Article

A Second-Order Network Structure Based on Gradient-Enhanced Physics-Informed Neural Networks for Solving Parabolic Partial Differential Equations

Kuo Sun [ID] and Xinlong Feng *

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

Abstract: Physics-informed neural networks (PINNs) are effective for solving partial differential equations (PDEs). This method of embedding partial differential equations and their initial boundary conditions into the loss functions of neural networks has successfully solved forward and inverse PDE problems. In this study, we considered a parametric light wave equation, discretized it using the central difference, and, through this difference scheme, constructed a new neural network structure named the second-order neural network structure. Additionally, we used the adaptive activation function strategy and gradient-enhanced strategy to improve the performance of the neural network and used the deep mixed residual method (MIM) to reduce the high computational cost caused by the enhanced gradient. At the end of this paper, we give some numerical examples of nonlinear parabolic partial differential equations to verify the effectiveness of the method.

Keywords: physics-informed neural networks; second-order neural network; deep mixed residual method; parabolic partial differential equations

1. Introduction

With the development of computational technology, neural network theory, and data science, deep learning algorithms have been remarkably successful in various application areas [1–3]. In physics and engineering, many important physical models are described as partial differential equations (PDEs), such as Navier–Stokes equations [4] for fluid mechanics and Maxwell equations [5] for electromagnetic field theory. When solving partial differential equations using traditional numerical methods, such as the Finite Difference Method (FDM) [6], Finite Element Method (FEM) [7], Finite Volume Method (FVM) [8], Radial Basis Function Method (RBF) [9], etc., problems such as high computational costs and the curse of dimensionality are often encountered. Over the last few years, the use of machine learning to solve partial differential equations has also rapidly expanded [10–13]. In 2018, Karniadakis and his research team were the first to put forward the concept of physics-informed neural networks (PINNs) [14,15], which solve PDEs by embedding them and their initial boundary value conditions in the loss function of the neural network thanks to automatic differentiation [16], following the physical laws described by nonlinear partial differential equations. Compared to traditional methods, PINNs are independent of the grid's quality, can avoid the curse of dimensionality [17], and offer significant advantages in solving forward and inverse problems for partial differential equations. They also have great advantages in high-dimensional problems and complex geometric region problems. They have been extensively applied in various domains involving partial differential equations [18–22].

Although PINNs have achieved good results in solving partial differential equations, there are still some aspects that need improvement, such as their accuracy and efficiency when used to solve complex problems. Several approaches have been proposed to improve PINNs' performance, for example, residual-based adaptive refinement (RAR) [23],

importance sampling [24], adaptive activation function [25–27], adaptive weight [28–30], etc. These strategies can go a long way toward improving the neural network's performance. Meng et al. proposed the parareal physics-informed neural network (PPINN) [31], which decomposes a long-time problem into multiple short-time problems for parallel iterative solutions, which can accelerate the training speed of PINNs. Additionally, a PINN algorithm based on domain decomposition was also proposed [32–35], as well as other extension algorithms [36–38]. Likewise, the theoretical analysis of PINNs presents numerous difficulties. Nevertheless, Mishra et al. derived a generalization error estimate according to the training error and the number of training samples by taking advantage of the stability characteristics of partial differential equations [39]. During this time, in [40], for a particular type of inverse problem, it was also proven that PINNs produce a strict estimation of the generalization error when approximating these inverse problems. Ryck et al. [41] placed a strict upper bound on the errors produced by PINNs and extended physics-informed neural networks (xPINNs) when solving Navier–Stokes equations under certain preconditions.

It is well known that increasing the width and depth of a network can improve network performance, and deep networks are generally better than shallow ones, but they are harder to train. A residual neural network [42,43] with a jump connection structure can be very effective in avoiding gradient vanishing [44] and can train deeper networks. Based on a residual network, Ruthotto et al. [45] established a connection between a deep residual convolutional neural network and a partial differential equation and constructed a second-order convolutional neural network for image recognition through a second-order hyperbolic equation. With this idea in mind, here, we constructed a second-order network structure based on a parametric light wave equation. In order to improve the performance of the neural network, we also used an adaptive activation function strategy and a gradient-enhanced strategy [46] and combined the deep mixed residual method (MIM) [47] to reduce the calculation cost of high-order derivatives. We applied this method to solve forward and inverse problems for partial differential equations. The heat conduction and diffusion of a substance appear to be processes of entropy change, which can be described by parabolic partial differential equations [48,49]. This is also a motivation for this paper to explore the deep learning algorithm for solving parabolic partial differential equations.

The rest of this document is arranged in the following manner: In Section 2, we take a brief look at solving differential equations through physics-informed neural networks. In Section 3, the methodology proposed in this paper is presented in detail. In Section 4, several numerical examples are given to test the efficiency of the proposed method in the resolution of differential equations, including forward and inverse problems. Finally, in Section 5, we summarize this work.

2. PINNs for Solving PDEs

This section introduces physics-informed neural networks (PINNs) for solving partial differential equations. Mathematically, a deep neural network can be regarded as a composite function of the input variable x, with a very high capacity for approximation. A neural network with a depth of L has only 1 input layer, $L-1$ hidden layers, and 1 output layer. Simultaneously, we assume that there are N_k neurons in the k layer, so the corresponding weight matrix and bias vectors are $W^k \in R^{N_k \times N_{k-1}}$ and $b^k \in R^{N_k}$, respectively. Then, we take into account a nonlinear activation function f_{act}, including logistic sigmoid $(1/(1+e^{-x}))$, hyperbolic tangent (tanh) $((e^x - e^{-x})/(e^x + e^{-x}))$, the rectified linear unit (ReLU) $(\max\{0, x\})$, etc. Finally, the neural network can be recursively expressed in the following way [25]:

$$y(x, \Theta) = W^L(f_{act}(W^{L-1}(\cdots f_{act}(W^2(f_{act}(W^1 x + b^1)) + b^2)\cdots) + b^{L-1})) + b^L, \quad (1)$$

where $\Theta = \left\{ W^i, b^i \right\}_{i=1}^{L}$ denotes the trainable parameters in the neural network. Additionally, we consider the following spatiotemporal PDEs for the solution $u(\pmb{x}, t)$ parameterized by the parameter λ defined on the domain $\Omega \times (0, T)$ [46]:

$$\begin{cases} \mathcal{F}(\pmb{x}, t, u, u_t, \nabla u, \Delta u, \lambda) = 0, (\pmb{x}, t) \in \Omega \times (0, T), \\ \qquad\qquad\qquad \mathcal{B}u = g, (\pmb{x}, t) \in \partial\Omega \times (0, T), \\ \qquad\qquad\qquad\quad u = h, (\pmb{x}, t) \in \Omega \times \{0\}, \end{cases} \tag{2}$$

where $(x) = (x_1, x_2, \cdots, x_d) \in R^d$, λ is a constant or can be a function of $\pmb{x}$ and t, $\Omega \subset R^d (d \in N^+)$ is the d-dimensional physical domain, T is the upper limit of the time interval, and $\mathcal{B}$ is a representation of the boundary conditions, including the Dirichlet boundary condition, Robin boundary condition, etc.

Physics-informed neural networks are a method in which a partial differential equation and its initial boundary value conditions are added to the loss function using automatic differentiation (AD) [16]. Our objective is to find the optimal weights for which the correctly defined loss function is minimized. For Equation (2), we typically use the Mean Square Error (MSE) to construct the loss function, which is formed as follows [50]:

$$\text{Loss} = \mathcal{L}_f + \mathcal{L}_b + \mathcal{L}_i, \tag{3}$$

where

$$\mathcal{L}_f = \frac{1}{\left|\mathcal{T}_f\right|} \sum_{(\pmb{x},t)\in\mathcal{T}_f} |\mathcal{F}(\pmb{x}, t, y, y_t, \nabla y, \Delta y, \lambda)|^2,$$

$$\mathcal{L}_b = \frac{1}{|\mathcal{T}_b|} \sum_{(\pmb{x},t)\in\mathcal{T}_b} |\mathcal{B}y - g|^2,$$

$$\mathcal{L}_i = \frac{1}{|\mathcal{T}_i|} \sum_{(\pmb{x},t)\in\mathcal{T}_i} |y - h|^2,$$

where $\mathcal{T}_f$, $\mathcal{T}_b$, and $\mathcal{T}_i$ represent the set of PDE residual training points, the set of boundary training points, and the set of initial training points, respectively. The PINN framework [23] can be found in Figure 1. In some PDEs, boundary conditions can be enforced accurately and automatically by modifying the network architecture [51–54], eliminating the term associated with the boundary conditions from the loss function (3).
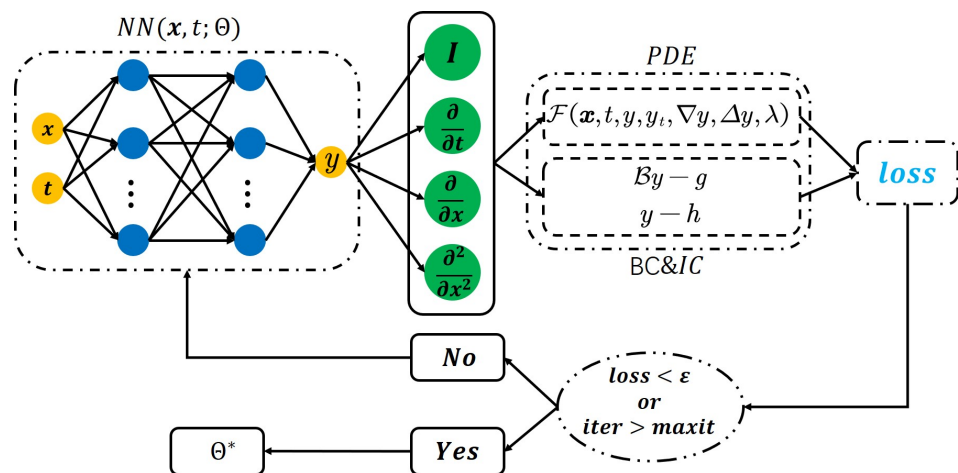

Figure 1. Schematic of PINNs for Equation (2).

3. Methodology

When PINNs are used to solve differential equations, sigmoid and tanh are typically used as nonlinear activation functions that take part in neural network training. If the input dataset is very small or very large, the sigmoid and tanh derivatives will be around 0, and

the gradient update will slow down during backpropagation [55]; the vanishing gradient problem [44] will inevitably occur. He et al. present a residual learning framework [42,43] to train deeper networks and avoid gradient vanishing. This neural network framework adopts residual machine learning every few hidden layers, and a building block is shown in Figure 2a [42]. The building block is defined as [42]:

$$S^{l+1} = \mathcal{F}\left(S^l, W\right) + S^l. \tag{4}$$

Here, S^l and S^{l+1} are the input and output vectors of the *l*-layers considered, and the function $\mathcal{F}\left(S^l, W\right)$ represents the residual mapping to be learned. The example in Figure 2a has one layer and $\mathcal{F}\left(S^l, W\right) = f_{act}(W^{l+1}S^l + b^{l+1})$.

(**a**) A building block of the residual network.

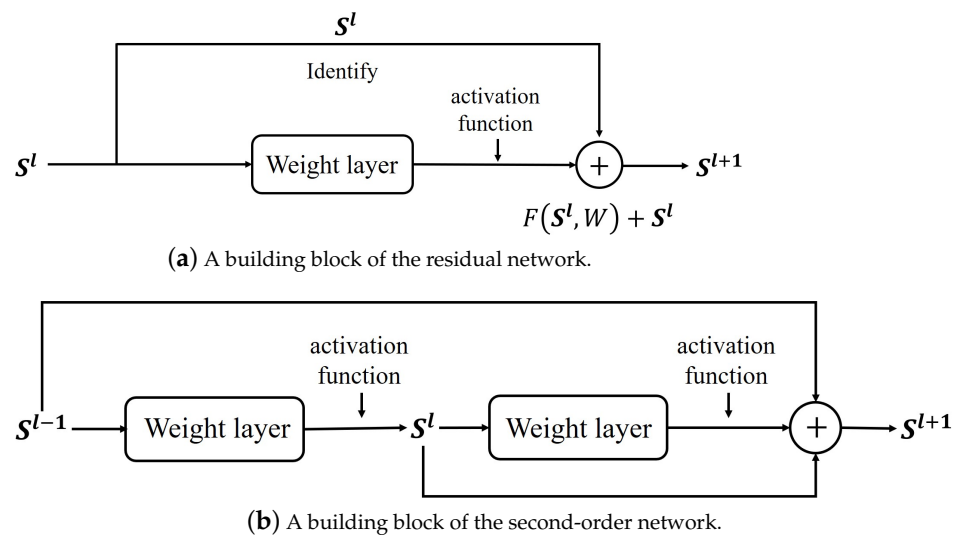(**b**) A building block of the second-order network.

Figure 2. A building block.

The residual network in Equation (4) can be seen as the forward Euler formula [45,56]:

$$\left.\frac{\partial S}{\partial t}\right|_{[l,\ l+1]} \approx \frac{S^{l+1} - S^l}{\delta_t} = \mathcal{F}\left(S^l, W\right). \tag{5}$$

Therefore, the residual network can be represented by a first-order differential equation.

Similarly, different network structures can be constructed by considering different numerical schemes for hyperbolic PDEs. For a second-order dynamical system of the form [45]:

$$\frac{\partial^2 S}{\partial t^2} = \mathcal{F}(S, W), \tag{6}$$

Ruthotto et al. established the relationship between a deep residual network and partial differential equations. They obtained a second-order convolutional neural network in the following form by discretizing the above second-order equation [45]:

$$S^{l+1} = 2S^l - S^{l-1} + \delta_t^2 \mathcal{F}\left(S^l, W\right). \tag{7}$$

The building block of the second-order network is shown in Figure 2b. In this study, the structure of the proposed second-order neural network was constructed using a parametric light wave equation in the following form [57]:

$$\frac{\partial S}{\partial t} + \varepsilon \frac{\partial^2 S}{\partial t^2} = \mathcal{F}(S, W). \tag{8}$$

We can obtain this by discretizing Equation (8) using the central difference:

$$\left(\frac{1}{2\delta_t} + \frac{\varepsilon}{\delta_t^2}\right)S^{l+1} = \mathcal{F}\left(S^l, W\right) + \frac{2\varepsilon}{\delta_t^2}S^l + \left(\frac{1}{2\delta_t} - \frac{\varepsilon}{\delta_t^2}\right)S^{l-1}. \tag{9}$$

It can be seen that Formulas (7) and (9) have the same form, except that the coefficients of each term are different. Therefore, the network represented by Formula (9) is still a second-order network.

We used a network structure like that in Formula (9) for PINNs. For this network structure, we used a simple linear combination and nonlinear mapping of the activation function for the first hidden layer, the residual connection shown in Figure 2a for the second hidden layer, and the connection mode given by Equation (9) and Figure 2b for other hidden layers. Finally, we added a linear output layer. The feedforward neural network can be recursively expressed as follows:

$$
\begin{aligned}
\text{input} &: S^0 = x; \\
\text{hidden layer } 1 &: S^1 = f_{act}(W^1 S^0 + b^1); \\
\text{hidden layer } 2 &: S^2 = \delta_t f_{act}(W^2 S^1 + b^2) + S^1; \\
\text{hidden layer } l &: S^l = \frac{2\delta_t^2}{\delta_t + 2\varepsilon}f_{act}(W^l S^{l-1} + b^l) + \frac{4\varepsilon}{\delta_t + 2\varepsilon}S^{l-1} + \frac{\delta_t - 2\varepsilon}{\delta_t + 2\varepsilon}S^{l-2}, \text{for } 3 \le l \le L-1; \\
\text{output} &: S^L = W^L S^{L-1} + b^L.
\end{aligned}
\tag{10}
$$

Here, $S^L = y(x, \Theta)$ is the neural network's output. It can be seen from the above expression that, compared with Equation (1), when the neural network has the same depth and width, the neural network expressed in Equation (10) has a stronger degree of nonlinearity, which can be said to be more expressive. This is a kind of neural network structure based on the difference scheme, and this is also a feasible scheme if pursuing structural diversity in neural networks. Ruthotto et al. [45] proved that the forward propagation of a second-order network is stable under certain conditions, which provides a foundation for our subsequent theoretical research. We hope that the continuous deep neural network (DNN) model can lead to some new breakthroughs in theory and algorithms, help simplify network architecture design, reduce trial and error, and improve training accuracy.

The adaptive activation function strategy [25–27] accelerates convergence and improves accuracy. We utilized Neuron-wise locally adaptive activation functions (N-LAAFs) [26] to enhance the performance of the neural network. It is assumed that x^{ij} is the input of the j-th neuron of the i-th hidden layer, y^{ij} is the output of this neuron, and a^{ij} is a hyperparameter that is added to this neuron in the following format [26]:

$$y^{ij} = f_{act}(na^{ij}x^{ij}), i = 1, 2, \cdots, L-1, j = 1, 2, \cdots, N_j. \tag{11}$$

In this case, $n \in N^+$ is a predetermined scale factor.

When the PDE residual equals zero, the corresponding gradient must equal zero. The gradient-enhanced PINNs (gPINNs) proposed in [46] force the derivatives of the PDE residual to be zero as well, i.e.,

$$\nabla \mathcal{F}(x, t) = \left(\frac{\partial \mathcal{F}}{\partial x_1}, \cdots, \frac{\partial \mathcal{F}}{\partial x_d}, \frac{\partial \mathcal{F}}{\partial t}\right) = 0, (x, t) \in \Omega \times (0, T). \tag{12}$$

This will add an additional loss term [46] to the loss function (3):

$$\mathcal{L}_g = \sum_{k=1}^{d+1}\left(\frac{1}{|\mathcal{T}_{g_k}|}\sum_{(x,t)\in\mathcal{T}_{g_k}}|\nabla_k\mathcal{F}|^2\right). \tag{13}$$

Here, $\nabla_k \mathcal{F}$ is the k-th element of $\nabla \mathcal{F}(\boldsymbol{x}, t)$, and $\mathcal{T}_{g_k}$ is the set of residual points for the derivative $\nabla_k \mathcal{F}$. In this paper, let us set $\mathcal{T}_{g_1} = \cdots = \mathcal{T}_{g_{d+1}} = \mathcal{T}_g$, $\mathcal{T}_g$, and $\mathcal{T}_f$ as the same dataset.

In order to reduce the high computational cost caused by gPINNs, we used the deep mixed residual method (MIM) [47]. We illustrate this with the heat equation, where $\mathcal{F}$ in Equation (2) is $\mathcal{F} := u_t - u_{x_1 x_1} - u_{x_2 x_2} - f = 0$. Then, we introduce the auxiliary variables p and q, and let $p := u_{x_1}$ and $q := u_{x_2}$; then, the PDEs' governing equations are rewritten as follows:

$$\begin{cases} p = u_{x_1}, q = u_{x_2} \\ u_t - p_{x_1} - q_{x_2} = f \end{cases}. \tag{14}$$

Here, the auxiliary variables p and q are the outputs of the feedforward neural network. In this paper, we use MIM and make $v = \mathcal{F}(\boldsymbol{x}, t, y, y_t, \nabla y, \Delta y, \lambda)$, and then the loss term $\mathcal{L}_g$ is rewritten as:

$$\mathcal{L}_g = \frac{1}{|\mathcal{T}_g|} \sum_{(\boldsymbol{x}, t) \in \mathcal{T}_g} |\mathcal{F} - v|^2 + \frac{1}{|\mathcal{T}_g|} \sum_{k=1}^{d+1} \left(\sum_{(\boldsymbol{x}, t) \in \mathcal{T}_g} |\nabla_k v|^2 \right). \tag{15}$$

Finally, we rewrite Formula (3) as follows:

$$\text{Loss} = \mathcal{L}_f + \mathcal{L}_b + \mathcal{L}_i + \mathcal{L}_g. \tag{16}$$

The algorithm framework can be found in Figure 3 and Algorithm 1. In this study, for neural networks $y(\boldsymbol{x}, \Theta_1)$ and $v(\boldsymbol{x}, \Theta_2)$, the same structural parameters are used unless otherwise specified. We used L-BFGS [58] as the optimization algorithm of the neural network to minimize the loss function (16) in the neural network training process. The initialization method for trainable parameters' weights is the Xavier method [59], and the biases are initialized to zero. All randomly sampled point locations were generated using a Latin Hypercube Sampling strategy [60]. Meanwhile, the initial parameter selection for the hyperparameters n and a^{ij} are chosen such that $na^{ij} = 1$ [25–27].
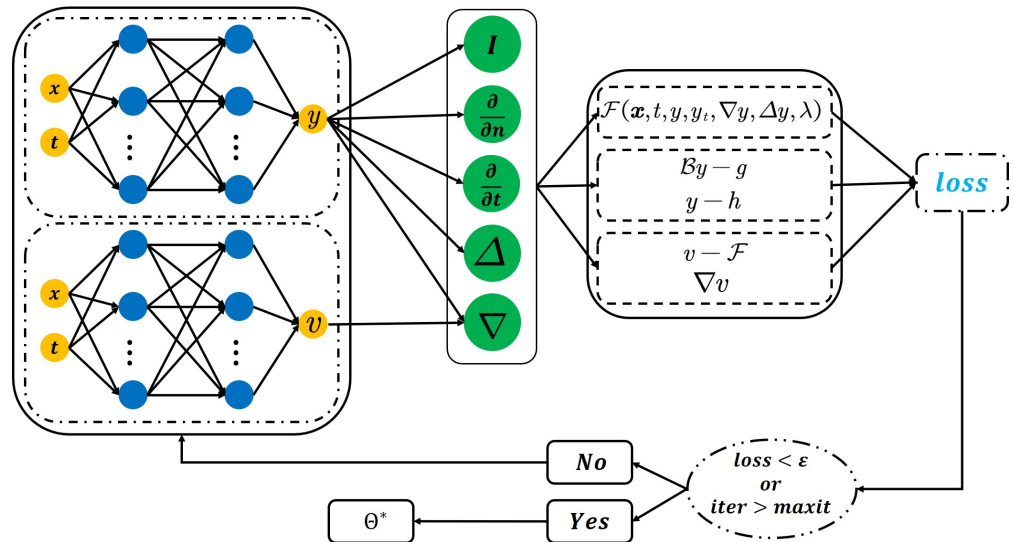

Figure 3. Schematic of gPINNs based on MIM for the Equation (2).

Algorithm 1 shows the specific steps of the presented method compared with the existing PINN algorithm [14,26,46,47]. The main difference is that in the second and third steps, using two neural networks, a neural network is used to approximate the solutions of differential equations, and another is used to approximate the PDEs' governing equations.

Algorithm 1 gPINN algorithm based on MIM.

1. Construct the training dataset $\mathcal{T}_f$, $\mathcal{T}_b$, $\mathcal{T}_i$, and $\mathcal{T}_g$ for the PDE residual, boundary conditions, initial conditions, and PDE residual gradient.
2. Construct two neural networks $y(\boldsymbol{x}, \Theta_1)$ and $v(\boldsymbol{x}, \Theta_2)$ with the parameter $\Theta = \{\Theta_1, \Theta_2\}$ using (10), where $\Theta_k = \left\{W_k^i, b_k^i\right\}_{i=1}^{L} \cup \left\{a_k^{ij}\right\}_{i=1:L-1}^{j=1:N_j}, (k = 1, 2)$.
3. Construct the PDE residual $\mathcal{F}$ and the PDE residual gradient $\widetilde{\nabla}\mathcal{F}$ using neural networks $y(\boldsymbol{x}, \Theta_1)$ and $v(\boldsymbol{x}, \Theta_2)$ in the governing equations using automatic differentiation.
4. Construct the loss function using the Mean Square Error.
5. Train the neural networks to find the best parameters Θ^* to minimize the loss function.

---

4. Numerical Examples

In this section, we give several examples to check the efficiency of the above method in the resolution of partial differential equations. On the test set, we used the Rel.L^2 error to evaluate the performance of neural network training [61]:

$$Rel.L^2\text{error} = \frac{\sqrt{\sum_{i=1}^{N}\left|y^*\left(\boldsymbol{x}^i, t^i\right) - u\left(\boldsymbol{x}^i, t^i\right)\right|^2}}{\sqrt{\sum_{i=1}^{N}\left|u\left(\boldsymbol{x}^i, t^i\right)\right|^2}}, \tag{17}$$

where y^* represents an optimal approximative solution derived from neural network training, i represents the test point number, and N represents the number of test points. In this part of the forward and inverse problem examples, the default values of ε and δ_t in Equation (10) are both set to 1.

4.1. Forward Problem

4.1.1. Burgers' Equation

Burgers' equation is a nonlinear partial differential equation that simulates the propagation and reflection of a shock wave. It is used in fluid mechanics, nonlinear acoustics, gas dynamics, etc. First, we chose the 2D viscous Burgers' equation as the first example in this document, which takes the following form [62]:

$$\begin{cases} u_t + uu_{x_1} + uu_{x_2} = \nu(u_{x_1 x_1} + u_{x_2 x_2}), \ (x_1, x_2, t) \in (-10, 10)^2 \times (0, 10), \\ \qquad\qquad u = g, \ (x_1, x_2, t) \in \partial(-10, 10)^2 \times (0, 10), \\ \qquad\qquad u = h, \ (x_1, x_2, t) \in (-10, 10)^2 \times \{0\}, \end{cases} \tag{18}$$

where the parameter ν is the diffusion coefficient, the viscosity of the material, and we set the parameter $\nu = 0.1$. The initial and boundary conditions are derived from the exact solution: $u(x_1, x_2, t) = \frac{1}{1+\exp((x_1+x_2-t)/2\nu)}$. Here, we aim to infer the entire spatiotemporal solution of the above Equation (18) for this example. For the neural network structure, we used five hidden layers, each with 50 neurons; the scale factor n is 50, and the tanh function is the activation function for this example. Meanwhile, for training, we constructed a training set of $3,000$ boundary training points, 500 initial training points, and $10,000$ PDE residual training points. We used the above configuration to form a neural network to obtain an approximate solution that satisfies Equation (18).

Next, we show the results of training. The Rel.L^2 error curve of Equation (18) on the test set over time is shown in Figure 4. It can be seen that the accuracy of the proposed method is about $7 \sim 8$ times higher than that of PINNs. Meanwhile, the exact solution, predicted solution, and error distributions at four time points, $t = 2.5$, $t = 5.0$, $t = 7.5$, and $t = 10.0$, are plotted in Figure 5. It can be seen in the error distribution diagram that the areas with large errors are mainly concentrated near the discontinuous point of the function, which can be improved by using a residual-based adaptive refinement (RAR)

method [23]. This will be a future research direction for discontinuous solutions and large gradient changes. Clearly, the proposed method provides a good approximation of the equation. We present a comparative analysis in the Appendix A, including the benchmark PINNs method, adaptive activation function, gPINNs method, and the method mentioned in section 3 of this paper.
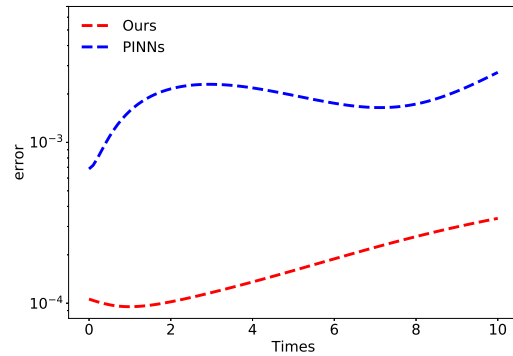

Figure 4. Burgers' equation: Rel.L^2 error over time.

(**a**) t = 2.5 (Rel.L^2 error: 1.0867×10^{-4})

(**b**) t = 5.0 (Rel.L^2 error: 1.5957×10^{-4})

(**c**) t = 7.5 (Rel.L^2 error: 2.4078×10^{-4})

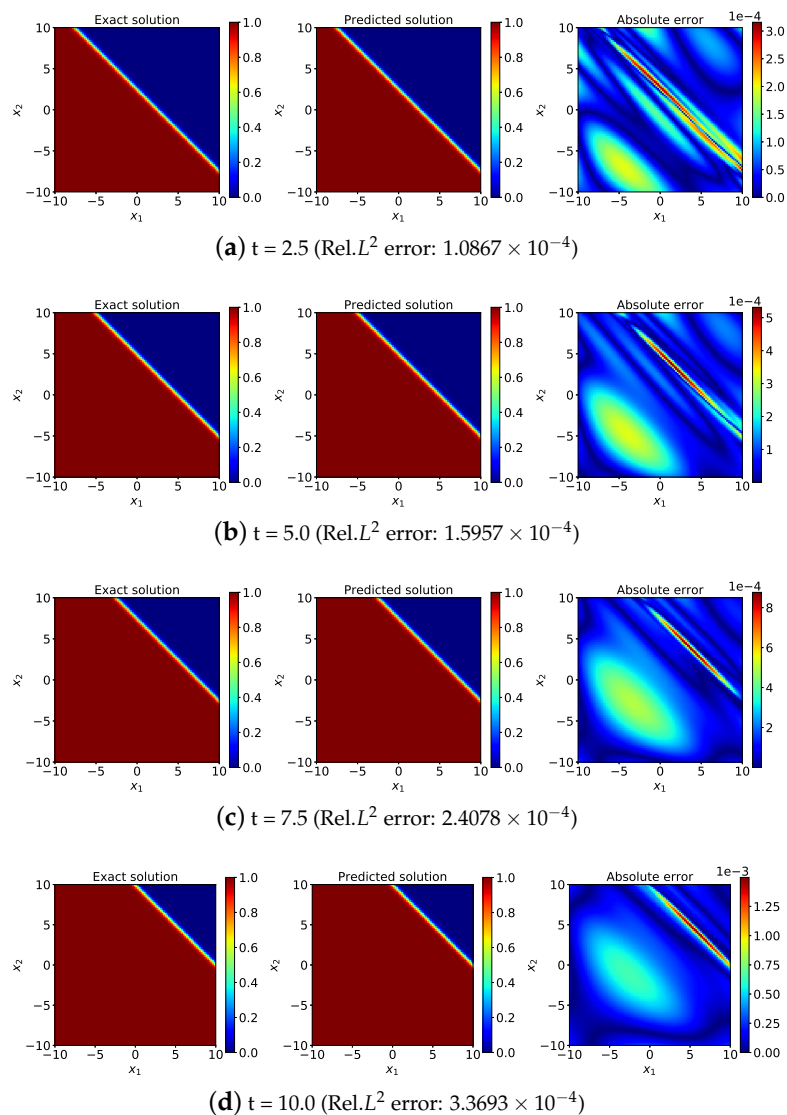(**d**) t = 10.0 (Rel.L^2 error: 3.3693×10^{-4})

Figure 5. Burgers' equation: exact solution, predicted solution, and absolute pointwise error at different times.

4.1.2. Taylor–Green Flow

When we consider incompressible Navier–Stokes equations (N-S equations), the governing equations can be written in a dimensionless form as [4]:

$$\begin{cases} \boldsymbol{u}_t + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} = -\nabla p + \nu \Delta \boldsymbol{u} + \boldsymbol{f}, \\ \nabla \cdot \boldsymbol{u} = 0, \end{cases} \tag{19}$$

where $\boldsymbol{u} = (u_1, u_2)$ is a velocity vector field, p denotes the fluid pressure, $\boldsymbol{f}$ is a given source term, $\nu = 1/Re$ denotes the kinematic viscosity of the fluid, and Re is the Reynolds number.

We proceeded to model a time-dependent flow problem, i.e., the Taylor–Green flow, using the proposed method. The computational domain is defined as $(x_1, x_2, t) \in (-\pi, \pi)^2 \times (0, 10)$. The exact solution is set as follows [21]:

$$\begin{cases} u_1 = -\cos(x_1)\sin(x_2)e^{-2t\nu}, \\ u_2 = \sin(x_1)\cos(x_2)e^{-2t\nu}, \\ p = -\dfrac{1}{4}[\cos(2x_1) + \cos(2x_2)]e^{-4t\nu}. \end{cases} \tag{20}$$

The second equation in Equation (19) is the continuity equation for an incompressible fluid and describes the mass conservation of the fluid. From the stream function expression of the two-dimensional incompressible Navier–Stokes equations, one can use a neural network to represent a scalar stream function $\psi(x_1, x_2, t)$ and assume $u_1 = \partial\psi/\partial x_2$ and $u_2 = -\partial\psi/\partial x_1$. In this way, we can only consider the residuals of the momentum equation since the continuity of the equation is already fully satisfied.

In simulations, we set $Re = 100$, and the initial and boundary conditions were obtained using Equation (20). In this example, the swish function is used as the activation function of the neural network, where the value of the scale factor n is 20. The training set of the neural network is composed of $10,000$ residual training points, $4,000$ boundary training points, and 1000 initial training points. We used a neural network with five hidden layers of 50 neurons each to simulate the dynamic behavior described by Equation (19). In Figure 6, we show the Rel.L^2 error curves of the velocity component compared to those of the PINNs in the left and right panels, respectively. Additionally, the exact solution $U = \sqrt{u_1^2 + u_2^2}$, the predicted solution $U^* = \sqrt{(u_1^*)^2 + (u_2^*)^2}$, and the error distributions $|U - U^*|$ at four time points are shown in Figure 7. It can be seen that the area with large errors is concentrated in the center, which is caused by the relative error used in the test set in this work, and the value of the denominator at the origin is 0. It can be seen that the proposed method achieves better accuracy than the PINNs in this example.
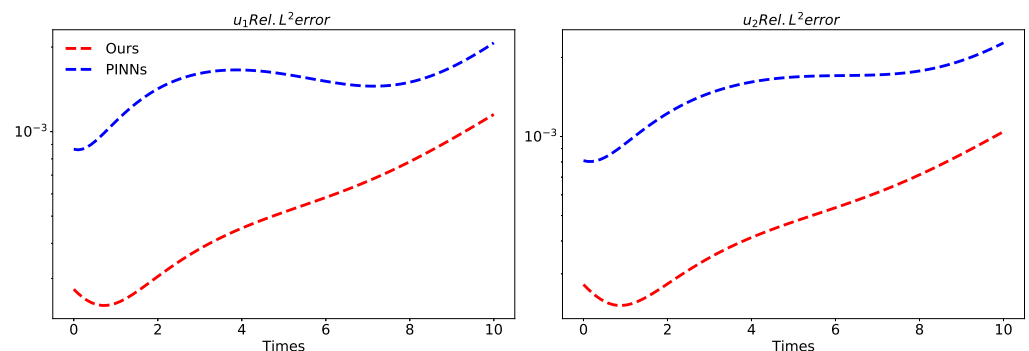

Figure 6. Taylor–Green flow: Rel.L^2 error over time.

(**a**) t = 2.5 (Rel.L^2 error: 2.6808×10^{-4})

(**b**) t = 5.0 (Rel.L^2 error: 3.9734×10^{-4})

(**c**) t = 7.5 (Rel.L^2 error: 5.5112×10^{-4})

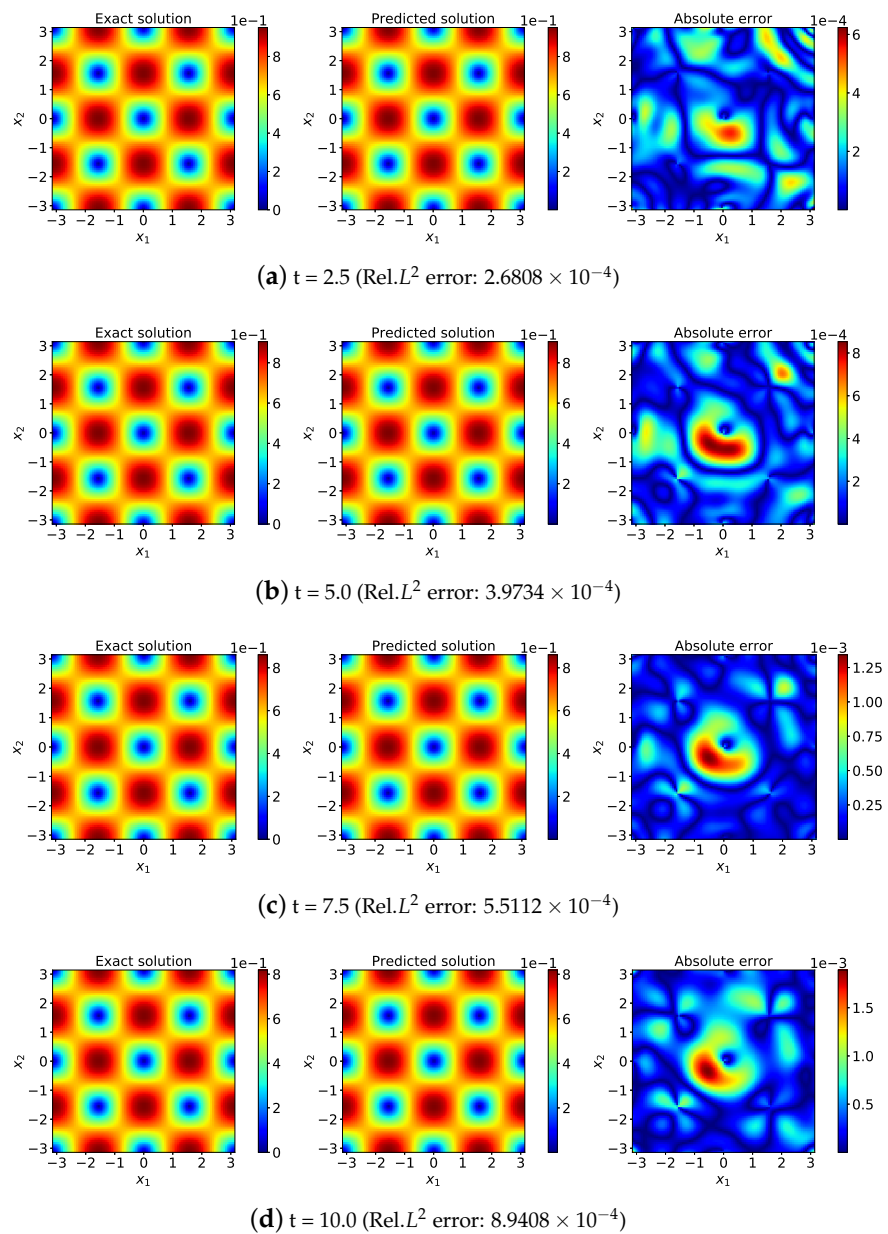(**d**) t = 10.0 (Rel.L^2 error: 8.9408×10^{-4})

Figure 7. Taylor–Green flow: exact solution, predicted solution, and absolute pointwise error at different times.

4.1.3. Cylinder Wake

For the third example of the forward problem, we simulated a 2D cylinder wake [63] using the method proposed in this paper. The PDEs' governing equations are given by Equation (19), and the Reynolds number is $Re = 3900$. High-fidelity data (the high-fidelity data are from https://github.com/Shengfeng233/PINN-for-turbulence, accessed on 25 September 2022) were used as a reference and provided the boundary and initial data. In this example, the training dataset contains $40,000$ residual training points, $10,000$ boundary training points, and 5000 initial training points. The structure of the neural network used has 6 hidden layers and 100 neurons in each layer. The tanh function is used as the activation function of the neural network, where the value of the scale factor n is 1. In Figure 8, we compare the Rel.L^2 error of the two velocity components in the left and right graphs, respectively, and in Figure 9, we show the predicted solution obtained by neural network training using the above setup and the corresponding error distribution. It can be seen that the proposed method improves the accuracy by $10\% \sim 30\%$ compared with

PINNs. In this example, we found an interesting phenomenon: the error decreases with time on the whole, which seems counterintuitive. The exploration of this problem will help us design better algorithms to further improve the performance of the neural network.
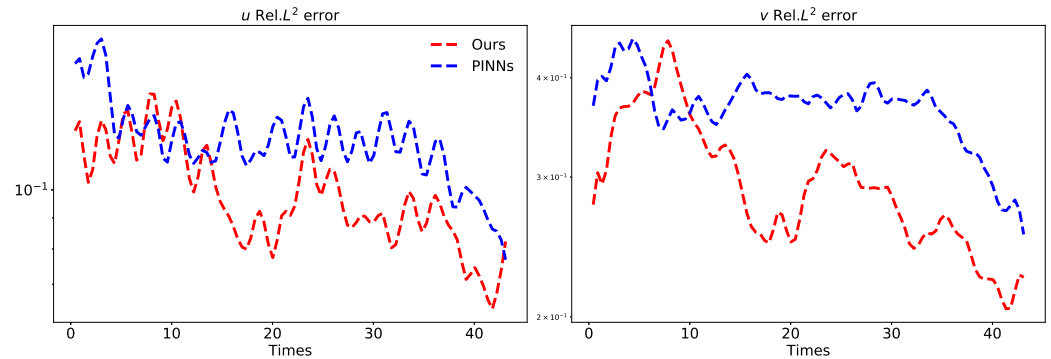

Figure 8. Cylinder wake: Rel.L^2 error over time.

(**a**) t = 8.2713 (Rel.L^2 error: 1.5684×10^1)

(**b**) t = 16.9780 (Rel.L^2 error: 1.0325×10^{-1})

(**c**) t = 25.6847 (Rel.L^2 error: 1.2424×10^{-1})

(**d**) t = 34.3913 (Rel.L^2 error: 1.1199×10^{-1})

(**e**) t = 43.0980 (Rel.L^2 error: 7.5796×10^{-2})
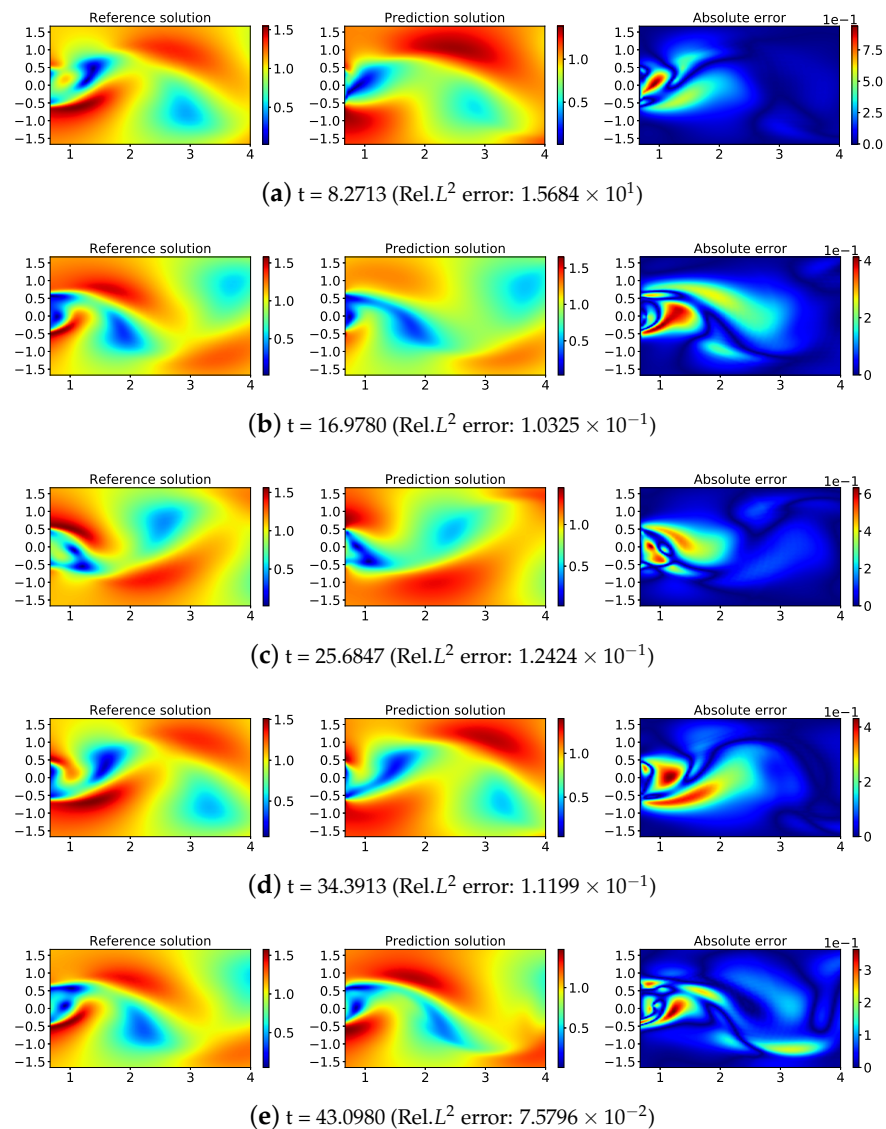
Figure 9. Cylinder wake: reference solution, predicted solution, and absolute pointwise error at different times.

4.1.4. Allen–Cahn Equation

The Allen–Cahn equation is a second-order nonlinear parabolic partial differential equation used to describe the phase separation process in binary alloys and anti-phase boundary movement in crystals [64]. It has been used extensively in different models. We consider an unsteady 3D Allen–Cahn equation as follows:

$$
\begin{cases}
u_t - (u_{x_1 x_1} + u_{x_2 x_2} + u_{x_3 x_3}) + u^3 - u = f, (x_1, x_2, x_3, t) \in (-1,1)^3 \times (0,1), \\
u = g, (x_1, x_2, x_3, t) \in \partial(-1,1)^3 \times (0,1), \\
u = h, (x_1, x_2, x_3, t) \in (-1,1)^3 \times \{0\}.
\end{cases}
\tag{21}
$$

The source term f and the initial boundary value conditions g and h can be simply obtained from the exact solution: $u(x_1, x_2, x_3, t) = e^{-t} \sin(x_1 + x_2 + x_3)$. We used a neural network with five hidden layers of 50 neurons each. In general, the neural network should have sufficient approximation capacity to consider the predicted complexity of u. We obtained the training data points of the neural network by random sampling. In particular, there are 500 initial training points, 2000 boundary training points, and 5000 PDE residual training points in the neural network training set. The value of the hyperparameter a^{ij} is set to 0.1, and the swish function is used as the activation function. The Rel.L^2 error curves of Equation (21) on the test set are plotted in Figure 10. Compared with PINNs, the accuracy is greatly improved, and the generalization is better. Meanwhile, the exact solution, predicted solution, and error distribution at four time points and $x_3 = 0$ are plotted in Figure 11. Using just a handful of initial data, the method is able to accurately capture the complex nonlinear behavior of the Allen–Cahn equation. Based on the results, it can be seen that the proposed method can accurately produce the solution.
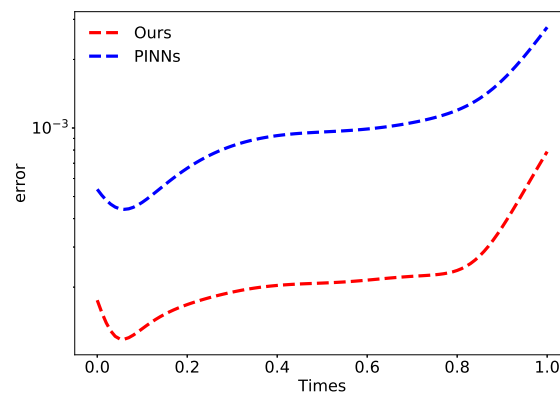

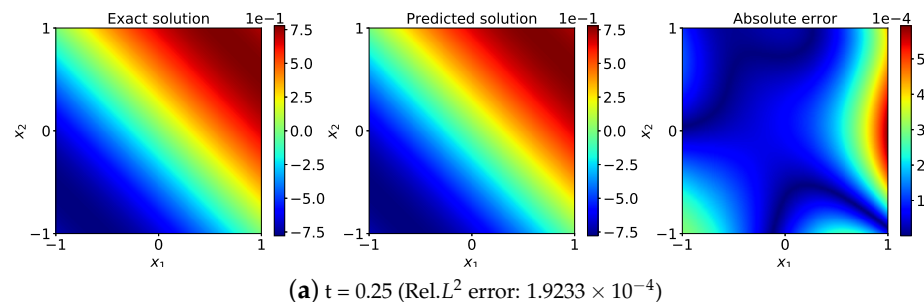**Figure 10.** Allen–Cahn equation: Rel.L^2 error over time.

(**a**) t = 0.25 (Rel.L^2 error: 1.9233×10^{-4})

Figure 11. *Cont.*

(b) t = 0.50 (Rel.L^2 error: 1.8332×10^{-4})

(c) t = 0.75 (Rel.L^2 error: 2.0538×10^{-4})

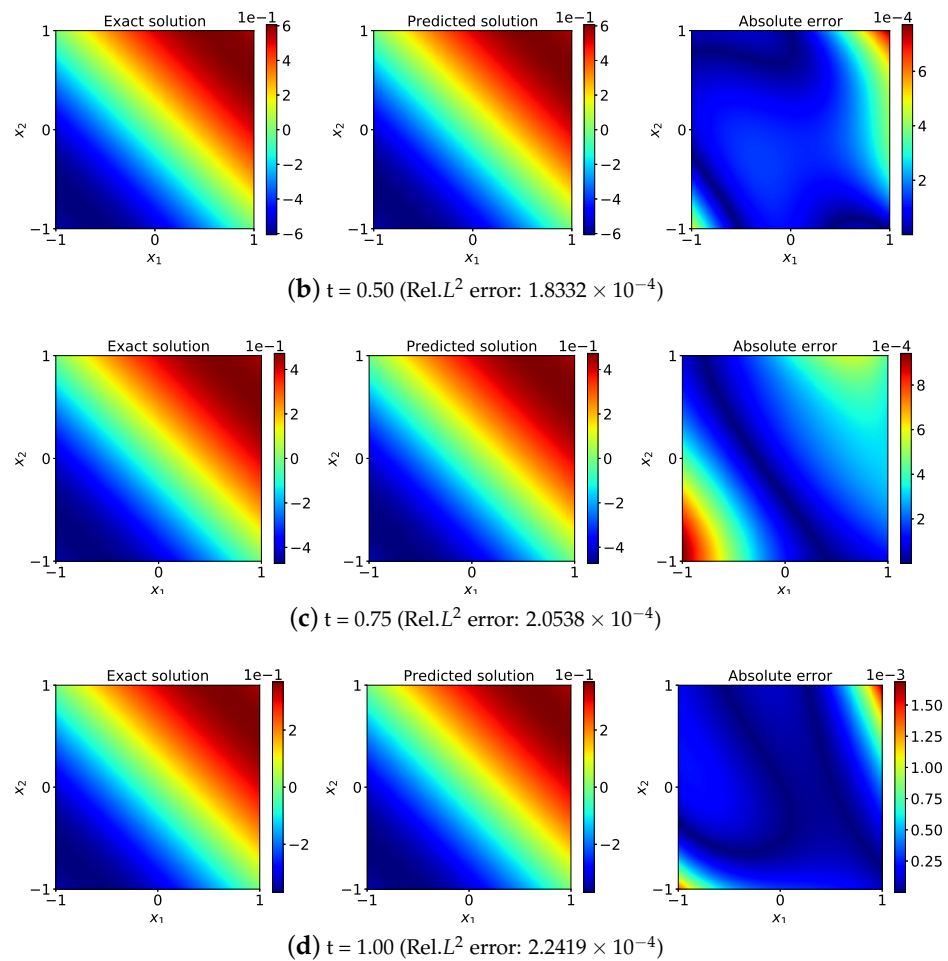**(d)** t = 1.00 (Rel.L^2 error: 2.2419×10^{-4})

Figure 11. Allen–Cahn equation: exact solution, predicted solution, and absolute pointwise error at different times when $x_3 = 0$.

4.1.5. Convection–Diffusion–Reaction Equation

Convection–diffusion–reaction (CDR) equations are basic partial differential equations for the simulation of heat and mass transfer. Their numerical solutions are critical research directions for numerical partial differential equations and computational fluid mechanics, with a broad range of applications [65,66]. For instance, we take the 3D unsteady nonlinear CDR equation as follows:

$$\begin{cases} u_t + \beta \cdot \nabla u - \nu \Delta u + \gamma u^2 = f, (x_1, x_2, x_3, t) \in (-1,1)^3 \times (0,1), \\ \qquad\qquad\qquad\qquad u = g, (x_1, x_2, x_3, t) \in \partial(-1,1)^3 \times (0,1), \\ \qquad\qquad\qquad\qquad u = h, (x_1, x_2, x_3, t) \in (-1,1)^3 \times \{0\}, \end{cases} \qquad (22)$$

where the unknown function u usually represents the temperature or concentration of the substance being transferred, $\nu \geq \nu_0 > 0$ is the diffusion coefficient, β is the convective coefficient, and γ is the reaction coefficient. When $\nu \ll \max |\beta|$, Equation (22) is called the convection-dominated diffusion problem. In this example, we consider a convection-dominated diffusion problem and use the coefficients $\nu = 10^{-4}$, $\beta = (1,1,1)$, and $\gamma = 1$, and the exact solution is given as $u(x_1, x_2, x_3, t) = e^{-t} \arctan(x_1) \arctan(x_2) \arctan(x_3)$. All the right-hand terms can be obtained from the exact solution.

We used a training set in which the number of residual points is 5000, 500 data points on $u|_{t=0}$ are randomly parsed from the exact solution, and 1000 boundary training points are also given for the boundary conditions of PDEs. We used a four-layer feedforward neural network with 50 neurons in each hidden layer; the value of the scale factor n is 100,

and the activation function is swish. The Rel.L^2 error curve on the test set can be seen in Figure 12. Meanwhile, the exact solution, predicted solution, and error distribution at four time points and $x_3 = 1.00$ are plotted in Figure 13. It is clearly observed that solutions with good precision are obtained for this equation.
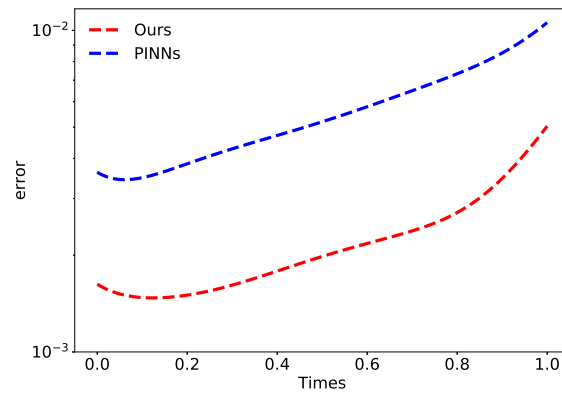

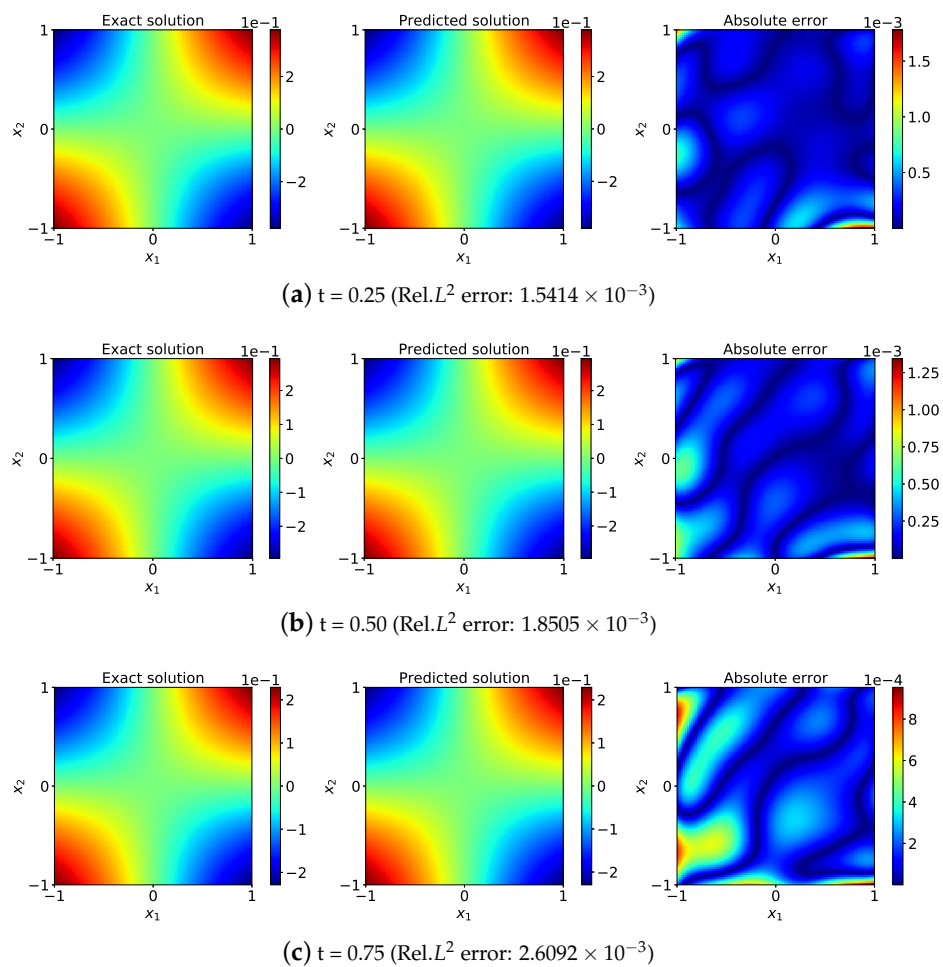**Figure 12.** CDR equation: Rel.L^2 error over time.

(**a**) t = 0.25 (Rel.L^2 error: 1.5414×10^{-3})

(**b**) t = 0.50 (Rel.L^2 error: 1.8505×10^{-3})

(**c**) t = 0.75 (Rel.L^2 error: 2.6092×10^{-3})

Figure 13. *Cont.*

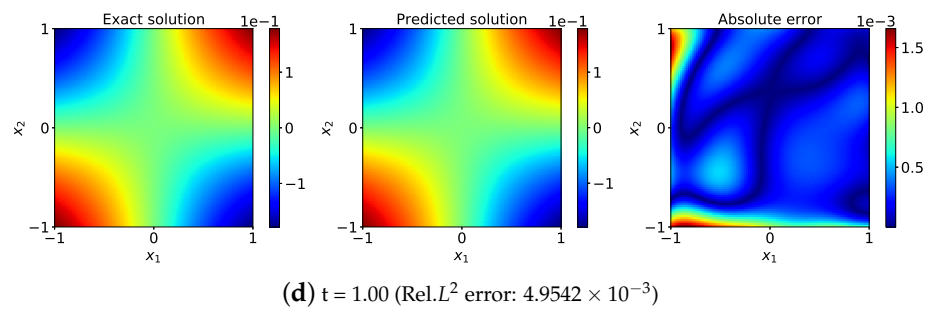**(d)** t = 1.00 (Rel.L^2 error: 4.9542×10^{-3})

Figure 13. CDR equation: exact solution, predicted solution, and absolute pointwise error at different times when $x_3 = 1$.

4.2. Inverse Problem

For the inverse problem, the parameter λ in Equation (2) is often unknown. With only a small set of observed data $\mathcal{T}_u$, the parameter λ can be determined while predicting the entire flow field. To achieve this, we simply add additional terms [23,46] to the loss function (16):

$$\mathcal{L}_u = \frac{1}{|\mathcal{T}_u|} \sum_{(\boldsymbol{x},t) \in \mathcal{T}_u} |u - y|^2. \tag{23}$$

4.2.1. Kolmogorov–Petrovskii–Piskunov-Equation

The Kolmogorov–Petrovskii–Piskunov equation (KPP equation) [67] is a kind of nonlinear reaction–diffusion equation commonly used in heat conduction, combustion theory, biology, ecology and other fields. The basic form of the KPP equation is:

$$u_t - Du_{x_1 x_1} - au - bu^m = 0, \ (x,t) \in (-\pi, \pi) \times (0,1), \tag{24}$$

where a, b, D, and m are arbitrary constants, and m is not equal to 1. The KPP equation has an exact solution as follows [67]:

$$u(x_1,t) = [\beta + \exp(\lambda t + \frac{\alpha x_1}{\sqrt{D}})]^{\frac{2}{1-m}},$$

where the parameters are given by:

$$\beta = \sqrt{-\frac{b}{a}},$$

$$\alpha = \sqrt{\frac{a(1-m)^2}{2(m+1)}},$$

$$\lambda = \frac{a(1-m)(m+3)}{2(m+1)}.$$

In this example, we set $m = 3$ and the parameters D, a, and b as unknown parameters in Equation (24). Given scattered and potentially noisy data, our goal is to identify the unknown parameters D, a, and b, as well as to obtain a qualitatively accurate reconstruction of the entire flow field $u(x_1,t)$. To illustrate the effectiveness of our approach and to highlight the ability of our method to learn from scattered and scarce training data, we created a training dataset by randomly generating $N_{train} = 2000$ points across the entire spatiotemporal domain from the exact solution corresponding to $D = 1$, $a = 1$, and $b = -1$. The neural network architecture used here consists of four hidden layers with 50 neurons in each layer; we set the value of the scale factor n to be equal to 50 in Equation (11), and we chose the sigmoid function as the activation function for this example. Upon training, the network was calibrated to predict the entire solution $u(x_1,t)$, as well as the unknown parameters D, a, and b that define the underlying dynamics. The network can identify

the underlying partial differential equations with remarkable accuracy, even when the scattered training data are corrupted by 1% and 5% uncorrelated noise. The results are summarized in Table 1 and in Figure 14, which shows the parameters D, a, and b with the iteration number. It is observed that results with good precision are obtained.

Table 1. KPP equation: Correct values obtained by learning D, a, and b and percentage errors in the identified parameters D, a, and b when the training data are corrupted by different noise levels.

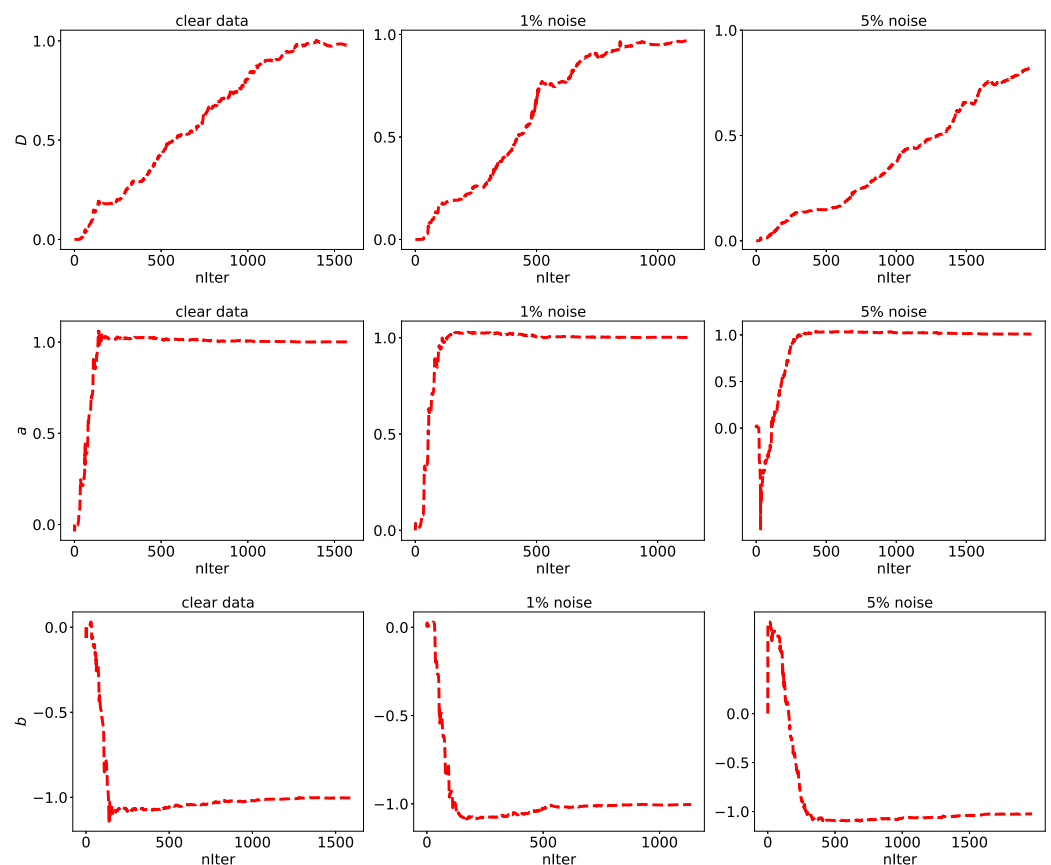|	Clear Data	1% Noise	5% Noise
Exact		$(D, a, b) = (1.000, 1.000, -1.000)$	
Identified	$(0.977, 1.001, -1.003)$	$(0.972, 1.002, -1.003)$	$(0.819, 1.010, -1.024)$
Error	$(2.26 \times 10^{-2}, 1.10 \times 10^{-3}, 3.41 \times 10^{-3})$	$(2.81 \times 10^{-2}, 1.52 \times 10^{-3}, 3.89 \times 10^{-3})$	$(1.81 \times 10^{-1}, 9.82 \times 10^{-3}, 2.38 \times 10^{-2})$

Figure 14. KPP equation: change curves of the parameters D, a, and b with the iteration number.

4.2.2. Burgers' Equation

Then, we take Equation (18) as an example and rewrite it in the following form:

$$u_t + \lambda_1(uu_{x_1} + uu_{x_2}) = \lambda_2(u_{x_1 x_1} + u_{x_2 x_2}), \ (x_1, x_2, t) \in (-1, 1)^2 \times (0, 1). \quad (25)$$

We set the parameters λ_1 and λ_2 as unknown parameters; we still used the same exact solution. To demonstrate how effective the approach is, we created a training dataset by randomly generating $N_{train} = 2000$ points on the whole space–time domain from the exact solution with $\lambda_1 = 1$ and $\lambda_2 = 0.1$. We used a neural network with four hidden layers, each with 50 neurons, to predict u in the entire computational domain with only a small number of high-fidelity datasets and, at the same time, to determine the unknown parameters λ_1 and λ_2 in Equation (25). The tanh function is set as the activation function, and the value of the scale factor n is 20. It is seen in Table 2 that, in this case, the method can correctly identify the unknown parameters λ_1 and λ_2 with very high accuracy, even when

the training data are corrupted by noise. The predictions remain robust even when the training data are corrupted by 1% and 5% uncorrelated Gaussian noise. Finally, Figure 15 shows the parameters λ_1 and λ_2 with the iteration number; the results show that the neural network can effectively find the unknown parameters in the equation. The results show that the proposed method makes it possible to obtain great accuracy.

Table 2. Burgers' equation: correct values obtained by learning λ_1 and λ_2 and percentage errors in the identified parameters λ_1 and λ_2 when the training data are corrupted by different noise levels.

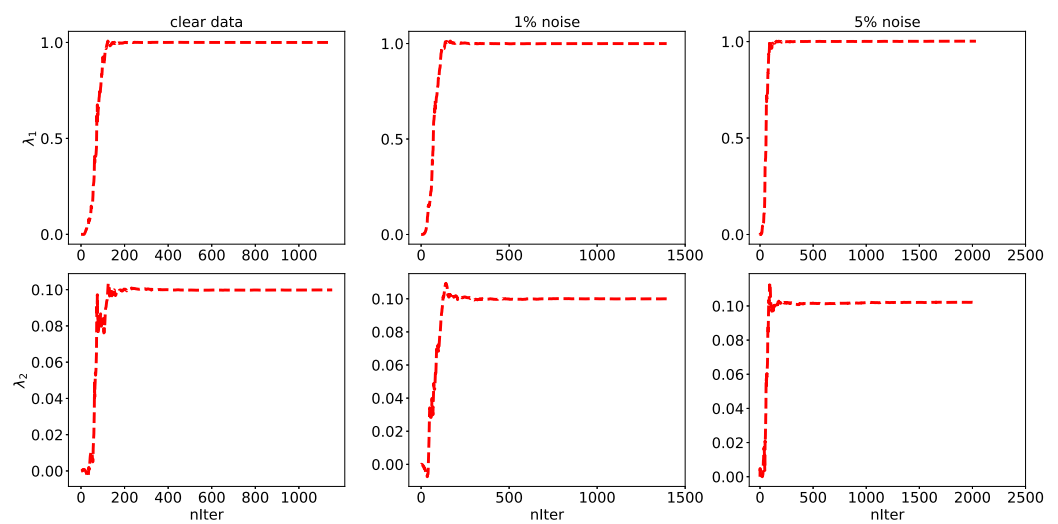|	Clear Data	1% Noise	5% Noise
Exact		$(\lambda_1, \lambda_2) = (1.000, 0.100)$	
Identified	$(1.0001, 0.0999)$	$(0.9991, 0.1001)$	$(1.0020, 0.1022)$
Error	$(1.81 \times 10^{-4}, 3.21 \times 10^{-4})$	$(5.95 \times 10^{-4}, 7.39 \times 10^{-3})$	$(2.02 \times 10^{-3}, 2.16 \times 10^{-2})$

Figure 15. Burgers' equation: change curves of the parameters λ_1 and λ_2 with the iteration number.

4.2.3. Diffusion–Reaction Equation

Finally, we consider a 3D unsteady nonlinear diffusion–reaction equation [68] given by:

$$u_t - \lambda_1 (u_{x_1 x_1} + u_{x_2 x_2} + u_{x_3 x_3}) + \lambda_2 u^2 = f, (x_1, x_2, x_3, t) \in (-1, 1)^3 \times (0, 1), \quad (26)$$

where u is the concentration of the substance, and $\lambda_1 > 0$ and $\lambda_2 > 0$ are the diffusion coefficient and reaction coefficient, respectively. The source term f can be obtained from the exact solution: $u = e^{-t} \sin(\pi x_1) \sin(\pi x_2) \sin(\pi x_3)$. We randomly selected 5000 training data points from the whole space–time region as the training data for the neural network and constructed a neural network with five hidden layers and 50 neurons in each layer. With only a small amount of data, we simulated the whole flow field and determined the unknown parameters λ_1 and λ_2 in the equation. We used the swish function as the activation function, where the scale factor n takes the value 20. Table 3 shows the values of λ_1 and λ_2 'discovered' through neural network training. Additionally, Figure 16 shows the change curves of parameter values in the process of neural network training. It can be seen that neural networks can effectively discover the unknown parameters in Equation (26).

Table 3. Diffusion–reaction equation: correct values obtained by learning λ_1 and λ_2 and percentage errors in the identified parameters λ_1 and λ_2 when the training data are corrupted by different noise levels.

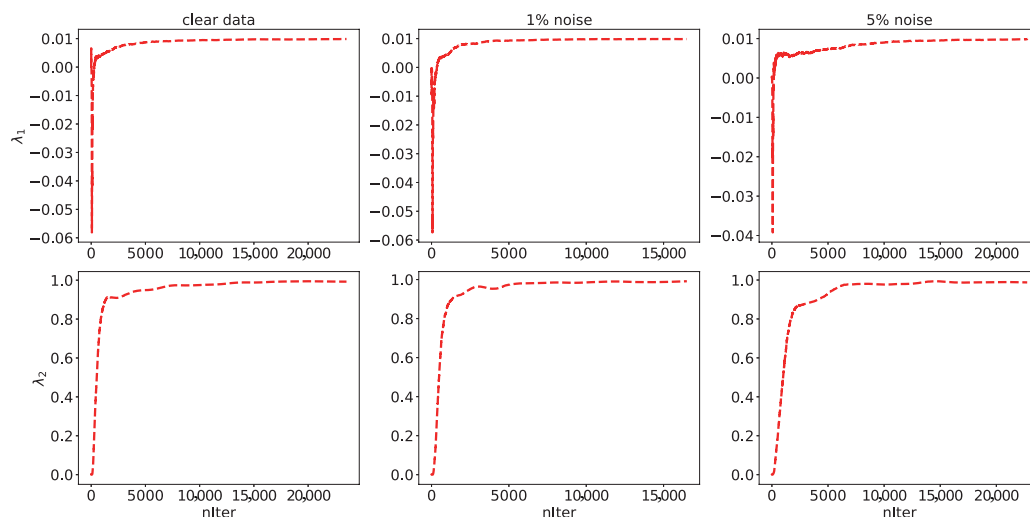|	Clear Data	1% Noise	5% Noise
Exact		$(\lambda_1, \lambda_2) = (0.0100, 1.0000)$	
Identified	$(0.0099, 0.9924)$	$(0.0098, 0.9914)$	$(0.0099, 0.9878)$
Error	$(1.31 \times 10^{-2}, 7.60 \times 10^{-3})$	$(1.53 \times 10^{-2}, 8.58 \times 10^{-3})$	$(1.36 \times 10^{-2}, 1.22 \times 10^{-2})$

Figure 16. Diffusion–reaction equation: change curves of the parameters λ_1 and λ_2 with the iteration number.

5. Conclusions

This paper presents the structure of a second-order neural network constructed through second-order parametric light wave equations. During this work, an adaptive activation function strategy was used to enhance the performance of the neural network, and gPINNs were used to enhance the generalization ability of the neural network. We found that using automatic differentiation techniques to obtain the higher-order derivatives of PDEs' control equations can cause high computational costs. Therefore, we introduced auxiliary variables to reduce computational costs. Finally, we give some numerical examples. Specifically, for forward problems, we simulated Burgers' equation, the Allen–Cahn equation, the CDR equation, and the Navier–Stokes equation. Additionally, for inverse problems, our method can effectively determine the unknown parameters of the equation and estimate and reconstruct the unknown parameters with high accuracy.

In this study, we did not consider the balance between the various parts of the loss function in the neural network. The weight setting of different parts has a great impact on the performance of the neural network, but this is beyond the scope of this paper. In the future, we will explore the use of adaptive weight strategy or grid search methods to determine the optimal weight of each term and further study the dependence of neural networks on hyperparameters. The proposed method is combined with other extended PINN algorithms, such as extended physics-informed neural networks (XPINNs), parallel physics-informed neural networks (PPINNs), and so on. In this paper, only the central difference of second-order differential equations is considered for constructing neural networks. We believe that the construction of neural networks using third-order differential equations or higher is also a feasible solution, which will be an upcoming research project. Additionally, we also hope to develop some innovations in theoretical analysis so as to provide a foundation for subsequent research. Similarly, we also found the computational cost caused by using automatic differentiation to calculate derivatives. It will also be useful

to study the use of numerical differentiation or Taylor-mode AD to calculate derivatives to prevent the excessive consumption of computing resources.

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

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The data that support the finding of this study are available from the corresponding author upon reasonable request.

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

Appendix A. Computational Cost Calculation

In this part, we use Burgers' Equation (18) as an example to calculate the computational cost of the method used. We used a neural network with only three hidden layers, each with only 50 neurons. The training dataset used included 500 initial training data points, 1000 boundary training points, and 5000 residual training points for the PDEs' governing equations. We used the Adam algorithm to train 50 k times. The training results are shown in Table A1. We compared the reference PINN equation, the adaptive PINN equation based on Neuron-wise locally adaptive activation functions (N-LAAF-PINNs), gPINNs, and the methods mentioned in Section 3. It can be seen from the results that both the adaptive activation function strategy and the gradient enhancement strategy can improve the prediction accuracy of the neural network, but the gradient enhancement strategy is time-consuming. We also performed the test without the MIM strategy, which doubled the prediction accuracy but also doubled the training time. This is only a preliminary comparison. As part of future work, we will summarize the existing algorithms and provide a large number of numerical examples for comparative analysis. Although this is a time-consuming process, it is also very important. This research content will help us to design appropriate algorithms for specific problems.

Table A1. Burgers' equation: Computational accuracy and computational cost calculations.

t	2.0	4.0	6.0	8.0	10.0	Training Times
PINNs	1.48×10^{-3}	1.32×10^{-3}	1.26×10^{-3}	1.31×10^{-3}	1.52×10^{-3}	456.46
gPINNs	4.26×10^{-4}	4.84×10^{-4}	4.26×10^{-4}	3.49×10^{-4}	3.85×10^{-4}	3042.68
N-LAAF-PINNs	1.78×10^{-4}	1.56×10^{-4}	1.41×10^{-4}	1.34×10^{-4}	1.41×10^{-4}	1228.73
Ours	7.52×10^{-5}	7.16×10^{-5}	7.43×10^{-5}	8.26×10^{-5}	9.64×10^{-5}	1804.28
Ours (Without MIM)	3.28×10^{-5}	3.58×10^{-5}	3.74×10^{-5}	5.00×10^{-5}	6.94×10^{-5}	3918.47

References

1. Mordohai, P.; Medioni, G.G. Tensor Voting: A Perceptual Organization Approach to Computer Vision and Machine Learning. *Synth. Lect. Image Video Multimed. Process.* **2006**, *2*, 136.
2. Chen, P.H.; Zafar, H.; Aizenberg, M.G.; Cook, T. Integrating Natural Language Processing and Machine Learning Algorithms to Categorize Oncologic Response in Radiology Reports. *J. Digit. Imaging Off. J. Soc. Comput. Appl. Radiol.* **2018**, *30*, 178–184. [CrossRef] [PubMed]
3. Alipanahi, B.; Delong, A.; Weirauch, M.T.; Frey, B.J. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nat. Biotechnol.* **2015**, *33*, 831–839. [CrossRef] [PubMed]
4. Temam, R.; Chorin, A. Navier Stokes Equations: Theory and Numerical Analysis. *J. Appl. Mech.* **1984**, *2*, 456. [CrossRef]

5. Rycroft, M.J. Computational electrodynamics, the finite-difference time-domain method. *J. Atmos. Terr. Phys.* **1996**, *58*, 1817–1818. [CrossRef]

6. Richtmyer, R.D.; Morton, K.W. Difference Methods For Initial-Value Problems. *Phys. Today* **1994**, *12*, 1817–1818. [CrossRef]

7. Chai, S.; Zou, Y.; Zhou, C.; Zhao, W. Weak Galerkin finite element methods for a fourth order parabolic equa-tion. *Numer. Meth. Partial Differ. Equ.* **2019**, *35*, 1745–1755. [CrossRef]

8. Yang, C.; Niu, R.; Zhang, P. Numerical analyses of liquid slosh by Finite volume and Lattice Boltzmann methods. *Aerosp. Sci. Technol.* **2021**, *113*, 106681. [CrossRef]

9. Fornberg, B.; Flyer, N. Solving PDEs with radial basis functions. *Acta Numer.* **2015**, *24*, 215–258. [CrossRef]

10. Zhang, T.; Xu, H.; Feng, X.L. A non-intrusive neural network model order reduction algorithm for pa-rameterized parabolic PDEs. *Comput. Math. Appl.* **2022**, *119*, 59–67. [CrossRef]

11. Cheng, F.X.; Xu, H.; Feng, X.L. Model order reduction method based on (r)POD-ANNs for parameterized time-dependent partial differential equations. *Comput. Fluids* **2022**, *241*, 105481. [CrossRef]

12. Hajialigol, N.; Daghigh, R. The evaluation of the first and second laws of thermodynamics for the pulsating MHD nanofluid flow using CFD and machine learning approach. *J. Taiwan Inst. Chem. Eng.* **2023**, 104782. [CrossRef]

13. Shah, M.A.; Pan, K.; Ibrahim, M.; Saeed, T. Use of neural network and machine learning in optimizing heat transfer and entropy generated in a cavity filled with nanofluid under the influence of magnetic field: A numerical study. *Eng. Anal. Bound. Elem.* **2022**, *139*, 113–131. [CrossRef]

14. 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]

15. Karniadakis, G.E.; Kevrekidis, I.G.; Lu, L.; Perdikaris, P.; Wang, S.; Yang, L. Physics-informed machine learning. *Nat. Rev. Phys.* **2021**, *3*, 422–440. [CrossRef]

16. 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.

17. Poggio, T.; Mhaskar, H.; Rosasco, L.; Miranda, B.; Liao, Q. Why and When Can Deep-but Not Shallow-networks Avoid the Curse of Dimensionality: A Review. *Int. J. Autom. Comput.* **2017**, *14*, 503–519. [CrossRef]

18. Jin, X.W.; Cai, S.Z.; Li, H.; Karniadakis, G.E. NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *J. Comput. Phys.* **2021**, *426*, 109951. [CrossRef]

19. Zhang, W.; Li, J. CPINNs: A coupled physics-informed neural networks for the closed-loop geothermal system. *Comput. Math. Appl.* **2023**, *132*, 161–179. [CrossRef]

20. Yang, M.Y.; John, T.F. Using Physics-Informed Neural Networks to Solve for Permeability Field under Two-Phase Flow in Heterogeneous Porous Media. *J. Mach. Learn. Model. Comput.* **2023**, *4*, 1–19. [CrossRef]

21. Strelow, E.L.; Gerisch, A.; Lang, J.; Pfetsch, M.E. Physics informed neural networks: A case study for gas transport problems. *J. Comput. Phys.* **2023**, *481*, 112041. [CrossRef]

22. Kovacs, A.; Exl, L.; Kornell, A.; Fischbacher, J.; Hovorka, M.; Gusenbauer, M.; Breth, L.; Oezelt, H.; Praetorius, D.; Suess, D.; et al. Magnetostatics and micromagnetics with physics informed neural networks. *J. Magn. Magn. Mater.* **2022**, *548*, 168951. [CrossRef]

23. Lu, L.; Meng, X.H.; Mao, Z.P.; Karniadakis, G.E. DeepXDE: A Deep Learning Library for Solving Differential Equations. *SIAM Rev.* **2021**, *63*, 208–228. [CrossRef]

24. Nabian, M.A.; Gladstone, R.J.; Meidani, H. Efficient training of physics-informed neural networks via importance sampling. *Comput.-Aided Civ. Infrastruct. Eng.* **2021**, *36*, 962–977. [CrossRef]

25. Jagtap, A.D.; Kawaguchi, K.J.; Karniadakis, G.E. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *J. Comput. Phys.* **2019**, *404*, 109136. [CrossRef]

26. Jagtap, A.D.; Kawaguchi, K.J.; Karniadakis, G.E. Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proc. R. Soc. A* **2020**, *476*, 20200334. [CrossRef] [PubMed]

27. Jagtap, A.D.; Shin, Y.; Kawaguchi, K.J.; Karniadakis, G.E. Deep Kronecker neural networks: A general framework for neural networks with adaptive activation functions. *Neurocomputing* **2022**, *468*, 165–180. [CrossRef]

28. Wang, S.F.; Teng, Y.J.; Perdikaris, P. Understanding and Mitigating Gradient Flow Pathologies in Phys-ics-Informed Neural Networks. *SIAM J. Sci. Comput.* **2021**, *43*, A3055–A3081. [CrossRef]

29. Wang, S.F.; Yu, X.L.; Perdikaris, P. When and why PINNs fail to train: A neural tangent kernel perspective. *J. Comput. Phys.* **2022**, *449*, 110768. [CrossRef]

30. Xiang, Z.X.; Peng, W.; Liu, X.; Yao, W. Self-adaptive loss balanced Physics-informed neural networks. *Neurocomputing* **2022**, *496*, 11–34. [CrossRef]

31. Meng, X.H.; Li, Z.; Zhang, D.K.; Karniadakis, G.E. PPINN: Parareal physics-informed neural network for time-dependent PDEs. *Comput. Methods Appl. Mech. Eng.* **2020**, *370*, 113250. [CrossRef]

32. Jagtap, A.D.; Kharazmi, E.; Karniadakis, G.E. Conservative physics-informed neural networks on discrete domains for conserva-tion laws: Applications to forward and inverse problems. *Comput. Methods Appl. Mech. Eng.* **2020**, *365*, 113028. [CrossRef]

33. 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.

34. Shukla, K.; Jagtap, A.D.; Karniadakis, G.E. Parallel Physics-Informed Neural Networks via Domain De-composition. *J. Comput. Phys.* **2021**, *447*, 110683. [CrossRef]
35. Wu, W.; Feng, X.L.; Xu, H. Improved Deep Neural Networks with Domain Decomposition in Solving Partial Differential Equations. *J. Sci. Comput.* **2022**, *93*, 1–34. [CrossRef]
36. Kharazmi, E.; Zhang, Z.Q.; Karniadakis, G.E. hp-VPINNs: Variational physics-informed neural networks with domain decomposition. *Comput. Methods Appl. Mech. Eng.* **2021**, *374*, 113547. [CrossRef]
37. Tang, S.P.; Feng, X.L.; Wu, W.; Xu, H. Physics-informed neural networks combined with polynomial in-terpolation to solve nonlinear partial differential equations. *Comput. Math. Appl.* **2023**, *132*, 48–62.
38. Peng, P.; Pan, J.G.; Xu, H.; Feng, X. RPINNs: Rectified-physics informed neural networks for solving stationary partial differential equations. *Comput. Fluids* **2022**, *245*, 105583. [CrossRef]
39. Mishra, S.; Molinaro, R . Estimates on the generalization error of physics-informed neural networks for approximating PDEs. *IMA J. Numer. Anal.* **2022**, *43*, 1–43. [CrossRef]
40. Mishra, S.; Molinaro, R. Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs. *IMA J. Numer. Anal.* **2021**, *42*, 981–1022. [CrossRef]
41. Ryck, T.D.; Jagtap, A.D.; Mishra, S. Error estimates for physics-informed neural networks approximating the Navier–Stokes equations. *IMA J. Numer. Anal.* **2023**. [CrossRef]
42. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016 ; pp. 770–778.
43. He, K.; Zhang, X.; Ren, S.; Sun, J. Identity Mappings in Deep Residual Networks. In *Computer Vision—ECCV 2016, Proceedings of the 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016*; Springer: Cham, Switzerland, 2016; Volume 9908, pp. 630–645.
44. Bengio, Y. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **2002**, *5*, 157–166. [CrossRef]
45. Ruthotto, L.; Haber, E. Deep Neural Networks Motivated by Partial Differential Equations. *J. Math. Imaging Vis.* **2020**, *62*, 352–364. [CrossRef]
46. Yu, J.; Lu, L.; Meng, X.H.; Karniadakis, G.E. Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems. *Comput. Methods Appl. Mech. Eng.* **2022**, *393*, 114823. [CrossRef]
47. Lyu, L.Y.; Zhang, Z.; Chen, M.X.; Chen, J. MIM: A deep mixed residual method for solving high-order partial differential equations. *J. Comput. Phys.* **2022**, *452*, 110930. [CrossRef]
48. Wu, J.Y. The logarithmic entropy formula for the linear heat equation on Riemannian manifolds. *Nonlinear Anal. Theory Methods Appl.* **2012**, *75*, 4862–4872. [CrossRef]
49. Maitournam, M.H. Entropy and temperature gradients thermomechanics: Dissipation, heat conduction inequality and heat equation. *Comptes Rendus Mec.* **2012**, *340*, 434–443. [CrossRef]
50. Chiu, P.H.; Wong, J.C.; Ooi, C.C.; Dao, M.H.; Ong, Y.S. CAN-PINN: A fast physics-informed neural network based on coupled-automatic-numerical differentiation method. *Comput. Methods Appl. Mech. Eng.* **2022**, *395*, 114909. [CrossRef]
51. Lagaris, I.E.; Likas, A. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.* **1998**, *9*, 987–1000. [CrossRef]
52. Schiassi, E.; Furfaro, R.; Leake, C.; De Florio, M.; Johnston, H.; Mortari, D. Extreme Theory of Functional Connections: A Fast Phys-ics-Informed Neural Network Method for Solving Ordinary and Partial Differential Equations. *Neurocomputing* **2021**, *457*, 334–356. [CrossRef]
53. Lu, L.; Pestourie, R.; Yao, W.J.; Wang, Z.; Verdugo, F.; Johnson, S.G. Physics-Informed Neural Networks with Hard Constraints for Inverse Design. *SIAM J. Sci. Comput.* **2021**, *43*, B1105–B1132. [CrossRef]
54. Lagari, P.L.; Tsoukalas, L.H.; Safarkhani, S.; Lagaris, I.E. Systematic Construction of Neural Forms for Solving Partial Differential Equations Inside Rectangular Domains, Subject to Initial, Boundary and Interface Conditions. *Int. J. Artif. Intell. Tools* **2020**, *29*, 2050009. [CrossRef]
55. Nielsen, R.H. Theory of the Backpropagation Neural Network. *Neural Netw.* **1998**, *1*, 445. [CrossRef]
56. Chen, R.T.Q.; Rubanova, Y.; Bettencourt, J.; Duvenaud, D.K. Neural Ordinary Differential Equations. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 1–18.
57. Rogers, C.; Moodie, T.B. *Wave Phenomena: Modern Theory and Applications*; North-Holland: Amsterdam, The Netherlands , 1984.
58. Byrd, R.H.; Lu, P.; Nocedal, J.; Zhu, C. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM J. Sci. Comput.* **1995**, *16*, 1190–1208. [CrossRef]
59. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. *JMLR Workshop Conf. Proc.* **2010**, *9*, 249–256.
60. Stein, M. Large Sample Properties of Simulations Using Latin Hypercube Sampling. *Technometrics* **1987**, *29*, 143–151. [CrossRef]
61. Xiang, Z.X.; Peng, W.; Zheng, X.H.; Zhao, X.; Yao, W. Self-adaptive loss balanced Physics-informed neural networks for the incompressible Navier-Stokes equations. *Acta Mech. Sin.* **2021**, *37*, 47–52.
62. Yang, X.; Ge, Y.; Zhang, L. A class of high-order compact difference schemes for solving the Burgers' equations. *Appl. Math. Comput.* **2019**, *358*, 394–417. [CrossRef]
63. Williamson, C.H.K. Vortex Dynamics in the Cylinder Wake. *Annu. Rev. Fluid Mech.* **1996**, *26*, 477–539. [CrossRef]

64. Allen, S.M.; Cahn, J.W. A microscopic theory for antiphase boundary motion and its application to anti-phase domain coarsening. *Acta Metall.* **1979**, *27*, 1085–1095. [CrossRef]
65. Ji, L.; Reutskiy, S.Y.; Lu, J. A novel meshless method for fully nonlinear advection-diffusion-reaction problems to model transfer in anisotropic media. *Appl. Math. Comput.* **2018**, *339*, 459–476.
66. Yun, D.F.; Hon, Y.C. Improved localized radial basis function collocation method for multi-dimensional convection-dominated problems. *Eng. Anal. Boundary Elem.* **2016**, *67*, 63–80. [CrossRef]
67. Wang, M.L.; Li, H.X.; Chen, X.; Chen, Y. Deep Learning-Based Model Reduction for Distributed Parameter Systems. *IEEE Trans. Syst. Man Cybern. Syst.* **2016**, *46*, 1664–1674. [CrossRef]
68. Conolly, B.W.; Britton, N.H. Reaction-Diffusion Equations and Their Applications to Biology. *Biometrics* **1989**, *45*, 1344 [CrossRef]