1. PINNs for solving PDEs

1.1 PINN Algorithm

Consider the PDE parameterized by $\theta$ for the solution $u(x)$ with $x = (x_1, \ldots, x_d) \in \Omega \subset \mathbb{R}^d$, with boundary conditions (BC) $\mathcal{B}(u, x) = 0$ on $\partial \Omega$. We consider time $t$ as a special component of $x$, and $\Omega$ contains temporal domain. The initial condition (IC) can be simply treated as a special type of Dirichlet boundary condition on the spatial-temporal domain.

![Figure 1: Schematic of a PINN for solving the diffusion equation](image)

Procedure 1: The PINN algorithm for solving differential equations.

1. Construct a neural network $u(x, \theta)$ with parameters $\theta$ as a surrogate of the solution $u(x)$.
2. Specify the two sets of 'residual points': $T^r \subset \Omega$ and $T^i \subset \Omega$ for the equation and boundary/initial conditions.
3. Specify a loss function by summing the weighted $L^2$ norm of both the PDE equation and boundary condition residuals.
4. Train the neural network to find the best parameters $\theta$ by minimizing the loss function $L(\theta)$.

To measure the discrepancy between the neural network $u$ and the PDE constraints, we consider the loss function:

$$
L(\theta) = \mathbb{E}_{(x, t) \in T^r} \left[ \frac{1}{2} \left( \nabla^2 u(x, \theta) - \mathcal{L}(\theta) \right)^2 + \left( u(x, \theta) - f(x, \theta) \right)^2 \right],
$$

where $
abla^2 u(x, \theta)$ is the Hessian matrix of the neural network.

1.2 Approximation theory

Whether there exists a neural network that can simultaneously and uniformly approximate a function and its partial derivatives? For $m = (m_1, \ldots, m_d) \in \mathbb{Z}_+^d$, we set $m = m_1 + \cdots + m_d$ and $\ell^d = \left[ \frac{m}{d} \right]$. Theorem 1 (Pinkus, 1999) Let $u \in C^m([0, 1]^d)$ and set $m = m_1 + \cdots + m_d$. If $u \in C^m([0, 1]^d)$ and $u$ is a polynomial. Then the space of single hidden layer neural nets $\mathcal{M}(x) = \text{span}(x, x^2, \ldots, x^m, \sin(x), \cos(x))$ is dense in $C^m([0, 1]^d)$.

1.3 Learning model

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.

![Figure 2: Convergence of the amplitude for each frequency during the training process.](image)

**Procedure 2:** RAR for improving the distribution of residual points.

1. Select the initial residual points $T^r$, and train the neural network for a limited number of iterations.
2. Estimate the mean PDE residual $\mathcal{E}_r$, from Monte Carlo integration, i.e., the average of values at a set of randomly sampled locations $S = \{x_1, x_2, \ldots, x_s\}$.
3. If $\mathcal{E}_r < \mathcal{E}_o$, otherwise, add new points with the largest residuals in $S \nabla T$, and go to Step 2.

2. DeepXDE (https://deepxde.readthedocs.io)

2.1 Usage

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

**Procedure 3:** Poisson equation over an L-shaped domain.

1. Geometry

2. PDE

3. BC

4. Data: geometry + PDE + BC + training points

5. network

6. model: data + network

7. train the model

Figure 3: Procedure for solving differential equations in DeepXDE.

3. Demonstration examples

3.1 Forward problem: Poisson equation

$$
\Delta u(x, y) = 1, \quad (x, y) \in \Omega, \quad u(x, y) = 0, \quad (x, y) \in \partial \Omega
$$

![Figure 4: Comparison of the PINN solution with the solution obtained by using spectral element method (SEM).](image)

**Procedure 4:** Solving the diffusion equation using deep learning.

1. Diffuse Neumann/Robin periodic/general BC & IC;
2. feed-forward network, and residual network.
3. Customize

**3.2 Inverse problems**

The Lorenz system:

$$
\frac{dx}{dt} = \sigma(y - x), \quad \frac{dy}{dt} = \rho z - y - xz, \quad \frac{dz}{dt} = x y - \beta z.
$$

A diffusion-reaction system on $x \in [0, 1], \ell \in [0, 60];$

$$
\frac{\partial p}{\partial t} = \Delta p + \frac{\partial}{\partial x} \left[ \frac{ \partial p }{ \partial x } \right],
$$

where $p(x, t) = \frac{\partial y}{\partial x} e^{-\beta t}$.

![Figure 5: Comparison of the PINN solution with the solution obtained by using spectral element method (SEM).](image)

**Procedure 5:** Solving the diffusion equation using deep learning.

1. Parameter space $\mathcal{X}$
2. training data
3. network
4. model: data + network
5. train the model

Figure 6: Comparison of the PINN solution with the solution obtained by using spectral element method (SEM). (A) Poisson equation, (B) PDE equation, (C) the absolute error.

Figure 7: Identified values of (A) the Lorenz system and (B) diffusion-reaction system converge to the true values.

References


Workshop on Mathematical Machine Learning and Application, Virtually, December 2020