physic informed Neural Network project report

Keyan Chen

March 2024

1 Introduction

The area of operator learning, which is changing quickly and growing fast in the field of computer science, has made great progress with new neural operators. These operators, which are based on how neural networks are built, have changed how we solve computer problems. Some of the main developments include Fourier Neural Operators (FNO) and Deep Operator Net (DeepONet). FNO uses a method called Fourier transform to work with data in a global way, while DeepONet uses two smaller networks - one called the branch net and the other called the trunk net - to understand data and make predictions similar to how attention in the brain works. This is my research topic about operator learning under Prof.Shandian Zhe's instruction this semester.

2 motivation

However, to work really well, neural operators need a lot of data for training. This can be hard, especially when the data is not easy to get or is expensive, like with inverse problems. To deal with this, a new area called physics-informed machine learning has become popular. This includes physics-informed neural networks (PINN), which use rules of physics to help train the model. This helps the model understand physics better without needing tons of data. Building on this, physics-informed neural operators (PINO) have been created. PINO uses physics rules to make models more accurate and less dependent on data, and it's used to train models like DeepONet and FNO.

Despite how well PINO works for many science problems, like studying fluids and chemical reactions, understanding the deep physics involved is still tough. This is especially true for complicated real-life problems like simulating circuits or climate systems, where it's hard to use traditional methods without detailed physics knowledge or a lot of data.

Therefore, we wish to improve the training ability when the dataset is small but which is high resolution.

3 progress

We first learned about FNO network structure to solve operator learning problems. This network structure is the most powerful and popular one recently.

3.1 Fourier Neural Operator

The Fourier Neural Operator (FNO) is a big improvement in how neural networks are designed, especially for solving complex math problems like partial differential equations (PDEs). When it gets a function as input, FNO first uses a simple network on each part of the function at its location, making the input more complex. The main part of FNO is the Fourier layer. This layer changes the input in a step-by-step way, first by a simple equation and then by making it more complex, shown by the formula:

$$h(\mathbf{x}) \leftarrow \sigma\left(\mathcal{W}h(\mathbf{x}) + \int \kappa(\mathbf{x}-\mathbf{x}')h(\mathbf{x}')d\mathbf{x}'\right).$$

In this formula, h is what we start with for the Fourier layer, k is a special function for integration, and $\mathcal{W}$ is the function that adds complexity. The step where it combines inputs in a special way is done efficiently through a math trick that involves changing the function into a different form, mixing it, and then changing it back:

$$\int \kappa(\mathbf{x}-\mathbf{x}')h(\mathbf{x}')d\mathbf{x}' = \mathcal{F}^{-1}[\mathcal{F}[\kappa]\cdot\mathcal{F}[h]](\mathbf{x})$$

This trick uses the Fast Fourier Transform (FFT) on h, mixes it with a kernel in a different form, and then brings it back with the inverse FFT. The simple equation part, Wh(x), is done using the usual mixing steps. After going through several Fourier layers, the final result comes from applying another simple network to each part, bringing the details back to the original form.

3.2 Methodology

When we don't have the physics equations (like not knowing the specific partial differential equations, PDEs), we can't use a physics-based loss term in the Physics-Informed Neural Operator (PINO) setup. To tackle the problem of learning about operators with very little data, especially in real situations, we suggest a two-part plan inspired by the need to discover the physical laws from the data we have. This is extremely useful in areas like climate study, where getting data is either too expensive or the data is too sparse, and understanding the full physics is challenging.

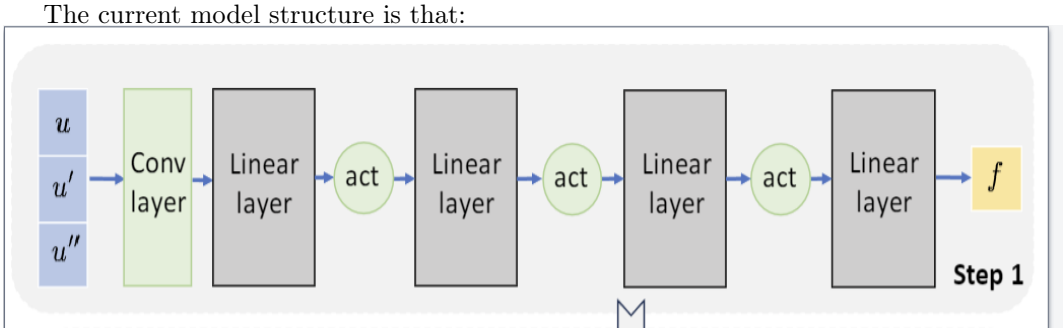
The first part of our plan introduces a new way to figure out the physics from a few pieces of data. We've noticed that although the connection between the input function f and the output u can be complex, needing data from the entire area, the actual PDE system can be simplified to a local mix of u and its

derivatives. We use a neural network, ϕ, to guess the broad form of the PDE system as it simplifies to a local combination of u and its derivatives. We use a neural network ϕ to approximate the general form of N,
$N[u](x) \approx \phi\left(x, u(x), S_1(u)(x), \ldots, S_Q(u)(x)\right)$
where S_Q are the derivative operators we think should be in the system. This approach turns a single input and output pair into many training points. For example, from data on a 128x128 grid, we can go from one pair being one data point to having 16,384 data points.We then use the loss to figure out the parameters for ϕ, written as
$L_\phi = \sum_{n=1}^{N} \sum_{j=1}^{M} \left[\phi(x_j, u_n(x_j), S_1(u_n)(x_j), \ldots, S_Q(u_n)(x_j)) - f_n(x_j)\right]^2$
where f_n and $u_n$nare the input and output functions for the n-th training example. To get the derivatives of each u_n, we use numerical differentiation, and then we input these into the neural network ϕ to predict the output. Since numerical differentiation can introduce errors, we include a convolution layer in ϕ to gather information from nearby points about u and its numerical derivatives, which helps correct these errors. Following this, we apply layers that do linear transformations and then add complexity, to predict the output at each point.

Although this method starts without known physics laws, it helps us learn them from the data. This approach is particularly useful in complex systems where we might only have data for steady states, even though the system changes over time. This shows how flexible and powerful this method can be in finding hidden physical laws from limited data.

The whole resources have been tested well and next, we will construct a training structure and figure out how to use these parameters like derivatives and combined with fno.

The current model structure is that:



Next, I will try to use the neural operator with pseudophysics regularization. Suppose our goal is to learn a function-to-function mapping $\psi : \mathcal{F} \to \mathcal{U}$, where $\mathcal{F}$ and $\mathcal{U}$ are two function spaces (*e.g.,* Banach spaces) and $\mathcal{U}$ is the solution of $\mathcal{F}$. The training dataset comprises pairs of discretized input and output functions, $\mathcal{D} = \{(\mathbf{f}_n, \mathbf{y}_n)\}_{n=1}^{N}$, where each $\mathbf{f}_n$ are samples of a function $f_n \in \mathcal{F}$, and $\mathbf{y}_n$ are samples of $\psi[f_n] \in \mathcal{U}$. We denote all the input function samples by $\mathbf{F} = [\mathbf{f}_1, \ldots, \mathbf{f}_N]^\top$ $(N \times d)$, and output function samples by $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_N]^\top$

$(N \times d)$.
We propose a co-training process based on a small set and high-resolution input $f_n \in \mathcal{F}$. That is, we model

$$\psi(\mathbf{f}_n) = \mathbf{u}_n^*$$

We will use FNO as our mapping model ψ to learn the mapping relationship between functions and their solutions. Suppose we only have a small set of samples; we will have mapping pairs $(\mathbf{f}_n, \mathbf{u}_n), 1 < n < N$, where N is the number of samples. We can use these pairs as training samples to train a model ψ.

First, we want to use the trained model to get predicted samples of $\mathbf{u}$ from our generated samples of $\mathbf{f}$. Here, the $\mathbf{f}$ samples were generated by using the following method: The function $\mathbf{f}^{(i)}$ for $i = [1, \ldots, I]$ were drawn independently from Gaussian Process with RBF kernel and length scale 0.2, the grid resolution can be chosen to be same with the training pair samples that we have. Now, we have related solutions $\mathbf{u}^{(i)}$ of $\mathbf{f}^{(i)}$ for $i = [1, \ldots, I]$ as generated samples $(\mathbf{f}^i, \mathbf{u}^i), 1 < i < I$, where $\mathbf{u}^{(i)}$ are the predicted result of $\psi(\mathbf{f}^{(i)})$

We then use the original pair samples $(\mathbf{f}_n, \mathbf{u}_n)$ inversely to train an FNN model ϕ, where we want the model to learn the mapping from $\mathbf{u}$ to $\mathbf{f}$.

$$\phi(\mathbf{u}_n) = \mathbf{f}_n^*$$

The derivative information $[\mathbf{u}, \frac{\partial \mathbf{u}}{\partial \mathbf{x}}, \frac{\partial \mathbf{u}}{\partial \mathbf{y}}, \frac{\partial^2 \mathbf{u}}{\partial \mathbf{x}^2}, \frac{\partial^2 \mathbf{u}}{\partial \mathbf{y}^2}, \mathbf{x}, \mathbf{y}]$ will be considered as one training example for one pixel suppose we have a $d \times d$ resolution function which stands for a $\mathbf{u}$ grid with shape (d, d). The calculation of the partial derivative uses the rows and columns to calculate the numerical difference along x and along y. The first derivative is calculated from the u grid and constructs two gradient grids with x and y. The second derivative is calculated from the first derivative grids and gets two more grids with x and y. An example is the following.

Second, we can use the new predicted $\mathbf{f}^{i*}$ from FNN model ϕ to participate in the third model:

$$\psi^{'}(\mathbf{f}_n) = \mathbf{u}_n^*$$

Where is the same model type we use as our first model ψ, but it has a different loss function. We will use the L_2 relative error loss function for training in model ψ and the MSE loss function for training in model *phi*. Since we need the output of mode *phi* to join the third model $\psi^{'}$ training process, we propose our third loss function:

$$Loss = L_2(\psi(\mathbf{f}_n), \mathbf{y}_n) + L_2(\phi(\mathbf{y}_n), \mathbf{f}_n) + \lambda \mathbb{E}_{\mathbf{P}(f)}(L_2(f^i, \phi(\psi(\mathbf{f}^i)))). \quad (1)$$

where $\mathbf{u}^i$ is the predicted solution results of our first model $\psi(\mathbf{f}^i)$ with our sampled $\mathbf{f}^i$. $\mathbf{f}_n$ is the original function training samples, $\mathbf{y}_n$ is the original solution training samples. λ is the constant parameter to control the value of our added regularization term. Moreover, L_2 stands for the calculation of L_2 relative error which is $L_2(\mathbf{a}, \mathbf{b}) = \sqrt{\frac{(\mathbf{a}-\mathbf{b})^2}{\mathbf{b}^2}}$. MSE stands for the mean square loss.

These loss functions means that we can add generated samples to enhance the learning ability of the model ψ.

4 data evaluation

Here is the data evaluation between FNO only and our methodology training, which runs on the Darcy equation dataset.

FNO only on darcy data:

```
fu Epoch [143/150], Training Loss: 0.0211, Test Loss: 0.1668
fu Epoch [144/150], Training Loss: 0.0206, Test Loss: 0.1657
fu Epoch [145/150], Training Loss: 0.0209, Test Loss: 0.1683
fu Epoch [146/150], Training Loss: 0.0219, Test Loss: 0.1669
fu Epoch [147/150], Training Loss: 0.0230, Test Loss: 0.1656
fu Epoch [148/150], Training Loss: 0.0234, Test Loss: 0.1693
fu Epoch [149/150], Training Loss: 0.0230, Test Loss: 0.1658
fu Epoch [150/150], Training Loss: 0.0235, Test Loss: 0.1690
```

our method on darcy data:

```
new fu Epoch [994/1000], Training Loss: 0.0032, Test Loss: 0.0677
new fu Epoch [995/1000], Training Loss: 0.0032, Test Loss: 0.0694
new fu Epoch [996/1000], Training Loss: 0.0032, Test Loss: 0.0677
new fu Epoch [997/1000], Training Loss: 0.0032, Test Loss: 0.0693
new fu Epoch [998/1000], Training Loss: 0.0032, Test Loss: 0.0676
new fu Epoch [999/1000], Training Loss: 0.0032, Test Loss: 0.0694
new fu Epoch [1000/1000], Training Loss: 0.0032, Test Loss: 0.0675
best error 0.06748652956073899
```

Here, we can see that the L2 error decreased from 0.169 to 0.0675. It shows that this method performs well on the Darcy data set.

5 future work

We need to find an appropriate real-world data set to try the training method here. Moreover, we could also try to make the training an alternative loop to see if it can perform like a circulation training.

6 Reference

Batlle, P., Darcy, M., Hosseini, B., and Owhadi, H. (2023). Kernel methods are competitive for operator learning. arXiv preprint arXiv:2304.13202.

Gupta, G., Xiao, X., and Bogdan, P. (2021). Multiwavelet- based operator learning for differential equations. Advances in neural information processing systems, 34:24048–24062.

Kovachki, N. B., Li, Z., Liu, B., Azizzadenesheli, K., Bhat- tacharya, K., Stuart, A. M., and Anandkumar, A. (2023). Neural operator: Learning maps between function spaces with applications to pdes. J. Mach. Learn. Res., 24(89):1– 97.

Lee, J. Y., Jang, J., and Hwang, H. J. (2023). op- pinn: Physics-informed neural network with operator learning to approximate solutions to the fokker-planck- landau equation. Journal of Computational Physics, 480:112031.

Li, S., Wang, Z., Kirby, R. M., and Zhe, S. (2022). Deep multi-fidelity active learning of high-dimensional out- puts. Proceedings of the Twenty-Fifth International Conference on Artificial Intelligence and Statistics.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhat- tacharya, K., Stuart, A., and Anandkumar, A. (2020a). Neural operator: Graph kernel network for partial differ- ential equations. arXiv preprint arXiv:2003.03485.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Stu- art, A., Bhattacharya, K., and Anandkumar, A. (2020b). Multipole graph neural operator for parametric partial differential equations. Advances in Neural Information Processing Systems, 33:6755–6766.

Li, Z., Kovachki, N. B., Azizzadenesheli, K., Bhat- tacharya, K., Stuart, A., Anandkumar, A., et al. (2020c). Fourier neural operator for parametric partial differen- tial equations. In International Conference on Learning Representations.

Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K., and Anandkumar, A. (2021). Physics-informed neural operator for learning partial dif- ferential equations. arXiv preprint arXiv:2111.03794.

Long, D., Mrvaljevic, N., Zhe, S., and Hosseini, B. (2022). A kernel approach for pde discovery and operator learn- ing. arXiv preprint arXiv:2210.08140.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. (2021). Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. Nature machine intelligence, 3(3):218–229.

Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., and Karniadakis, G. E. (2022). A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. Computer Methods in Applied Mechanics and Engineering, 393:114778.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems in- volving nonlinear partial differential equations. Journal of Computational physics, 378:686–707.

Rosofsky, S. G., Al Majed, H., and Huerta, E. (2023). Ap- plications of physics informed neural operators. Machine Learning: Science and Technology, 4(2):025022.

Seidman, J., Kissas, G., Perdikaris, P., and Pappas, G. J. (2022). Nomad: Nonlinear manifold decoders for opera- tor learning. Advances in Neural Information Processing Systems, 35:5601–5613.

Sethian, J. A. (1999). Fast marching methods. SIAM review, 41(2):199–235.