Physics-Informed Neural Networks (PINNs)

For Solving Partial Differential Equations

Vincent Scholz & Dr. Yaohua Zang

Professorship for Data-driven Materials Modeling

May 28, 2025

Overview

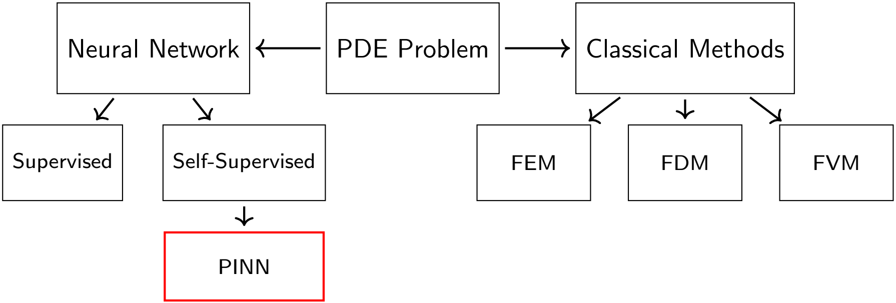

Neural Network
PDE Problem
Classical Methods
Supervised
Self-Supervised
PINN
FEM
FDM
FVM

Table of Contents

Table of Contents

Partial Differential Equations (PDEs)

A PDE in a 2D domain is typically written in residual form:

$$F(x, y, u, u_x, u_y, u_{xx}, u_{yy}, \cdots, \mu) = 0, \quad (x, y) \in \Omega \subset \mathbb{R}^d \tag{1}$$

where:

- $u(x, y)$ denotes the desired solution
- μ represents parameters in the PDE
- Boundary conditions must be imposed

Classifications of PDE Problems

Forward (Direct) Problem

Given μ and boundary conditions, solve for $u(x, y)$

$$u_{NN} : (x, y) \to u(x, y)$$

Inverse Problem

Given boundary conditions and partial observations of u, infer μ

$$\mu_{NN} : (x, y) \to \mu(x, y)$$

Parametric PDE Problem

Solve PDE for a range of μ values as additional input parameters

$$\mathcal{G}_{NN} : \mu(x, y) \to u(x, y)$$

Example: 1D Heat Equation

The 1D heat equation with Dirichlet boundary and initial conditions:

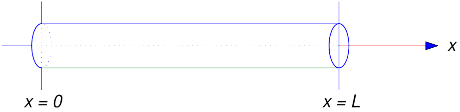
$$u_t - \partial_x(\mu u_x) = 0, \quad x \in \Omega_T = [-1, 1] \times [0, 1] \tag{2}$$

$$u(x \pm 1, t) = g_R(x, t), \quad x \in \partial\Omega_T = \{\pm 1\} \times [0, 1] \tag{3}$$

$$u(x, t = 0) = g_D(x), \quad x \in \Omega, t = 0 \tag{4}$$

where:

- $u(x, t)$ indicates temperature
- μ (m^2/s) represents thermal diffusivity



Heat Equation: Problem Types

- **Forward PDE Problem**:
  - Given $\mu(x)$ (e.g., $\mu(x) = x^2 + 2x + 4$)
  - Solve for $u(x, t)$
- **Inverse Problem**:
  - Recover $\mu(x)$ from sparse, noisy observations of $u(x, t)$
- **Parametric PDE Problem**:
  - Compute $u(x, t; a, b)$ for different $\mu(x)$
  - Parameterized as $\mu(x) = 1 + ax + bx^2$ where $a, b > 0$
  - Treating (a, b) as input variables

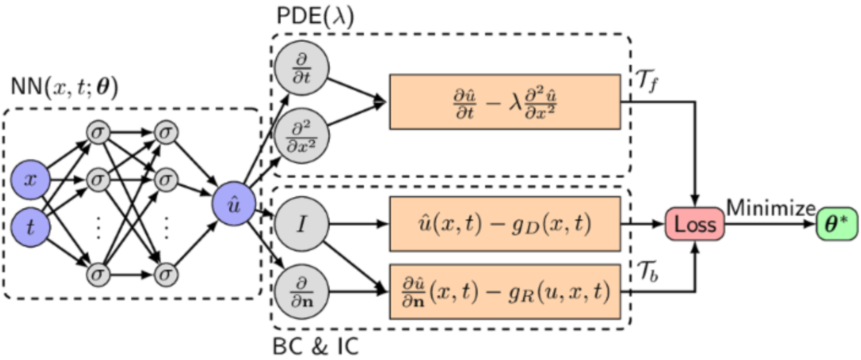
Table of Contents

Concept of PINNs

PINNs solve PDEs by approximating the solution using a neural network, enforcing both:

- **Physical laws**
- **Boundary conditions**

as loss terms during training.



Step 1: Neural Network Approximation

A multi-layer perceptron approximates $u(x,t)$ as $u_\theta(x,t)$:

- **Input layer**: receives spatial-temporal coordinates (x,t)
- **Hidden layers**: apply transformations using activation functions
- **Output layer**: provides estimated solution $u_\theta(x,t)$

where θ represents the network parameters.

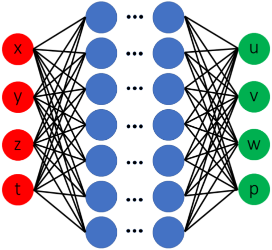


Figure: Neural Network to estimate solution.

Step 2: Classical vs. Self-Supervised Learning

Classical Supervised Learning

Dataset of N_{data} known input-output pairs: $\mathcal{D} = \{(x_i, t_i), u_i\}_{i=1}^{N_{data}}$

$$\theta^* = \arg\min_{\theta} L_{data}(\theta) = \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} |u_\theta(x_i, t_i) - u_i|^2$$

Self-Supervised PINN Method

Solves PDEs **without requiring input-output pairs** inside domain $\Omega/\partial\Omega$
Relies on:

- Boundary/initial conditions
- Physical laws

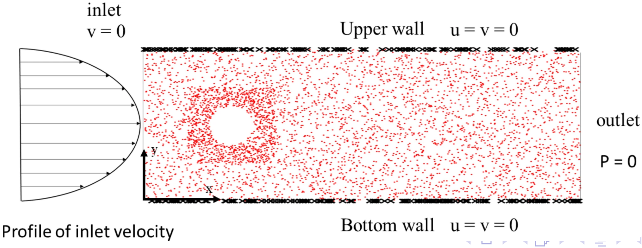
Sample points from boundary and initial conditions:

$$\mathcal{D}_{bd} = \{(x_i, t_i), g_R(x_i, t_i)\}_{i=1}^{N_{bd}} \tag{5}$$

$$\mathcal{D}_{ic} = \{(x_i, t_i = 0), g_D(x_i)\}_{i=1}^{N_{ic}} \tag{6}$$

Boundary/Initial condition loss:

$$L_{bd/ic}(\theta) = \frac{1}{N_{bd}} \sum_{i=1}^{N_{bd}} |u_\theta(x_i, t_i) - g_R(x_i, t_i)|^2 + \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} |u_\theta(x_i, 0) - g_D(x_i)|^2 \tag{7}$$



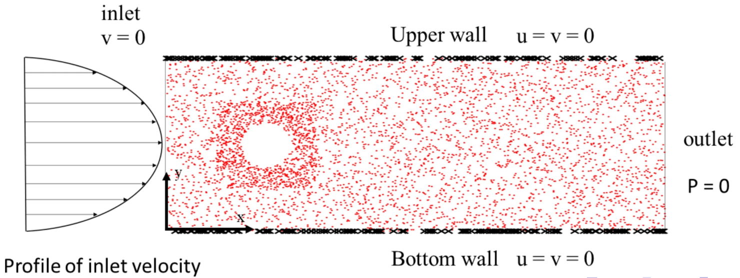
Residual (PDE) Loss

PDE loss defined as strong-form residuals:

$$L_{PDE}(\theta) = \frac{1}{N_c} \sum_{i=1}^{N_c} |F(x_i, t_i, u_\theta, \partial_x u_\theta, \partial_t u_\theta, \partial^2_{xx} u_\theta, \partial^2_{tt} u_\theta, \cdots, \mu)|^2 \qquad (8)$$

where:

- Evaluation performed on N_c **collocation points** (x_i, t_i)
- Residuals measure how well the neural network satisfies the PDE

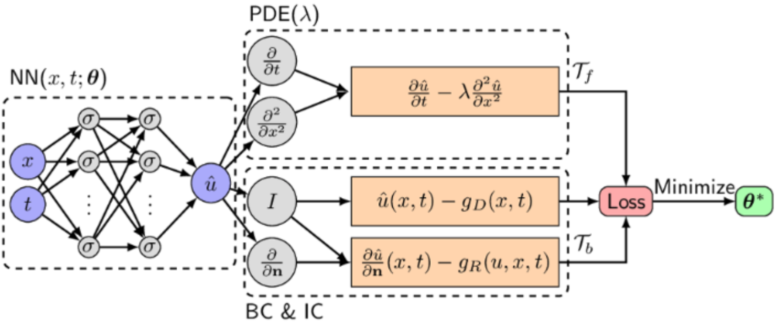


Total Loss Function

Composite total loss function:

$$L(\theta) = \omega_{PDE} L_{PDE}(\theta) + \omega_{bd/ic} L_{bd/ic}(\theta) \tag{9}$$

where ω_{PDE} and $\omega_{bd/ic}$ are weights to balance the loss terms.



Step 3: Training with Gradient Descent

Key Insight

The PINN method transfers solving the PDE problem to the minimization problem:

$$\min_{\theta} L(\theta)$$

Minimization accomplished through (stochastic) gradient descent:

$$\theta_{t+1} = \theta_t - l_r \nabla_\theta L(\theta_t) \tag{10}$$

where:

- t denotes the iteration (epoch)
- l_r is the learning rate parameter
- Algorithm continues until convergence or maximum iterations

Table of Contents

Inverse Problems: Problem Setup

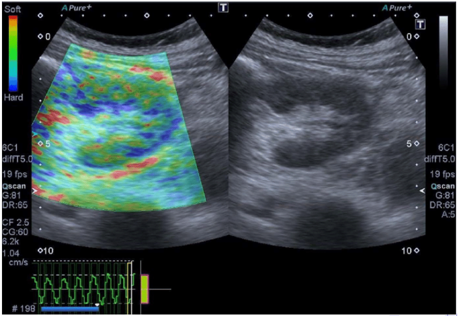
Consider a PDE with unknown parameter $\mu(x,t)$:

$$F(x, t, u, u_t, u_x, u_{xx}, \ldots, \mu) = 0, \quad (x,t) \in \Omega_T$$

Given:

- Boundary and initial conditions
- **Sparse and possibly noisy observations** $u(x_i, t_i)$ at certain locations

Goal: Approximate both $u(x,t)$ and $\mu(x,t)$ using neural networks



Neural Network Architecture for Inverse Problems

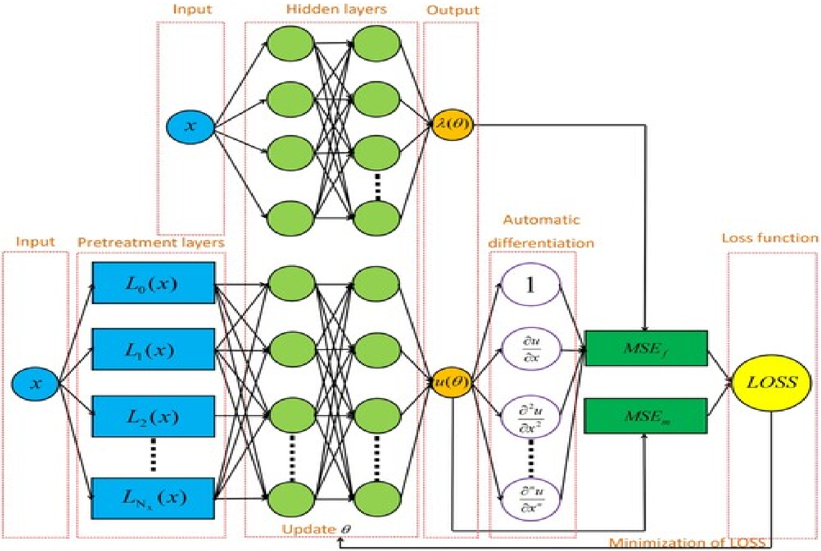
Define two neural networks:

- **Solution network**: $u_\theta(x, t)$ parameterized by θ
- **Parameter network**: $\mu_\phi(x, t)$ parameterized by ϕ

Both networks are trained **simultaneously** to satisfy:

- Given observed data (noisy & sparse)
- Governing PDE

Neural Network Architecture for Inverse Problems



Loss Function for Inverse Problems

Three loss terms:

Data Loss

$$L_{\text{data}}(\theta) = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} |u_\theta(x_i, t_i) - u_i|^2$$

PDE Residual Loss

$$L_{\text{PDE}}(\theta, \phi) = \frac{1}{N_c} \sum_{i=1}^{N_c} |F(x_i, t_i, u_\theta, \partial_x u_\theta, \partial_t u_\theta, \mu_\phi)|^2$$

Boundary/Initial Condition Loss

$$L_{\text{BC/IC}}(\theta) = \frac{1}{N_{bd}} \sum_{i=1}^{N_{bd}} |u_\theta(x_i, t_i) - g(x_i, t_i)|^2$$

Total Loss and Training

Total loss function (weighted sum):

$$L(\theta, \phi) = \omega_{\text{data}} L_{\text{data}} + \omega_{\text{PDE}} L_{\text{PDE}} + \omega_{\text{BC/IC}} L_{\text{BC/IC}} \tag{11}$$

Minimization problem:

$$\min_{\theta,\phi} L(\theta, \phi)$$

Gradient descent updates:

$$\theta_{t+1} = \theta_t - l_r \nabla_\theta L(\theta_t, \phi_t) \tag{12}$$

$$\phi_{t+1} = \phi_t - l_r \nabla_\phi L(\theta_t, \phi_t) \tag{13}$$

Table of Contents

Advantages of PINNs

- **Mesh-free**: No need for grid generation
- **Handles high-dimensional problems**: Useful where classical methods suffer from curse of dimensionality
- **Solves inverse problems effectively**: Learns unknown PDE parameters from data
- **Easy to implement**: no need to consider assembling complex system matrices and handling various boundary conditions in detail

Challenges of PINNs

- **Training instability**: Requires careful tuning of hyperparameters
- **Limited generalization**: May struggle with sharp gradients or discontinuities
- **Computational cost**: Training deep networks can be expensive
- **Lack of theoretical analyses:** Currently no rigorous theoretical analysis to guarantee its convergence and error control

Table of Contents

Summary

Physics-Informed Neural Networks (PINNs) provide a powerful framework for:

- Solving forward PDE problems without traditional mesh generation
- Tackling inverse problems to identify unknown parameters
- Handling high-dimensional problems effectively
- Incorporating physical laws directly into the learning process

Key Innovation: Self-supervised learning approach that leverages:

- PDE residuals as physics constraints
- Boundary/initial conditions
- Sparse observational data (for inverse problems)