\tilde{\mathbf{\Theta}}^{k-1}, w_u^{k-1}, w_f^{k-1}, w_b^{k-1}, w_0^{k-1}\right).$$

According to (7)

$$\left|w_0^{k+1} - w_0^k\right| = \eta_w^k MSE_0^k, \left|w_0^k - w_0^{k-1}\right| = \eta_w^{k-1} MSE_0^{k-1},$$

$$\eta_w^k MSE_0^k \leq \eta_w^{k-1} MSE_0^{k-1}. \tag{8}$$

Therefore $\left|w_0^{k+1} - w_0^k\right| \leq \left|w_0^k - w_0^{k-1}\right|$. According to the principle of compression mapping, $\{w_0^k\}$ is convergent. $\{w_0^k\}$ has an upper bound. The analysis for $w_u^k$,$w_b^k$,$w_f^k$ is the same as $w_0^k$.

**Remark 1.** *From Formula (8), it can be seen that the convergence of the sequence* $\{w_0^k\}$ *depends on the monotonous decrease in the* $MSE_0^k$. *The mean square error is theoretically monotonically decreased, so the weight sequence is theoretically convergent. However, in actual training, the mean square error is not strictly monotonically decreasing, which may have a little adverse effect on the performance of our method.*

In order to overcome the above problems as much as possible, Algorithm 1 is adopted. To strengthen the condition of balancing weight update, we update the weight only when the condition $MSE_0^{k+1} < MSE_0^k$ is satisfied. As far as possible, to make the weight sequence meet the convergence conditions, reducing the fluctuation of $MSE$ has an adverse effect on our method. In our implementation of dwPINNs, we use Tensorflow with a fixed number of iterations of Adam followed by another fixed number of iterations of the L-BFGS quasi-newton method [27,28]. This is consistent with the PINNs formulation in [9], as well as follow-up literature [11]. The adaptive weights are only updated in the Adam training steps and are held constant during L-BFGS training. The dwPINNs algorithm is summarized.

**Algorithm 1:** Dynamic weights strategy for PINNs

```
Step 1: Set Training steps K, the learning rate $\eta, \eta_w$, initial values balance weights
$\mathbf{w} = \left\{w_u, w_0, w_b, w_f\right\}$ and neural network parameters $\tilde{\mathbf{\Theta}}$.
Step 2: Consider a physics-informed neural network to define the weighted loss function
$\mathcal{J}\left(\tilde{\mathbf{\Theta}}, w_u, w_0, w_b, w_f\right)$ based on (4).
Step 3: Then use K steps of a gradient descent algorithm to update the parameters $\mathbf{w}$
and $\tilde{\mathbf{\Theta}}$ as:
for $k = 1$ to $K$ do
    if $MSE_0^{k+1} < MSE_0^k$ and $MSE_u^{k+1} < MSE_u^k$ and $MSE_b^{k+1} < MSE_b^k$ and $MSE_f^{k+1} < MSE_f^k$
        Tune the balance weights $\mathbf{w}$ via Adam to maximize the meeting constraints
            $\mathbf{w}_{k+1} \leftarrow \text{Adam1}\left(\mathcal{J}\left(\mathbf{w}_k; \tilde{\mathbf{\Theta}}; K\right); \eta; \eta_w\right)$
    Update the parameters $\tilde{\mathbf{\Theta}}$ via Adam to minimize $\mathcal{J}$
            $\tilde{\mathbf{\Theta}}_{k+1} \leftarrow \text{Adam2}\left(\mathcal{J}\left(\mathbf{w}_k; \tilde{\mathbf{\Theta}}_k; K\right); \eta; \eta_w\right)$
end for
```

### 3.2. A Brief Note on the Errors Involved in the dwPINNs Methodology

Let $\mathcal{F}$ and $u$ be the family of functions that can be represented by the chosen neural network and the exact solution of PDE. Then, we define $u_a = \arg\min_{f \in \mathcal{F}} \|f - u\|$ as the best approximation to the exact solution $u$. Let $u_g = \arg\min_{\tilde{\Theta}} \mathcal{J}(\tilde{\Theta})$ be the solution of net at global minimum and $u_t = \arg\min_{\tilde{\Theta}} \mathcal{J}(\tilde{\Theta})$ be the solution of net at local minimum. Therefore, the total error consists of an approximation error $\mathcal{E}_{\text{app}} = \|u - u_a\|$, the optimization error $\mathcal{E}_{\text{opt}} = \|u_t - u_g\|$ and the generalization error $\mathcal{E}_{\text{gen}} = \|u_g - u_a\|$. In PINNs, the number and location (distribution) of residual points are two important factors that affect the generalization error [13]. The optimization error is introduced due to the complexity of the loss function. The performance of PINNs is closely related to the appropriate combination of loss terms, which may avoid local optimization. Thus, the total error in PINNs as

$$\mathcal{E}_{PINN} := \|u_t - u\| \leq \|u_t - u_g\| + \|u_g - u_a\| + \|u_a - u\|. \tag{9}$$

### 3.3. Advantages of Dynamic Weight Strategy for Physics-Informed Neural Networks

1. The optimization error can be reduced by using the dynamic weight strategy for physics-informed neural networks. During training, each part of the loss function can be dropped more evenly, and the loss can become smaller and converge faster.

2. This method can reduce the generalization error by increasing the weights of hard-to-train points during training. It also makes the error of such hard-to-train points smaller.

## 4. Numerical Examples

In this section We apply the proposed dwPINNs to simulate different incompressible Navier–Stokes flows. First, we consider two-dimensional unsteady equations with the analytic solution to investigate the effectiveness of dwPINNs. Then, we employ the dynamic weights strategy to steady lid-driven cavity flow in two dimensions. Compared with other PINNs methods, the efficiency of this method is highlighted. Finally, the inverse problem of the flow around a cylinder is solved by the dwPINNs method. In the numerical experiments, we strictly control the non-experimental variables of the two test methods to be the same, such as training data, network structure, optimization method, etc. The activation functions used in the following numerical experiments are tanh. G.Cybenkot [29] found that it has a strong linear superposition approximation ability and is more suitable for function approximation. We keep the amount of training data roughly equal to the parameters of the neural network to avoid the overfitting problem. To illustrate the efficiency of the proposed method, the accuracy of the trained model is assessed through the relative $L_2$ error of the exact value $\bar{u}(\mathbf{x}_i, t_i)$ and the trained approximation $u(\mathbf{x}_i, t_i)$ inferred by the network at the data $\{t^i, \mathbf{x}^i\}_{i=1}^{N_t}$, $N_t$ is the number of test data. The deep learning framework used in this experiment is tensorflow2.3. In terms of hardware, the CPU is Intel CORE i5 7th Gen, the memory is 4G, and the GPU is NVIDIA GeForce 940MX.

### 4.1. Navier–Stokes Equations with Analytic Solution

We first solve forward problems. We use the 2D unsteady Navier–Stokes with an analytical solution as the first test case to demonstrate the feasibility of dwPINNs. The analytical solution is as follows

$$\begin{aligned} u^* &= -\sin(t)\sin^2(\pi x)\sin(\pi y)\cos(\pi y), \\ v^* &= \sin(t)\sin(\pi x)\cos(\pi x)\sin^2(\pi y), \\ p^* &= \sin(t)\sin(\pi x)\cos(\pi y). \end{aligned} \tag{10}$$

Based on Section 3.1, the problem loss is defined as follows

$$\begin{aligned} MSE_0 &= \tfrac{1}{N_0}\textstyle\sum_{i=1}^{N_0}\left|\mathbf{u}\left(t_0^i, \mathbf{x}_0^i, \tilde{\mathbf{\Theta}}\right) - \mathbf{u}_0^i\right|^2, \\ MSE_b &= \tfrac{1}{N_b}\textstyle\sum_{i=1}^{N_b}\left|\mathbf{u}\left(t_b^i, \mathbf{x}_b^i, \tilde{\mathbf{\Theta}}\right) - \mathbf{u}_b^i\right|^2, \\ MSE_f &= \tfrac{1}{N_f}\textstyle\sum_{i=1}^{N_f}\left(\left|\mathbf{f}_1\left(t_f^i, \mathbf{x}_f^i, \tilde{\mathbf{\Theta}}\right)\right|^2 + \left|\mathbf{f}_2\left(t_f^i, \mathbf{x}_f^i, \tilde{\mathbf{\Theta}}\right)\right|^2\right). \end{aligned} \tag{11}$$

$\lambda_1 = 1, \lambda_2 = 0.01$. The computational domain is defined by $\Omega = [0,1] \times [0,1]$ and the time interval is $[0,1]$. There are 40 points with fixed spatial coordinate on each boundary, $N_b = 4 \times 40$. For computing the equation loss of dwPINNs, 10,000 points are randomly selected inside the domain. Adam training time is 10,000.

The numerical results of the two methods are shown in Table 1. We see that the dwPINNs perform better than the PINNs, and also that applying the dynamic weights can improve the simulation accuracy. A snapshot of the velocity fields together with the absolute errors at $t = 1$ is displayed in Figure 1. It shows that this method is feasible and qualitatively accurate. The convergence of dynamic weights during the training process is displayed in Figure 2.

Next, considering whether dwPINNs can still work in the middle and later stages of training, we design the following experiment. After PINNs are normally trained 5000 times, we conduct dynamic weight strategy training 5000 times and take PINNs as the control group of the experiment. The results are shown in Figure 3. It can be clearly seen from the figure that the dynamic weight strategy can accelerate the convergence of error in the middle and later stages of training.

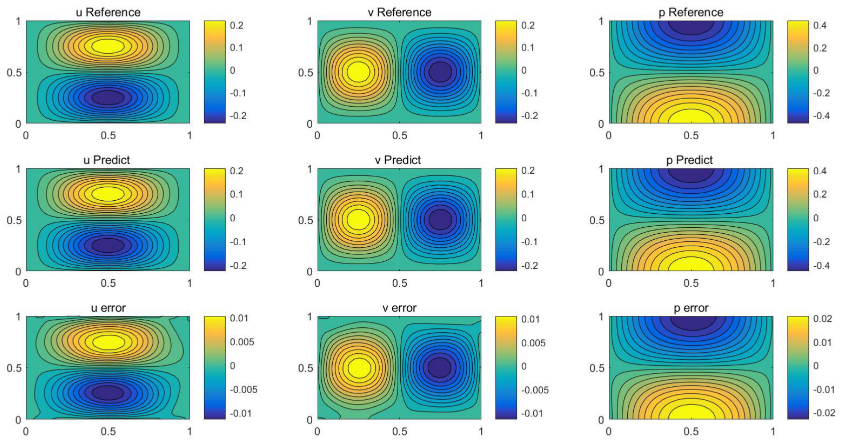

**Figure 1.** Navier–Stokes: A snapshot of analytical solution (**top**) prediction solution (**middle**) and error (**bottom**) at $t = 1$.

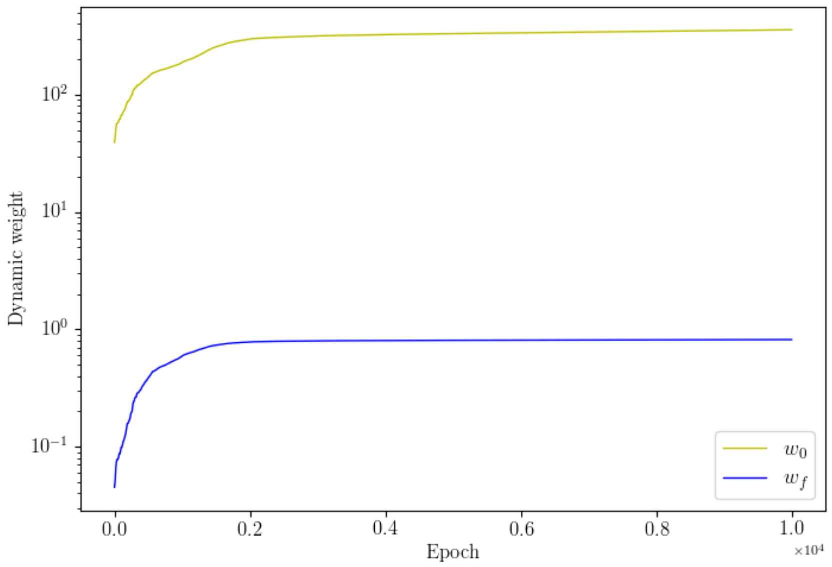

**Figure 2.** Navier–Stokes: Dynamic weights $w_f$, $w_u$ diagrams are shown.

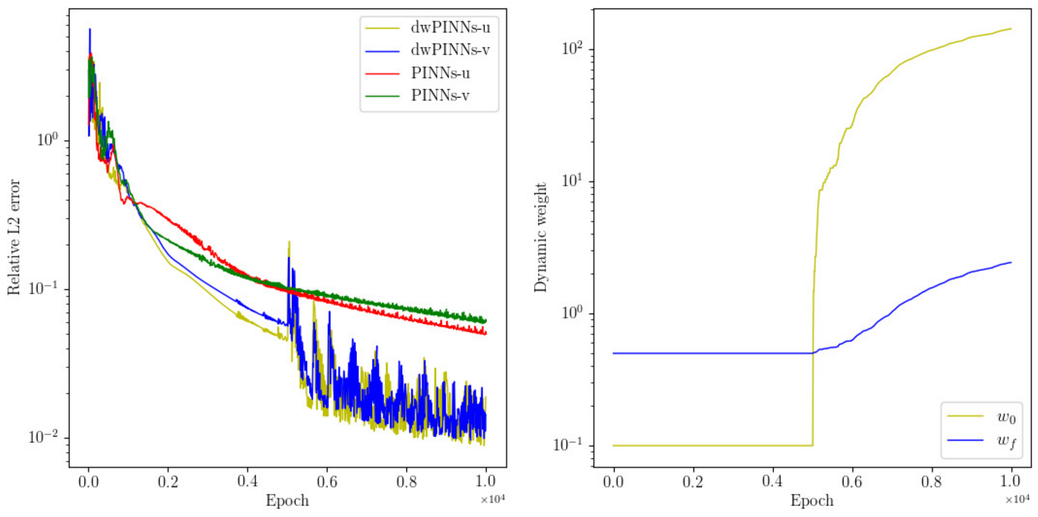

**Figure 3.** Navier–Stokes: The history of relative L2 error (**left**) of dwPINNs and PINNs and the training process of dynamic weights (**right**).

**Table 1.** Relative L2 Error and Training Time of dwPINNs and PINNs. The NN size is 4 × 50.

| | **Error *u*** | **Error *v*** | **Error *p*** | **Training Time (s)** |
|---|---|---|---|---|
| dwPINNs | $2.890 \times 10^{-3}$ | $2.973 \times 10^{-3}$ | $5.522 \times 10^{-1}$ | 5412.53 |
| PINNs | $1.180 \times 10^{-2}$ | $1.204 \times 10^{-2}$ | $7.761 \times 10^{-1}$ | 5314.67 |

### *4.2. Comparison of the Different PINNs Methods for 2D Navier–Stokes Equations*

The lid-driven cavity flow is a standard test case for verifying the accuracy of new computational methods for incompressible Navier–Stokes equations. Although, there are many papers in the literature that present results of the lid-driven cavity with different formulations, grids and numerical methods, we shall compare the results with Wang [18], where they used a Learning rate annealing for PINNs. The domain is $[0,1] \times [0,1]$ and no-slip boundary conditions are applied at the left, bottom and right boundaries. The top boundary moves with constant velocity in the positive $x$ direction. $\mathbf{u}(\mathbf{x}) = (u(x), v(x))$ is a velocity vector field, $p$ is a scalar pressure field. $\lambda_1 = 1, \lambda_2 = 0.01$.

Parameters during training are as follows: $N_f = 10{,}000$, $N_b = 3000$. The network structure is a 4-layer deep fully-connected network with 50 neurons per layer. The results of this experiment are summarized in Table 2. It represents average relative L2 error of $|\mathbf{u}(\mathbf{x})| = \sqrt{u^2(x) + v^2(x)}$ across 10 runs with random restarts. Figure 4 summarizes the results of our experiment, which shows the magnitude of the predicted solution $u$, $v$ and absolute point-wise error predicted by dwPINNs, after 32,000 stochastic gradient descent updates using the Adam optimizer. Figure 5 is the convergence process of dynamic weights and the relative L2 error of $u$ and $v$.

**Table 2.** Relative L2 errors of velocity the different PINNs methods. Adam training time is 32,000. The network architecture is fixed to 4 layers with 50 neurons per hidden layer.

| | **dwPINNs** | **PINNs** | **SAPINNs** | **Learning Rate Annealing for PINNs** |
|---|---|---|---|---|
| Relative L2 error | $6.710 \times 10^{-2}$ | $2.713 \times 10^{-1}$ | $3.415 \times 10^{-1}$ | $2.492 \times 10^{-1}$ |

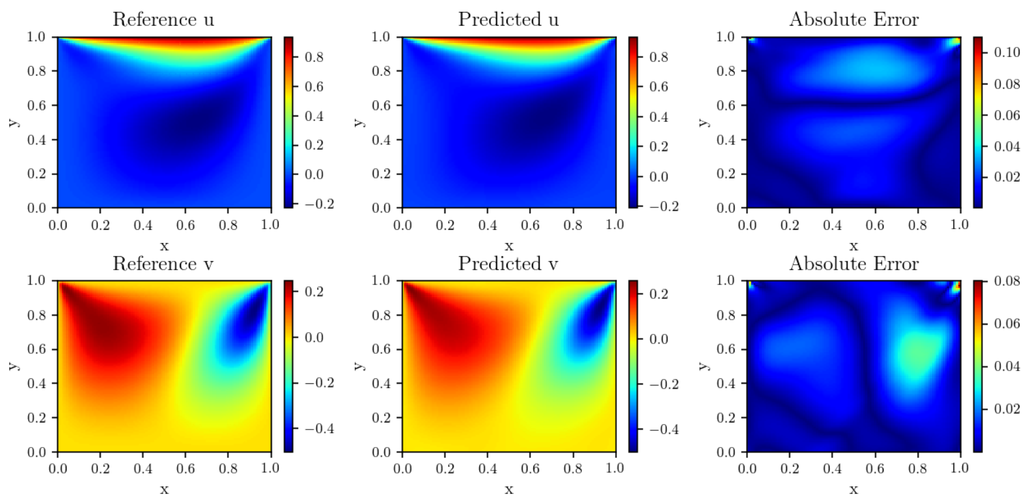

**Figure 4.** Flow in a lid-driven cavity: Reference solution using a comsol solver, prediction of dynamic weights strategy of PINNs, and absolute point-wise error. The relative L2 error of u is $6.512 \times 10^{-2}$. The relative L2 error of v is $8.973 \times 10^{-2}$.

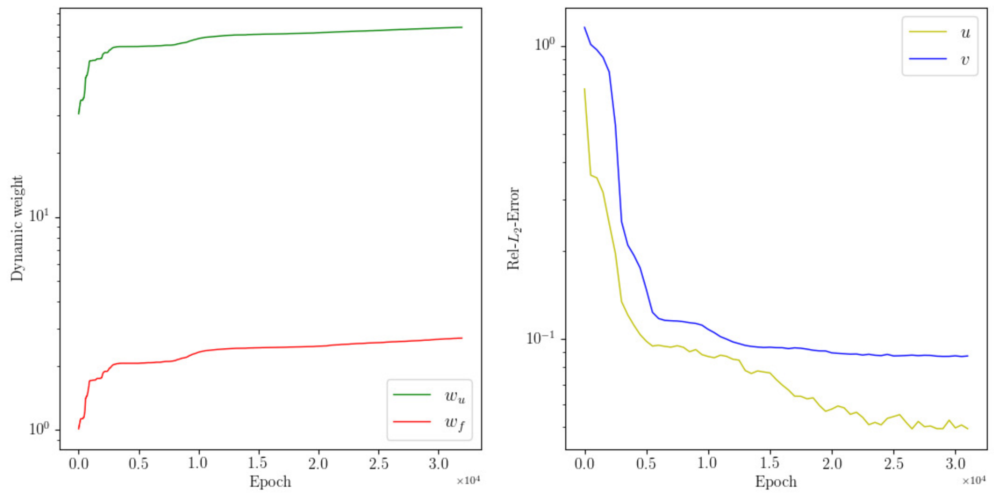

**Figure 5.** The training process of dynamic weights (**left**) and the relative L2 error of $u$ and $v$ (**right**).

Learning rate annealing for PINNs is one of the important loss balancing methods [18]. Compared with Learning rate annealing for PINNs, the error of dwPINNs is reduced. The SAPINNs outperformed other state-of-the-art PINN algorithms in L2 error by a wide margin [21]. However, the SAPINNs failed to solve the Navier–Stokes equations. Therefore, the performance of dwPINNs is the best at present.

In order to further analyze the performance of our method, we carried out the following systematic research to quantify its prediction accuracy for different numbers of training points and configuration points, as well as different neural network structures. In Table 3, we report the relative L2 error obtained under the conditions of different initial and boundary training data $N_u$ and different configuration points $N_F$, while keeping the 4-layer network architecture fixed. With a sufficient number of configuration points $N_f$, the overall trend of prediction accuracy continues to improve with the increase in the total number of training data $N_u$. Finally, Table 4 shows the resulting relative L2 for different numbers of hidden layers, and different numbers of neurons per layer, while the total number of training and collocation points is kept fixed to $N_u = 3000$ and $N_f = 10,000$, respectively. As expected, we observed that the prediction accuracy improved with the increase in the number of layers and neurons.

**Table 3.** Relative L2 error between the predicted and the exact $|\mathbf{u}|$ for different number of initial and boundary training data $N_u$, and different number of collocation points $N_f$. Here, the network architecture is fixed to 4 layers with 50 neurons per hidden layer.

| | **2000** | **4000** | **8000** | **10,000** |
|---|---|---|---|---|
| 200 | $3.2 \times 10^{-1}$ | $2.7 \times 10^{-1}$ | $1.3 \times 10^{-1}$ | $1.5 \times 10^{-1}$ |
| 1000 | $3.1 \times 10^{-1}$ | $2.5 \times 10^{-1}$ | $9.1 \times 10^{-2}$ | $9.0 \times 10^{-2}$ |
| 3000 | $1.4 \times 10^{-1}$ | $1.2 \times 10^{-1}$ | $8.2 \times 10^{-2}$ | $6.7 \times 10^{-2}$ |

**Table 4.** Relative L2 error between the predicted and the exact $|\mathbf{u}|$ for different numbers of hidden layers and different numbers of neurons per layer. Here, the total number of training and collocation points is fixed to $N_u = 3000$ and $N_f = 10,000$, respectively.

| | **20** | **30** | **40** | **50** |
|---|---|---|---|---|
| 2 | $3.5 \times 10^{-1}$ | $1.5 \times 10^{-1}$ | $1.3 \times 10^{-1}$ | $1.5 \times 10^{-1}$ |
| 3 | $2.7 \times 10^{-1}$ | $1.3 \times 10^{-1}$ | $9.7 \times 10^{-2}$ | $8.2 \times 10^{-2}$ |
| 4 | $1.4 \times 10^{-1}$ | $1.0 \times 10^{-1}$ | $7.4 \times 10^{-2}$ | $6.7 \times 10^{-2}$ |

### *4.3. Inverse Problem: Two-Dimensional Navier-Stokes Equations*

Here, we use a dynamic weight strategy for Physics-informed neural networks to simulate the 2D vortex shedding behind a circular cylinder at Re = 100. The cylinder diameter D is 1. The simulation domain size is $[0,8] \times [-2,-2]$. In this example, $\lambda_1 = 1.0, \lambda_2 = 0.01$. High-fidelity data from [9] is used as a reference and for providing training data for dwPINNs. We sample $N_f = 5000$ collocation points, $N_u = 5000$ exact points. Our goal is to identify the unknown parameters $\lambda_1, \lambda_2$, as well as to obtain a qualitatively accurate reconstruction of the entire pressure field $p(t,x,y)$ in the cylinder wake, which by definition can only be identified up to a constant. Each loss item is as follows

$$\begin{aligned} MSE_u &= \frac{1}{N_u}\sum_{i=1}^{N_u}\left|\mathbf{u}\left(t_u^i, \mathbf{x}_u^i, \tilde{\boldsymbol{\Theta}}\right) - \mathbf{u}_u^i\right|^2, \\ MSE_f &= \frac{1}{N_f}\sum_{i=1}^{N_f}\left(\left|\mathbf{f}_1\left(t_f^i, \mathbf{x}_f^i, \tilde{\boldsymbol{\Theta}}\right)\right|^2 + \left|\mathbf{f}_2\left(t_f^i, \mathbf{x}_f^i, \tilde{\boldsymbol{\Theta}}\right)\right|^2\right). \end{aligned} \tag{12}$$

Numerical results of the Navier–Stokes equation in the way of the dwPINNs are displayed in Table 5. Compared with PINNs, dwPINNs have less training time and higher prediction accuracy. Then, after applying noise to the original training data, the calculation was performed using the dwPINNs method. We observed that even if the training data are corrupted with 1% uncorrelated Gaussian noise, the method can identify unknown parameters very accurately $\lambda_1$ and $\lambda_1$, indicating that our method has good stability.

Figure 6 shows the loss history of two methods and dynamic weights history. Additionally, the convergence of loss dwPINNs is quicker than PINNs in Figure 6. The scalar $w_u$ increases rapidly and is more punitive to the $MSE_u$, which leads to faster convergence. The loss of PINNs would attain $9.268 \times 10^{-1}$, while dwPINNs would converge to $1.206 \times 10^{-1}$. Also plotted are representative snapshots of the predicted velocity components $u(t,x,y)$, $v(t,x,y)$ after the model was trained in Figure 7. Based on the predicted versus instantaneous pressure field $p(x,y,t)$ shown in Figure 7, the error between the predicted value and the true value is extremely low in the entire calculation domain. An interesting result is that, in the absence of any training data on the pressure itself, the network can provide a qualitatively accurate prediction of the entire pressure field $p(t,x,y)$. The neural network architecture used here consists of 4 hidden layers with 50 neurons in each layer. The predicted $\lambda_1 = 1.00065, \lambda_2 = 0.00991$.

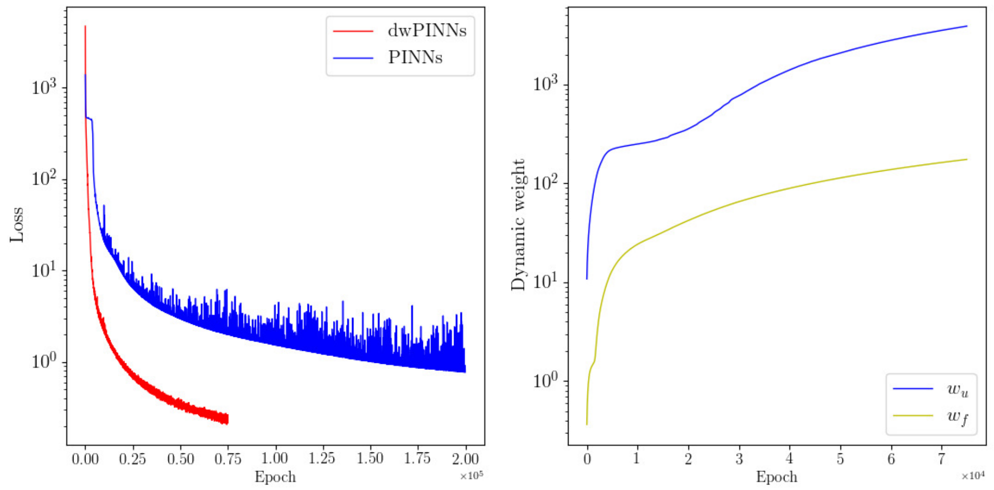

**Figure 6.** Flow past a circular cylinder: Loss history (**left**) two methods and dynamic weights history (**right**).

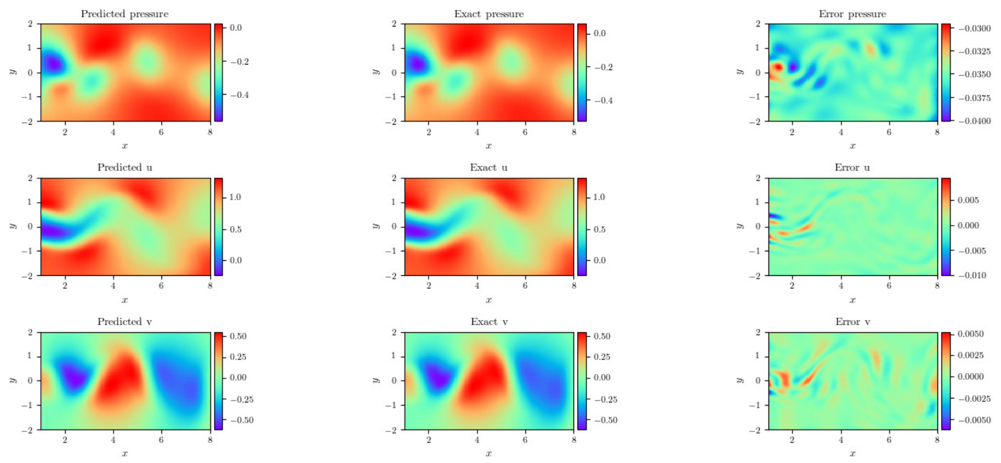

**Figure 7.** Flow past a circular cylinder: A snapshot of reference solution (**left**) with the result approximated by dwPINNs (**middle**) based on the error (**right**) at t = 10.

**Table 5.** Relative $L_2$ error of velocity field $u$, $v$ and relative error unknown parameters $\lambda_1$, $\lambda_2$.

| | $u$ | $v$ | $\lambda_1$ | $\lambda_2$ | Training Time (s) |
|---|---|---|---|---|---|
| dwPINNs (clean) | $1.421 \times 10^{-3}$ | $3.832 \times 10^{-3}$ | 0.06% | 0.9% | 30,574 |
| dwPINNs (1% noise) | $2.811 \times 10^{-3}$ | $5.215 \times 10^{-3}$ | 0.23% | 2.1% | 30,575 |
| PINNs | $2.103 \times 10^{-3}$ | $6.813 \times 10^{-3}$ | 0.99% | 2.30% | 51,475 |

## 5. Conclusions

In this paper, we introduced a dynamic weights strategy for loss balanced in physics-informed neural networks. The strategy is helpful to the training of PINNs in solving incompressible Navier–Stokes equations. It can effectively improve unbalanced back-propagated gradients during model training. The most appropriate weights are added to the optimization system through the objective function minimax alternating optimization, which makes the training more balanced. It is theoretically analyzed that the balanced weights are convergent under the condition of monotonic mean square error. The advantages of the method are verified by studying the forward and inverse problems of the Navier–Stokes equation. Various experimental results can support our view that the loss function decays slightly faster and the relative error is lower than the state-of-the-art PINNs. This paper also provides a reference for the application of adaptive mechanisms in the PINNs framework. Code and data accompanying this manuscript are publicly available at https://github.com/1shirong/dwPINNs.git (accessed on 7 July 2022). In this method, the convergence of the maximum system and the convergence of the minimum system are interdependent. Adam and other algorithms can not guarantee to find the global minimum, which will limit the optimal performance of this method. How to design an optimization algorithm specifically for PINNs is an open and meaningful problem.

**Author Contributions:** Conceptualization, X.F.; methodology, S.L.; software, S.L.; validation, X.F.; formal analysis, X.F. and S.L.; investigation, S.L.; resources, X.F.; data curation, S.L.; writing—original draft preparation, S.L.; writing—review and editing, X.F. and S.L.; visualization, S.L.; supervision, X.F.; project administration, X.F.; funding acquisition, X.F. and S.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work is in parts supported by the Research Fund from Key Laboratory of Xinjiang Province (No. 2020D04002).

**Acknowledgments:** The authors would like to thank Xu Hui of Shanghai Jiaotong University for his suggestions on experimental design. His help and discussion have enabled me to learn more about machine learning. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions that helped improve the quality of this manuscript.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Huang, P.; Feng, X.; He, Y. Two-level defect-correction Oseen iterative stabilized finite element methods for the stationary Navier–Stokes equations. *Appl. Math. Model.* **2013**, *37*, 728–741. [CrossRef]
2. Feng, X.; Li, R.; He, Y.; Liu, D. $P_1$-Nonconforming quadrilateral finite volume methods for the semilinear elliptic equations. *J. Sci. Comput.* **2012**, *52*, 519–545. [CrossRef]
3. Feng, X.; He, Y. H1-Super-convergence of center finite difference method based on P1-element for the elliptic equation. *Appl. Math. Model.* **2014**, *38*, 5439–5455. [CrossRef]
4. Elia, M.D.; Perego, M.; Yeneziani, A. A variational data assimilation procedure for the incompressible four equations in hemodynamics. *J. Sci.* **2012**, *52*, 340–359.
5. Baker, N.; Alexander, F.; Bremer, T.; Hagberg, A.; Kevrekidis, Y.; Najm, H.; Parashar, M.; Patra, A.; Sethian, J.; Wild, S.; et al. *Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence*; USDOE Office of Science (SC): Washington, DC, USA, 2019.
6. Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.* **1989**, *2*, 359–366. [CrossRef]
7. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. TensorFlow: A System for Large-Scale Machine Learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
8. Baydin, A.G.; Pearlmutter, B.A.; Radul, A.A.; Siskind, J.M. Automatic differentiation in machine learning: A survey. *J. Mach. Learn. Res.* **2018**, *18*, 1–43.
9. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [CrossRef]
10. Wight, C.L.; Zhao, J. Solving allen-cahn and cahn-hilliard equations using the adaptive physics informed neural networks. *arXiv* **2020**, arXiv:2007.04542.
11. Jagtap, A.D.; Kharazmi, E.; Karniadakis, G.E. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Comput. Method. Appl. Mech. Eng.* **2019**, *365*, 113028. [CrossRef]
12. Mao, Z.; Jagtap, A.D.; Karniadakis, G.E. Physics-informed neural network for high-speed flows. *Comput. Method. Appl. Mech. Eng.* **2020**, *360*, *112789*. [CrossRef]
13. Jagtap, A.D.; Karniadakis, G.E. Extended Physics-Informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition Based Deep Learning Framework for Nonlinear Partial Differential Equations. *Commun. Comput. Phys.* **2020**, *28*, 2002–2041.
14. Yang, L.; Zhang, D.; Karniadakis, G.E. Physics-informed generative adversarial networks for stochastic differential equations. *SIAM J. Sci. Comput.* **2020**, *42*, A292–A317. [CrossRef]
15. Raissi, M. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. *arXiv* **2018**, arXiv:1804.07010.
16. Zhang, D.; Lu, L.; Guo, L.; Karniadakis, G.E. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *J. Comput. Phys.* **2019**, *397*, 108850. [CrossRef]
17. Zhang, D.; Guo, L.; Karniadakis, G.E. Learning in modal space: Solving time-dependent stochastic PDEs using physics-informed neural networks. *SIAM J. Sci. Comput.* **2020**, *42*, A639–A665. [CrossRef]
18. Wang, S.; Teng, Y.; Perdikaris, P. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM J. Sci. Comput.* **2021**, *43*, A3055–A3081. [CrossRef]
19. Elhamod, M.; Bu, J.; Singh, C.; Redell, M.; Ghosh, A.; Podolskiy, V.; Lee, W.C.; Karpatne, A. CoPhy-PGNN: Learning physics-guided neural networks with competing loss functions for solving eigenvalue problems. *ACM Trans. Intell. Syst. Technol.* **2020**. [CrossRef]
20. Wang, S.; Yu, X.; Perdikaris, P. When and why PINNSs fail to train: A neural tangent kernel perspective. *J. Comput. Phys.* **2022**, *449*, 110768. [CrossRef]
21. Mcclenny, L.; Braga-Neto, U. Self-Adaptive Physics-Informed Neural Networks using a Soft Attention Mechanism. In Proceedings of the AAAI-MLPS 2021, Stanford, CA, USA, 22–24 March 2021.
22. Lu, L.; Meng, X.; Mao, Z.; Karniadakis, G.E. DeepXDE: A deep learning library for solving differential equations. *SIAM Rev.* **2021**, *63*, 208–228. [CrossRef]
23. Wang, F.; Jiang, M.; Qian, C.; Yang, S.; Li, C.; Zhang, H.; Wang, X.; Tang, X. Residual attention network for image classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 3156–3164.
24. Dissanayake, M.; Phan-Thien, N. Neural-network-based approximations for solving partial differential equations. *Commun. Numer. Methods Eng.* **1994**, *10*, 195–201. [CrossRef]
25. Barakat, A.; Bianchi, P. Convergence and dynamical behavior of the ADAM algorithm for nonconvex stochastic optimization. *SIAM J. Optim.* **2021**, *31*, 244–274. [CrossRef]

26. Li, J.; Cheng, J.H.; Shi, J.Y.; Huang, F. Brief introduction of back propagation (BP) neural network algorithm and its improvement. In *Advances in Computer Science and Information Engineering*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 553–558.
27. Kingma, D.; Ba, J. Adam: A Method for Stochastic Optimization. *Comput. Sci.* **2014**. [CrossRef]
28. Liu, D.C.; Nocedal, J. On the limited memory bfgs method for large scale optimization. *Math. Program* **1989**, *45*, 503–528. [CrossRef]
29. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control Signal Syst.* **1989**, *2*, 303–314. [CrossRef]