

# Physics-Informed Neural Networks (PINNs)

Krishna Kumar

University of Texas at Austin

*krishnak@utexas.edu*

- 1 The PINN Concept: Beyond Data-Only
- 2 Enforcing Boundary Conditions
- 3 Inverse Problems: Discovering Physics

# Outline

- 1 The PINN Concept: Beyond Data-Only
- 2 Enforcing Boundary Conditions
- 3 Inverse Problems: Discovering Physics

# Learning Objectives

- Understand why standard neural networks fail for physics problems
- Learn how to incorporate physics into neural network training
- Master automatic differentiation for computing derivatives
- Compare data-driven vs physics-informed approaches

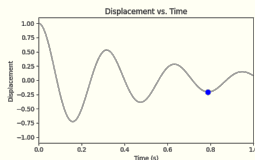
► Open Notebook: PINN

# The Problem: A Damped Harmonic Oscillator

A mass  $m$  on a spring (constant  $k$ ) with damping (coefficient  $c$ ). The displacement  $u(t)$  satisfies:

$$m \frac{d^2 u}{dt^2} + c \frac{du}{dt} + ku = 0$$

with initial conditions:  $u(0) = 1$ ,  $\frac{du}{dt}(0) = 0$ .



A classic physics problem to illustrate PINNs.

**The Challenge:** Reconstruct the full solution  $u(t)$  from a few sparse, noisy data points.

# Stage 1: The Data-Only Approach

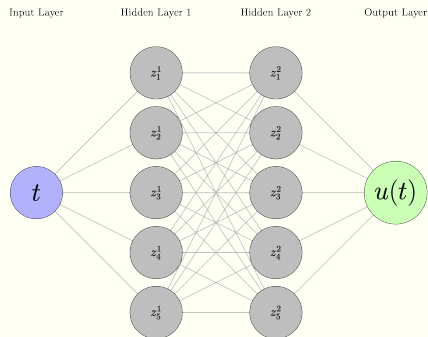
**Idea:** Train a standard neural network to fit the sparse data.

**Loss Function:** Mean Squared Error

$$\mathcal{L}_{\text{data}}(\theta) = \frac{1}{N} \sum_{i=1}^N |\hat{u}_{\theta}(t_i) - u_i|^2$$

**Architecture:**

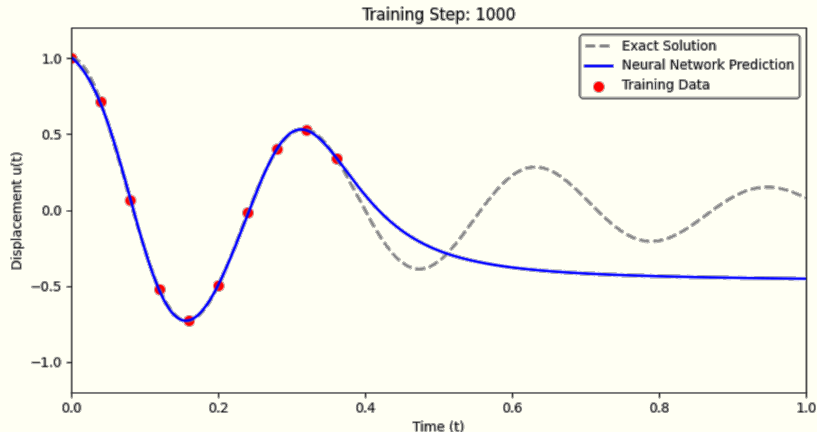
- Input: Time  $t$
- Hidden Layers: Tanh activations
- Output: Displacement  $\hat{u}_{\theta}(t)$



Standard NN for function fitting.

# The Failure of the Data-Only Approach

**Result:** The network fits the training points but fails catastrophically between them.



The network overfits to the sparse data and does not respect the underlying physics.

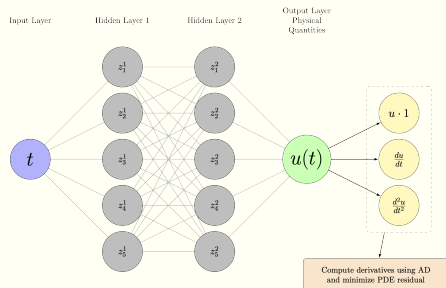
# Stage 2: Enter Physics-Informed Neural Networks

**The Key Insight:** Don't just fit data. Enforce the differential equation itself!

**Physics Residual:** We define a residual based on the ODE:

$$\mathcal{R}_\theta(t) = m \frac{d^2 \hat{u}_\theta}{dt^2} + c \frac{d \hat{u}_\theta}{dt} + k \hat{u}_\theta$$

If the solution is correct,  $\mathcal{R}_\theta(t)$  should be zero.



PINN architecture with physics loss.



# The Complete PINN Loss Function

The total loss is a combination of data fit and physics enforcement.

**Total Loss:**

$$\mathcal{L}_{\text{total}}(\theta) = \mathcal{L}_{\text{data}}(\theta) + \lambda \mathcal{L}_{\text{physics}}(\theta)$$

## Data Loss

Ensures the solution passes through the measurements.

$$\mathcal{L}_{\text{data}}(\theta) = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} |\hat{u}_{\theta}(t_i) - u_i|^2$$

## Physics Loss

Ensures the solution obeys the ODE at random "collocation" points.

$$\mathcal{L}_{\text{physics}}(\theta) = \frac{1}{N_{\text{colloc}}} \sum_{j=1}^{N_{\text{colloc}}} |\mathcal{R}_{\theta}(t_j)|^2$$

# The Secret Weapon: Automatic Differentiation (AD)

**Critical Question:** How do we compute  $\frac{d\hat{u}_\theta}{dt}$  and  $\frac{d^2\hat{u}_\theta}{dt^2}$ ?

**Answer: Automatic Differentiation**

AD provides **exact** derivatives of the neural network output with respect to its input, by applying the chain rule through the computational graph.

- No finite difference errors.
- Computationally efficient (especially reverse-mode AD).
- Built into modern frameworks (PyTorch, TensorFlow, JAX).

**Example:** For  $u(t) = \sin(t)$ , AD can compute  $u'(t) = \cos(t)$  and  $u''(t) = -\sin(t)$  to machine precision.

# Theoretical Foundation: UAT for Sobolev Spaces

**Classical UAT:** NNs can approximate any *continuous function*.

**Problem:** For PDEs, we need to approximate functions **and their derivatives**.

## Extended Universal Approximation Theorem

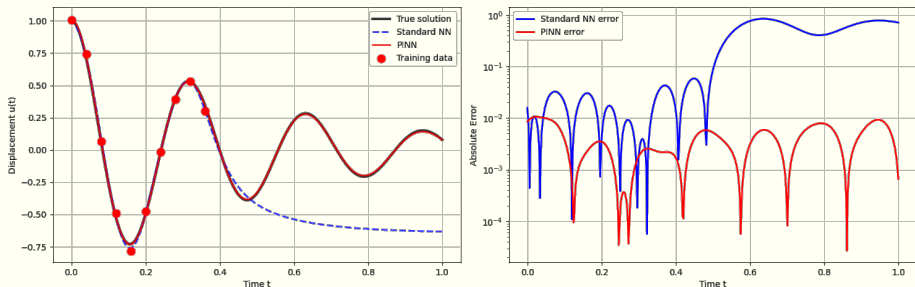
Neural networks with sufficiently smooth activation functions (e.g., tanh, not ReLU) can approximate functions in **Sobolev spaces**  $H^k(\Omega)$ .

$$\|u - \hat{u}_\theta\|_{H^k} < \epsilon$$

The Sobolev norm  $\|u\|_{H^k}^2 = \sum_{|\alpha| \leq k} \|D^\alpha u\|_{L^2}^2$  measures the error in the function and all its derivatives up to order  $k$ .

**Why this matters:** For a  $k^{\text{th}}$ -order ODE/PDE, we need an activation function that is at least  $k$  times differentiable ( $C^k$ ). For our 2nd-order oscillator, we need a  $C^2$  activation like tanh.

# The Moment of Truth: Standard NN vs. PINN



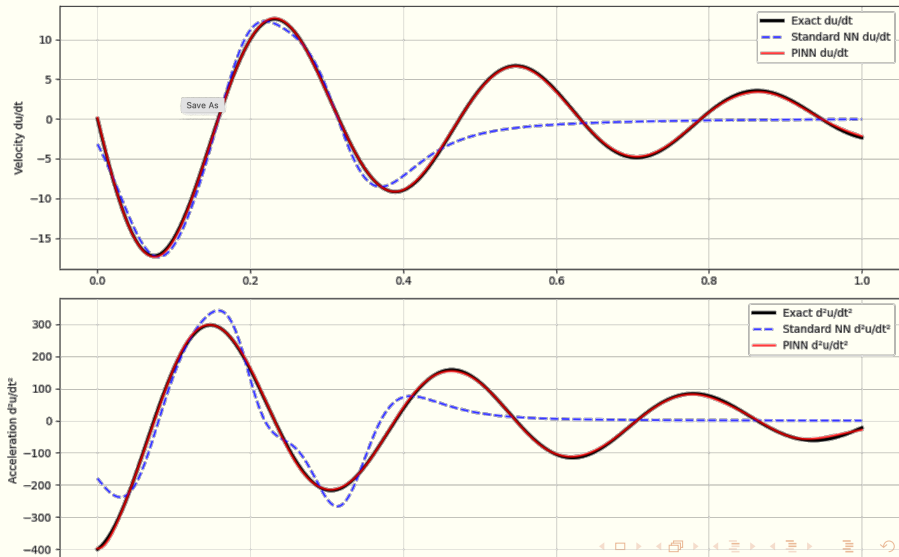
Placeholder for the comparison plot from the notebook.

## Observation:

- **Standard NN:** Fits data points, but fails to generalize.
- **PINN:** Fits data points AND follows the physics, resulting in a globally accurate solution.

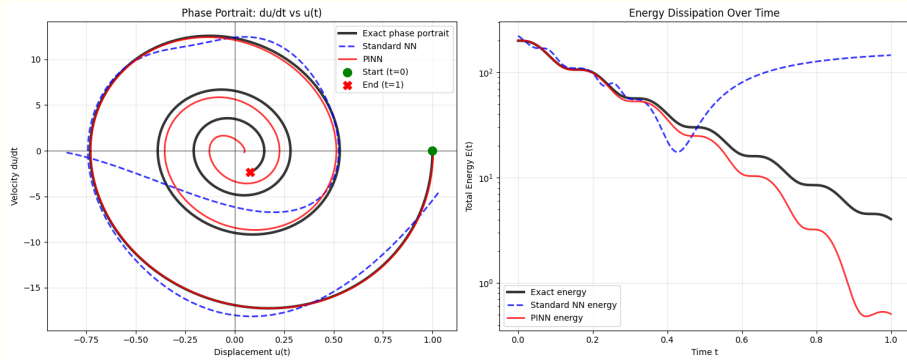
# Deep Dive: Derivative and Phase Portrait Analysis

The ultimate test: Does the PINN learn physically consistent derivatives?



# Deep Dive: Derivative and Phase Portrait Analysis

The ultimate test: Does the PINN learn physically consistent derivatives?



Phase portrait plots.

**Result:** The phase portrait (velocity vs. displacement) traces the correct physical trajectory (a spiral for a damped oscillator).

# Outline

- 1 The PINN Concept: Beyond Data-Only
- 2 Enforcing Boundary Conditions
- 3 Inverse Problems: Discovering Physics

# The 1D Poisson Problem

We now tackle a boundary value problem, the 1D Poisson equation:

$$\frac{d^2 u}{dx^2} + \pi \sin(\pi x) = 0, \quad \text{for } x \in [0, 1]$$

with Dirichlet boundary conditions (BCs):

$$u(0) = 0 \quad \text{and} \quad u(1) = 0$$

**Goal:** Train a PINN to find the solution using only the governing equation and its BCs.

► [Open Notebook: 1D Poisson](#)



# Method 1: Soft Constraints

Treat the boundary conditions as another component of the loss function.

**Total Loss:**

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

## Boundary Loss

A mean squared error term that penalizes violations of the BCs.

$$\mathcal{L}_{BC} = \frac{1}{N_{BC}} \sum_{i=1}^{N_{BC}} |\hat{u}_{\theta}(x_i) - u_{BC}|^2$$

## Pros & Cons

- + **Flexible:** Easy to implement for any type of BC (Dirichlet, Neumann, etc.).
- **Approximate:** Satisfaction is not guaranteed, only encouraged.
- **Tuning:** Requires careful tuning of the weight  $\lambda_{BC}$ .

## Method 2: Hard Constraints

Modify the network architecture to satisfy the BCs *by construction*.

For our problem with  $u(0) = 0$  and  $u(1) = 0$ , we can define a trial solution  $\tilde{u}(x)$ :

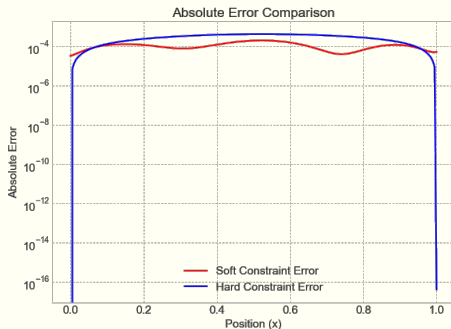
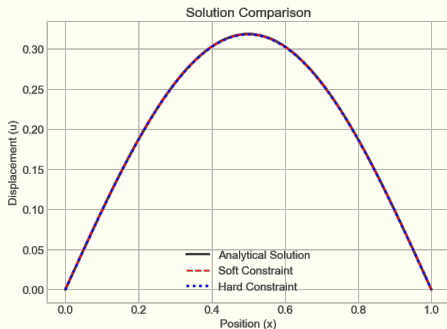
$$\tilde{u}(x) = \underbrace{x(1-x)}_{D(x)} \cdot \underbrace{\text{NN}(x; \theta)}_{\text{Network Output}}$$

The distance function  $D(x)$  is zero at the boundaries, forcing  $\tilde{u}(x)$  to be zero there, regardless of the network's output.

### Pros & Cons

- + **Exact:** BCs are satisfied perfectly.
- + **Simpler Loss:** No need for  $\mathcal{L}_{BC}$  or  $\lambda_{BC}$ , leading to more stable training.
- **Inflexible:** Requires designing a specific trial function for the problem's geometry and BCs, which can be difficult for complex cases.

# Comparison: Soft vs. Hard Constraints



Comparison plot of soft vs. hard constraint solutions and errors.

## Conclusion:

- Both methods achieve high accuracy.
- The hard constraint method shows slightly lower error and, by design, has zero error at the boundaries.
- For simple geometries and Dirichlet BCs, **hard constraints are often**

# Outline

- 1 The PINN Concept: Beyond Data-Only
- 2 Enforcing Boundary Conditions
- 3 Inverse Problems: Discovering Physics

# Forward vs. Inverse Problems

## Forward Problem

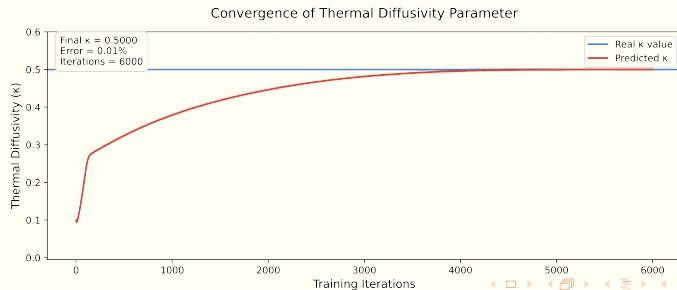
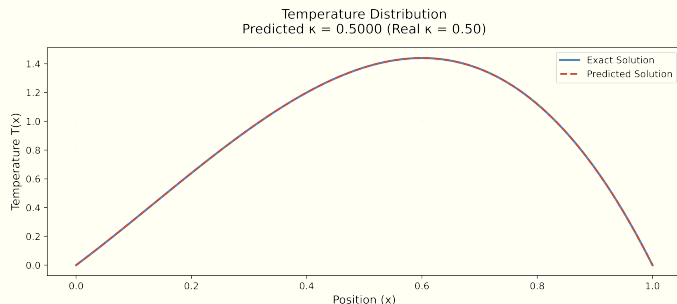
- **Given:** Full physical model (equations + parameters).
- **Find:** The solution  $u(x, t)$ .
- *Example: Simulate temperature given thermal conductivity.*

## Inverse Problem

- **Given:** Sparse measurements of the solution  $u(x, t)$ .
- **Find:** Unknown physical parameters in the model.
- *Example: Infer thermal conductivity from temperature measurements.*

► Open Notebook: Inverse Heat

# Forward vs. Inverse Problems



# The PINN Approach to Inverse Problems

**The Key Insight:** Treat the unknown physical parameters as additional trainable variables in the network.

**Problem:** 1D steady-state heat conduction.

$$-k \frac{d^2 T}{dx^2} = f(x)$$

Here, the thermal diffusivity  $k$  is **unknown**.

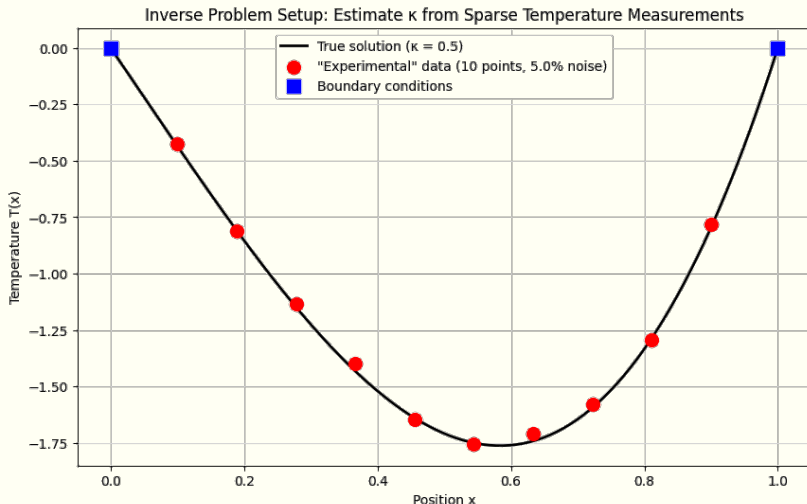
**PINN Framework:**

- The neural network learns the temperature field:  $\hat{T}_\theta(x)$ .
- A new trainable parameter is introduced:  $\hat{k}$ .
- The optimizer updates both the network weights  $\theta$  and the parameter  $\hat{k}$  simultaneously.

**Loss Function:**  $\mathcal{L}(\theta, \hat{k}) = \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{BC}}$  where the physics loss now includes the trainable parameter  $\hat{k}$ :

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_f} \sum \left| -\hat{k} \frac{d^2 \hat{T}_\theta}{dx^2}(x_j) - f(x_j) \right|^2$$

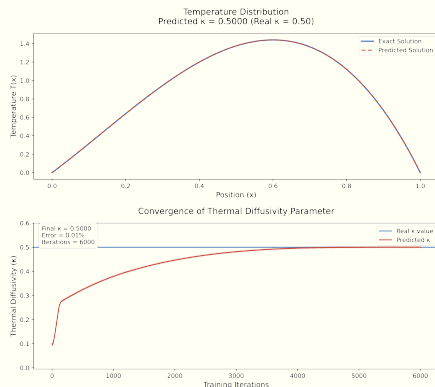
# Implementation: Parameter Estimation



"Experimental Setup" plot, showing the true solution and sparse, noisy data points.



# Results: Parameter Recovery



Parameter convergence and temperature field reconstruction.

- The estimated parameter  $\hat{k}$  converges to the true value.
- The PINN simultaneously reconstructs the full, continuous temperature field accurately.
- This is achieved from very sparse and noisy data, showcasing the regularizing effect of the physics loss.

# Summary: Why PINNs are Powerful

## 1. Regularization Effect:

- Physics constraints prevent overfitting and guide the solution in data-sparse regions.

## 2. Data Efficiency:

- Physics provides a strong inductive bias, allowing PINNs to learn from very few measurements.

## 3. Accurate Derivatives:

- Automatic differentiation provides exact derivatives, which are learned correctly as a consequence of enforcing the physics.

## 4. Versatility:

- The same framework can solve forward problems, inverse problems, and handle various boundary conditions.

**PINNs = Universal Function Approx. + Physics Constraints + Auto. Diff.**

Thank you!

**Contact:**

Krishna Kumar

*krishnak@utexas.edu*

University of Texas at Austin