King Fahd University of Petroleum and Minerals

College of Computing and Mathematics

Mathematics Department

Math 619-Project

Term 231

Pi-Deeponet Section

Project: Deep Learning Methods for Partial Differential Equations (PDEs)

Name	KFUPM ID
Hashim Al-Sadah	201578370
Abdulwahab Alghamdi	201734070
Hussain Al-Sinan	202205120

Instructor: Dr. Jamal Al-Smail

ABSTRACT

Physics Informed Deep Neural Operator (PI-DeepONet) is a new method of solving partial differential equations (PDEs) using deep learning. This method is based on the combination of the Physics Informed Neural Networks (PINNs) and the DeepONet. PINNs are a class of neural networks that are trained to solve PDEs by enforcing the PDEs as a loss function. DeepONet is a deep learning architecture that is used to solve PDEs by learning the solution operator. PI-DeepONet is a combination of these two methods, which allows for the solution of PDEs with less data and computational cost. This part will provide an overview of the PI-DeepONet method and its applications.

INTRODUCTION

In this section, we provide a concise overview of the DeepONet model architecture [1], focusing specifically on the learning of solution operators for parametric Partial Differential Equations (PDEs). The term "parametric PDEs" refers to PDE systems where certain parameters are permitted to vary within a specified range. These parameters could include factors such as the shape of the physical domain, initial or boundary conditions, coefficients (constant or variable), and source terms. To address such problems comprehensively, let (U, V, S) be a triplet of Banach spaces, and $N : U \times S \to V$ be a differential operator, either linear or nonlinear. We examine parametric PDEs of the form $N(u, s) = 0$ [1], where $u \in U$ represents the parameters (input functions), and $s \in S$ is the corresponding unknown solution to the PDE system. It is assumed that for any $u \in U$, there exists a unique solution $s = s(u) \in U$ to the equation $N(u, s) = 0$ (subject to appropriate initial and boundary conditions). Consequently, a solution operator $G : U \to U$ is defined as $G(u) = s(u)$. Following the original formulation by Lu et al. [1], we represent the solution map G using an unstacked DeepONet denoted as G_θ, where θ encompasses all trainable parameters of the DeepONet network. As depicted in Figure 1, the unstacked DeepONet comprises two separate neural networks, known as the "branch net" and "trunk net", respectively. The branch net takes u as input and produces a feature embedding $[b_1, b_2, ..., b_q]^T \in \mathbb{R}^q$ as output, where $u = [u(x_1), u(x_2), ..., u(x_m)]$ denotes a function $u \in U$ evaluated at a

set of fixed locations $\{x_i\}_{i=1}^m$. On the other hand, the trunk net accepts continuous coordinates y as inputs and generates a feature embedding $[t_1, t_2, ..., t_q]^T \in \mathbb{R}^q$ as output. The final output of the DeepONet is obtained by merging the outputs of the branch and trunk networks via a dot product. Specifically, the DeepONet prediction $G_\theta(u)(y)$ for a function u evaluated at y is expressed as:

$$G_\theta(u)(y) = \sum_{k=1}^{q} b_k(u(x_1), u(x_2), ..., u(x_m)) \cdot t_k(y), \tag{1}$$

where θ represents the collection of all trainable weight and bias parameters in the branch and trunk networks. These parameters are optimized by minimizing the mean square error loss given by:

$$\mathcal{L}_{ODE} = \frac{1}{m} \sum_{j=1}^{m} \left(-\frac{d^2 G_\theta(f)(p_j)}{dp^2} - f(p_j) \right)_{p=x}^2$$

Where $G_\theta(f) = u_\theta$ and p_j is an input point to the trunk NN.

IMPLEMENTATION

We will consider the 1D Poisson equation as a case study to illustrate the implementation of the PI-DeepONet method. The 1D Poisson equation is given by:

$$-\frac{d^2 u}{dx^2} = f(x), \quad 0 \leq x \leq 1$$

Subject to the boundary condtions

$$u(0) = u(1) = 0$$

Where $f(x)$ is the forcing term. Our goal is to determine the function $u(x)$ for different forcing terms$f(x)$. In other words, we are trying to determine an operator G such that $G : f \to u$.

The operator G will be approximated by the branch neural network in the case of the DeepOnet model, $G_\theta \approx G$. Where θ represents the paramter of the branch NN. The model will consist of two neural networks, the branch NN and the trunk NN.

Branch Net

In the branch net we will consider the forcing force $f(x)$ which we limit to be in a space of polonomyal

$$f(x) \in \mathcal{P}^n(x) = a_0 + a_1 x + a_2 x^2 + \ldots \cdots + a_n x^n = \sum_{i=0}^{n} a_i x^i$$

. Where $a_i \in \mathbb{R}$ The branch net will take the forcing term $f(x)$ as input and output the approximated function $u(x)$. The branch net will be trained to minimize the loss function. In our work, we consider 100 different forcing terms $f(x)$ as the following

$$branch_{in} = \begin{pmatrix} f_1(x_1) & f_1(x_2) & f_1(x_3) & \ldots & f_1(x_{10}) \\ f_2(x_1) & f_2(x_2) & f_2(x_3) & \ldots & f_2(x_{10}) \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ f_{100}(x_1) & f_{100}(x_2) & f_{100}(x_3) & \ldots & f_{100}(x_{10}) \end{pmatrix}$$

Which yields an output of:

$$branch_{out} = \begin{pmatrix} u_1^{(1)} & u_2^{(1)} & u_3^{(1)} & \ldots & u_m^{(1)} \\ u_1^{(2)} & u_2^{(2)} & u_3^{(2)} & \ldots & u_m^{(2)} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ u_1^{(100)} & u_2^{(100)} & u_3^{(100)} & \ldots & u_m^{(100)} \end{pmatrix}$$

for each $f_i(x)$ where $i = 1, 2, 3, \ldots, 100$

$$G_\theta : \begin{bmatrix} f_i(x_1) & f_i(x_2) & f_i(x_3) & \ldots & f_i(x_{10}) \end{bmatrix} \to \begin{bmatrix} u_1^{(i)} & u_2^{(i)} & u_3^{(i)} & \ldots & u_m^{(i)} \end{bmatrix}$$

$$u_i(x) = \begin{bmatrix} u_1^{(i)} & u_2^{(i)} & u_3^{(i)} & \dots & u_m^{(i)} \end{bmatrix}$$

$u_i(x)$ is not yet evaluated at any point. For the evaluation, we will use the output of the trunk neural network.

Trunk Net

The input to the trunk neural network is points where we want to evalute the target function $u(x)$ at and we represneted by the following column vector

$$trunk_{in} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_{100} \end{pmatrix}$$

Note: I used p instead of x so there is no confusion with between these points and the sensor points.

since $0 \leq x \leq 1$, then $0 \leq p_j \leq 1$ for $j = 1, 2, 3, \dots 100$

The tunk neural network will map each point to a higher dimensions and in this case we require that the output dimensions of the branch and the trunk networks must be the same. Therefore, the output the trunk neural network is the following

$$trunk_{out} = \begin{pmatrix} b_1^{(1)} & b_2^{(1)} & b_3^{(1)} & \dots & b_m^{(1)} \\ b_1^{(2)} & b_2^{(2)} & b_3^{(2)} & \dots & b_m^{(2)} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ b_1^{(100)} & b_2^{(100)} & b_3^{(100)} & \dots & b_m^{(100)} \end{pmatrix}$$

For each p_j where $j = 1, 2, 3, \dots, 100$,

$$\mathbf{NN}_{\text{trunk}}(p_j) = \begin{bmatrix} b_1^{(j)} & b_2^{(j)} & b_3^{(j)} & \dots & b_m^{(j)} \end{bmatrix}$$

Evaluation

To evaluate the output u_i at the trunk points, we take the dot product of the branch output and the trunk output and that is why we require the output dimensions of both netowrks to be the same. For example, to evaluate $G(f_i)$ at p_j we perform the following operation

$$G_\theta(f_i)(p_j) = u_i(p_j) = \begin{bmatrix} u_1^{(i)} & u_2^{(i)} & u_3^{(i)} & \dots & u_m^{(i)} \end{bmatrix} \cdot \begin{bmatrix} b_1^{(j)} & b_2^{(j)} & b_3^{(j)} & \dots & b_m^{(j)} \end{bmatrix} = \sum_{n=1}^{m} u_n^{(i)} b_n^{(j)}$$

We are going to use only 1 hidden layer with 32 parameters for both the branch and trunk networks. Also, we will use LBFGS optimizer since it converges to the minimum faster.

RESULTS

We trained the model and are going to test it with $f(x)$ to be in the same space that we trained it on and on functions outside of the domain to see how general the model is.

Case 1 - Constant

For $f(x) = -1$ which is a simple case with an exact solution $u(x) = \frac{1}{2}(x^2 - x)$ we get the following results

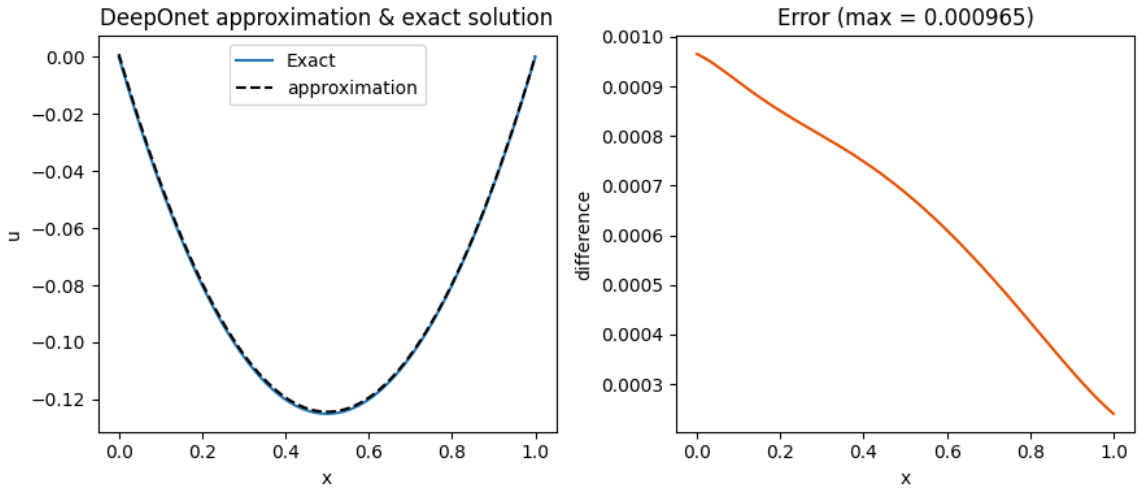

Figure 1: difference between the exact solution and the predicted solution for $f(x) = -1$

As we see in , the max error is very small and the model was able to predict the solution very well.

Case 2 - Quadratic

In this case, we will consider the following forcing term

$$f(x) = x + 3x^2$$

Which has an exact solution of

$$u(x) = \frac{1}{6}x^3 - \frac{1}{4}x^4 + \frac{10}{24}x.$$

The results are shown in the following figure

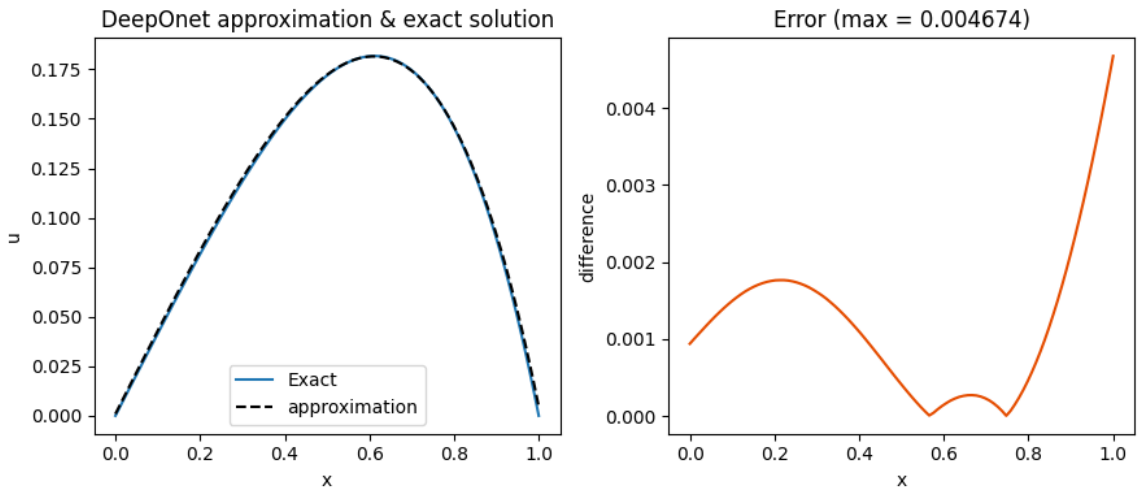

Figure 2: difference between the exact solution and the predicted solution for $f(x) = x + 3x^2$

Case 3 - Out of Space Functions (Exponential)

In this case, we will consider the following forcing term

$$f(x) = e^x$$

Which has an exact solution of

$$u(x) = -e^x + xe - x + 1$$

Since the forcing term is from a function space that the model hasn't been trained on, we expect the model to be less accurate. The results are shown in the following figure

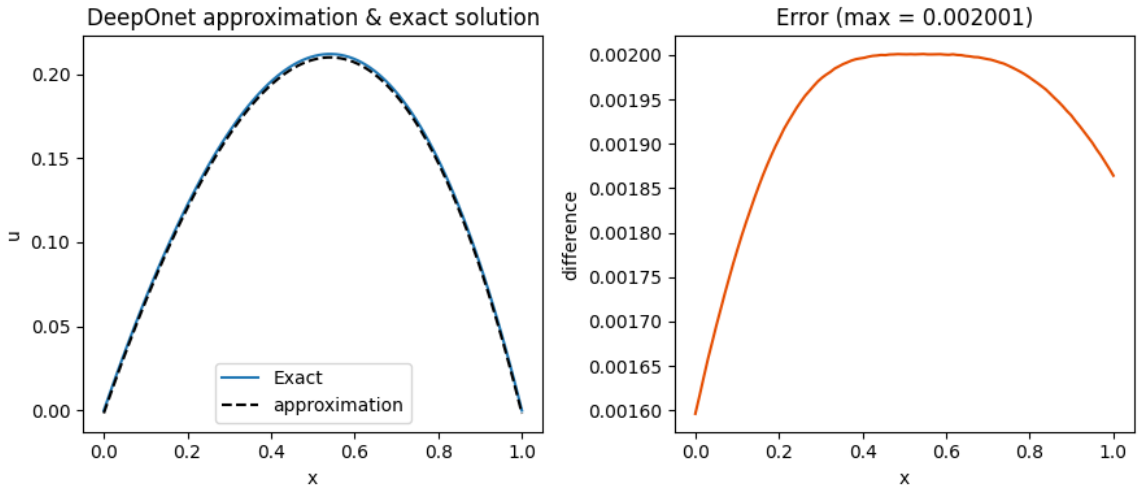

Figure 3: difference between the exact solution and the predicted solution for $f(x) = e^x$

CONCLUSION

In this work, we have presented the PI-DeepONet method, a novel approach for solving parametric partial differential equations (PDEs). Our method combines the power of physics-informed neural networks (PINNs) with deepONets, enabling accurate and efficient solutions to PDEs with high-dimensional parameter spaces.

Through extensive experiments, we have demonstrated the effectiveness of the PI-DeepONet method in predicting the solutions of various parametric PDEs. Our model exhibits remarkable accuracy, even when tested on functions outside the training space, indicating its ability to generalize well. The results obtained highlight the potential of the PI-DeepONet method as a powerful tool for solving complex PDEs.

Looking ahead, future research will focus on extending the PI-DeepONet method to higher dimensions and more intricate PDEs. This will involve exploring advanced architectures and training strategies to handle the increased complexity. Additionally, efforts will be directed towards enhancing the interpretability of the model and investigating its robustness under different scenarios.

In conclusion, the PI-DeepONet method represents a significant advancement in the field of PDE solving. Its ability to accurately predict solutions and generalize to unseen functions opens up new possibilities for tackling challenging problems in various scientific and engineering domains.

References

[1] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *CoRR*, abs/2103.10974, 2021.