1. PINNs for solving PDEs

1.1 PINN Algorithm

Consider the PDE parameterized by \( \theta \) for the solution \( u(x) \) with \( x = (x_1, x_2) \) defined on a domain \( \Omega \subset \mathbb{R}^2 \). The PDE is:

\[
\frac{\partial u}{\partial t} + \nabla \cdot (a(x, \theta) \nabla u) = f(x, \theta), \quad x \in \Omega, \quad \theta \in \Theta.
\]

with boundary conditions (BC) \( \partial u/\partial n = g(x) \) on \( \partial \Omega \).

We consider time \( t \) as a special component of \( x \), and \( \Omega \) contains the temporal domain. The initial condition (IC) can be simply treated as a special type of Dirichlet boundary condition on the spatio-temporal domain.

1.3 Learning mode

Recent studies show that for function approximation, neural networks learn target functions from low to high frequencies, but we show that the learning mode of PINNs is different due to the existence of high-order derivatives.


2.1 Usage

Solving differential equations in DeepXDE is no more than specifying the problem using the build-in modules, including computational domain (geometry and time), PDE equations, BC/IC, constraints, training data, network architecture, and training hyperparameters.

2.2 Customizability

All the components are loosely coupled, and thus DeepXDE is well-structured and highly configurable.

3. Demonstration examples

3.1 Forward problem: Poisson equation

\[-\Delta u(x, y) = 1, \quad (x, y) \in \Omega; \quad u(x, y) = 0, \quad (x, y) \in \partial \Omega.\]

3.2 Inverse problems

The Lorenz system:

\[
\begin{align*}
\frac{dx}{dt} &= \sigma(y - x), \\
\frac{dy}{dt} &= x(z - y) + \beta y, \\
\frac{dz}{dt} &= xy - \gamma z.
\end{align*}
\]

A diffusion-reaction system on \( x \in [0,1], \quad y \in [0,10] \):

\[
\frac{\partial u}{\partial t} = \frac{\partial}{\partial x} \left( k_{12} \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left( k_{21} \frac{\partial u}{\partial y} \right) + E u.
\]

4. data: geometry + PDE or data + network architecture + training hyperparameters.

5. network:

\[
\text{network} = \text{deepxde.Model(data, f, \theta, \phi, \psi, \xi, \eta, \zeta)}.\]

6. model: data + network:

\[
\text{model} = \text{deepxde.Model(data, \text{network})}.\]

7. train the model

\[
\text{model.train(..., ...)}.
\]

References