PINN code quick rundown of good to know things

May 2024

1 Introduction

The goal is to create a **modular design** that allows different partial differential equations (PDEs) to be easily imported and integrated into the main class.

2 Main Class

The main class consolidates all key components, including the training scheme, sampling type, learning rate scheduling, and the model itself. Additionally, it manages all parameter settings, such as weight balancing and any extra PDE coefficients that may be required.

3 Model/PINN Types

The models are customizable, allowing users to define the depth, width, activation functions, and mapping when calling the PINN class. This setup slightly differs for the transformer-based PINN. The key feature of both PINN types is that users input the PDE, boundary conditions (BC), and initial conditions (IC) functions into the model to compute the losses. This design eliminates the need to modify the PINN code when testing different function sets. Additionally, NTK weight balancing uses the PDE function, **which is why the forward method includes parameters** even if the PINN doesn't directly use them. This design ensures compatibility with the NTK-functionalized model forward method, allowing the reuse of the PDE function.

The models can handle higher-dimensional PDEs, though weight balancing functions and domain samplers are currently implemented for two inputs only. The main class can also handle **multiple PDE coefficients.**

Note: Transformer-based PINN non-dimensionalization is not supported due to differing dimensions compared to MLP-based PINN. For transformer PINN, define the domains as [1,1]. Implementing non-dimensionalization by checking dimensions could be computationally heavy. The purpose of the domains input is to non-dimensionalize the inputs to the range [0,1].

4 Weight balancing

NTK and NTK2 differ in matrix computation, with NTK2 not using the PDE equation. NTK2 and GradNorm work with transformer-based PINNs, while MLP PINNs support all implemented weight balancing schemes.

5 Domain sampler and Training schemes

The domain sampler offers two modes: dynamic sampling, which generates new samples from the domain for each epoch, and **static sampling, which provides sorted samples to support causality training scheme**. Use the static domain sampler for causality weighting schemes to avoid sorting the sample each time.

The **transformer sampler works exclusively with transformer-type PINNs**, creating pseudo-sequences. The causality weighting training scheme adjusts the PDE residual by the causality coefficient. Currently, the system matrix A's dimension are hard-coded to match the PDE residual dimension, as dynamically creating the matrix is computationally intensive. **Adjusting the number of PDE samples requires changing the matrix dimension accordingly.**

6 Optimizers

Implemented optimizers include Adam, SGD, and LBFGS. LBFGS operates differently from Adam and SGD. This is why the training schemes do not include points as inputs for the train cycle function, aligning with the closure() function signature required by LBFGS. Note that LBFGS does not utilize learning rate schemes or initial learning rates.

7 Expanding the code

The main limitation is that **domain samplers** and **NTK** schemes are implemented **only for 1D PDE equations.** To extend the code to higher dimensions, consider offloading torch.cat operations to the sampler side. Another challenge is ensuring non-dimensionalization works correctly, providing inputs in the range [0,1] while allowing equations to operate within the desired domain. NTK argument operations could potentially be handled in a loop.