Physics-Informed Neural Networks

Hamza El Otmani
LJ2424105

June 22, 2024

Abstract

This report explores the use of Physics-Informed Neural Networks (PINNs) for solving partial differential equations (PDEs), specifically the heat and wave equations. Unlike traditional neural networks, PINNs integrate physical laws directly into the network's loss function, ensuring data-efficient and physically consistent solutions. We demonstrate the implementation of PINNs using the DeepXDE library to solve these equations under specified initial and boundary conditions, showcasing the method's accuracy and adherence to physical constraints. Despite promising results, challenges such as training complexity, hyperparameter tuning, and generalization persist, indicating areas for future research. This study underscores PINNs' potential in merging machine learning with physical sciences, promising advancements in solving complex scientific and engineering problems.

Contents

1 Introduction **2**

2 Traditional Neural Networks vs. PINNs **2**
2.1 Traditional Neural Networks (NNs) 2
2.2 Physics-Informed Neural Networks (PINNs) 3
2.3 Example 3

3 Physics-Informed Neural Networks **4**

4 Application **5**
4.1 The Heat Equation 6
4.1.1 Implementation Steps 6
4.1.2 Results 7
4.2 Wave Equation 8
4.2.1 Implementation Steps 8
4.2.2 Results 10

5 Discussion **10**

6 Challenges Associated with PINNs **10**

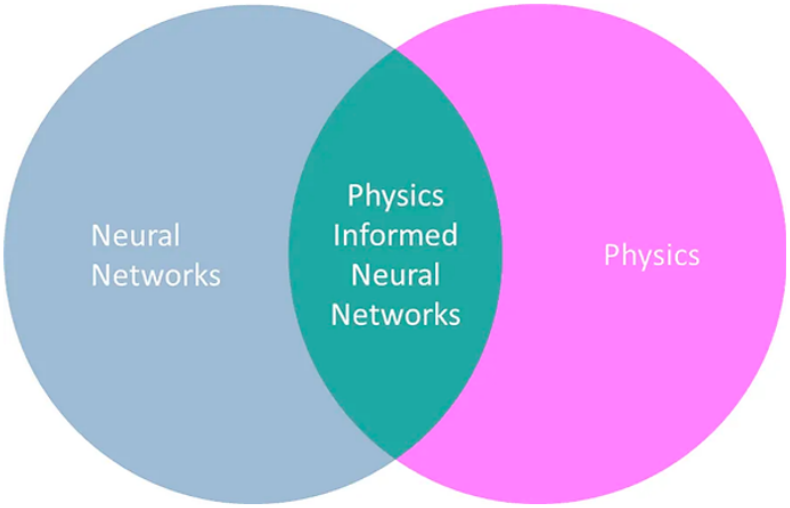
7 Conclusion **11**

1 Introduction

Neural networks have revolutionized various fields by offering powerful tools for pattern recognition, data analysis, and prediction. However, traditional neural networks primarily rely on large datasets to learn complex relationships, which can be a limitation in domains where data is scarce or expensive to obtain. To address this, Physics-Informed Neural Networks (PINNs) have emerged as a novel approach that incorporates physical laws into the learning process, providing a robust framework for solving scientific and engineering problems governed by partial differential equations (PDEs).

PINNs leverage the underlying physical principles of a system by embedding these principles directly into the neural network's loss function. This integration ensures that the network's predictions adhere to the known laws of physics, leading to models that are not only accurate but also generalizable with limited data. This characteristic makes PINNs particularly valuable for forward and inverse problems in scientific computing, where the exact solution must satisfy certain physical constraints.

In this report, we explore the concept and mechanisms of PINNs and demonstrate their application through two case studies: the heat equation and the wave equation. The heat equation is a fundamental PDE in thermal physics that describes the distribution of heat in a given region over time. The wave equation, on the other hand, is a fundamental PDE in physics that describes the propagation of waves, such as sound waves, light waves, and water waves, through a medium.



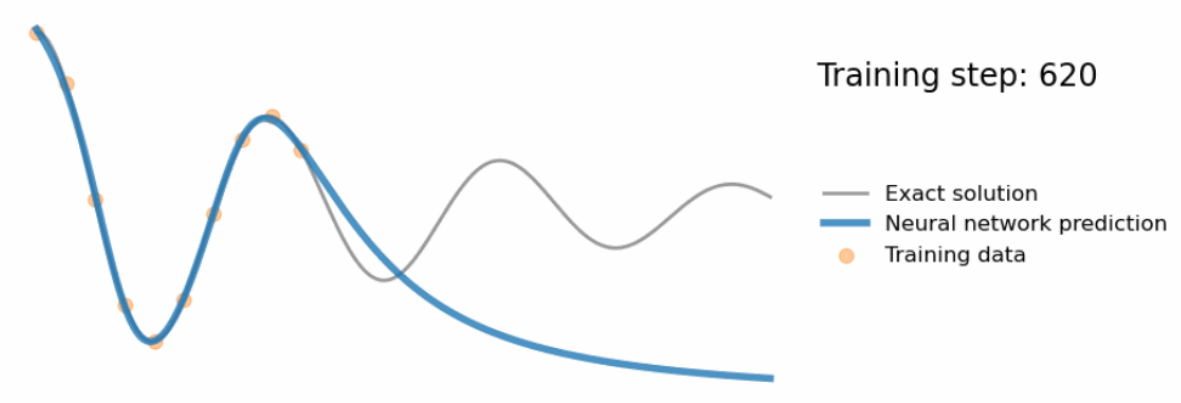
2 Traditional Neural Networks vs. PINNs

2.1 Traditional Neural Networks (NNs)

Traditional neural networks (NNs) are powerful tools for modeling complex relationships from data. They are trained by minimizing a loss function that measures the discrepancy between the network's predictions and true values in a labeled dataset. However, they have limitations:

- **Data Dependency**: Require large datasets for training, which may not be available.
- **Lack of Physical Consistency**: Predictions may not adhere to known physical laws.

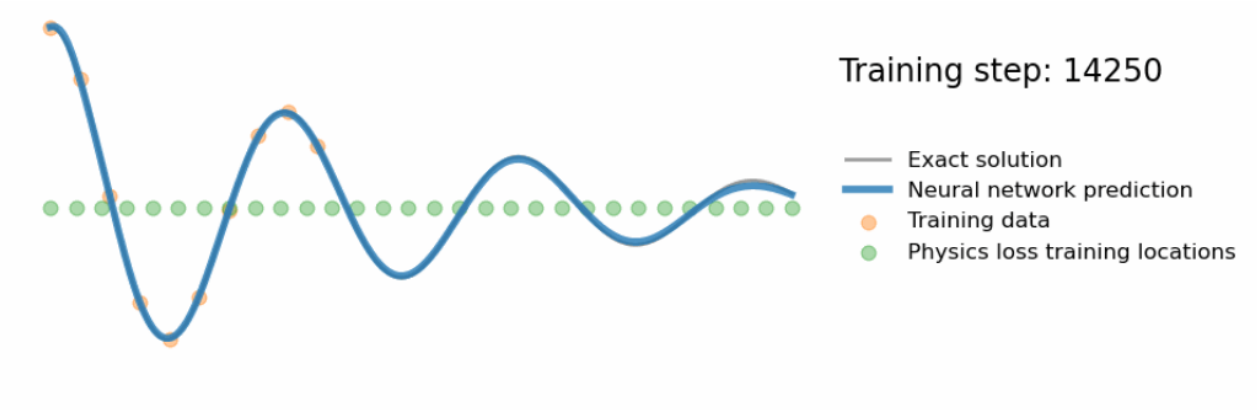
- **Generalization**: May struggle to generalize well to new, unseen data, especially with limited training data.



2.2 Physics-Informed Neural Networks (PINNs)

Physics-Informed Neural Networks (PINNs) integrate physical laws directly into the training process by embedding partial differential equations (PDEs) into the loss function. This hybrid approach combines data-driven learning with physical principles:

- **Reduced Data Dependency**: Achieve accurate results with limited data by leveraging physical laws.
- **Physical Consistency**: Ensure predictions are physically plausible and adhere to PDE constraints.
- **Generalization**: Often generalize better to new scenarios due to the guidance provided by the physical laws.



2.3 Example

For the heat equation, a traditional NN might need extensive data to learn temperature distribution. A PINN, by incorporating the heat equation into its training, reduces data requirements and ensures temperature predictions obey physical laws.

3 Physics-Informed Neural Networks

Problem Setup: Consider a general partial differential equation (PDE) in a domain $\Omega \subset R^d$:

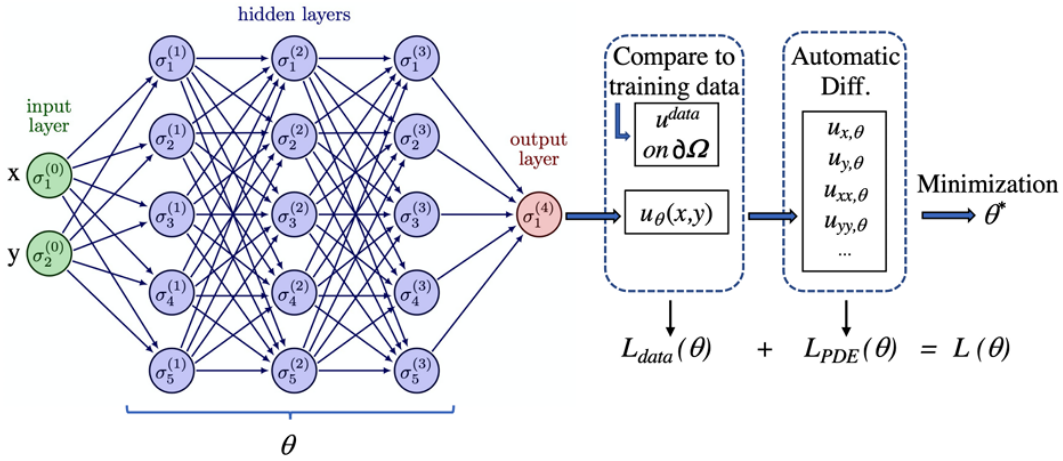
$$\mathcal{F}(u, x, \nabla u, \nabla^2 u, \ldots) = 0, \quad x \in \Omega$$

where $u : \Omega \to R$ is the unknown solution, x denotes the spatial (and possibly temporal) coordinates, ∇u represents the gradient of u, and $\nabla^2 u$ represents the Hessian (second-order partial derivatives).

Boundary conditions (BCs) and initial conditions (ICs) are also imposed as:

$$u(x) = g(x), \quad x \in \partial\Omega$$

Neural Network Approximation: Let $u_\theta(x)$ be a neural network parameterized by θ that approximates the solution $u(x)$. The goal is to find the parameters θ such that $u_\theta(x)$ satisfies the PDE and the boundary/initial conditions.



Loss Function: The loss function for training the PINN is constructed to enforce both the PDE and the boundary/initial conditions.

PDE Residual Loss:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_\Omega} \sum_{i=1}^{N_\Omega} \left| \mathcal{F}(u_\theta(x_i), x_i, \nabla u_\theta(x_i), \nabla^2 u_\theta(x_i), \ldots) \right|^2$$

where $\{x_i\}_{i=1}^{N_\Omega}$ are collocation points in the domain Ω.

Boundary/Initial Condition Loss:

$$\mathcal{L}_{\text{BC}} = \frac{1}{N_{\partial\Omega}} \sum_{j=1}^{N_{\partial\Omega}} |u_\theta(x_j) - g(x_j)|^2$$

where $\{x_j\}_{j=1}^{N_{\partial\Omega}}$ are points on the boundary $\partial\Omega$.

Total Loss:

$$\mathcal{L} = \mathcal{L}_{\text{PDE}} + \lambda \mathcal{L}_{\text{BC}}$$

where λ is a weighting parameter to balance the importance of the PDE residual and boundary condition losses.

Automatic Differentiation: To compute $\nabla u_\theta(x)$ and $\nabla^2 u_\theta(x)$, we use automatic differentiation, which is efficiently handled by modern machine learning frameworks like TensorFlow or PyTorch. This allows us to backpropagate through the PDE residuals and boundary/initial condition errors to update the neural network parameters θ.

Training: The parameters θ are optimized by minimizing the total loss $\mathcal{L}$ using gradient-based optimization algorithms such as Adam or L-BFGS:

$$\theta^* = \arg\min_\theta \mathcal{L}$$

Example

Consider a simple example of a one-dimensional PDE, the Poisson equation:

$$u_{xx} = f(x), \quad x \in (0, 1)$$

with boundary conditions:

$$u(0) = u(1) = 0$$

Neural Network Approximation: Let $u_\theta(x)$ be the neural network approximation.

PDE Residual Loss:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{\partial^2 u_\theta}{\partial x^2}(x_i) - f(x_i) \right|^2$$

Boundary Condition Loss:

$$\mathcal{L}_{\text{BC}} = |u_\theta(0)|^2 + |u_\theta(1)|^2$$

Total Loss:

$$\mathcal{L} = \mathcal{L}_{\text{PDE}} + \lambda \mathcal{L}_{\text{BC}}$$

The neural network $u_\theta(x)$ is trained to minimize this total loss, thereby approximating the solution to the Poisson equation with the given boundary conditions.

4 Application

Physics-Informed Neural Networks (PINNs) leverage the DeepXDE library to solve partial differential equations (PDEs) by integrating the governing physical laws directly into the network's loss function. In this section, we demonstrate the application of PINNs to solve the heat equation and the wave equation using DeepXDE.

4.1 The Heat Equation

Consider the heat equation in one spatial dimension:

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

where $u(x, t)$ represents the temperature at position x and time t, and α is the thermal diffusivity constant.

The domain is $x \in [-1, 1]$ and $t \in [0, 1]$ with the following conditions:

- **Initial Condition:** $u(x, 0) = f(x)$
- **Boundary Conditions:** $u(-1, t) = 0$ and $u(1, t) = 0$

4.1.1 Implementation Steps

1. **Setup and Imports**:

```
import deepxde as dde
import numpy as np

# Define the geometry and time domain
geom = dde.geometry.Interval(-1, 1)
timedomain = dde.geometry.TimeDomain(0, 1)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)
```

2. **Define the PDE and Boundary Conditions**:

```
# Define the PDE (heat equation with alpha = 0.3)
def pde(x, y):
    dy_t = dde.grad.jacobian(y, x, j=1)
    dy_xx = dde.grad.hessian(y, x)
    return (dy_t - dy_xx * 0.3)

# Initial condition u(x, 0) = sin(pi * x) * exp(-x)
def func(x):
    return np.sin(np.pi * x[:, 0:1]) * np.exp(-x[:, 1:])

# Boundary condition
bc = dde.DirichletBC(geomtime, func, lambda _, on_boundary: on_boundary)

# Initial condition
ic = dde.IC(geomtime, func, lambda _, on_initial: on_initial)
```

3. **Create the Dataset and Model**:

```
data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc, ic],
    num_domain=4000,
```

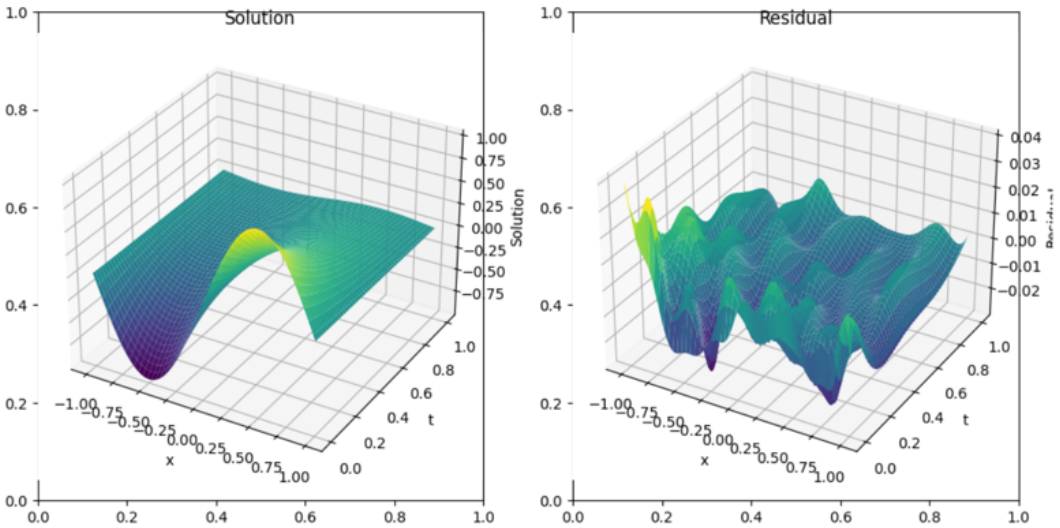
```
    num_boundary=2000,
    num_initial=1000,
    solution=func,
    num_test=1000,
)

layer_size = [2] + [32] * 3 + [1]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

model = dde.Model(data, net)
model.compile("adam", lr=0.01, metrics=["mse"])

losshistory, train_state = model.train(epochs=10000)
```

4. **Predict the Solution and Calculate Residuals**:

```
x_data = np.linspace(-1, 1, num=100)
t_data = np.linspace(0, 1, num=100)
test_x, test_t = np.meshgrid(x_data, t_data)
test_domain = np.vstack((np.ravel(test_x), np.ravel(test_t))).T

predicted_solution = model.predict(test_domain)
residual = model.predict(test_domain, operator=pde)
```

4.1.2 Results

The trained PINN accurately approximates the temperature distribution, respecting both initial and boundary conditions, and adheres to the heat equation's physical laws.



4.2 Wave Equation

The wave equation is a fundamental PDE describing the propagation of waves through a medium. It is given by:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \tag{1}$$

where $u(x,t)$ represents the wave displacement at position x and time t, and c is the wave speed.

The domain is $x \in [-1, 1]$ and $t \in [0, 1]$ with the following conditions:

Boundary Conditions:

- Dirichlet BC: $u(-1,t) = u(1,t) = 0$

Initial Conditions:

- IC: $u(x,0) = \sin(\pi x)$
- IC: $u_t(x,0) = 0$

4.2.1 Implementation Steps

1. **Setup and Imports**:

```
import numpy as np
import deepxde as dde

# Define geometry and time domain
geom = dde.geometry.Interval(-1, 1)
timedomain = dde.geometry.TimeDomain(0, 1)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)
```

2. **Define the PDE and Boundary Conditions**:

```
# Define the PDE (wave equation with c = 1)
def pde(x, y):
    dy_tt = dde.grad.hessian(y, x, i=1, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    return dy_tt - dy_xx

# Initial condition u(x, 0) = sin(pi * x)
def func_u0(x):
    return np.sin(np.pi * x[:, 0:1])

# Initial condition u_t(x, 0) = 0
def func_ut0(x):
    return np.zeros_like(x[:, 0:1])

# Boundary condition u(-1, t) = u(1, t) = 0
def boundary_func(x, on_boundary):
    return on_boundary and (np.isclose(x[0], -1) or np.isclose(x[0], 1))
```

```
# Create Dirichlet boundary condition
bc = dde.DirichletBC(geomtime, lambda x: 0, boundary_func)

# Create initial conditions
ic_u = dde.IC(geomtime, func_u0, lambda _, on_initial: on_initial)
ic_ut = dde.OperatorBC(
    geomtime, lambda x, y, _: dde.grad.jacobian(y, x, j=1), lambda _, on_
)
```

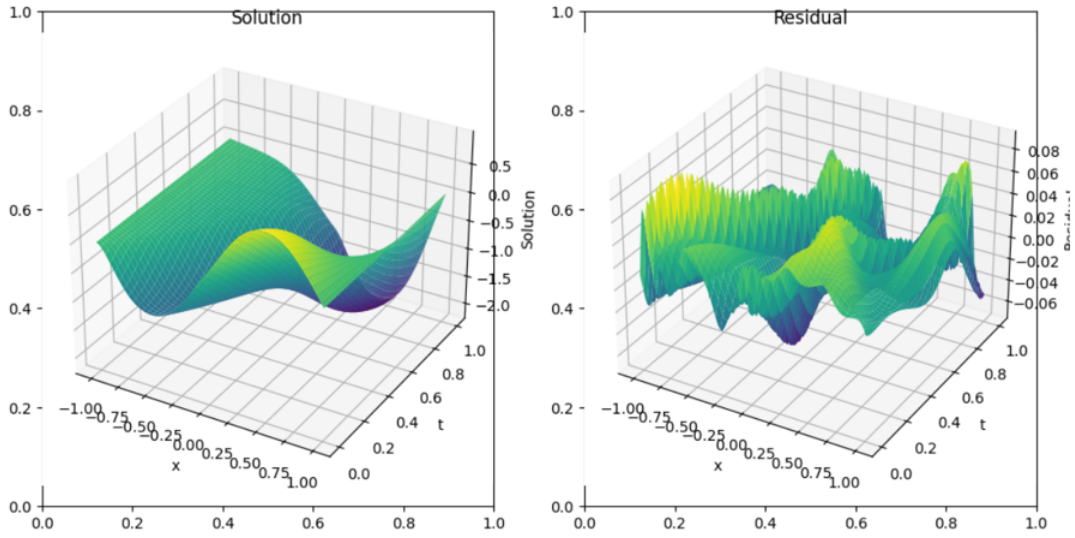
3. **Create the Dataset and Model**:

```
# Define the data
data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc, ic_u, ic_ut],
    num_domain=4000,
    num_boundary=2000,
    num_initial=1000,
    solution=func_u0,  # This is just for test purposes
    num_test=1000,
)

# Define the neural network
layer_size = [2] + [32] * 3 + [1]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

# Create the model
model = dde.Model(data, net)
model.compile("adam", lr=0.01, metrics=["mse"])

# Train the model
losshistory, train_state = model.train(epochs=10000)
```

4. **Predict the Solution and Calculate Residuals**:

```
# Create test data
x_data = np.linspace(-1, 1, num=100)
t_data = np.linspace(0, 1, num=100)
test_x, test_t = np.meshgrid(x_data, t_data)
test_domain = np.vstack((np.ravel(test_x), np.ravel(test_t))).T

# Predict the solution
predicted_solution = model.predict(test_domain)
residual = model.predict(test_domain, operator=pde)
```

4.2.2 Results



5 Discussion

The PINN effectively learned the solution to the heat equation and the wave eqution , satisfying both the initial and boundary conditions. The results demonstrate that PINNs can accurately approximate the temperature distribution over time, guided by the physics encoded in the PDE. By comparing the predicted solution with the analytical or numerical solution (if available), we can validate the performance of the PINN. In practice, the accuracy of PINNs can be influenced by factors such as the network architecture, the number of collocation points, and the training duration.

6 Challenges Associated with PINNs

1. **Training Complexity**: Requires significant computational resources and time for training, especially for complex PDEs and large domains.
2. **Hyperparameter Tuning**: Sensitive to choices of network architecture, learning rate, and activation functions, necessitating careful tuning.
3. **Collocation Point Selection**: The distribution and number of collocation points significantly affect accuracy and convergence.
4. **Convergence Issues**: May struggle with convergence or get trapped in local minima, leading to suboptimal solutions.
5. **Residual Minimization**: Ensuring low residuals across the entire domain can be challenging, especially in regions with complex dynamics.
6. **Generalization**: Generalizing well to unseen scenarios or different boundary conditions can be difficult.
7. **Scalability**: Scaling PINNs to high-dimensional problems or multiple coupled PDEs remains challenging.

7 Conclusion

In this report, we explored the application of Physics-Informed Neural Networks (PINNs) for solving partial differential equations (PDEs), focusing on the heat equation and the wave equation. Traditional neural networks, while effective for various tasks, face limitations in data dependency and physical consistency, making them less suitable for problems governed by physical laws. PINNs address these limitations by integrating physical laws directly into the training process, leading to models that are both data-efficient and physically accurate.

Through our implementations using the DeepXDE library, we demonstrated that PINNs could effectively solve the heat and wave equations, adhering to the initial and boundary conditions and accurately capturing the underlying physics. This approach not only reduces the need for extensive data but also ensures that the solutions are consistent with the known physical laws.

However, PINNs are not without challenges. Training complexity, hyperparameter tuning, collocation point selection, convergence issues, residual minimization, generalization, and scalability remain significant hurdles. Addressing these challenges will be crucial for the broader adoption and application of PINNs in scientific and engineering domains.

Despite these challenges, the potential of PINNs is immense. By leveraging both data and physical principles, they offer a powerful tool for solving complex PDEs in areas such as fluid dynamics, material science, and beyond. As computational resources and algorithms continue to advance, PINNs are poised to play a pivotal role in the future of scientific computing.

In conclusion, PINNs represent a significant advancement in the integration of machine learning and physical sciences. Their ability to produce physically consistent and data-efficient solutions makes them a valuable asset in tackling complex problems where traditional methods fall short. Continued research and development in this field will undoubtedly lead to further innovations and broader applications, enhancing our ability to solve some of the most challenging problems in science and engineering.