Physics-Informed Neural Networks for PDEs

Amir Mohammad Tavakkoli
Kahlert School of Computing, University of Utah

1 Introduction

Physics-Informed Neural Networks (PINNs) [2] represent a transformative approach in computational science, particularly in solving Partial Differential Equations (PDEs). PDEs are everywhere in scientific and engineering disciplines such as physics, chemistry, biology, and finance. They are used to model various phenomena, including heat transfer, fluid dynamics, and electromagnetics. Traditionally, numerical methods like finite element and finite difference methods have been employed to find approximate solutions to these equations. However, these methods can be computationally expensive and may not scale well with the problem's complexity or dimensionality. Developing PINNs is motivated by addressing several challenges in traditional numerical simulation methods for solving PDEs.

PINNs introduce a novel approach by leveraging deep learning capabilities to approximate complex functions. The integration of physical laws directly into the learning process of neural networks as part of optimizing loss function ensures that the solutions respect the governing equations. This is particularly significant in scientific fields where data collection is expensive or infeasible which guides the learning process and enables the network to predict physically plausible solutions even with limited data.

Overall, PINNs offer the potential to revolutionize the way simulations, and predictions are performed in various fields by:

- **Reducing Computational Costs:** Once trained, PINNs can rapidly make predictions, preventing repetitive and expensive simulations typically required in iterative numerical solvers.
- **Improving Solution Accuracy:** By constraining solutions within the bounds of known physical laws, PINNs can enhance the accuracy and reliability of predictions, especially in extrapolation to unseen conditions.

This project provide implementation of PINNs to solve various well-known PDEs using the Pytorch framework. The focus is on demonstrating the capability of PINNs while ensuring that the solutions satisfy physical laws. Each implementation solves the PDEs and serves as a benchmark for showcasing the effectiveness of PINNs in different scenarios ranging from heat transfer to fluid dynamics. In the following sections, we detail the specific implementations for each PDE, highlight the unique aspects of using PINNs for these equations, and discuss the results and insights gained from these models.

2 Implementation Details

2.1 Manufactured Solutions

An important aspect of developing and validating Physics-Informed Neural Networks (PINNs) involves the use of manufactured solutions. Manufactured solutions are artificially created scenarios where the exact solution to a PDE is known in advance and can be used to rigorously test and validate the numerical methods employed by the neural network. This methodology is crucial for validating the reliability of PINNs before they are applied to real-world problems where true solutions are unknown.

2.2 Experiemnts setup

All experiments are done on CHPC machines using NVIDIA RTX A6000 GPU and AMD EPYC 7513 32-Core Processor.

3 Partial Differential Equations (PDEs)

3.1 2D Heat Equation

The 2D Heat Equation, a parabolic PDE, models the distribution of heat (or variations in temperature) in a given area over time. It is generally expressed as:

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u + f(x, y, t)$$

where u represents the temperature, t represents time, α is the thermal diffusivity of the material, and ∇^2 is the Laplacian operator representing the sum of the second partial derivatives with respect to the spatial variables.

3.1.1 Manufactured Solution

To validate our PINN, we employ a manufactured solution where the exact form of $u(x, y, t)$ is predetermined. For the 2D Heat Equation, we choose:

$$u(x, y, t) = \sin(\pi x)\sin(\pi y)e^{-2\pi^2 \alpha t}$$

This function satisfies the boundary conditions and the initial conditions typically required for the heat equation over a domain bounded by $0 \leq x, y \leq 1$ and $t \geq 0$.

3.1.2 Derivatives and Loss Function

For the 2D Heat Equation using the chosen manufactured solution:

$$u(x, y, t) = \sin(\pi x)\sin(\pi y)e^{-2\pi^2 \alpha t}$$

the exact derivatives are computed as follows:

$$u_t = -2\pi^2 \alpha \sin(\pi x)\sin(\pi y)e^{-2\pi^2 \alpha t}$$

$$u_{xx} + u_{yy} = -2\pi^2 \sin(\pi x)\sin(\pi y)e^{-2\pi^2 \alpha t}$$

For the predictions made by the neural network, we denote the predicted derivatives as $u_{t_{\text{pred}}}$, $u_{xx_{\text{pred}}}$, and $u_{yy_{\text{pred}}}$. The predicted source term f_{pred} is then calculated:

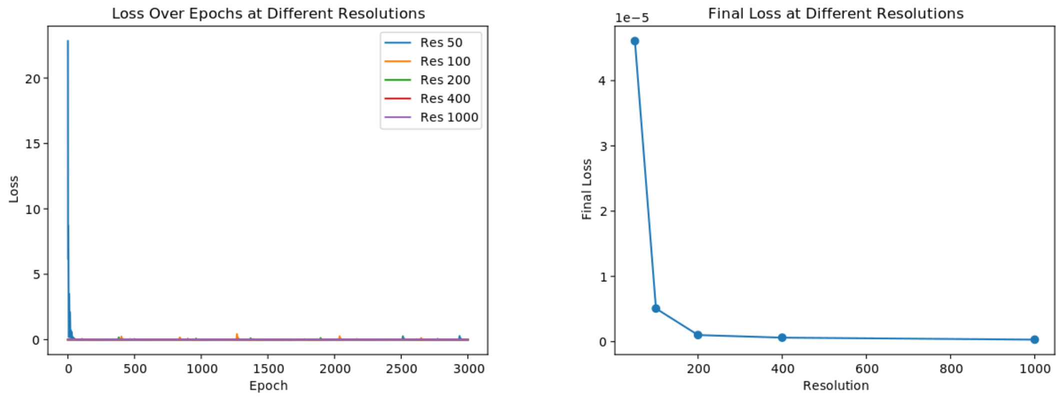
$$f_{\text{pred}} = u_{t_{\text{pred}}} - \alpha(u_{xx_{\text{pred}}} + u_{yy_{\text{pred}}})$$

The neural network is trained to minimize the difference between f_{pred} and f_{true}, by minimizing the squared error, forming the following loss function:

$$\text{Loss} = (f_{\text{pred}} - f_{\text{true}})^2$$

3.1.3 Training and Validation

The training uses the Adam optimizer with a learning rate of 0.001 for 3000 epochs for resolutions from 50 to 1000 points over the spatial and temporal domains and $\alpha = 100$.



(a) Loss Convergence (b) Error Convergence

Figure 1: Result of 2D heat Equation based on various resolutions for the grid

3.2 2D Poisson Equation

The 2D Poisson Equation, an elliptic PDE, is widely used to model potential fields, such as electrostatic or gravitational fields. It is expressed as:

$$\nabla^2 u = f(x, y)$$

where u represents the potential field and $f(x, y)$ is the source term, which may vary across the domain.

3.2.1 Manufactured Solution

For validating our PINN model, we employ a manufactured solution for the 2D Poisson Equation. The solution is chosen based on its simplicity and ability to meet typical boundary conditions. We define:

$$u(x, y) = \sin(\pi x)\cos(\pi y)$$

This function satisfies the Dirichlet boundary conditions $u = 0$ on the edges of the domain, assuming the domain is $0 \le x, y \le 1$.

3.2.2 Derivatives and Loss Function

For the 2D Poisson Equation using the chosen manufactured solution, the exact Laplacian $\nabla^2 u$ is calculated as:

$$u_{xx} = -\pi^2 \sin(\pi x)\cos(\pi y)$$

$$u_{yy} = -\pi^2 \sin(\pi x)\cos(\pi y)$$

Adding these, we get:

$$\nabla^2 u = u_{xx} + u_{yy} = -2\pi^2 \sin(\pi x)\cos(\pi y)$$

We define the source term $f(x, y)$ based on this Laplacian, which should match the right-hand side of the Poisson equation:

$$f(x, y) = -2\pi^2 \sin(\pi x)\cos(\pi y)$$

For the neural network predictions, we denote the predicted Laplacian as $\nabla^2 u_{\text{pred}}$. The predicted source term f_{pred} is then:

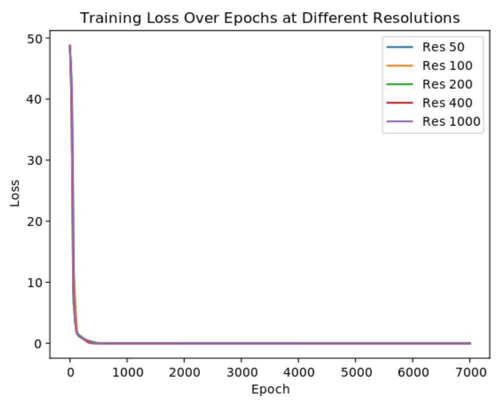
$$f_{\text{pred}} = \nabla^2 u_{\text{pred}}$$

The neural network is trained to minimize the difference between f_{pred} and the true source term $f(x, y)$, using the squared error as the loss function:

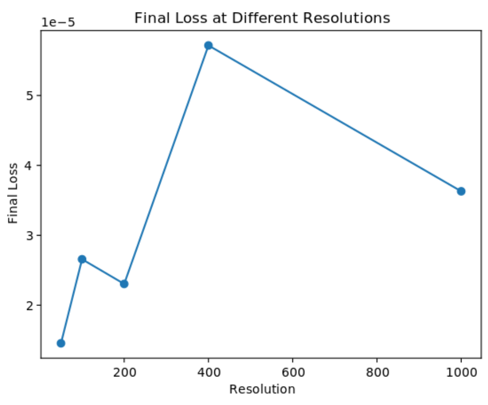
$$\text{Loss} = (f_{\text{pred}} - f(x, y))^2$$

3.2.3 Training and Validation

The training uses the Adam optimizer with a learning rate of 0.001 for 7000 epochs for resolutions from 50 to 1000 points over the spatial and temporal domains.



(a) Loss Convergence



(b) Error Convergence

Figure 2: Result of 2D Poisson Equation based on various resolutions for the grid

3.3 2D Wave Equation

The 2D Wave Equation is a fundamental equation in physics for modeling wave propagation, such as sound waves or light waves, in two dimensions. It can be expressed as:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u$$

where $u(x, y, t)$ represents the wave function, c is the wave speed, t is time, and ∇^2 denotes the Laplacian in two spatial dimensions.

3.3.1 Manufactured Solution

For testing and validating our PINN, I have chosen a manufactured solution known to satisfy the 2D Wave Equation under ideal conditions. The solution is given by:

$$u(x, y, t) = \sin(\pi x)\sin(\pi y)\cos(\omega t)$$

This function represents a wave that oscillates in time with frequency ω and spatially along both x and y directions. The choice of sinusoidal functions ensures the satisfaction of typical boundary conditions for a rectangular domain where u vanishes at the boundaries.

3.3.2 Derivatives and Loss Function

The second time derivative (u_{tt}) and the Laplacian ($u_{xx} + u_{yy}$) of the manufactured solution are:

$$u_{tt} = -\omega^2 \sin(\pi x)\sin(\pi y)\cos(\omega t)$$

$$u_{xx} + u_{yy} = -2\pi^2 \sin(\pi x)\sin(\pi y)\cos(\omega t)$$

From these derivatives, we calculate the true source term f_{true}:

$$f_{\text{true}} = u_{tt} - c^2(u_{xx} + u_{yy}) = (\omega^2 - 2\pi^2 c^2)\sin(\pi x)\sin(\pi y)\cos(\omega t)$$

For the neural network predictions, we define f_{pred} as the difference between the network's predicted second time derivative and the predicted Laplacian:

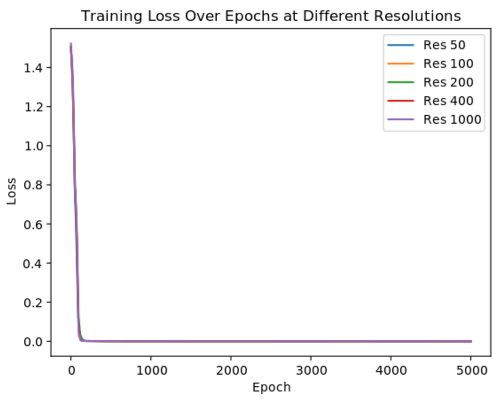
$$f_{\text{pred}} = u_{tt_{\text{pred}}} - c^2(u_{xx_{\text{pred}}} + u_{yy_{\text{pred}}})$$

The loss function for training the neural network is thus the mean squared error between f_{pred} and f_{true}:

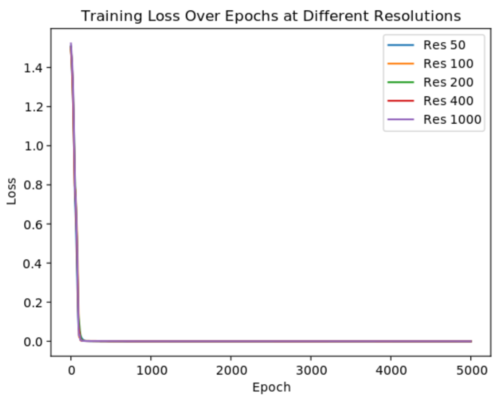
$$\text{Loss} = (f_{\text{pred}} - f_{\text{true}})^2$$

3.3.3 Training and Validation

The training uses the Adam optimizer with a learning rate of 0.001 for 5000 epochs for resolutions from 50 to 1000 points over the spatial and temporal domains with $c = 0.5$.



(a) Loss Convergence



(b) Error Convergence

Figure 3: Result of 2D Wave Equation based on various resolutions for the grid

3.4 Burgers' Equation

Burgers' Equation is a fundamental nonlinear partial differential equation used extensively in various fields such as fluid mechanics, traffic flow, and gas dynamics. It combines both advection and diffusion effects and is expressed as:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$

where $u(x,t)$ represents the fluid velocity, ν is the kinematic viscosity, and t and x represent time and space, respectively.

3.4.1 Manufactured Solution

For validation of our PINN, we select a manufactured solution that is known to satisfy Burgers' Equation under specific conditions. The chosen solution is:

$$u(x,t) = \sin(\pi x)e^{-\nu\pi^2 t}$$

3.4.2 Derivatives and Loss Function

The implementation of the PINN requires calculating the time derivative (u_t) and the second spatial derivative (u_{xx}) of the manufactured solution. These derivatives are key to constructing the nonlinear and diffusive terms in Burgers' Equation.

Given the manufactured solution, we calculate the true source term f_{true} by substituting into Burgers' Equation:

$$f_{\text{true}} = u_t + uu_x - \nu u_{xx}$$

Automatic differentiation tools in PyTorch [1] can be used to ensure accuracy when computing these derivatives for the PINN. After computation, the derivatives are used to form the predicted source term f_{pred} as follows:

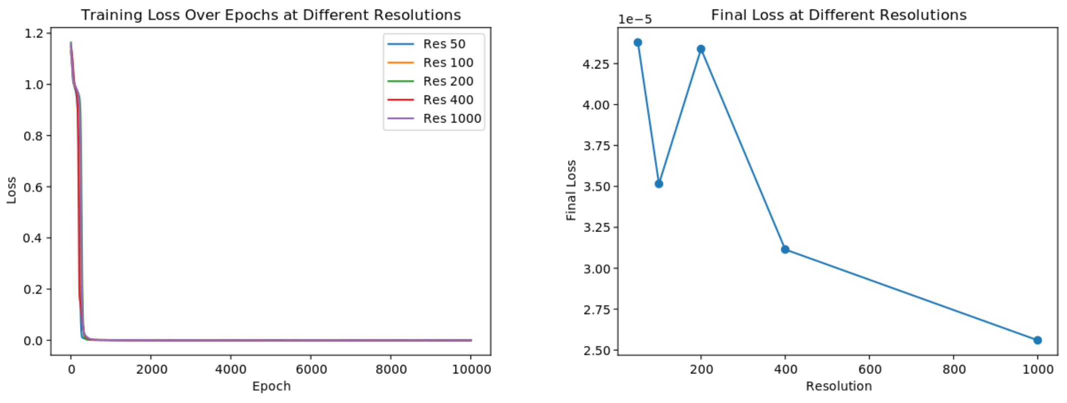
$$f_{\text{pred}} = u_{t_{\text{pred}}} + u_{\text{pred}} u_{x_{\text{pred}}} - \nu u_{xx_{\text{pred}}}$$

The loss function for the PINN is then the mean squared error between f_{pred} and f_{true}:

$$\text{Loss} = (f_{\text{pred}} - f_{\text{true}})^2$$

3.4.3 Training and Validation

The training uses the Adam optimizer with a learning rate of 0.001 for 10000 epochs for resolutions from 50 to 1000 points over the spatial and temporal domains with $\nu = \frac{0.01}{\pi}$.



(a) Loss Convergence (b) Error Convergence

Figure 4: Result of Burgers' Equation based on various resolutions for the grid

3.5 Helmholtz Equation

The Helmholtz Equation often arises in problems involving wave propagation and vibration, and it is used to describe phenomena in both acoustics and electromagnetics. The equation is generally expressed as:

$$\nabla^2 u + k^2 u = 0$$

where $u(x, y)$ represents the wave function, and k is the wave number, which is related to the frequency of the wave.

3.5.1 Manufactured Solution

To test the effectiveness of our PINN, we employ a manufactured solution that satisfies the boundary conditions and the Helmholtz Equation. The chosen solution for this purpose is:

$$u(x, y) = \sin(\pi x)\sin(\pi y)$$

This solution is particularly suitable because it naturally satisfies zero boundary conditions for a domain where x, y range from -1 to 1.

3.5.2 Derivatives and Loss Function

To implement the PINN, we first compute the Laplacian ($\nabla^2 u$) of the manufactured solution

$$\nabla^2 u = -2\pi^2 \sin(\pi x)\sin(\pi y)$$

Given the manufactured solution, we calculate the true source term f_{true} by substituting into the Helmholtz equation:

$$f_{\text{true}} = \nabla^2 u + k^2 u = -2\pi^2 \sin(\pi x)\sin(\pi y) + k^2 \sin(\pi x)\sin(\pi y)$$

For the neural network predictions, the predicted source term f_{pred} is defined as:

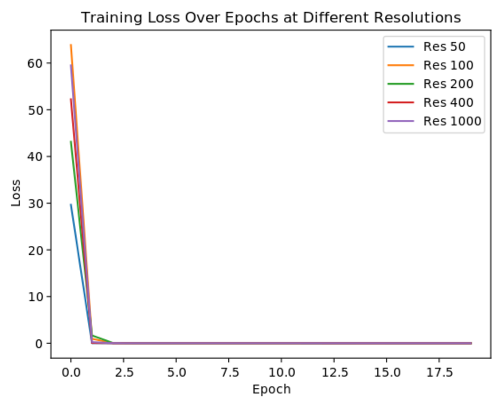
$$f_{\text{pred}} = \nabla^2 u_{\text{pred}} + k^2 u_{\text{pred}}$$

The loss function for training the neural network is thus the mean squared error between f_{pred} and f_{true}:

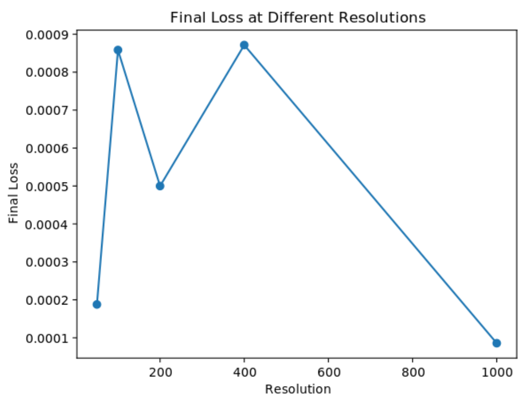
$$\text{Loss} = (f_{\text{pred}} - f_{\text{true}})^2$$

3.5.3 Training and Validation

The training uses the Adam optimizer with a learning rate of 0.001 for 1000 epochs for resolutions from 50 to 1000 points over the spatial and temporal domains with $k = 3$.



(a) Loss Convergence



(b) Error Convergence

Figure 5: Result of Helmholtz Equation based on various resolutions for the grid

3.6 Navier-Stokes Equation

The Navier-Stokes Equation describes the motion of viscous fluid substances and is a fundamental equation in fluid dynamics. It combines the conservation laws of momentum and mass (continuity equation) and is expressed for an incompressible flow as:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f}$$

$$\nabla(\mathbf{u}) = 0$$

where $\mathbf{u}$ represents the velocity field, p is the pressure field, ρ is the density, ν is the kinematic viscosity, and $\mathbf{f}$ is the body forces acting on the fluid.

3.6.1 Manufactured Solution

To validate our PINN, we use a manufactured solution that is known to satisfy the Navier-Stokes Equation under specific conditions. The solution components are defined as follows:

$$u(x, y, t) = -\sin(\pi x)\cos(\pi y)e^{-2\pi^2 \nu t}$$

$$v(x, y, t) = \cos(\pi x)\sin(\pi y)e^{-2\pi^2 \nu t}$$

$$p(x, y, t) = \sin(\pi x)\sin(\pi y)e^{-4\pi^2 \nu t}$$

These functions represent the velocity components in the x and y directions, and the pressure field, respectively.

3.6.2 Derivatives and Loss Function

For the PINN implementation, we first compute the necessary derivatives and terms needed for the Navier-Stokes equations, focusing on incompressibility ($\nabla \cdot \mathbf{u} = 0$), momentum, and continuity equations.

Given the manufactured solution, we calculate the true source terms $\mathbf{f}_{\text{true}}$ by substituting into the Navier-Stokes equations. For the neural network predictions, the predicted source terms $\mathbf{f}_{\text{pred}}$ are defined as:

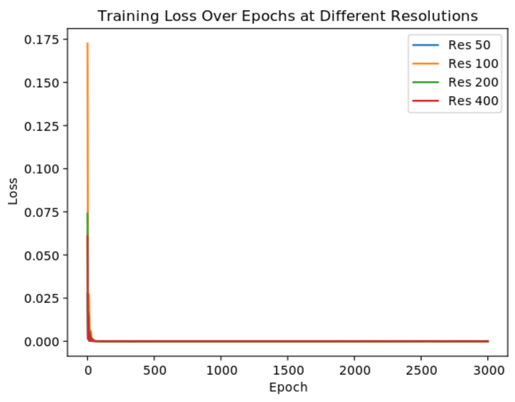
$$\mathbf{f}_{\text{pred}} = \frac{\partial \mathbf{u}_{\text{pred}}}{\partial t} + (\mathbf{u}_{\text{pred}} \cdot \nabla)\mathbf{u}_{\text{pred}} + \frac{1}{\rho}\nabla p_{\text{pred}} - \nu \nabla^2 \mathbf{u}_{\text{pred}}$$

The loss function for the PINN is then the mean squared error between $\mathbf{f}_{\text{pred}}$ and $\mathbf{f}_{\text{true}}$:

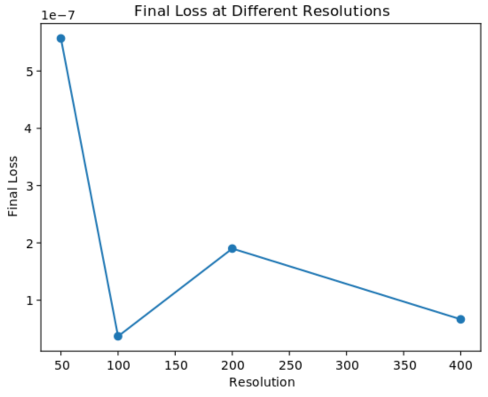
$$\text{Loss} = (\mathbf{f}_{\text{pred}} - \mathbf{f}_{\text{true}})^2$$

3.6.3 Training and Validation

The training uses the Adam optimizer with a learning rate of 0.001 for 3000 epochs for resolutions from 50 to 1000 points over the spatial and temporal domains with $\rho = 1$ and $\nu = 0.1$.



(a) Loss Convergence



(b) Error Convergence

Figure 6: Result of Navier-Stokes Equation based on various resolutions for the grid

4 Conclusion

This work demonstrates how Physics-Informed Neural Networks can be effectively applied to solve complex Partial Differential Equations, bridging the gap between deep learning techniques and physical sciences. Future work will focus on improving the models' accuracy and efficiency and extending their applicability to more complex and unexplored PDEs.

References

[1] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[2] Maziar Raissi, Paris Perdikaris, and George 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*, 378:686–707, 2019.