UE3 GPU Project: Physics-informed Neural Operators (PINO) on 2D Wave equation

Abhishek Purandare, HPC-AI

March 2023

Problem statement

Physics-informed Neural Operator

The approach involves using a neural network to approximate the solution to a PDE, while enforcing the physical constraints of the system through the incorporation of the governing PDE as a constraint in the training process. In PINO, the neural network is trained to learn the solution to the PDE by minimizing the difference between the predicted solution and the true solution, as well as the difference between the predicted and true physical constraints. The physical constraints are incorporated into the loss function as a penalty term, and can be formulated using a variety of techniques, such as residual networks or Fourier neural operators.

The advantage of PINO over traditional numerical methods is that it can learn complex solutions to PDEs directly from data, without the need for manual feature engineering or discretization of the domain. This makes it a powerful tool for solving PDEs in complex domains where traditional numerical methods may struggle, such as in multi-physics systems or systems with high-dimensional inputs.

2D wave equation

The 2D wave equation is a PDE that describes the behavior of waves propagating through a two-dimensional medium, such as water or air. It can be written in terms of the displacement $u(x, y, t)$ of the medium from its equilibrium position at each point (x, y) and time t, as:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

where c is the wave speed. This equation states that the second partial derivative of the displacement with respect to time is equal to the square of the wave speed times the Laplacian of the displacement, which is the sum of the second partial derivatives of the displacement with respect to each spatial dimension.

Dataset Generation

Gaussian Random Fields

Mathematically, a GRF can be defined as a function $Z(x)$ that assigns a random variable to each point x in a spatial domain D. The random variables $Z(x)$ are assumed to be drawn from a Gaussian distribution with mean $\mu(x)$ and variance $\sigma^2(x)$:

$$Z(x) \sim \mathcal{N}(\mu(x), \sigma^2(x))$$

The paper [1] utilized Gaussian random fields (GRFs) to produce randomized spatial data. I have carried out a similar operation, generating a set of random fields that will be used to solve an equation. To train the model, I have created approximately 1000 samples, and for validation, I have generated 100 samples, all of which adhere to periodic boundary conditions. The grid size for both the x and y axes is 128.

Training

I trained a stationary PDE where the operator for IC (t=0) to a solution (t=100) is derived by the PINO. I kept most of the hyperparameters intact for this part and tested on a few samples before starting the main training phase. I use two gradient methods, exact, and fourier derivatives.

- Exact derivatives are straightforward and are calculated using torch's autograd.
- Fourier derivatives can be calculated using the Fourier transform. Let's say we have a function $f(x)$ and we want to calculate its derivative df/dx. We can take the Fourier transform of $f(x)$ to obtain its Fourier transform $F(k)$, where k is the frequency domain variable. Then, We can use the fact that the Fourier transform of the derivative of $f(x)$ is $ik\mathcal{F}(f(k))$ (up to a constant factor depending on the Fourier transform conventions used). Finally, we can take the inverse Fourier transform of $ik\mathcal{F}(f(k))$ to obtain the Fourier derivative of $f(x)$, which gives us df/dx as a function of x.

By default, Modulus computes the solution loss implicitly. Both the residual loss, and the solution loss (data loss) is then assigned some weighting to finally contribute to a global loss sum which is minimized by the network.

I varied the number of steps taken between 10,000-50,000. It takes no more than 15–20 minutes for 10,000 steps. For the learning rate, I started with a small value of 0.0001 with a gradual decay rate of 0.95. Finally, I set decoder layers, and FNO layers to 1, and 4 respectively. I see no major improvement in the performance of the model when I increased the number of layers, and their latent space sizes.

Results

Using GPUs for training the model, the batch size was increased up to 64 resulting in significant performance degradation as shown in 1. To reduce the training time, a batch size of 32 was chosen. It is worth noting that the losses were high and the model's performance plateaued at 0.4 and 0.5 errors for exact and Fourier, respectively, as shown in 2. However, predicted solutions appear to be more or less similar to the true solutions 3 4. This is quite unexpected considering how poorly the operator is performing in terms of metrics. Looking at the predictions from both the exact and fourier derivative operators, exact solution outperforms fourier. There is one more aspect about the GRFs worth mentioning is that initial conditions generated always have some stripes in them.

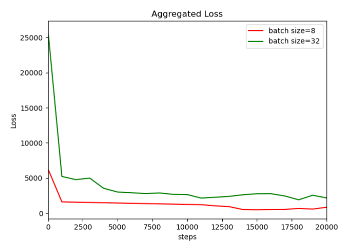

Figure 1: Batch size loss comparison

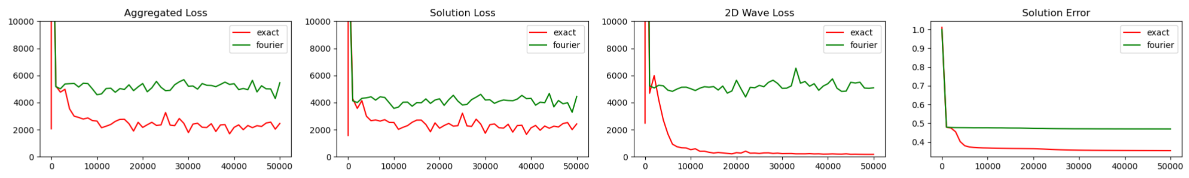

Figure 2: Losses and Error for Exact, and Fourier derivatives

Conclusion

- Given the size of the training set (1000), and testing set (100), the problem at hand is quite trivial for GPUs to solve. And using the batch size of 32 may be too small.
- Regarding the losses, it is still unclear on why the values are high. It could be the architecture, or some equation-specific settings that might have been overlooked.
- I also attempted to incorporate dynamic PDE that is evolving over time. I implemented the logic (reflected from [1]) to compute the temporal derivatives in the forward pass of the mode. However, FNO does not have any temporal dimension in its input.

References

[1] Rosofsky, Shawn, et al. Applications of Physics Informed Neural Operators. 8 Dec. 2022.

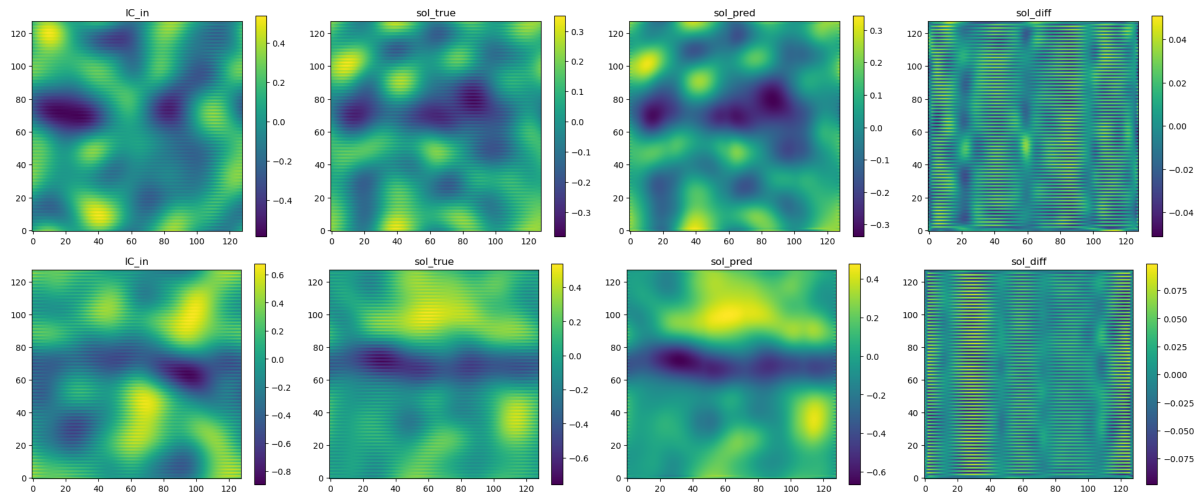

Figure 3: Using Exact derivatives

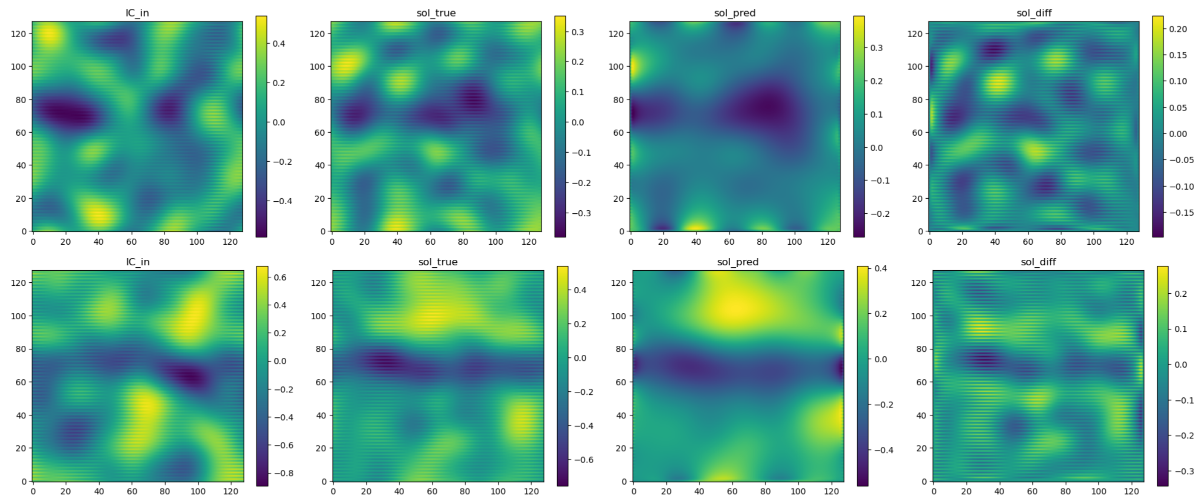

Figure 4: Using Fourier derivatives