Bachelor's Thesis Project Report 7th Semester

Keshav Saxena (2101MC57) and Shantanu Punde (2101MC53)
Supervised by Dr. Kuldip Singh Patel

1st December 2024

1 Problem Statement

Solving partial differential equations (PDEs) is a critical task in many fields of science and engineering.

In this context, DeepONet, a deep learning-based framework, has emerged as a powerful tool to solve PDEs. DeepONet leverages the universal approximation theorem, which ensures that deep neural networks can approximate any continuous operator. This makes DeepONet an ideal candidate for solving PDEs, as it can learn mappings between infinite-dimensional function spaces. By training on data generated from PDEs, DeepONet learns the underlying operator and provides an efficient solution to complex PDE problems, particularly those that are parametric in nature.

2 DeepONet: Deep Operator Network

DeepONet is structured as a two-branch architecture, where one branch processes the input function (such as boundary or initial conditions), and the other branch processes the spatial coordinates (such as positions in space and time). These two branches combine their outputs to generate the desired solution to the PDE. This architecture is especially well-suited for solving parametric PDEs, where the problem depends on varying parameters (e.g., material properties or initial conditions). By learning a mapping from these parameters to the solution, DeepONet can efficiently predict solutions for new parameter values without the need to re-solve the entire problem from scratch.

One of the key advantages of DeepONet over traditional numerical methods is its ability to handle high-dimensional and complex problems. In classical numerical approaches, solving PDEs often requires discretization of the domain and solving large systems of equations, which becomes computationally expensive as the number of dimensions increases. DeepONet, on the other hand, bypasses this need for discretization, offering a more scalable and computationally efficient approach, particularly for high-dimensional or time-dependent problems.

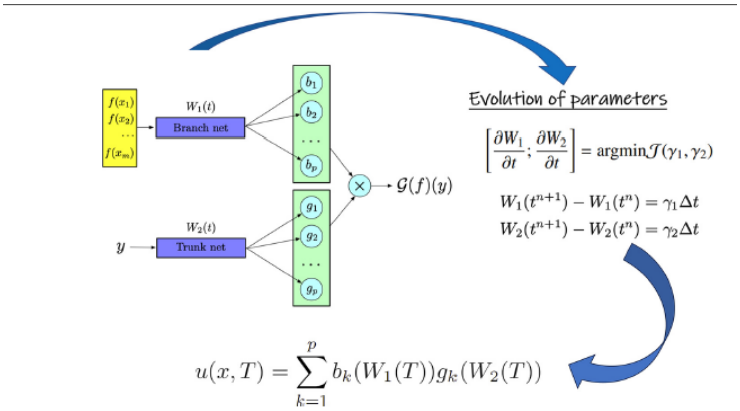

Figure 1: How DeepONet works

In addition to its efficiency, DeepONet has demonstrated generalization capabilities, meaning that once it is trained on a sufficient dataset, it can generalize well to new and unseen problems. This makes it ideal for real-time simulations, uncertainty quantification, and optimization tasks. DeepONet has been applied to various complex problems, including the solution of nonlinear PDEs, inverse problems, and multi-physics problems, demonstrating its potential in both research and practical applications.

3 Past Work

Paper Name	**Authors**	**Year**	**Purpose**
Neural operator: learning maps between function spaces	N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, A. Anandkumar	2021	Introduces the DeepONet framework for learning maps between function spaces, leveraging deep neural networks for operator learning.
Neural operator: graph kernel network for partial differential equations	Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar	2020	Proposes a graph kernel network approach for solving partial differential equations using neural operators.
Fourier neural operator for parametric partial differential equations	Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar	2020	Introduces the Fourier neural operator (FNO) for solving parametric PDEs with high efficiency.
Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators	L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis	2021	Demonstrates that DeepONet can approximate nonlinear operators and solve parametric problems, leveraging the universal approximation theorem.

Table 1: Summary of Papers on DeepONet

4 Heat Equation

4.1 PDE, Boundary Conditions, Spatial and Temporal Domain

We begin by considering the simple heat equation with various initial conditions for which we already possess exact solutions. A one-dimensional heat equation system can be described by the following partial differential equation (PDE):

$$u_t = u_{xx}, \quad u(x,0) = u_0, \quad u(0,t) = u(2,t) = 0.$$

Using the method of separation of variables, we can derive the solution to the heat equation. If we set the initial condition $u_0(x) = a\sin(\pi x)$, the solution is given by:

$$u(x,t) = a\sin(\pi x)e^{-\pi^2 t}, \quad a \in [1,2].$$

where u_x represents the spatial derivative of u, and Δx is the grid spacing.

4.2 Data Generation and Model Training

To generate initial data samples, we select 51 points uniformly from the interval $[0,2]$ for x and 50 random values of a from the range $[1,2]$. For each value of a, we use the following procedure:

- We define the initial condition as $u_0(x) = a\sin(\pi x)$ where a is a value randomly chosen from $[1,2]$.
- The time-dependent PDE is solved using DeepONet, with the time grid and the spatial domain chosen accordingly.
- We use 4000 points in the domain and 400 points for boundary conditions, with 50 initial conditions randomly chosen.

The neural network used in the training process is a Feed-Forward Neural Network (FNN) with 3 hidden layers, each consisting of 75 neurons, and the activation function is set to `tanh`. The network is compiled with the Adam optimizer and a learning rate of 1×10^{-3}, and trained for 10,000 iterations.

The model is trained for the following values of a:

$$a = 1.0, 1.5, 1.8, 2.5.$$

Although $a = 2.5$ is outside the range of the training data, the model performs well even for this value.

4.3 Graphs Comparison

We compare the solutions for different values of a where u is the solution obtained by EDE-DeepONet, and $\hat{u}$ is the reference solution. We calculate the MSE for different values of a and present the results in Table 1.

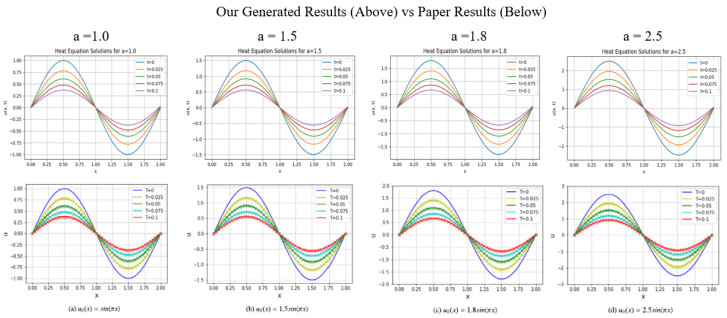

Figure 2: Comparison of solutions obtained by EDE-DeepONet for different values of a.

4.4 Error Matrix Comparison

The performance of the model is evaluated based on the mean squared error (MSE) defined as:

$$e(u, \hat{u}) = \frac{1}{N} \sum_{k=1}^{N} (u(x_k) - \hat{u}(x_k))^2,$$

a	**T = 0.025**	**T = 0.05**	**T = 0.075**	**T = 0.1**
1.0	3.2×10^{-5}	2.2×10^{-5}	1.4×10^{-5}	7.0×10^{-6}
1.5	8.9×10^{-5}	1.3×10^{-5}	8.0×10^{-6}	3.4×10^{-5}
1.8	7.8×10^{-5}	2.1×10^{-5}	1.8×10^{-5}	3.1×10^{-5}
2.5	2.9×10^{-5}	1.11×10^{-4}	6.3×10^{-5}	2.9×10^{-5}

Table 2: Error Matrix (DeepONet, our simulated)

a	**T = 0.025**	**T = 0.05**	**T = 0.075**	**T = 0.1**
1.0	1.47×10^{-5}	1.33×10^{-5}	1.32×10^{-5}	1.29×10^{-5}
1.5	5.11×10^{-6}	7.05×10^{-6}	7.40×10^{-6}	9.10×10^{-6}
1.8	1.46×10^{-5}	1.69×10^{-5}	1.79×10^{-5}	1.85×10^{-5}
2.5	2.20×10^{-4}	1.34×10^{-4}	6.02×10^{-5}	1.72×10^{-5}

Table 3: Error Matrix (from research paper)

5 Parametric Heat Equation

5.1 PDE, Boundary Conditions, Spatial and Temporal Domain

We now consider the one-dimensional parametric heat equation, which can be described by the following partial differential equation (PDE):

$$u_t = cu_{xx}, \quad u(x,0) = \sin(\pi x), \quad u(0,t) = u(2,t) = 0,$$

where c is a constant parameter and u_x represents the spatial derivative of u. The solution to the PDE depends on the value of the parameter c, and the boundary conditions are set at $x = 0$ and $x = 2$.

5.2 Data Generation and Model Training

To generate initial data samples, we select 51 points uniformly from the interval $[0, 2]$ for x and 50 random values of c from the range $[1, 2]$. For each value of c, we use the following procedure:

- We define the initial condition as $u_0(x) = \sin(\pi x)$, which represents the initial temperature distribution.
- The time-dependent PDE is solved using DeepONet, with the time grid and the spatial domain chosen accordingly.
- We use 4000 points in the domain and 400 points for boundary conditions, with 50 initial conditions randomly chosen.

The neural network used in the training process is a Feed-Forward Neural Network (FNN) with 4 hidden layers, each consisting of 50 neurons, and the activation function is set to `tanh`. The network is compiled with the Adam optimizer and a learning rate of 1×10^{-3}, and trained for 10,000 iterations.

The model is trained for the following values of c:

$$c = 1.2, 1.5, 1.8, 2.5.$$

Although $c = 2.5$ is outside the range of the training data, the model performs well even for this value.

5.3 Graphs Comparison

We compare the solutions for different values of c where u is the solution obtained by EDE-DeepONet, and $\hat{u}$ is the reference solution.

5.4 Error Matrix Comparison

The performance of the model is evaluated based on the mean squared error (MSE).

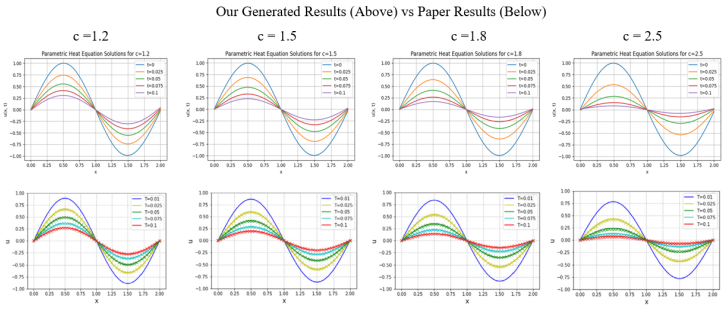

Figure 3: Comparison of solutions obtained by EDE-DeepONet for different values of c.

c	**T = 0**	**T = 0.025**	**T = 0.05**	**T = 0.1**
1.2	1.2×10^{-4}	2.3×10^{-5}	2.3×10^{-5}	3.1×10^{-4}
1.5	6.6×10^{-5}	9.3×10^{-6}	2.3×10^{-6}	4.4×10^{-5}
1.8	6.9×10^{-6}	5.5×10^{-5}	9.0×10^{-6}	2.4×10^{-4}
2.5	1.9×10^{-4}	3.9×10^{-5}	4.0×10^{-6}	6.6×10^{-6}

Table 4: Error Matrix (DeepONet, our simulated)

c	**T = 0.025**	**T = 0.05**	**T = 0.075**	**T = 0.1**
1.2	1.30×10^{-5}	1.43×10^{-5}	1.35×10^{-5}	1.20×10^{-5}
1.5	1.35×10^{-5}	1.27×10^{-5}	1.35×10^{-5}	1.80×10^{-6}
1.8	1.17×10^{-5}	1.03×10^{-5}	7.88×10^{-5}	7.08×10^{-6}
2.5	2.20×10^{-4}	1.34×10^{-4}	6.02×10^{-5}	7.08×10^{-6}

Table 5: Error Matrix (from research paper)

6 Advection Diffusion Equation

6.1 PDE, Boundary Conditions, Spatial and Temporal Domain

$$\frac{\partial u}{\partial t} + b\frac{\partial u}{\partial x} - a\frac{\partial^2 u}{\partial x^2} = 0$$

where: - $u(x,t)$ is the concentration or value of the substance at position x and time t,
- a is the diffusion coefficient (describing how the substance spreads)
- b is the convection (or advection) coefficient (describing the speed of movement in the x-direction).

The problem is defined on a spatial domain $x \in [0, L]$ and a temporal domain $t \in [0, T]$. In this case, we consider the boundary conditions:

$$u(0,t) = 0, \quad u(L,t) = 0, \quad u(x,0) = 3\sin(4\pi x)$$

where $u(x, 0)$ represents the initial condition of the substance's distribution. The boundaries are fixed at zero, which means that at $x = 0$ and $x = L$, the concentration is always zero for all times t.

The spatial domain is given as $x \in [0, 1]$, and the temporal domain extends from $t = 0$ to $t = 1$. The PDE is solved numerically using a physics-informed neural network (PINN) approach, which allows solving for the solution $u(x, t)$ over both time and space.

6.2 Graphs Comparison

The results of the Advection-Diffusion equation are compared between the analytical solution and the predicted solution obtained from a neural network model trained on the physical problem.

The analytical solution can be expressed as a series of terms that incorporate both convection and diffusion effects. It is derived using techniques such as separation of variables and Fourier series expansion. The solution is plotted as a 3D surface over the spatial and temporal domains to visualize how the concentration evolves over time.

In comparison, the PINN-based model solves the same equation by approximating the solution directly from the underlying PDE and initial/boundary conditions. The trained neural network model provides a numerical solution to $u(x, t)$, and the comparison between the predicted solution and the exact analytical solution can be plotted in a similar manner.

Both solutions are compared visually through 3D surface plots. These plots show the concentration $u(x, t)$ as a function of both space and time, highlighting how the substance propagates through the medium and how well the neural network model approximates the analytical solution.

subcaption

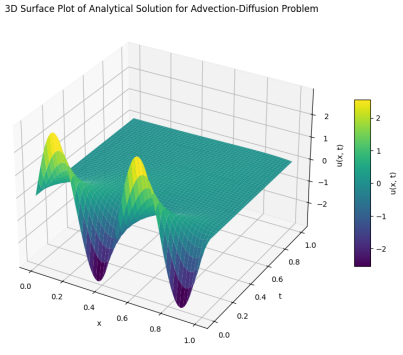

Figure 4: 3D Surface Plot of Analytical Solution for the Advection-Diffusion Equation.

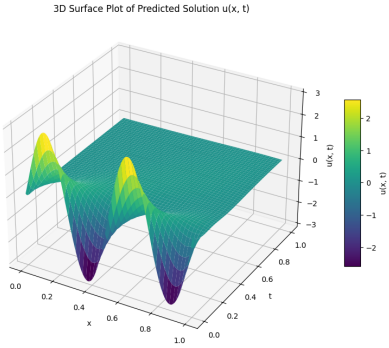

Figure 5: 3D Surface Plot of Predicted Solution using DeepXDE.

6.3 Error Matrix Comparison

The error matrix compares the predicted solution from the neural network with the analytical solution. The error is computed as the absolute difference between the predicted and exact values at each grid point across both space and time.

To visualize the error, a 3D surface plot is generated, showing the magnitude of the error between the two solutions. This provides insight into the accuracy of the neural network model and how closely it approximates the true solution. The error matrix comparison highlights regions where the model performs well and regions where there may be deviations, indicating areas where further model refinement or adjustments may be needed.

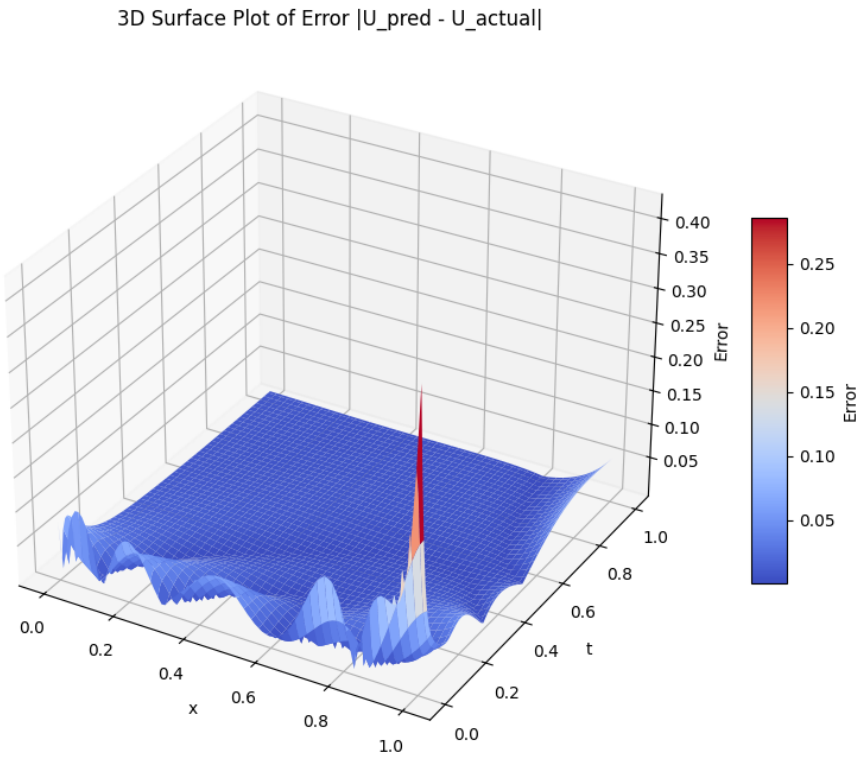

Figure 6: 3D Surface Plot of Error between the Analytical and Predicted Solutions.

7 Convection Diffusion Equation in 1D

7.1 PDE, Boundary Conditions, Spatial and Temporal Domain

We consider the one-dimensional convection-diffusion equation, defined as:

$$u_t + u_x - \frac{1}{1000} u_{xx} - 0.5 = 0$$

where: - u_t is the time derivative of u, - u_x and u_{xx} are the first and second spatial derivatives of u, - The equation models convection with a small diffusion term and a constant source of -0.5.

The spatial domain is $x \in [0, 1]$ and the temporal domain is $t \in [0, 0.25]$.

The initial condition is:

$$u(x, 0) = \begin{cases} 8(x - 0.25), & 0.25 \leq x < 0.375, \\ 8(0.5 - x), & 0.375 \leq x \leq 0.5, \\ 0, & \text{otherwise.} \end{cases}$$

The boundary conditions are:

$$u(0, t) = 0, \quad u(1, t) = 0$$

The neural network used in the training process is a Feed-Forward Neural Network (FNN) with 3 hidden layers, each consisting of 75 neurons, and the activation function is set to `tanh`. The network is compiled with the Adam optimizer and a learning rate of 1×10^{-3}, and trained for 10,000 iterations.

7.2 Graphs and Error Comparison

Unfortunately we did not have the exact solution for the equation which would act as a reliable measure for testing the accuracy for our method hence this work was left incomplete and we explored other equations.

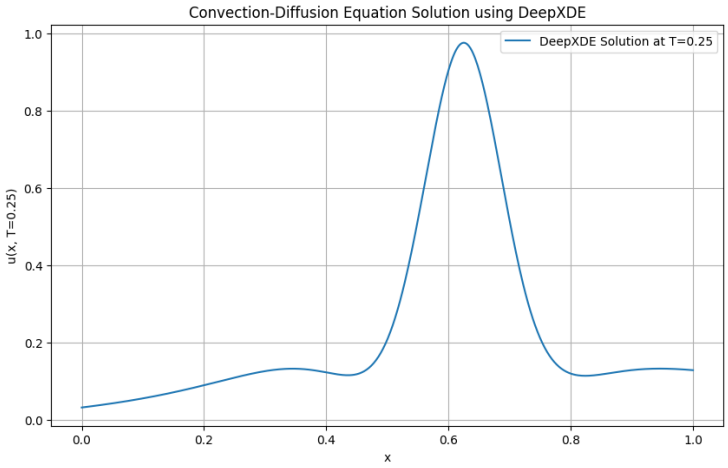

Figure 7: Predicted solution of the Convection-Diffusion Equation at $t = 0.25$

8 Further Work to be Pursued

Target 1: 2D Parabolic Reaction-Diffusion PDE

Solve the following PDE:

$$u_t(x, y, \epsilon, t) - \epsilon \Delta u(x, y, \epsilon, t) + (2 + x + y)u(x, y, \epsilon, t) = f(x, y, t), \tag{1}$$

where $u(x, y, \epsilon, t) \in \Omega \times (0, T]$, and the function $f(x, y, t)$ is defined as:

$$f(x, y, t) = 1 - 2\exp(-t)h(x)h(y) + \exp(-t)(h(x) + h(y)) - 1. \tag{2}$$

The function $h(x)$ is given by:

$$h(x) = \frac{\exp\left(\frac{T}{\epsilon} - \frac{x}{\epsilon}\right) + \exp\left(\frac{T}{\epsilon} - \frac{1-x}{\epsilon}\right)}{1 + \exp\left(\frac{T}{\epsilon} - \frac{1}{\epsilon}\right)}. \tag{3}$$

Initial and Boundary Conditions:

$$u(x, y, \epsilon, 0) = 0, \tag{4}$$

$$u(x, y, \epsilon, t) = 0 \quad \text{on the boundary of } \Omega. \tag{5}$$

Target 2: Black-Scholes PDE for Barrier Call Options

Solve the Black-Scholes PDE:

$$\frac{\partial u}{\partial t}(S, t) - \frac{\sigma^2 S^2}{2}\frac{\partial^2 u}{\partial S^2}(S, t) - rS\frac{\partial u}{\partial S}(S, t) + ru = 0, \tag{6}$$

where $(S, t) \in (S_{\min}, S_{\max}) \times (0, T]$.

Initial and Boundary Conditions:

$$u(S,0) = \max(S-K,0), \tag{7}$$

$$u(S_b,t) = 0, \tag{8}$$

$$u(S_{\max},t) = 0. \tag{9}$$