Physics Informed Neural Networks

PaAC

Abhivansh Gupta
23112004
abhivansh_g@ch.iitr.ac.in
Chemical Engineering
+91 8302905563

INTRODUCTION

Differential equations play a pivotal role in modeling physical, chemical, and biological systems. Traditional methods like finite difference schemes, finite element methods, or spectral methods are effective but often computationally expensive, particularly in high-dimensional systems. Physics-Informed Neural Networks (PINNs) have emerged as a novel solution by embedding the governing laws of physics directly into the neural network's loss function.

This project focuses on solving a time-dependent partial differential equation (PDE) using a PINN. The equation models phenomena such as heat conduction or wave propagation, characterized by a combination of sinusoidal and exponential terms. PINNs allow for a data-efficient solution by enforcing the PDE residuals as soft constraints during the training process, minimizing the reliance on labeled data.

The goal is to approximate the solution *u(x, t)* within a defined spatial-temporal domain using a neural network trained with the physics-informed approach. This demonstrates PINNs' potential as a general-purpose solver for PDEs.

MATHEMATICAL FORMULATION

The specific PDE studied is:

$(\partial u/\partial t) - \boldsymbol{\Delta} u = 0$,

where **Δ** denotes the Laplacian operator, and the equation is defined over the domain $[0,1] \times [0,1]$. The problem setup includes:

1. <u>Initial Condition</u>: $u(x,0) = \sin(\pi x)$
2. <u>Boundary Conditions</u>: $u(0, t) = u(1, t) = 0$.

The equation describes a diffusive process where the solution's time evolution is governed by the Laplacian's impact and the initial state.

Traditional methods to solve this PDE involve discretizing the domain and approximating derivatives using schemes such as:

- Finite Difference Methods (FDM): Simplifies the problem into a system of linear equations.
- Separation of Variables: Derives an exact solution for specific boundary conditions.
- Software Solutions: MATLAB or Python libraries like SciPy offer tools for solving PDEs numerically.

While accurate, these methods may face challenges like the curse of dimensionality or grid dependency, which PINNs aim to mitigate.
Assumptions

- The system is described by a well-posed PDE with sufficient smoothness.
- The initial and boundary conditions are precisely defined and adhered to.
- The PINN can generalize sufficiently within the provided domain.

PINN DESIGN

Neural Network Architecture

The PINN used in this project is a fully connected feed-forward neural network designed as follows:

- Layers: 3 hidden layers.
- Neurons: 50 neurons per layer.
- Activation Function: Hyperbolic tangent (*tanh*), chosen for its ability to approximate smooth functions effectively and handle vanishing gradients.
- Output Layer: Single neuron representing u(x, t).

Training Process

- Data Preparation: Synthetic training data was sampled uniformly within the spatial-temporal domain. This included both collocation points for physics loss computation and labeled points for supervised learning.

- Loss Function:
  - Physics Loss: Residuals of the PDE and boundary conditions.
  - Data Loss: Mean squared error (MSE) between the predicted and exact solutions.
  - Combined loss: L=L_data +λ*L_physics , where λ balances the two terms.
- Optimizer: Adam optimizer with an initial learning rate of 1 ×10−3.
- Epochs: 5000 iterations were performed to minimize the combined loss function.

Implementation

Tools and Libraries Used:

- TensorFlow: For building and training the neural network.
- NumPy: For data manipulation and mathematical computations.
- Matplotlib: For visualizing results, including 3D scatter plots.

Code Description

Data Preparation:

  - Randomly sampled *(x, t)* inputs and computed exact solutions using the analytical form of the PDE. These serve as benchmarks for comparison.
  - Model Definition: The `PINN` model was implemented as a custom subclass of `tf.keras.Model`. Layers and activations were defined with an overriding call method for forward passes.
  - Training Loop: Training combined labeled data and collocation points to simultaneously minimize supervised and unsupervised losses. Gradient computation used `tf.GradientTape`.

- Visualization: Results were plotted in 3D to show the alignment between predicted and exact solutions. Heat-maps were used to highlight error distribution across the domain.

Challenges Encountered

1. Gradient Calculation: Accurate computation of the PDE's derivatives required careful use of TensorFlow's automatic differentiation tools.
2. Hyperparameter Tuning: Balancing *L_data* and *L_physics* was critical to avoid overfitting to data points or collocation points.
3. Convergence Issues: The loss landscape proved challenging, leading to slow convergence initially. Reducing the learning rate and using adaptive optimizers resolved this.

RESULTS

Quantitative Results

- The PINN achieved a mean absolute error (MAE) below 10^−3, indicating high accuracy relative to the exact solution.
- Loss components decreased steadily, demonstrating the model's ability to satisfy both data fidelity and PDE constraints.

Visual Results

3D Scatter Plots:

- Displayed excellent overlap between predicted and exact solutions.
- Heatmaps: Error distributions were primarily concentrated near the domain's boundaries, highlighting potential improvements in boundary loss handling.

CONCLUSION

Interpretation

The PINN successfully solved the PDE, demonstrating its capability as a flexible and efficient tool for solving physics-constrained problems. The results closely matched the exact solution, validating the model's accuracy.

Limitations

- Computational overhead: Training PINNs is computationally intensive compared to traditional methods for low-dimensional problems.
- Domain sensitivity: PINNs struggled with regions near boundary discontinuities.

Future Improvements

1. Loss Balancing: Introduce adaptive weights to dynamically balance data and physics losses.
2. Enhanced Architectures: Use residual networks or Fourier feature mappings for improved convergence.
3. Multi-GPU Training: Scale the model to handle more complex PDEs using parallel computing.