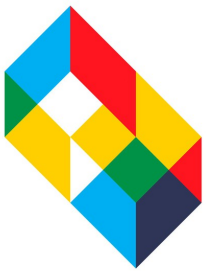
Physics-Informed Neural Networks: A Data-Driven Approach to Physics-Based Modeling

ME-371: Data-Driven Problem Solving in Mechanical Engineering

Dr. Masoud Masoumi

Authors: Gabriel Kret, Adin Sacho-Tanzer
gabriel.kret@cooper.edu
adin.sachotanzer@cooper.edu



THE COOPER UNION

The Cooper Union for The Advancement of Science and Art
New York, NY
Fall 2024

1 Introduction

Physics-Informed Neural Networks (PINNs) are a computational framework that bridges the gap between machine learning and traditional physics-based modeling. PINNs embed physical laws, often expressed as PDEs, directly into the loss functions of neural networks, allowing these networks to solve both forward and inverse problems [10].

Forward problems involve predicting the solution to PDEs given known parameters, while inverse problems aim to infer unknown parameters or equations from observed data [8]. These capabilities make PINNs versatile and applicable to fields ranging from fluid dynamics to quantum mechanics. The traditional methods for solving PDEs, such as finite difference and finite element methods, rely heavily on domain discretization. These methods become computationally expensive for high-dimensional problems or complex geometries. PINNs, by contrast, are mesh-free, relying on neural networks' ability to approximate functions flexibly. Formerly introduced by Raissi et al. [10], PINNs have rapidly gained popularity for their ability to seamlessly integrate data-driven learning and physics-based modeling.

2 Mathematical Foundations of PINNs

Before delving into the workings of PINNs, it is crucial to understand the mathematical underpinnings that form their basis. This section discusses PDEs, boundary and initial conditions, and the role of neural networks as function approximators.

2.1 Partial Differential Equations (PDEs)

At the highest level, PDEs relate a function of several variables to its partial derivatives. They are often used to describe physical phenomena. Common examples include The Heat Equation used for heat conduction and diffusion, The Navier-Stokes Equations used for fluid dynamics, and The Maxwell Equations used for electromagnetism. A general Partial Differential Equation (PDE) can be written as:

$$N[u(x)] = f(x)$$

where:

- N: A mathematical operation involving derivatives (a differential operator).
- $u(x)$: The solution we are trying to find.
- $f(x)$: A given function or value (the "source term").
- x: Represents the variables, such as space and time, that the solution depends on.

Examples of PDEs include:

- **Heat Equation:**
$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T,$$
which models the distribution of temperature u over time t in a material with thermal diffusivity α [3].
- **Wave Equation:**
$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u,$$
which describes wave propagation with speed c.
- **Navier-Stokes Equations:** The Incompressible Navier-Stokes Equations used in the case of the lid driven cavity (A common test case for PINNs) are defined as [1] [7]:
$$\begin{aligned} \nabla \cdot \vec{v} &= 0 \\ \frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\vec{v} &= -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v} \end{aligned}$$
where

- $\vec{v}$ is the velocity vector of the fluid,
- t is time,
- p is the pressure,
- ρ is the fluid density,
- ν is the kinematic viscosity of the fluid,
- ∇ is the del operator (gradient),
- ∇^2 is the Laplacian operator (divergence of the gradient).

2.2 Boundary and Initial Conditions

Boundary and initial conditions are fundamental requirements for solving Partial Differential Equations (PDEs). These conditions provide the necessary constraints to ensure that the problem is well-posed, meaning it has a unique and stable solution. In the context of Physics-Informed Neural Networks (PINNs), these conditions are incorporated into the loss function to guide the network toward physically valid predictions.

2.2.1 Boundary Conditions

Boundary conditions specify the behavior of the solution at the edges of the domain. The most common type is the Dirichlet boundary condition, which directly prescribes the value of the solution along the boundary:

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

where:

- $u(x)$ is the solution to the PDE.
- $g(x)$ is a known function defining the solution's value at the boundary.
- $\partial\Omega$ represents the boundary of the domain Ω.

For example, in a heat transfer problem, Dirichlet boundary conditions might specify a fixed temperature at the surface of a material, such as $u(x) = 100°C$. These conditions are straightforward to apply and are widely used in problems where physical constraints directly dictate the value of the solution at specific locations.

While Dirichlet conditions prescribe the value of the solution, other types of boundary conditions such as Neumann, Robin, and Periodic, can provide alternative constraints. In PINNs, these boundary conditions are incorporated into the loss function as a penalty term, ensuring that the neural network predictions adhere to the prescribed physical constraints.

2.2.2 Initial Conditions

For time-dependent problems, initial conditions specify the state of the system at the start of the simulation (typically $t = 0$). These conditions are essential for propagating the solution forward in time and take the form:

$$u(x, 0) = f(x), \quad x \in \Omega,$$

where $f(x)$ is a known function defining the initial state. For example, in a wave equation describing vibrations on a string, the initial conditions might specify the initial displacement $u(x, 0)$ and initial velocity $\frac{\partial u}{\partial t}(x, 0)$ of the string.

2.2.3 Role in Physics-Informed Neural Networks (PINNs)

Boundary and initial conditions are seamlessly incorporated into PINNs through specific terms in the loss function. Described further in Section 3.3, these terms guide the neural network to learn solutions that satisfy both the governing PDEs and the prescribed conditions, ensuring physically consistent results.

3 How PINNs Work

3.1 The Universal Approximation Theorem in PINNs

A cornerstone of PINNs is the universal approximation theorem, which states that a sufficiently large neural network can approximate any continuous function on a compact domain, given appropriate weights and biases. This property allows neural networks to approximate the solution to a PDE, as well as the associated boundary and initial conditions. [6]. By leveraging this theorem, PINNs use neural networks as flexible function approximators that inherently satisfy the physical constraints embedded in the loss function. Unlike traditional numerical methods, which impose boundary and initial conditions explicitly during the solution process, PINNs incorporate these conditions as part of the optimization problem.

3.2 Neural Networks as Function Approximators

A neural network approximates a function by learning a mapping between inputs and outputs. In the context of PINNs:

- The input layer accepts spatial coordinates (x, y, z) and time t.
- Hidden layers perform nonlinear transformations using activation functions like tanh, Sigmoid, or ReLU.
- The output layer predicts the solution $u(x, t)$ to the PDE.

3.3 Loss Functions

Physics-Informed Neural Networks (PINNs) solve Partial Differential Equations (PDEs) by minimizing a loss function that combines data-driven and physics-based terms:

$$\mathcal{L} = \mathcal{L}_{data} + \mathcal{L}_{PDE} + \mathcal{L}_{BC}.$$

3.3.1 Data Loss ($\mathcal{L}_{data}$)

The data loss term measures the difference between the predicted solution and observed data. While there are many options for loss functions, a common choice for this is the Mean Squared Error (MSE):

$$\mathcal{L}_{data} = \frac{1}{N_d} \sum_{i=1}^{N_d} \left(u(x_i) - u_{true}(x_i) \right)^2,$$

where $u_{true}(x_i)$ is the observed value at location x_i, and N_d is the number of data points. This term ensures that the network predictions align with ground truth data where available.

3.3.2 PDE Loss ($\mathcal{L}_{PDE}$)

The PDE loss ensures that the network's output satisfies the governing equations:

$$\mathcal{L}_{PDE} = \frac{1}{N_c} \sum_{i=1}^{N_c} \left(\mathcal{N}[u(x_i)] - f(x_i) \right)^2,$$

where $\mathcal{N}[u(x_i)]$ represents the application of the PDE operator to the predicted solution at collocation points x_i, and $f(x_i)$ is the known source term at these points. N_c is the total number of collocation points. These points are selected within the spatial and temporal domain of the PDE and serve as locations where the network enforces the governing equations.

3.3.3 Boundary Condition Loss ($\mathcal{L}_{BC}$)

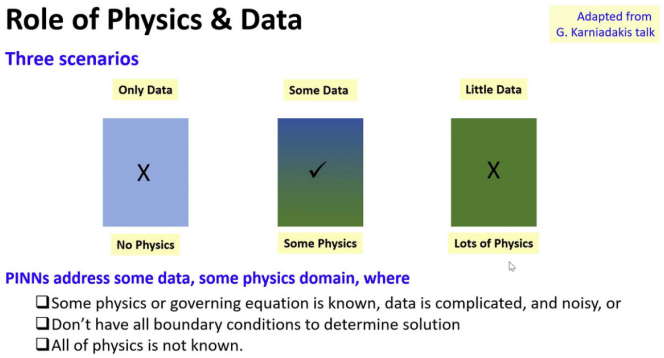
The boundary condition loss enforces compliance with prescribed boundary values:

$$\mathcal{L}_{BC} = \frac{1}{N_b} \sum_{i=1}^{N_b} \left(u(x_i) - g(x_i) \right)^2,$$

where $g(x_i)$ represents the boundary condition value at x_i, and N_b is the number of boundary points. This concept also extends to initial conditions in time-dependent problems, ensuring the solution respects the required starting conditions.

3.4 Training Without Ground Truth Data

In scenarios where ground truth data ($\mathcal{L}_{data}$) is unavailable, the PINN relies entirely on the physics-based loss terms ($\mathcal{L}_{PDE}$ and $\mathcal{L}_{BC}$). This makes PINNs particularly powerful for theoretical and simulation-based studies, where direct experimental data might be scarce or infeasible to obtain [8].



3.5 Automatic vs Numerical Differentiation

PINNs often use Automatic Differentiation to compute derivatives required for the PDE loss ($\mathcal{L}_{PDE}$), offering several advantages over traditional numerical differentiation methods.

3.5.1 Automatic Differentiation (A-PINNs)

Automatic differentiation leverages the chain rule of calculus to compute derivatives exactly, up to machine precision, by decomposing functions into elementary operations. AD is integrated into modern deep learning frameworks like TensorFlow and PyTorch, enabling efficient computation of high-order derivatives with minimal overhead. For example, given $u(x) = \sin(x^2)$, AD computes the derivative $\frac{du}{dx}$ as:

$$\frac{du}{dx} = \cos(x^2) \cdot 2x,$$

without relying on finite differences.

3.5.2 Numerical Differentiation (N-PINNs)

Numerical differentiation approximates derivatives using finite differences (can us forward, backwards, or central difference scheme), e.g.:

$$\frac{du}{dx} \approx \frac{u(x+h) - u(x)}{h},$$

where h is a small step size. While simple to implement, this method introduces truncation and round-off errors, which can accumulate significantly in higher-order derivatives or complex PDEs. By using AD, PINNs can avoid approximation errors and achieve precise enforcement of PDEs, even in high-dimensional or complex domains. [10] However, in complex cases such as the Navier-Stokes equations, there may be some merit to using numerical differentiation. This is an area that physicists and computer scientists are currently researching. What is better for solving complex flow scenarios? A-PINNs, N-PINNs, CAN(Combined Automatic-Numerical)-PINNs, or something completely different?

4 Applications of PINNs

PINNs have been applied to various fields:

- **Computational Fluid Dynamics:** Solving Navier-Stokes equations for laminar and turbulent flows. This is likely the most applicable and furthest developed use for PINNs since the Navier-Stokes equations do not have a known analytical solution [7].
- **Quantum Mechanics:** Simulating wavefunctions using Schrödinger equations [5].
- **Structural Engineering:** Modeling stress and deformation in materials [3].
- **Biomedicine:** Simulating blood flow dynamics and other physiological systems [5].

5 Challenges and Future Directions

Physics-Informed Neural Networks (PINNs) have demonstrated significant promise, but several challenges limit their broader application:

5.1 Challenges

- **Training Stability:** Balancing loss terms (e.g., data and physics) can cause optimization issues. Techniques such as adaptive weighting and domain decomposition are under active investigation.
- **Computational Cost:** Solving high-dimensional problems is resource-intensive, requiring improved training algorithms to reduce runtime without sacrificing accuracy.
- **Sparse or Noisy Data:** PINNs struggle with limited datasets since gathering data for complex physics phenomena, such as turbulent fluid flow, is difficult and expensive. Advances in robust training methods and data augmentation aim to mitigate this [8].

5.2 Future Directions

- **Hybrid Modeling Approaches:** Combining PINNs with traditional numerical methods, such as finite element or finite difference methods, can leverage the strengths of both approaches, enhancing accuracy and efficiency [9].
- **Types of Neural Networks:** Developing specialized neural network architectures, including graph neural networks and attention mechanisms, can improve the capability of PINNs to model more complex phenomena and capture long-range dependencies [2].
- **Uncertainty Quantification:** Integrating uncertainty quantification into PINNs is crucial for assessing the reliability of predictions, especially in safety-critical applications. This involves developing methods to quantify and propagate uncertainties through the network [4].
- **Computational Efficiency and Hardware Utilization:** Training PINNs can be time-consuming and computationally intensive, often necessitating the use of advanced hardware accelerators like GPUs. Working to optimize utilization of such hardware is crucial to reduce training times and energy consumption.
- **Theoretical Foundations:** Strengthening the theoretical understanding of PINNs, including convergence analysis and error bounds, is essential for establishing their reliability and guiding the development of more robust algorithms [11].

By addressing these challenges, PINNs can become a transformative tool for solving PDEs in science and engineering. Future research can focus on hybrid methods that combine PINNs with traditional numerical solvers and developing more efficient network architectures that have more wider and generic use cases [3].

6 Conclusion

Physics-Informed Neural Networks provide a revolutionary approach to solving PDEs by embedding physical laws directly into neural network training. Their flexibility and ability to work without data make them a transformative tool for computational science. With ongoing advancements, PINNs are poised to address increasingly complex problems across scientific disciplines.

Appendix A

The report has exceeded the 5 page limit imposed in the assignment. Therefore, examples have been omitted from the body of the report. Please find a simple sample PINNs implementation here:

References

[1] Lorena Barba and Gilbert Forsyth. Cfd python: the 12 steps to navier-stokes equations. *Journal of Open Source Education*, 2(16):21, 2019. `https://doi.org/10.21105/jose.00021`.

[2] Marien Chenaud, José Alves, and Frédéric Magoulès. Physics-informed graph convolutional networks: Towards a generalized framework for complex geometries. *arXiv preprint arXiv:2310.14948*, 2023.

[3] Salvatore Cuomo et al. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92:88, 2022.

[4] Salvatore Cuomo, Vincenzo Schiano di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics-informed neural networks: Where we are and what's next. *arXiv preprint arXiv:2201.05624*, 2022.

[5] Amer Farea et al. Understanding physics-informed neural networks: Techniques, applications, trends, and challenges. *AI*, 5:1534–1557, 2024.

[6] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, jan 1989.

[7] Qinghua Jiang et al. Applications of finite difference-based physics-informed neural networks to steady incompressible isothermal and thermal flows. *International Journal for Numerical Methods in Fluids*, 2023.

[8] Zaharaddeen Karami Lawal et al. Physics-informed neural network (pinn) evolution and beyond: A systematic literature review and bibliometric analysis. *Big Data and Cognitive Computing*, 6:140, 2022.

[9] Numeryst. Physics-informed neural networks and classical numerical methods. *Numeryst*, 2023.

[10] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear pdes. *arXiv preprint arXiv:1711.10561*, 2017.

[11] Edward Small. An analysis of physics-informed neural networks. *arXiv preprint arXiv:2303.02890*, 2023.