Introduction to Physics-informed Neural Network

Yaohua Zang

June 23, 2025

0.1 Introduction to Related PDE Problems

A partial differential equation (PDE) in a 2D domain is typically written in the 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 and μ represents parameters in the PDE, which can be a scalar or a function $\mu(x, y)$. Boundary conditions must also be imposed depending on the problem.

0.1.1 Classifications of Related PDE Problems

The Forward (Direct) Problem: Given μ and boundary conditions, solve for $u(x, y)$. Therefore, the goal is to find an approximation of $u(x, y)$, i.e.:

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

The Inverse Problem: Given boundary conditions and partial observations of u, infer the parameter μ. Therefore, the goal is to find the approximation of the coefficient $\mu(x, y)$, i.e.:

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

The Parametric PDE Problem: Solve the PDE for a range of μ values, treating them as additional input parameters. In this case, the goal is to seek an approximation for the operator $\mathcal{G} : \mu \to u$, which maps the coefficient μ to the PDE solution u. Therefore, the goal of the parametric PDE problem can be summarized as learning an approximation of the map $\mathcal{G}$, i.e.:

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

0.1.2 Example: 1D Heat Equation

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

$$\begin{aligned} u_t - \partial_x(\mu u_x) = 0, &\quad x \in \Omega_T = [-1, 1] \times [0, 1] \\ u(x \pm 1, t) = g_R(x, t), &\quad x \in \partial\Omega_T = \{\pm 1\} \times [0, 1] \\ u(x, t = 0) = g_D(x), &\quad x \in \Omega,\ t = 0 \end{aligned} \tag{2}$$

where $u(x, t)$ indicates temperature and μ (m^2/s) represents thermal diffusivity. The associated problems are:

- **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.

0.2 The Physics-Informed Neural Networks (PINNs)

0.2.1 Concept of PINNs

PINNs solve PDEs by approximating the solution using a neural network, enforcing both **physical laws** and **boundary conditions** as loss terms during training.

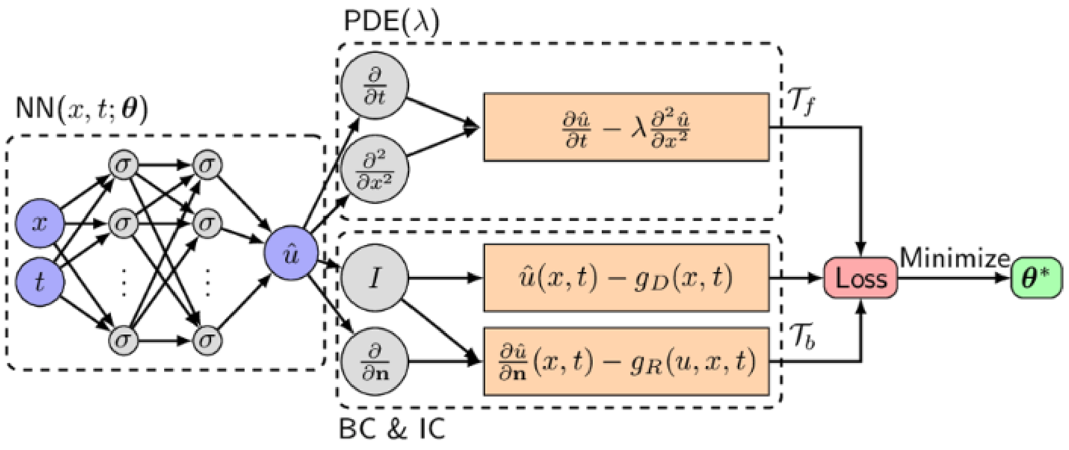
PDE(λ)
NN($x, t; \boldsymbol{\theta}$)
x
t
σ
$\hat{u}$
$\frac{\partial}{\partial t}$
$\frac{\partial^2}{\partial x^2}$
$\frac{\partial \hat{u}}{\partial t} - \lambda \frac{\partial^2 \hat{u}}{\partial x^2}$
$\mathcal{T}_f$
I
$\hat{u}(x,t) - g_D(x,t)$
$\frac{\partial}{\partial \mathbf{n}}$
$\frac{\partial \hat{u}}{\partial \mathbf{n}}(x,t) - g_R(u,x,t)$
$\mathcal{T}_b$
Loss
Minimize
$\boldsymbol{\theta}^*$
BC & IC

Figure 1: The architecture of PINNs for 1D heat equation

0.2.2 Procedure of Solving PDEs with PINNs

Step 1: Approximating the Solution with a Neural Network A multi-layer perceptron (or other neural network architectures such as ResNet, CNN, and Transformer) is used to approximate $u(x,t)$ as $u_\theta(x,t)$, where θ represents the network parameters:

- The input layer receives spatial-temporal coordinates (x,t).
- Hidden layers apply transformations using activation functions.
- The output layer provides the estimated solution $u_\theta(x,t)$.

Step 2: Defining the Loss Function

- **Classical supervised learning approach:** assumes that we have a dataset of N_{data} known input-output pairs $((x,t),u)$:

$$\mathcal{D} = \{(x_i,t_i),u_i\}_{i=1}^{N_{data}}.$$

The u_θ is considered to be a good approximation of u if predictions $u_\theta(x_i,t_i)$ are close to target outputs u_i for every data sample i. We want to minimize the prediction error on the dataset. Hence, it's natural to search for a value θ^* solution of the following optimization problem:

$$\theta^* = \arg\min_\theta L_{data}(\theta) = \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} |u_\theta(x_i, t_i) - u_i|^2 \tag{3}$$

where L_{data} is the loss function.

- **Self-supervised learning of the PINN method:** the PINN method solves the PDEs **without requiring any input-output pairs** $((x,t),u)$ **inside the domain** $\Omega/\partial\Omega$, which makes it different from classical supervised learning approaches. Instead, the PINN method relies on two kinds of training data: the **boundary/initial conditions** and the **physical laws**.

  – **Boundary/initial condition loss:** The boundary condition indicates the value of solution $u(x,t)$ on the domain boundary $\partial\Omega_T = \partial\Omega \times [0,T]$, and the initial condition represents the value of $u(x,t)$ on Ω at $t=0$. To utilize both conditions for defining the loss, we can sample N_{bd} points (x,t) from the boundary $\partial\Omega_T$ and N_{ic} points x from the domain Ω. This leads to the following datasets:

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

  The model u_θ is expected to align with $g_R(x,t)$ for any (x_i,t_i) from dataset $\mathcal{D}_{bd}$ and align with $g_D(x_i)$ for any (x_i,t_i) from dataset $\mathcal{D}_{ic}$. This leads to the following loss term for boundary/initial conditions:

  $$L_{bd/ic}(\theta) = \frac{1}{N_{bd}} \sum_{i=1}^{N_{bd}} |u_\theta(x_i,t_i) - g_R(x_i,t_i)|^2 + \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} |u_\theta(x_i,0) - g_D(x_i)|^2 \tag{4}$$

  – **Residual (PDE) loss:** The PINNs method defines the PDE loss L_{PDE} as the strong-form residuals:

  $$L_{PDE}(\theta) = \frac{1}{N_c} \sum_{i=1}^{N_c} \left| F(x_i, t_i, u_\theta, \partial_x u_\theta, \partial_t u_\theta, \partial^2_{xx} u_\theta, \partial^2_{tt} u_\theta, \cdots, \mu) \right|^2 \tag{5}$$

  where the evaluation of residuals is performed on a set of N_c points denoted as (x_i, t_i). These points are commonly referred to as **collocation points**.

- **Total Loss:** A composite total loss function is typically formulated as follows:

$$L(\theta) = \omega_{PDE} L_{PDE}(\theta) + \omega_{bd/ic} L_{bd/ic}(\theta) \tag{6}$$

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

Step 3: Training with Gradient Descent With the loss function (6), the PINN method transfers the solving of the PDE problem (2) to the following minimization problem:

$$\min_{\theta} L(\theta) \tag{7}$$

Solving the above minimization problem is typically accomplished through a (stochastic) gradient descent algorithm. The algorithm is iteratively applied until convergence towards the minimum is achieved, either based on a predefined accuracy criterion or a specified maximum iteration number:

$$\theta_{t+1} = \theta_t - l_r \nabla_\theta L(\theta_t) \tag{8}$$

for the t-th iteration, also called an **epoch** in the literature, where l_r is the learning rate parameter.

0.3 Solving Inverse Problems with PINNs

Inverse problems in PDEs involve identifying unknown parameters in the equation given partial, often noisy, observations of the solution. Unlike forward problems, where the PDE and boundary conditions are fully known and we seek the solution, inverse problems require us to infer missing information about the PDE itself. PINNs provide a natural framework for solving such problems by **treating unknown parameters as learnable variables**.

0.3.1 Problem Setup

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

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

where:

- $u(x,t)$ is the solution to the PDE,
- $\mu(x,t)$ is an unknown function or parameter that we need to infer,
- Boundary and initial conditions are given, and
- We have access to **sparse and possibly noisy observations** $u(x_i, t_i)$ **at certain locations**.

Our goal is to **approximate both** $u(x,t)$ **and** $\mu(x,t)$ using neural networks.

0.3.2 Neural Network Architecture

To estimate both the solution $u(x,t)$ and the unknown parameter $\mu(x,t)$, we define two neural networks:

- A solution network $u_\theta(x,t)$ parameterized by θ.
- A parameter network $\mu_\phi(x,t)$ parameterized by ϕ.

These networks are trained simultaneously to satisfy both the given data and the governing PDE.

0.3.3 Loss Function Formulation

To train the PINN for inverse problems, we define three loss terms:

- **Data Loss**: Enforces agreement with observed measurements of $u(x,t)$ at sampled points $\{(x_i,t_i),u_i\}$:
$$L_{\text{data}}(\theta) = \frac{1}{N_{\text{data}}}\sum_{i=1}^{N_{\text{data}}} |u_\theta(x_i,t_i) - u_i|^2 \tag{9}$$

- **PDE Residual Loss**: Ensures that the learned functions satisfy the governing PDE:
$$L_{\text{PDE}}(\theta,\phi) = \frac{1}{N_c}\sum_{i=1}^{N_c} |F(x_i,t_i,u_\theta,\partial_x u_\theta,\partial_t u_\theta,\mu_\phi)|^2 \tag{10}$$

- **Boundary and Initial Condition Loss**: Enforces the solution to satisfy given boundary/initial conditions:
$$L_{\text{BC/IC}}(\theta) = \frac{1}{N_{bd}}\sum_{i=1}^{N_{bd}} |u_\theta(x_i,t_i) - g(x_i,t_i)|^2 \tag{11}$$

The **total loss function** is a weighted sum:
$$L(\theta,\phi) = \omega_{\text{data}}L_{\text{data}} + \omega_{\text{PDE}}L_{\text{PDE}} + \omega_{\text{BC/IC}}L_{\text{BC/IC}} \tag{12}$$
where the weights $\omega_{\text{data}},\omega_{\text{PDE}},\omega_{\text{BC/IC}}$ control the contribution of each term.

0.3.4 Training with Gradient Descent

For the inverse problem, the minimization problem is formulated as:
$$\min_{\theta,\phi} L(\theta,\phi)$$
and (stochastic) gradient descent algorithms (such as LBFGS, SGD, and Adam) can be used to update the network parameters:
$$\theta_{t+1} = \theta_t - l_r\nabla_\theta L(\theta_t,\phi_t)$$
$$\phi_{t+1} = \phi_t - l_r\nabla_\phi L(\theta_t,\phi_t)$$
where l_r is the learning rate.

0.4 Advantages and Challenges of PINNs

0.4.1 Advantages

- **Mesh-free**: No need for grid generation.
- **Handles high-dimensional problems**: Useful for problems where classical methods suffer from the curse of dimensionality.
- **Solves inverse problems**: Learns unknown PDE parameters from data.

0.4.2 Challenges

- **Training instability**: Requires careful tuning of hyperparameters.
- **Limited generalization**: May struggle with sharp gradients or discontinuities.
- **Computational cost**: Training deep networks can be expensive.