Enhancing Stability in Deep Learning for PDEs: A Residual-Feedforward Approach

Amir Noorizadegan*, C.S. Chen*†

Abstract

This paper introduces a novel strategy designed to enhance the stability of adaptive feedforward architectures in deep learning, particularly focusing on applications in partial differential equations (PDEs). The proposed approach utilizes a distinctive variant of the Residual Network, termed the squared residual network (SqrResNet). Through the integration of SqrResNet with physics, resulting in the PI-SqrResNet model, this strategy effectively addresses inherent stability challenges encountered in the domain of deep learning for PDE applications. The comprehensive evaluation of the PI-SqrResNet method encompasses a diverse set of PDE scenarios, including both time-independent and time-dependent examples such as Poisson equations, elastostatic problems, Burgers equations, and Schrödinger equations. Empirical findings underscore the superior performance of the proposed approach when compared to the DeepXDE package [9], leading to improved convergence, increased accuracy, and greater stability. An intriguing observation reveals that deeper layers contribute to achieving better accuracy, where the DeepXDE may struggle during training at this level of deep network, further emphasizing the potential of the PI-SqrResNet approach in enhancing deep learning methodologies for solving PDEs.

Keywords: Residual network, squared residual network, Partial differential equations, deep learning for PDEs, stability.

1 Introduction

To do ...

2 Neural Networks

The feedforward neural network, also commonly referred to as a multilayer perceptron (MLP), serves as a foundational architectural framework within the realm of artificial

*Department of Civil Engineering, National Taiwan University, 10617, Taipei, Taiwan
†Corresponding authors: `dchen@ntu.edu.tw`

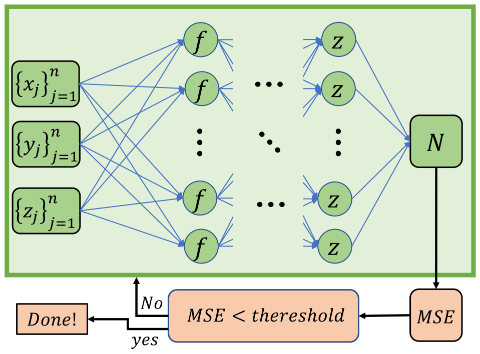


Figure 1: A feedforward neural network architecture designed for a three-dimensional problem in the domain of (x, y, z), incorporating n sample points

neural networks. These networks are characterized by their interconnected layers of neurons, where the flow of information moves unidirectionally, starting from the input layer and traversing through various hidden layers before reaching the output layer. This unidirectional data flow, often described as "feedforward," plays a crucial role in transforming input data into precise and valuable output predictions.

At the core of a feedforward neural network lie its individual neurons. Each neuron is tasked with calculating a weighted sum of its inputs, which is subsequently modified by the inclusion of a bias term. Following this summation, an activation function is applied to the result.

For a neuron situated in a layer i with n input data points, where these inputs are represented as $\mathcal{P} = [\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n]$, the respective weights as $\mathbf{w} = [w_1, w_2, \ldots, w_n]$, and the bias as b_i, the output z_i is determined as follows:

$$z_i = \sum_{j=1}^{n} w_j \mathbf{p}_j + b_i \tag{1}$$

Here, $\mathbf{p}_j$ exists in a 3-dimensional space, represented as $\mathbf{p} = (x, y, z)$. Subsequently, the output undergoes a transformation using the `tanh` activation function denoted as $h(\cdot)$:

$$y_i = h(z_i) \tag{2}$$

It is crucial to note that while hidden layers make use of activation functions to introduce non-linearity, the final layer, typically referred to as the output layer, often does not employ an activation function for its outputs. Figure 1 provides a schematic representation of a feedforward neural network designed for function interpolation, with f defined as follows:

$$f(\mathbf{p}) = h\left(\sum_{j=1}^{n} w_j \mathbf{p}_j + b\right) \tag{3}$$

This represents a combination of the linear transformation Eq. (1) and the nonlinear activation function Eq. (2).

3 Residual Network

Residual networks, commonly recognized as ResNets [2, 3], have emerged as a dominant architectural paradigm in neural networks. They are notably distinguished by their residual modules, denoted as f_i, and the integration of skip connections to bypass these modules, enabling the assembly of deep networks. This construction facilitates the creation of residual blocks, constituting a cluster of layers within the network [4, 6]. Unlike the depiction of the basic neural network in Fig. 2a, Fig. 2b illustrates the incorporation of ResNet attributes into the network architecture. To simplify our discourse, we shall omit discussions of the initial pre-processing and final steps. Consequently, the definition of the output g_i for the i-th layer is as expressed below:

$$\begin{cases} g_i = f_i(g_{i-1}) + g_{i-1}, & \text{for} \quad i = 1, 3, 5, \dots \\ g_i = f_i(g_{i-1}), & \text{for} \quad i = 2, 4, 6, \dots \end{cases} \tag{4}$$

where f is defined in Eq. (3) and i is the layer's index (as shown in Fig. 2b). In [5], Veit et al. found that:

- Residual networks can be conceptualized as a collection of numerous paths rather than a single ultra-deep network as it can be seen from (7).
- Through lesion studies, they demonstrated that despite being trained jointly, these paths exhibit weak interdependence and showcase ensemble-like behavior. The performance smoothly correlates with the number of valid paths.
- Additionally, they revealed that the paths contributing to gradient during training are shorter than anticipated. Surprisingly, deep paths are not necessary during training as they do not contribute any gradient. Thus, residual networks do not address the vanishing gradient problem by preserving gradient flow throughout the entire depth of the network.

In this research, we introduce an innovative approach known as the "SqrResNet." This variant is characterized by an adjusted recursive definition of the form:

$$\begin{cases} g_i = f_i(g_{i-1}) + g_{i-1}^2, & \text{for} \quad i = 1, 3, 5, \dots \\ g_i = f_i(g_{i-1}), & \text{for} \quad i = 2, 4, 6, \dots \end{cases} \tag{5}$$

This distinctive configuration, portrayed in Fig. 2c, introduces the incorporation of a power term, g_{i-1}^2, for specific layers, augmenting the network's expressive capabilities.

To facilitate comparative assessment among Plain NN, ResNet, and SqrResNet (portrayed in Figs. 2a,b,c, respectively), we evaluate the output of the third hidden layer with

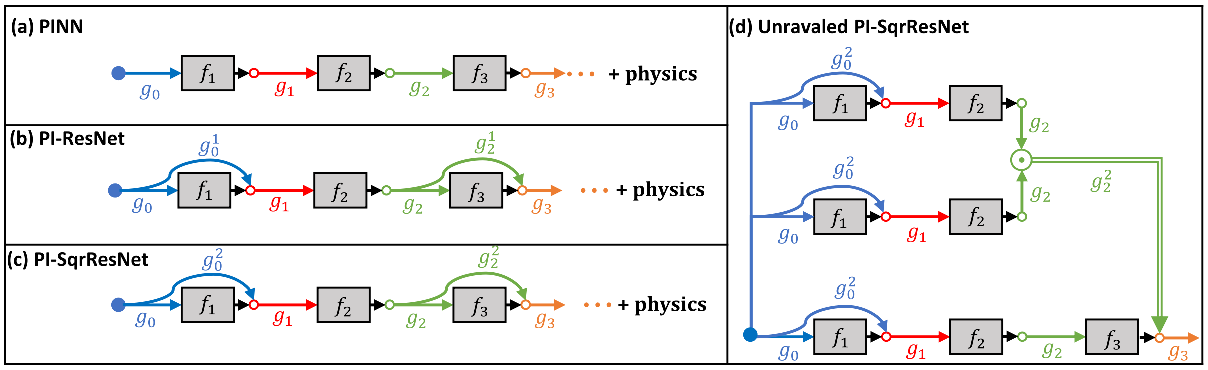


Figure 2: Three neural network architectures for physics-informed with a (a) conventional neural network, (b) residual network with a skipped layer for the added residual term, (c) squared residual network with a skipped layer for the added residual term, and (d) unraveled PI-SqrResNet where $\odot$ denotes element-wise multiplication.

respect to the input $g_0 = \mathcal{P}$. The outcomes for the plain neural network are outlined below:

$$\begin{aligned} g_3 &= f_3(g_2) \\ &= f_3(f_2(g_1)) \\ &= f_3(f_2(f_1(g_0))) \end{aligned} \tag{6}$$

In contrast, the corresponding ResNet formulation, as articulated by Veit et al. [5], can be summarized as:

$$\begin{aligned} g_3 &= f_3(g_2) + g_2 \\ &= f_3(f_2(g_1) + g_1) + [f_2(g_1) + g_1] \\ &= f_3(f_2(f_1(g_0) + g_0) + f_1(g_0) + g_0) \\ &\quad + [f_2(f_1(g_0) + g_0) + f_1(g_0) + g_0] \end{aligned} \tag{7}$$

Finally, the formulation for the initial three hidden layers within the SqrResNet is defined as follows:

$$\begin{aligned} g_3 &= f_3(g_2) + g_2^2 \\ &= f_3(f_2(g_1)) + [f_2(g_1)]^2 \\ &= f_3(f_2(f_1(g_0) + g_0^2)) + \left[f_2(f_1(g_0) + y_0^2)\right]^2 \end{aligned} \tag{8}$$

Fig. 2d visually represents the intricate "expression tree," offering a comprehensive illustration of the data flow from input to output. Here, the symbol $\odot$ signifies element-wise multiplication. This graphical depiction underscores the existence of diverse data pathways that facilitate or bypass particular residual modules as also described by [5].

4 Gradient Pathologies

To do: Make a comparison between back-propagated gradients for PINN, PI-ResNet,and PI-SqrResNet following reference [8].

5 Numerical Results

In this study, we use the notations NP, NL, and NN to denote the number of data points (training), layers, and neurons in each layer, respectively. Throughout all subsequent examples, unless otherwise specified, we consider 100^2 validation data points. We compare three algorithms:

- `PINN (Physics-informed Neural Network):`
  A plain neural network for physics-informed tasks.
- `PI-ResNet (Physics-informed Residual network):`
  A physics-informed residual network where the residual terms are added to every other layer.
- `PI-SqrResNet (Physics-informed Squared Residual network):`
  A physics-informed squared residual network where the squared power of residual terms is added to every other layer.

The Deepxde package [9] is utilized for PINN with the following settings: `pytorch` selected as the backend, and a random seed of `12345` is used for point generation. Computations are performed using `float 32` as the default in deepxde. For optimization, the `L-BFGS` (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) optimizer is employed with a maximum of `5000` iterations, a gradient tolerance of 1×10^{-8}, and a learning rate of 1×10^{-3}. Additionally, the Adam (Adaptive Moment Estimation) optimizer is used with the same maximum iterations, gradient tolerance, and a learning rate of 1×10^{-3}. The neural network is configured with the activation function `tanh`, and the initializer is set to `Glorot uniform`. Each hidden layer has a fixed number of 50 neurons, unless specified otherwise. The Deepxde's source code has been updated to incorporate ResNet and SqrResNet algorithms.

Tables in this section present train loss and test loss, indicating the mean square error $\frac{1}{n}\sum_{i=1}^{n}(u_i - N_i)^2$ over function interpolation and PDEs, respectively. The error and test metrics are based on L2 norm error $\frac{\|u-N\|_2}{\|u\|_2}$ where u and N denote exact and approximated solutions, respectively.
In tables, the term "status" denotes three situations as follows:

- `completed:` Indicates that all predefined steps for the network have been successfully executed, or the network has converged even in a smaller number of steps.

- `not trained:` Denotes a scenario where, from the beginning, a large loss was observed, and the situation further deteriorates as the steps progress.
- `diverged:` Signifies that the computations diverged after some initially successful iterations. Divergence can take two forms: (i) a significant increase in loss and error, or (ii) the occurrence of NaN due to SVD convergence issues.

The numerical experiments were conducted on a computer with an Intel(R) Core(TM) i9-9900 CPU operating at 3.10 GHz and equipped with 64.0 GB of RAM. Additionally, `Google Colaboratory` with a `T4 GPU` was occasionally utilized. Table 1 provides a brief overview of the examples designed for this study.

Table 1: Study Examples Overview

Example	Equation	Dimension	Time-dependent	CPU/L-BFGS	GPU/Adam
1	Poisson	1D	No	Yes	Yes
2	Poisson	2D	No	Yes	No
3	Burgers	1D	Yes	Yes	No
4	Diffusion-reaction	1D	Yes	Yes	No
5	Diffusion	1D	Yes	Yes	No
6	Heat	1D	Yes	Yes	Yes
7	Schrödinger	1D	Yes	Yes	No
8	Elastostatic	2D	No	Yes	No

Example 1 First we solve a Poisson equation in 1D given by:

$$-\Delta u = \pi^2 \sin(\pi x), \quad x \in [-1, 1], \tag{9}$$

with the Dirichlet boundary conditions:

$$u(-1) = 0, \quad u(1) = 0. \tag{10}$$

The exact solution is $u(x) = \sin(\pi x)$.

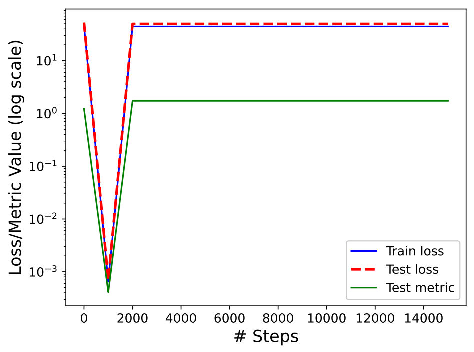
Table 2: Example 1: 1D Poisson equation, CPU/L-BFGS- A comparison of PINN, PI-ResNet, and PI-SqrResNet.

method	NL	train loss	test loss	error	t(s)	status
PINN	5	4.14E-07	6.21E-07	8.96E-06	30	completed
	10	✗	✗	✗	✗	not trained
	15	✗	✗	✗	✗	not trained
	20	6.51e-04	6.96e-04	4.08e-04	81	diverged*
	30	✗	✗	✗	✗	not trained
PI-ResNet	5	2.71e-06	5.25e-06	2.30e-05	32	completed
	10	✗	✗	✗	✗	not trained
	15	✗	✗	✗	✗	not trained
	20	✗	✗	✗	✗	not trained
	30	✗	✗	✗	✗	not trained
PI-SqrResNet	5	2.19e-07	1.12e-06	8.10e-06	33	completed
	10	3.10e-06	1.08e-05	2.31e-05	56	completed
	15	2.19e-07	1.12e-06	8.10e-06	34	completed
	20	**8.37e-08**	**1.08e-07**	**3.16e-06**	**97**	**completed**
	30	9.84e-07	1.14e-06	5.40e-06	128	completed

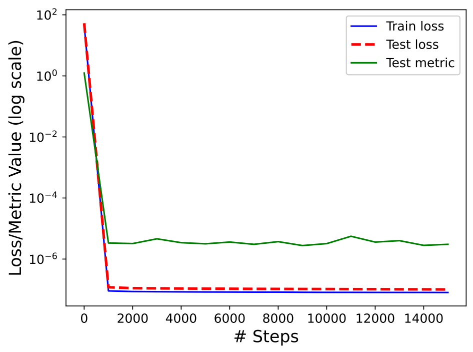
* Diverged after step = 2000.

Table 3: Example 1: 1D Poisson equation, GPU/Adam- A comparison between PINN, PI-ResNeT, and PI-SqrResNet.

	PINN			PI-ResNet			PI-SqrResNet		
NL	train loss	error	status	train loss	error	status	train loss	error	status
5	3.3e-4	4.0e-3	completed	1.4e-4	2.8e-3	completed	**3.5e-5**	**8.3e-5**	**completed**
10	2.3e-3	1.6e-2	completed	5.7e-4	3.4e-4	completed	2.4e-5	3.4e-4	completed
15	2.6e-3	1.3e-2	completed	3.0e-3	9.6e-1	completed	5.6e-5	2.6e-4	completed
20	1.1e-3	6.9e-3	completed	✗	✗	not trained	4.9e-5	7.0e-4	completed
30	7.7e-3	1.6e-2	completed	✗	✗	not trained	1.4e-4	3.2e-4	completed
40	9.2e-3	1.3e-2	completed	✗	✗	not trained	4.6e-3	8.4e-3	completed
50	✗	✗	not trained	✗	✗	not trained	7.6e-6	3.7e-3	completed



(a) PINN



(b) PI-SqrResNet

Figure 3: Example 1: 1D Poisson equation, CPU/L-BFGS- A comparison of PINN (left) and PI-SqrResNet (right) for NL=20.

For this scenario, we utilize a total of 18 training points, comprising 16 points within the domain and 2 on the boundary and L-BFGS optimizer. Table 2 presents the losses, errors, and computational times for various methods, with respect to the number of hidden layers (NL). In the table, ✗ designates cases that were not trained. Key observations include:

- In the case of PINN, effective training and accuracy are achieved at NL=5. However, for NL=10, 15, and 30, the network fails to train. For NL=20, the training diverges after 1000 steps.
- In PI-ResNet executes NL=5 successfully. However, the network fails to train for larger NL.
- In PI-SqrResNet, the proposed network successfully trains for all NL cases. Increasing the number of hidden layers not only affects the result but may improve it.

In summary, PI-SqrResNet demonstrates stable results, with NL=20 achieving the best accuracy. Although the CPU time for PI-SqrResNet is about 20% larger than PINN, stability is significantly improved.

The training outcomes, with 15000 steps, for both PINN and the proposed PI-SqrResNet are depicted in Fig.3 for the case of NL=20. In Fig.3(a), a substantial increase in loss/metric values is evident after step 1000 for PINN. Conversely, PI-SqrResNet exhibits good performance for all loss and metric parameters with respect to the step. Notably, the loss and metric values for PI-SqrResNet at step 1000 are smaller than the corresponding results achieved with PINN implying that the proposed PI-SqrResNet converges faster than PINN.

Moreover, Table 3 presents results obtained with PINN, PI-ResNet, and PI-SqrResNet using Google Colab in GPU mode with the Adam optimizer. The table reveals more stable results for the PINN compared to those obtained with a CPU-based system using L-BFGS, as shown in Table 2. From Table 3 we can see that the training loss and error with PI-SqrResNet are smaller than with the other two methods, highlighting the stability of this algorithm across different system configurations.

Example 2 Next, we will solve a Poisson equation over an L-shaped domain given by:

$$-u_{xx} - u_{yy} = 1, \quad \Omega = [-1,1]^2 \setminus [0,1]^2 \tag{11}$$

with the Dirichlet boundary conditions:

$$u(x,y) = 0, \quad (x,y) \in \partial\Omega. \tag{12}$$

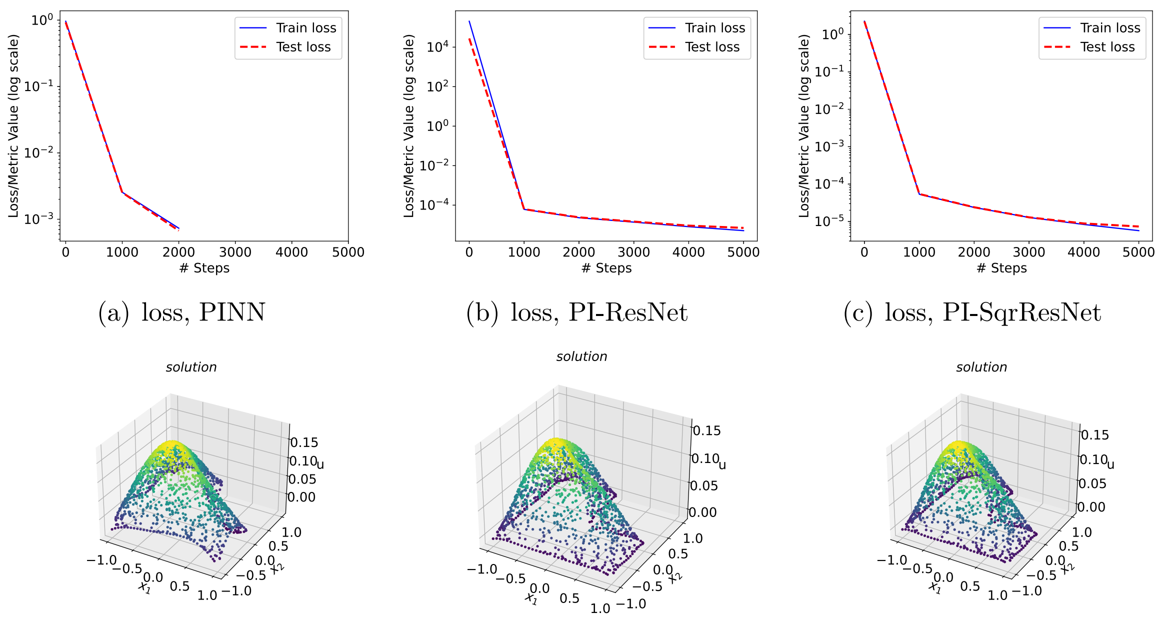
Table 4: Example 2: 2D Poisson equation, CPU/L-BFGS- A comparison of PINN, PI-ResNet, and PI-SqrResNet.

	PINN			PI-ResNet			PI-SqrResNet		
NL	train loss	test loss	status	train loss	test loss	status	train loss	test loss	status
5	5.5e-5	5.7e-5	completed	5.7e-5	5.9e-5	completed	2.4e-5	2.6e-5	completed
10	1.6e-5	1.5e-5	completed	3.6e-5	3.7e-5	completed	7.1e-6	7.7e-6	completed
15	3.9e-5	4.1e-5	completed	7.4e-6	9.8e-6	completed	5.2e-6	6.3e-6	completed
20	7.3e-4	6.7e-4	diverged[*]	5.1e-6	7.0e-6	completed	5.9e-6	7.4e-6	completed
30	✗	✗	not trained	✗	✗	not trained	**2.9e-6**	**3.7e-6**	**completed**

[*] Diverged after step = 2000.

In this example, an exact solution is not available, and only train loss and test loss are taken into consideration. The total number of training points is 1320, consisting of 1200 interior points and the remainder on the boundary. Table 4 presents the results for PINN, PI-ResNet, and PI-SqrResNet concerning the number of hidden layers (NL). Based on the status, both PINN and PI-ResNet exhibit one case labeled as not trained when NL=30. Additionally, PINN diverged after step=2000 for NL=20. In contrast, the proposed PI-SqrResNet successfully completed the training process for all NL cases. A comparison of losses reveals that the smallest loss occurs for NL=30 when using PI-SqrResNet. This suggests that deeper networks not only affect the training but also lead to better training outcomes with PI-SqrResNet.

Figure 4 displays the corresponding results of Table 4 for the case with NL=20. The top panel illustrates the loss over train and test data, while the bottom panel shows the solution at the final steps. From Fig. 4(a), it can be observed that the training process stops at Step=2000 when using PINN. The corresponding solution plotted in Fig. 4(d) also exhibits poor performance on the boundary condition where a value of zero is expected. On the other hand, when using ResNet and SqrResNet, the training completes the 5000 steps, and the corresponding solution in the bottom panel shows acceptable performance. From this example, we can conclude that PI-SqrResNet demonstrates better training and stability compared to other networks.



(a) loss, PINN (b) loss, PI-ResNet (c) loss, PI-SqrResNet

(d) solution, PINN (e) solution, PI-ResNet (f) solution, PI-SqrResNet

Figure 4: Example 2: 2D Poisson equation, CPU/L-BFGS- A comparison of PINN (left panel), PI-ResNet (middle panel) and PI-SqrResNet (right) with NL=20.

Example 3 We will solve a Burgers equation given by:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1], \quad t \in [0, 1] \tag{13}$$

with the Dirichlet boundary conditions and initial conditions:

$$u(-1, t) = u(1, t) = 0, \tag{14}$$

$$u(x, 0) = -\sin(\pi x). \tag{15}$$

The errors are calculated using the exact solution provided by the DeepXDE package [9]. Table 5 highlights the poor performance of PINN, where even a small number of hidden layers, such as 5, could not be completed. The results frequently diverged for this example using PINN. In contrast, both PI-ResNet and PI-SqrResNet exhibit good stability and performance. Notably, the best accuracy is achieved at NL=20 using PI-SqrResNet. Overall, PI-SqrResNet demonstrates higher accuracy across different network depths compared to the other two methods.

Example 4 We will solve the following 1D diffusion-reaction equation:

$$\frac{\partial u}{\partial t} = d\frac{\partial^2 u}{\partial x^2} + e^{-t}\left(3\frac{\sin(2x)}{2} + \frac{8\sin(3x)}{3} + \frac{15\sin(4x)}{4} + \frac{63\sin(8x)}{8}\right) \tag{16}$$

with the initial condition:

$$u(x, 0) = \sin(x) + \frac{\sin(2x)}{2} + \frac{\sin(3x)}{3} + \frac{\sin(4x)}{4} + \frac{\sin(8x)}{8}, \quad x \in [-\pi, \pi] \tag{17}$$

Table 5: Example 3: Burger's equation, CPU/L-BFGS– A comparison of PINN, PI-ResNet, and PI-SqrResNet.

	PINN			PI-ResNet			PI-SqrResNet		
NL	train loss	error	status	train loss	error	status	train loss	error	status
5	5.6e-4	1.7e-1	diverged[*]	1.8e-5	1.3e-2	completed	5.6e-6	1.5e-2	completed
10	1.1e-5	5.2e-3	completed	5.2e-6	3.3e-2	completed	1.3e-6	6.0e-3	completed
15	2.9e-5	7.6e-3	diverged[†]	1.4e-6	5.7e-2	completed	6.7e-7	6.1e-3	completed
20	4.9e-3	8.2e-2	diverged[‡]	3.6e-6	6.7e-3	completed	**9.6e-7**	**1.5e-3**	**completed**
30	✗	✗	not trained	1.7e-6	3.0e-3	completed	1.3e-6	2.6e-3	completed

[*] Diverged after step = 1000.
[†] Diverged after step = 2000.
[‡] Diverged after step = 1000.

and the Dirichlet boundary condition:

$$u(t,-\pi) = u(t,\pi) = 0, \quad t \in [0,1] \tag{18}$$

We also specify the following parameters for the equation:

$$d = 1 \tag{19}$$

The exact solution is:

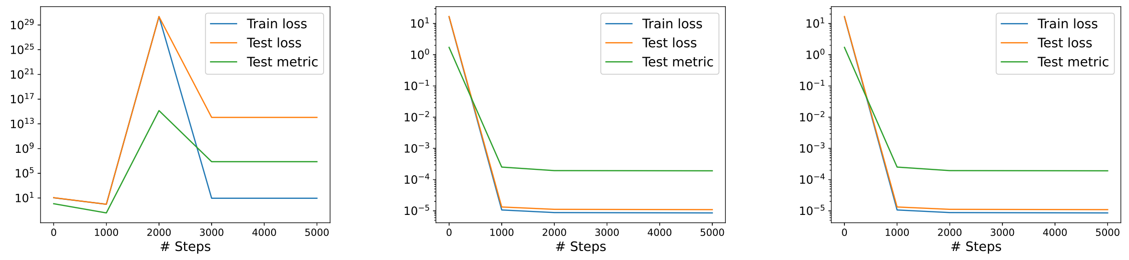
$$u(x,t) = e^{-t}\left(\sin(x) + \frac{\sin(2x)}{2} + \frac{\sin(3x)}{3} + \frac{\sin(4x)}{4} + \frac{\sin(8x)}{8}\right) \tag{20}$$

Table 6: Example 4: Diffusion-reaction equation, CPU/L-BFGS- A comparison of PINN, PI-ResNet, and PI-SqrResNet.

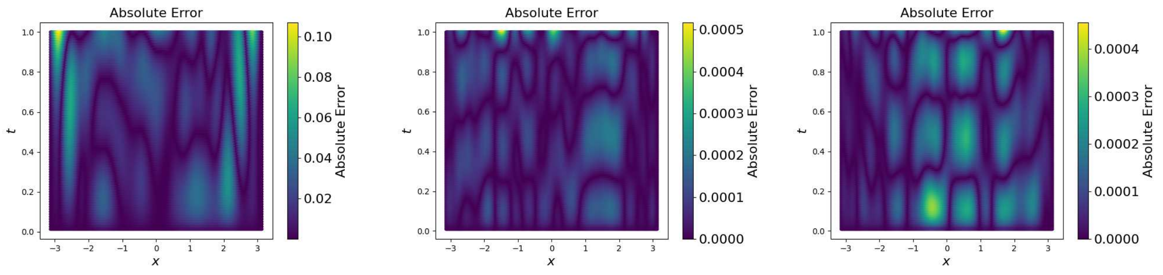
	PINN			PI-ResNet			PI-SqrResNet		
NL	train loss	error	status	train loss	error	status	train loss	error	status
5	2.5e-4	7.4e-4	completed	3.5e-6	9.4e-5	completed	8.0e-6	2.9e-4	completed
10	9.8e-1	3.9e-2	diverged[*]	6.9e-6	1.3e-4	completed	8.5e-6	1.9e-4	completed
15	2.6e-5	3.0e-4	completed	2.3e-6	1.0e-4	completed	**1.5e-6**	**6.5e-5**	**completed**
20	✗	✗	not trained	3.9e-6	1.1e-4	completed	2.8e-6	1.0e-4	completed
30	✗	✗	not trained	✗	✗	not trained	3.5e-6	1.1e-4	completed

[*] Diverged after step = 1000.

Table 6 compiles the results obtained using PINN, PI-ResNet, and PI-SqrResNet with respect to NL. It is evident that PINN has only successfully completed two cases, namely NL=5 and 15. Conversely, PI-ResNet exhibits improved performance but encounters failure at NL=30. Notably, PI-SqrResNet demonstrates the best overall performance by completing the training, achieving high accuracy, and maintaining stable performance. Fig.5 illustrates the loss and metric of NL=10 over the course of the training steps (top



(a) loss, PINN (b) loss, PI-ResNet (c) loss, PI-SqrResNet



(d) error, PINN (e) error, PI-ResNet (f) error, PI-SqrResNet

Figure 5: Example 4: Diffusion-reaction equation, CPU/L-BFGS- A comparison of PINN (left panel), PI-ResNet (middle panel) and PI-SqrResNet (right) with NL=10.

panel) and the maximum absolute error (bottom panel) corresponding to the smallest error during training. Analyzing Fig.5(a), we observe divergence after 1000 steps and large absolute error in Fig.5(d). On the other hand both residual based methods, PI-ResNet and PI-SqrResNet, exhibit robust performance on training data and small absolute error over validation data.

Example 5 We will solve a diffusion equation with hard initial and boundary conditions:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} - e^{-t}(\sin(\pi x) - \pi^2 \sin(\pi x)), \quad x \in [-1, 1], \quad t \in [0, 1] \tag{21}$$

with the initial condition:

$$u(x, 0) = \sin(\pi x) \tag{22}$$

and the Dirichlet boundary condition:

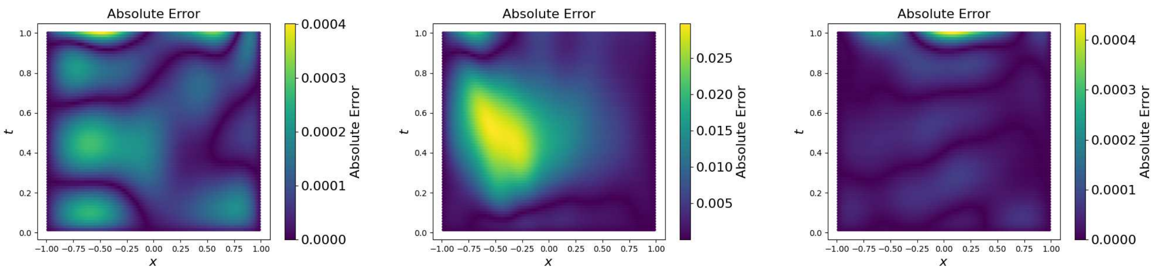
$$u(-1, t) = u(1, t) = 0. \tag{23}$$

The reference solution is:

$$u = e^{-t} \sin(\pi x) \tag{24}$$

Table 7: Example 5: Diffusion equation, CPU/L-BFGS- A comparison of PINN, PI-ResNet, and PI-SqrResNet.

	PINN			PI-ResNet			PI-SqrResNet		
NL	train loss	error	status	train loss	error	status	train loss	error	status
5	✗	✗	not trained	1.0e-6	9.6e-5	completed	1.0e-6	1.1e-4	completed
10	3.6e-6	2.1e-4	completed	8.7e-7	1.2e-3	completed	4.1e-7	8.4e-5	completed
15	3.7e-6	2.5e-4	completed	5.5e-10	2.5e-2	completed	6.1e-7	1.1e-4	completed
20	**7.3e-7**	**8.0e-5**	**completed**	✗	✗	not trained	2.7e-7	8.4e-4	completed
30	2.4e-6	2.4e-4	completed	✗	✗	not trained	5.3e-7	1.1e-4	completed



(a) PINN (b) PI-ResNet (c) PI-SqrResNet

Figure 6: Example 5: Diffusion equation, CPU/L-BFGS- A comparison of PINN (left), PI-ResNet (middle) and PI-SqrResNet (right) with NL=15.

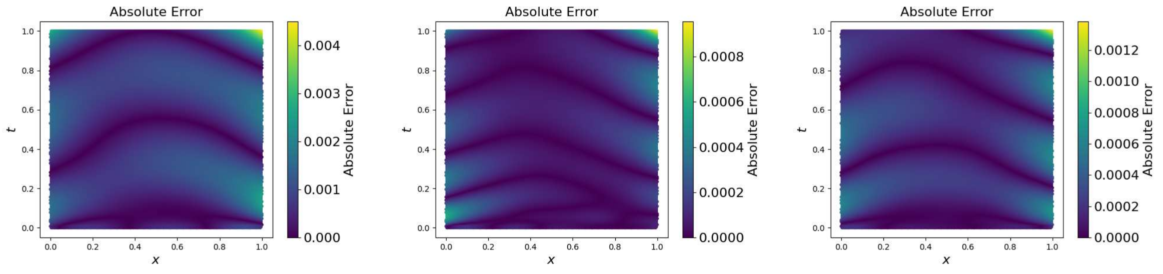
Table 7 presents the results for PINN, PI-ResNet, and PI-SqrResNet. While PINN struggles to train with the smallest number of hidden layers in our list, it demonstrates stable performance for deeper networks. In contrast, the performance of PI-ResNet , in terms of error, decreases as the number of hidden layers increases, even though the train loss diminishes. The situation worsens for PI-ResNet , where the last two NL values fail to be trained. On the other hand, PI-SqrResNet remains stable and successfully completes training for all cases.

Based on the performance of PINN, we conclude that selecting an appropriate hyperparameter, such as the number of hidden layers in the neural network, is a problem-dependent factor. For instance, a small number of NL fails here, while NL=30 works fine with a small error. Therefore, a robust algorithm like the proposed PI-SqrResNet , capable of handling various problems, is essential.

The absolute error for the case with NL=15 using PINN, PI-ResNet, and PI-SqrResNet is depicted in Fig. 6. It is evident that PI-ResNet exhibits the largest error among our methods. Additionally, we observe that the error distribution for PI-SqrResNet is more evenly spread compared to PINN.

Example 6 We will solve a heat equation given by:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1], \quad t \in [0, 1] \tag{25}$$



(a) PINN (b) PI-ResNet (c) PI-SqrResNet

Figure 7: Example 6: Heat equation, CPU/L-BFGS- A comparison of PINN (left), PI-ResNet (middle) and PI-SqrResNet (right) with NL=15.

where $\alpha = 0.4$ is the thermal diffusivity constant. With Dirichlet boundary conditions:

$$u(0,t) = u(1,t) = 0 \tag{26}$$

and periodic (sinusoidal) initial condition:

$$u(x,0) = \sin\left(\frac{n\pi x}{L}\right), \quad 0 < x < L, \quad n = 1, 2, \ldots \tag{27}$$

The exact solution is:

$$u(x,t) = e^{-\frac{n^2\pi^2\alpha t}{L^2}} \sin\left(\frac{n\pi x}{L}\right) \tag{28}$$

where $L = 1$ is the length of the bar, and $n = 1$ is the frequency of the sinusoidal initial conditions.

Table 8: Example 6: Heat equation, CPU/L-BFGS- A comparison of PINN, PI-ResNet, and PI-SqrResNet.

	PINN			PI-ResNet			PI-SqrResNet		
NL	train loss	error	status	train loss	error	status	train loss	error	status
5	1.0e-6	7.0e-4	completed	5.0e-7	5.3e-4	completed	4.8e-7	4.2e-4	completed
10	3.6e-6	1.9e-3	diverged*	5.8e-7	9.4e-4	completed	6.5e-7	4.9e-4	completed
15	5.2e-6	3.5e-3	diverged†	6.7e-7	5.9e-4	completed	7.4e-7	9.3e-4	completed
20	1.2e-6	7.9e-4	completed	**2.3e-7**	**2.6e-4**	**completed**	5.3e-7	6.7e-4	completed
30	✗	✗	not trained	8.5e-7	7.8e-4	completed	4.6e-7	5.8e-4	completed

* Diverged after step = 1000.
† Diverged after step = 1000.

A total of 2780 collocation points, including 2540 interior, 80 boundary, and 160 initial points, are considered for this example. Table 8 presents the results for PINN, PI-ResNet, and PI-SqrResNet with respect to the NL. PINN completed 5000 steps only for NL=5 and 20, while NL=30 failed to train, producing large loss and error from the beginning of computation. Two cases, NL=10 and 15, produced large loss values after step=1000. On the other hand, both PI-ResNet and PI-SqrResNet completed the predefined number of

Table 9: Example 6: Heat equation, GPU/Adam- A comparison of PINN and ResNet-PINN.

	PINN			PI-ResNet			PI-SqrResNet		
NL	train loss	error	status	train loss	error	status	train loss	error	status
5	1.6e-5	3.7e-3	completed	2.3e-5	2.4e-3	completed	3.6e-6	1.8e-3	completed
10	2.4e-5	3.2e-3	completed	2.4e-5	2.7e-3	completed	5.3e-6	1.0e-3	completed
15	3.6e-5	9.3e-3	completed	1.2e-4	1.0e-2	completed	**2.9e-6**	**7.1e-4**	**completed**
20	3.0e-5	8.4e-3	completed	2.0e-2	1.9e-1	completed	3.0e-6	1.0e-3	completed
30	1.0e-4	1.6e-2	completed	✗	✗	not trained	4.7e-6	2.3e-3	completed
40	✗	✗	not trained	✗	✗	not trained	4.8e-6	1.5e-3	completed
50	✗	✗	not trained	✗	✗	not trained	7.4e-6	1.2e-3	completed

iterations (5000) for all NL cases. However, the error for all cases with completed status is very close, with the best accuracy achieved for NL=20 using PI-ResNet.

Moreover, Fig. 7 presents the profile of the absolute error for the case with NL=15, using PINN, PI-ResNet, and PI-SqrResNet. The figure showcases the best training results concerning the training loss value with respect to the steps. From these plots, it can be observed that PI-ResNet produces the smallest maximum absolute error, which occurred at step=5000. On the other hand, the plot shows the profile of the error for PINN at step=1000, which is very large compared with the other two methods due to incomplete training processes.

Table 9 presents the results using the Adam optimizer on the GPU of the Google Colab platform. In comparison to Table 8, the performance regarding the completion of the training process is more stable for PINN up to NL=30. However, as NL increases, both accuracy and train loss also increase; nonetheless, the last two NL values, i.e., NL=40 and 50, fail to train. PI-ResNet exhibits poor performance as NL increases, in contrast to the corresponding results using CPU/L-BFGS in Table 8. On the other hand, the proposed PI-SqrResNet demonstrates very stable results concerning the number of hidden layers for both train loss and error.

Example 7 This example aims to solve a nonlinear Schrödinger equation along with periodic boundary conditions which is given by

$$i\frac{\partial h}{\partial t} + 0.5\frac{\partial^2 h}{\partial x^2} + |h|^2 h = 0, \quad x \in [-5, 5], \quad t \in [0, \pi/2], \tag{29}$$

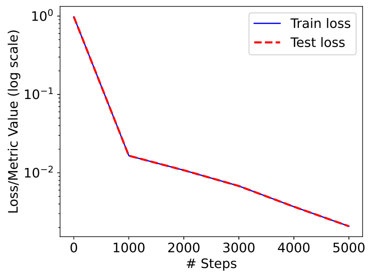
where $h(0, x) = 2\text{sech}(x)$, $h(t, -5) = h(t, 5)$, and $\frac{\partial h}{\partial x}(t, -5) = \frac{\partial h}{\partial x}(t, 5)$. Here, $h(t, x)$ is the complex-valued solution. Let us define $f(t, x)$ to be given by

$$f := i\frac{\partial h}{\partial t} + 0.5\frac{\partial^2 h}{\partial x^2} + |h|^2 h, \tag{30}$$

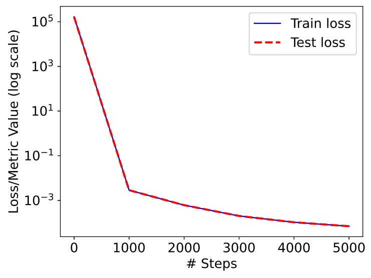
and proceed by placing a complex-valued neural network prior on $h(t, x)$. the exact solution derived from reference [1].

Table 10: Example 7: nonlinear Schrödinger equation, CPU/L-BFGS- A comparison of PINN, PI-ResNet, and PI-SqrResNet.

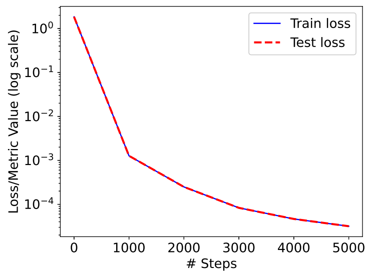
	PINN				PI-ResNet				PI-SqrResNet			
NL	train loss	ϵ(u)	ϵ(v)	status	train loss	ϵ(u)	ϵ(v)	status	train loss	ϵ(u)	ϵ(v)	status
5	1.4e-4	5.5e-2	9.2e-2	completed	8.1e-5	3.9e-2	6.6e-2	completed	1.1e-5	4.7e-3	7.1e-3	completed
10	3.8e-5	6.8e-3	1.1e-2	completed	4.2e-5	1.1e-2	1.9e-2	completed	1.1e-5	2.6e-3	3.4e-3	completed
15	✗	✗	✗	not trained	3.3e-5	1.5e-2	2.6e-2	completed	**6.7e-6**	**1.9e-3**	**2.6e-3**	**completed**
20	1.1e-4	2.6e-2	4.4e-2	completed	4.2e-5	1.3e-2	2.2e-2	completed	3.5e-5	5.7e-3	9.6e-3	completed
30	2.0e-3	3.1e-1	5.0e-1	completed	6.9e-5	1.5e-2	2.4e-2	completed	3.1e-5	1.2e-2	2.2e-2	completed



(a) PINN



(b) PI-ResNet



(c) PI-SqrResNet

Figure 8: Example 7: Nonlinear Schrödinger equation, CPU/L-BFGS- A comparison of PINN (left), PI-ResNet (middle) and PI-SqrResNet (right) with NL=30.

In this example, 10,000 interior points, 20 boundary points, and 200 points for the initial condition are considered. Table 10 presents the train loss. It also lists the error on the real and imaginary terms shown as ϵ(u) and ϵ(v). In this example, PINN behaves more stably than in previous examples in terms of completing the training process. However, still at NL=15, a divergence after step=1000 occurred. It can also be seen that there is a rise in errors at NL=30. The residual-based method, PI-ResNet, behaves more stably by keeping the order of error magnitudes at -2. However, the best performance can be seen for PI-SqrResNet, where the smallest errors compared to the two other methods are observed, and the results are stable as NL increases. Note that the best accuracy happened at NL=15 using PI-SqrResNet. Fig. 8 shows the loss and metric values with respect to the step. Clearly, PI-SqrResNet leads to the best training performance in terms of accuracy and convergence, and PINN is the worst one.

Example 8 We will solve a 2D linear elasticity solid mechanics problem:

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} + f_x = 0, \quad x \in [0,1], \quad y \in [0,1], \tag{31}$$

$$\frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + f_y = 0, \tag{32}$$

where the linear elastic constitutive model is defined as:

$$\sigma_{xx} = (\lambda + 2\mu)\epsilon_{xx} + \lambda\epsilon_{yy}, \tag{33}$$
$$\sigma_{yy} = (\lambda + 2\mu)\epsilon_{yy} + \lambda\epsilon_{xx}, \tag{34}$$
$$\sigma_{xy} = 2\mu\epsilon_{xy}, \tag{35}$$

with the kinematic relation:

$$\epsilon_{xx} = \frac{\partial u_x}{\partial x}, \tag{36}$$
$$\epsilon_{yy} = \frac{\partial u_y}{\partial y}, \tag{37}$$
$$\epsilon_{xy} = \frac{1}{2}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right). \tag{38}$$

The 2D square domain is subjected to body forces:

$$\begin{aligned} f_x &= \lambda[4\pi^2 \cos(2\pi x)\sin(\pi y) - \pi\cos(\pi x)Qy^3] \\ &\quad + \mu[9\pi^2\cos(2\pi x)\sin(\pi y) - \pi\cos(\pi x)Qy^3], \end{aligned} \tag{39}$$
$$\begin{aligned} f_y &= \lambda[-3\sin(\pi x)Qy^2 + 2\pi^2\sin(2\pi x)\cos(\pi y)] \\ &\quad + \mu[-6\sin(\pi x)Qy^2 + 2\pi^2\sin(2\pi x)\cos(\pi y) + \pi^2\sin(\pi x)Qy^4/4], \end{aligned} \tag{40}$$

with displacement boundary conditions:

$$u_x(x,0) = u_x(x,1) = 0, \tag{41}$$
$$u_y(0,y) = u_y(1,y) = u_y(x,0) = 0, \tag{42}$$

and traction boundary conditions:

$$\sigma_{xx}(0,y) = \sigma_{xx}(1,y) = 0, \tag{43}$$
$$\sigma_{yy}(x,1) = (\lambda + 2\mu)Q\sin(\pi x). \tag{44}$$

We set parameters $\lambda = 1$, $\mu = 0.5$, and $Q = 4$. The exact solution is $u_x(x,y) = \cos(2\pi x)\sin(\pi y)$ and $u_y(x,y) = \sin(\pi x)Qy^4/4$.

In this example, we consider 500 interior points and 500 points on the boundary. The results presented in Table 11 indicate a decrease in the performance of PINN as NL increases, with the last two NL values (NL=20 and 30) not being trained successfully. On the contrary, the residual-based methods exhibit better performance, with all cases successfully implemented. Notably, PI-SqrResNet maintains the error's order of magnitude at -3, showcasing a more stable computation across various NL. Additionally, the smallest error is observed at NL=10 using PI-SqrResNet.

Table 11: Example 8: Elastostatic equation, CPU/L-BFGS- A comparison of PINN, PI-ResNet, and PI-SqrResNet with NN=200.

	PINN			PI-ResNet			PI-SqrResNet		
NL	train loss	error	status	train loss	error	status	train loss	error	status
5	3.6e-5	3.2e-3	completed	1.1e-4	9.2e-3	completed	1.9e-5	2.6e-3	completed
10	5.7e-5	4.1e-3	completed	2.5e-5	2.8e-3	completed	**4.1e-5**	**2.7e-3**	**completed**
15	1.9e-4	6.5e-3	completed	5.7e-5	4.5e-3	completed	8.1e-5	4.8e-3	completed
20	✗	✗	not trained	6.7e-5	3.5e-3	completed	1.5e-4	7.1e-3	completed
30	✗	✗	not trained	3.7e-4	1.2e-2	completed	7.9e-5	5.8e-3	completed

[*] Diverged at step = 2000.

6 Conclusion

In this paper, we introduce a groundbreaking approach, PI-SqrResNet, aimed at bolstering stability in deep learning applications for partial differential equations (PDEs). The method integrates squared residual terms into the Residual Network architecture, providing a unique perspective on stability challenges. Our systematic evaluation across a spectrum of PDE scenarios, ranging from Poisson equations to Schrödinger equations, reveals the method's superior performance compared to traditional Residual Networks or standalone feedforward networks. By combining the robustness of squared residual terms with adaptive feedforward architectures, PI-SqrResNet addresses inherent stability issues, enhancing accuracy and convergence in the numerical approximation of PDE solutions.

Empirical results highlight the method's efficacy across diverse scenarios, showcasing its potential impact on computational physics and scientific computing. The proposed approach not only presents a viable solution to stability challenges in deep learning for PDEs but also contributes to a broader understanding of the interplay between architecture choices and performance outcomes. PI-SqrResNet emerges as a robust and stable framework, demonstrating its potential to advance the field of deep learning for PDEs and foster applications in computational physics with improved stability and accuracy.

Acknowledgments

Authors gratefully acknowledge the financial support of the Ministry of Science and Technology (MOST) of Taiwan under grant numbers 111-2811-E002-062 and 111-2221-E-002 -054 -MY3.

References

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

[2] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, Identity mappings in deep residual networks. arXiv preprint arXiv:1603.05027, 2016.

[4] H. Li, Z. Xu, G. Taylor, C. Studer, T. Goldstein, Visualizing the Loss Landscape of Neural Nets, arxiv.1712.09913v3, 2018.

[5] A. Veit, M. Wilber, and S. Belongie, Residual networks behave like ensembles of relatively shallow networks, in Proceedings of the 30th International Conference on Neural Information Processing Systems. Curran Associates Inc.: Barcelona, Spain. p. 550–558, 2016.

[6] S. Jastrzębski, Arpit D, Ballas N, Verma V, Che T, Bengio Y: Residual Connections Encourage Iterative Inference. arXiv:1710.04773, 2017.

[7] L. Lu, M. Dao, P. Kumar, U. Ramamurty, G.E. Karniadakis, and S. Suresh, Extraction of mechanical properties of materials through deep learning from instrumented indentation. Proceedings of the National Academy of Sciences, 117(13): p. 7052-7062, 2020.

[8] S. Wang, Y. Teng, and P. Perdikaris, Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks. SIAM Journal on Scientific Computing, 43(5): p. A3055-A3081, 2021.

[9] L. Lu, X. Meng, Z. Mao, and G.E. Karniadakis, DeepXDE: A Deep Learning Library for Solving Differential Equations. SIAM Review, 2021. 63(1): p. 208-228.

[10] S. Wang, H. Wang, and P. Perdikaris, Improved Architectures and Training Algorithms for Deep Operator Networks. Journal of Scientific Computing, 2022. 92(2): p. 35.