Deep Neural Network Performance in Solving the Helmholtz Dirichlet Problem

Konstantine Butbaia

Free University of Tbilisi

Abstract

In this paper we will present our ongoing work and progress where we explore the performance and challenges of various deep neural networks in learning solutions to the Dirichlet problem for the Helmholtz equation. We consider a simple problem defined on a region $[0,1] \times [0,1]$. We conduct numerical experiments on these neural networks and qualitatively compare their convergence to the true solution with the number of epochs needed to achieve it. As this is an ongoing work, lastly I will present future goals of this paper in the conclusion.

1 Introduction

The Helmholtz equation is a time-independent form of the wave equation. Solutions to the Helmholtz equation represent the spatial distribution and behavior of waves. This equation arises in various fields of physics, such as electromagnetism, acoustics, and quantum mechanics. The Dirichlet problem for the Helmholtz equation is a fundamental problem involving specific boundary conditions and is essential in modeling various physics phenomena. In recent years, there has been a growing interest in using artificial neural networks to solve complex physics problems like the Helmholtz equation BVPs. Neural networks have shown great potential in learning solutions to partial differential equations that govern physical phenomena. Through techniques like automatic differentiation, neural networks can be trained to minimize errors in the governing equation, making them a promising tool for solving PDEs. Such a framework is referred to as Physics Informed Neural Networks (PINNs). PINNs have been shown to be effective in learning solutions to Helmholtz equation for the problem of modeling seismic wave propagation [1]. Unlike FD and FE methods, which require discretizing the problem domain into a grid or mesh, which can be computationally expensive and challenging to implement for complex boundary shapes, PINNs do not require such discretization. Instead, PINNs directly learn the solution over the entire domain. Additionally, PINNs are flexible because learning the solution to a different PDE only requires modifying the loss function. PINNs have also been successfully used to tackle inverse problems, a class of problems in which the goal is to determine the parameters of a system based on observed data. For example in [9], They tackle inverse scattering problem in photonic metamaterials and nano-optics technologies using PINNs, where they determine the permittivity $\epsilon(x, y)$ by minimizing the loss function derived from the differential equation $\Delta E + k^2 \epsilon(x, y) E = 0$.

While PINNs offer several advantages over traditional numerical methods like FD and FE, they also come with their own set of challenges. One drawback of PINNs is their high computational cost and long training times, especially for problems with complex boundary conditions. Another challenge that arises when dealing with PINNs is the choice of the hyperparameters, activation function and the network architecture as the convergence of the model is highly dependent on it. An improper choice of hyperparameters can lead to poor convergence or cause the model to get stuck in a local minimum. The purpose

of the following sections is to explore the convergence of the model by using various methods and neural network architectures to speed up the learning process, in the context of solving the Dirichlet problem for the Helmholtz equation. Such a task of finding the optimal design of a neural network, balancing the representational power required and the computational resources used, is called neural network search. For example the paper [5], tackles the problem of finding the optimal design choice for the neural network by introducing a method that uses neural architecture search to create emulators, speeding up scientific simulations. Another challenge with PINNs is designing an appropriate loss function that ensures the model converges to the global minimum. The error function in PINNs is typically a composite loss function that combines multiple terms, including the residuals of the governing partial differential equations (PDEs), boundary conditions, and initial conditions. One of the main challenges is balancing different components of the error function to ensure that each term contributes appropriately to the learning process. Such a problem is usually tackled by introducing weighing coefficients for the terms in the loss function to balance the error terms in the training. Hyperparameter tuning is necessary for selecting appropriate values for these coefficients. However, no straightforward method exists for this process, and manually searching for values is often inefficient. In [3], This problem is addressed by simultaneously training both the weights of the neural network and the coefficients. The idea behind training these coefficients is to increase their values as the corresponding losses increase, Therefore prioritizing terms with larger losses. [6] Addresses this problem by also introducing an adaptive algorithm that adjusts the weights of different loss terms based on gradient magnitudes, balancing their contributions. Convergence of the model also depends on the initialization of the weights in the neural network and the choice of the activation function. For our purposes of solving the Helmholtz equation, which involves representing a solution with oscillatory behavior, a periodic activation function is more useful [7]. Sinusoidal types of activation functions introduce a non-linearity that allows the network to learn periodic patterns. While other non-linear functions like ReLU, sigmoid and tanh are common, they don't capture periodic behavior. Using a periodic function allows the network to have a broader range of representable functions, including those that need to oscillate. The initialization of the weights is an important step in training a neural network because it influences how effectively the model converges to the global minimum during training. A common way of initializing the weights in a neural network is Xavier initialization [8], which is a method of setting the initial weights of a neural network by taking them from a uniform distribution $w_i \sim \mathcal{U}(-a, a)$. Such initialization ensures that the weights are set up in a way that keeps the variance of activations and gradients consistent across layers, which helps prevent the vanishing or exploding gradient problems.

In the following sections, I will first describe the specific Dirichlet problem being considered. Then, I will explain the neural network architectures and training processes used, followed by the results obtained from training these networks to solve the problem.

2 Problem and Numerical Experiments

In the following sections the performance of the PINNs method is explored with various different architectures and error functions on the interior 2D Dirichlet problem for the Helmholtz equation defined as:

$$\begin{aligned} &u_{xx}(\mathbf{x}) + u_{yy}(\mathbf{x}) + k^2 u(\mathbf{x}) = f(\mathbf{x}), \quad \mathbf{x} \in \Omega \\ &u(\mathbf{x}) = g(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega \end{aligned} \tag{1}$$

Where the region $\Omega = [0, 1] \times [0, 1]$, the function g represents the boundary condition on the edges of the square and the function f represents the source function. We will consider either a $f(\mathbf{x}) = 0$ or a gaussian source function positioned at the center of the square and defined with 2 coefficients A and σ:

$$f(\mathbf{x}) = Ae^{-\frac{(x-0.5)^2+(y-0.5)^2}{\sigma}}$$

We would like the neural network to learn the solution $u(\mathbf{x})$ of the problem (1), Therefore the inputs to the neural network are coordinates $\mathbf{x} = (x, y)$ and the output is the prediction of the neural network of the value of the solution at that point $\hat{u}(\mathbf{x})$. We use a fully connected network and discuss results with different number of nodes in hidden layers. In all cases we employ the Adam optimizer for the training process.

2.1 PINNs Solution

For training the Physics-Informed Neural Networks, we need to select a set of input points within the domain $[0, 1] \times [0, 1]$ that will be fed into the neural network. These points are where the neural network will learn the underlying solution to the problem, ensuring that the network can approximate the solution accurately across the entire region. The choice of input points can significantly affect the training process, as the distribution and density of points influence the network's ability to capture key features of the solution. There may be regions where the solution's behavior is more complex or where higher accuracy is needed, requiring more training points in that region. This problem is addressed in [4] by adjusting the distribution of the training points depending on the residuals, Therefore focusing on areas which require more accuracy. For simplicity, We will choose a grid of points as presented in the Fig 1.

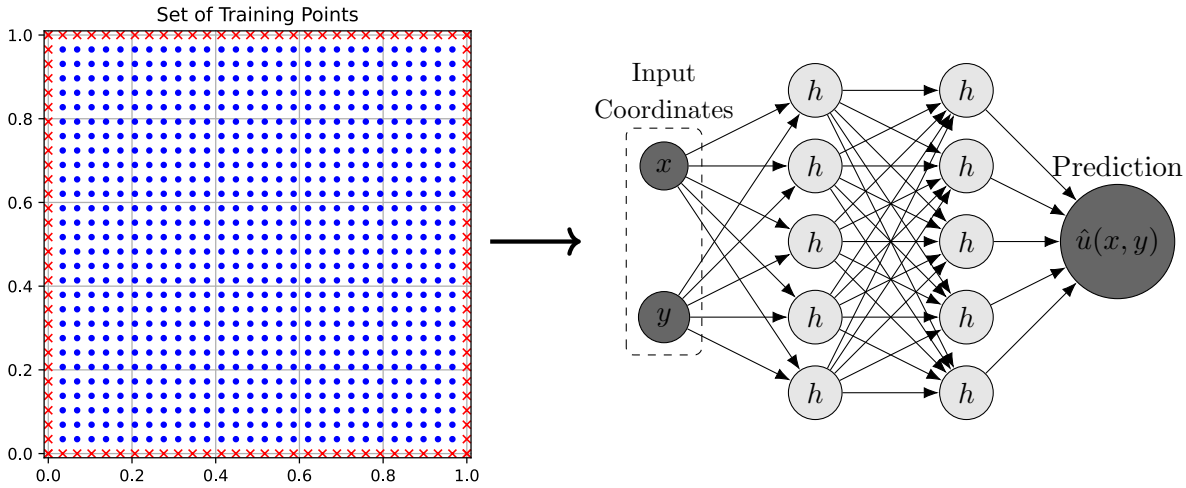
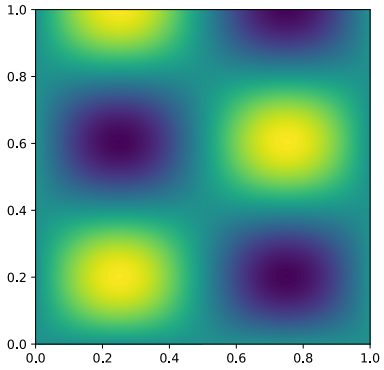


Figure 1: Equally spaced grid of points on $[0, 1] \times [0, 1]$ used in the process of training PINNs for the solution to the dirichlet problem.

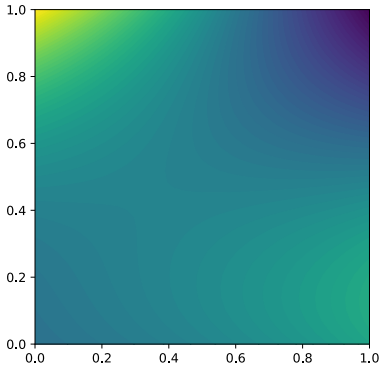
2.2 Sine Edge Problem

First, we examine a simple case of Problem (1), where the source function is $f(\mathbf{x}) = 0$ and the boundary conditions for the edges of $[0, 1] \times [0, 1]$ are chosen as a sine function on the top edge of the square, with zeros on the remaining edges. We take boundary condition $g(\mathbf{x})$ as:

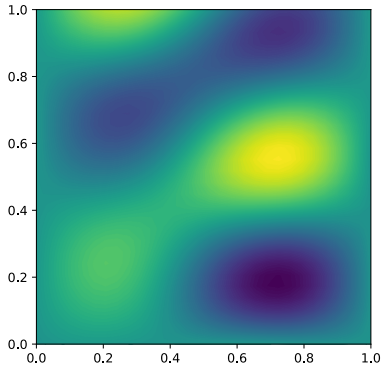
$$\begin{aligned} g(x, 0) &= 0, \quad g(x, 1) = \sin(2\pi x) \\ g(0, y) &= 0, \quad g(1, y) = 0 \end{aligned} \tag{2}$$



(a) Ground Truth



(b) PINNs result without weighing error terms



(c) PINNs result with weighing error terms

Figure 2: The graphs above show results for a neural network with 6 hidden layers, each containing 50 nodes. The training was performed using the Adam optimizer over 6000 epochs, with $k = 10$ and Xavier Initialization. The ground truth graph is constructed from the analytical solution to the problem (1) in (a). In (b) we train the neural network without weighing coefficients which is $\lambda_b = \lambda_{pde} = 1$. In (c) we chose $\lambda_b = 100$ and $\lambda_{pde} = 1$. The code for PyTorch implementation of this neural network can be found on github link in the conclusion section.

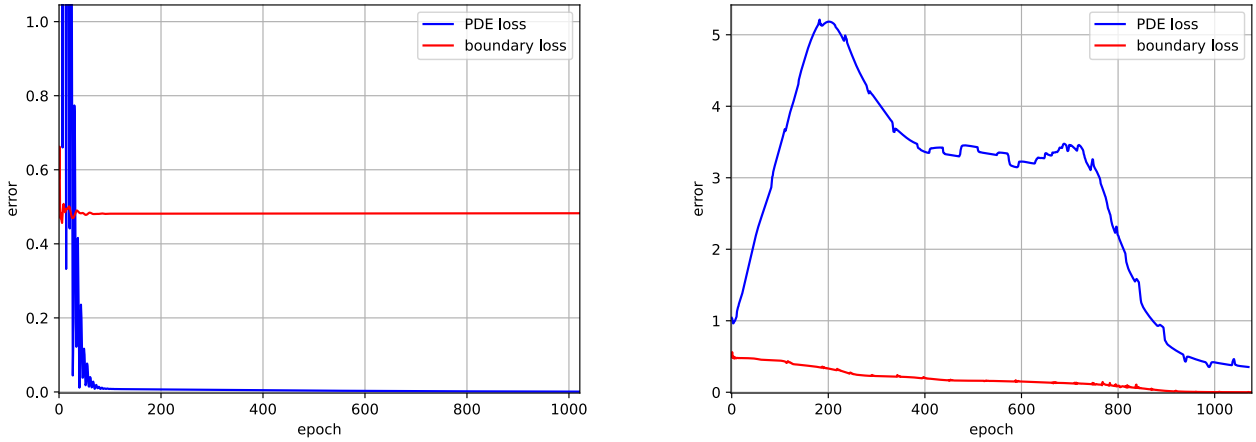
To train the neural network, we define the error function as the sum of the residuals E_{pde} from the differential equation and E_b from the boundary condition, along with their respective weighting coefficients λ_{pde} and λ_b

$$E_{pde} = \frac{1}{N}\sum_{i=1}^{N}\left[\hat{u}_{xx}(\mathbf{x}_i) + \hat{u}_{yy}(\mathbf{x}_i) + k^2\hat{u}(\mathbf{x}_i) - f(\mathbf{x}_i)\right]^2$$

$$E_b = \frac{1}{N_b}\sum_{i=1}^{N_b}\left[\hat{u}(\mathbf{x}_i) - g(\mathbf{x}_i)\right]^2$$

Where N is the total number of training points and N_b is the number of points on the boundaries. Then we will define the error function as $E = \lambda_{pde}E_{pde} + \lambda_b E_b$.

The choice of the weighing coefficients for the error terms $\lambda_{pde} = 1$ and $\lambda_b = 100$ in Fig. 2(c) were chosen manually by observing the convergence of the error function terms to zero during training. This highlights the importance of carefully selecting the weighing coefficients, as the convergence of the PINNs model to the true solution depends on them. This also emphasizes the fact that minimizing the error function does not necessarily guarantee convergence to the true solution of the BVP. This observation can be seen on the error graphs in Fig. 3. It can be observed in Fig. 3(a) that when no coefficients are used, the training primarily focuses on minimizing the loss from the differential equation, while the boundary condition loss is largely ignored and remains almost constant throughout the training. As a result, no significant progress is made toward learning the actual ground truth solution. However, when coefficients are applied to the loss terms, with $\lambda_{pde} = 1$ and $\lambda_b = 100$, the contribution of the boundary condition loss becomes more significant. As shown in Fig. 3(b), the training process minimizes both the boundary loss and the PDE loss terms, rather than focusing on just one.



(a) Without weighing error terms

(b) With weighing error terms

Figure 3: The graphs above show error values E_{pde} and E_b for every 5 epochs during training with Adam optimizer over 6000 epochs in total. In (a) we don't use weighing of these error terms, in (b) we have $\lambda_{pde} = 1$ and $\lambda_b = 100$

When using error functions in the context of Physics-Informed Neural Networks that include multiple terms and need to satisfy several conditions in addition to the PDE, it can be challenging to balance these terms. This imbalance may interfere with the learning process, potentially causing the model not to converge to the true solution and leading to an inaccurate approximation. One way to tackle this challenge is by embedding the boundary condition information in the neural network's output and only training the network to satisfy the differential equation. This approach eliminates the need to select appropriate values for the coefficients of multiple error terms, as there is only a single error term, which is the PDE error.

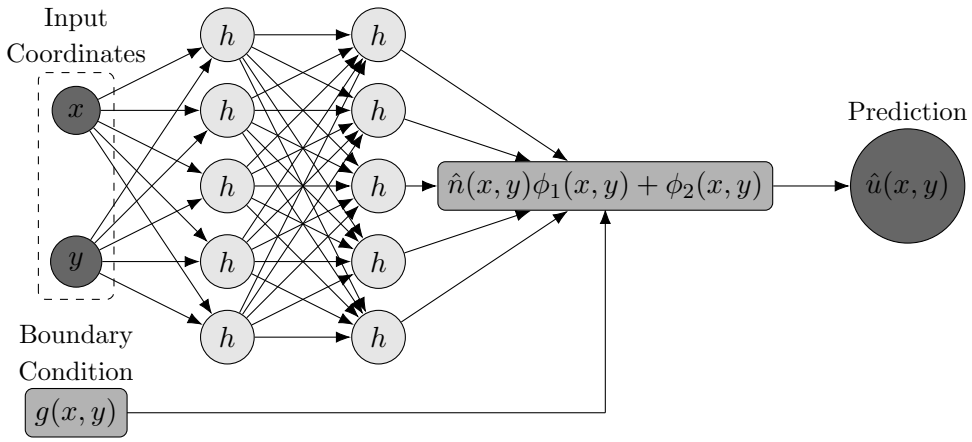
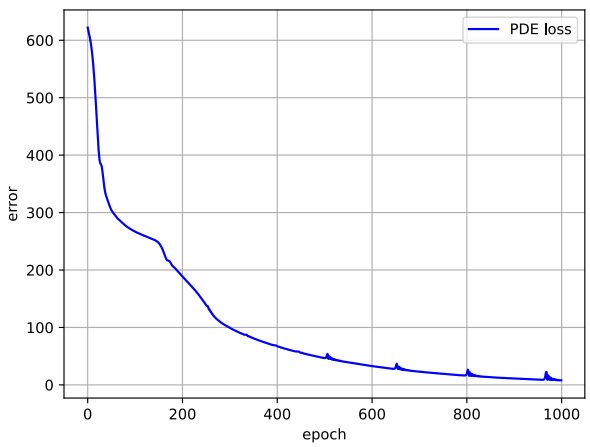
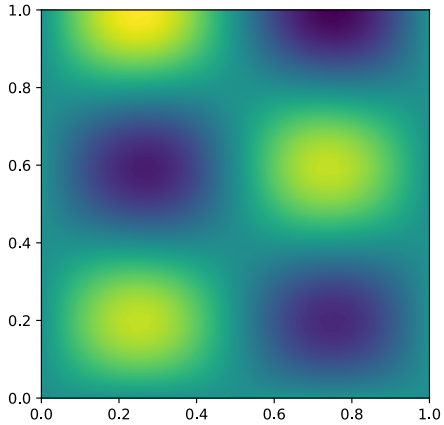


Figure 4: The graph of the neural network which incorporates boundary condition information at the output of the hidden layers. Therefore, the boundary conditions are inherently satisfied and the learning process is solely focused on satisfying the differential equation. Here $\hat{n}(x, y)$ is the output of the neural network itself and ϕ_1 and ϕ_2 are functions which are determined from the boundary condition information $g(x, y)$



(a) PDE error over epochs



(b) Learned solution $\hat{u}$

Figure 5: Results of training a neural network described above on the problem (1) with boundary conditions (2). Here we also have $k = 10$ and training was done with 1000 epochs of the Adam optimizer. The result of the learned solution is on the graph (b) and (a) is the loss evolution over epochs during training.

This idea is expressed in the graph illustrated in Fig. 4. The objective is to incorporate the boundary condition information into the output of the neural network in such a manner that the boundary conditions are automatically satisfied for any input point (x, y) within the network. Thus, the error function is now represented as E_{pde} and the training process focuses solely on minimizing the PDE-related error in the network's weights. To accomplish this, we introduce two differentiable functions, ϕ_1 and ϕ_2, which will be applied to the output of the neural network in a manner that ensures the boundary conditions are consistently satisfied. Therefore we define the output of the PINNs as:

$$\hat{u}(x, y) = \hat{n}(x, y)\phi_1(x, y) + \phi_2(x, y) \quad \text{such that} \quad \forall \hat{n}\ \forall (x, y) \in \partial\Omega : \hat{n}(x, y)\phi_1(x, y) + \phi_2(x, y) = g(x, y)$$

For our problem with the boundary conditions defined in (2). We can choose ϕ_1 and ϕ_2 the following way:

$$\hat{u}(x, y) = xy(x - 1)(y - 1)\hat{n}(x, y) + y \sin(2\pi x)$$

The above expression can be checked that it satisfies the boundary conditions presented in (2) for any function $\hat{n}$. Since the boundary condition information is now incorporated into the neural network, The learning process becomes more efficient and the model conveges to the true solution in less epochs. This can be observed on Fig. 5. Incorporating boundary condition information directly into the neural network training process not only improves efficiency by reducing the number of training epochs but also leads to more accurate solutions, as an improper choice of error coefficients in the previous method could result in inaccurate results.

Rather than training the neural network to solve the Dirichlet problem at every point inside the computational domain, we can leverage the fundamental solution of the Helmholtz equation, denoted as $\phi(\mathbf{x})$. By doing so, we only need to train the neural network at the boundary points, where the boundary conditions are specified. This approach is presented in this paper [2]. This method greatly reduces the number of points where the network has to be trained by reducing the dimensionality of the problem by one. But as we'll see, this approach comes with its own challenges regarding computation time, as it requires numerically calculating the value of an integral, which is costly.

In our case for solving the dirichlet problem (1) with boundary conditions (2), We can express it's solution $u(\mathbf{x})$ using the double layer potential along with it's jump condition on the boundaries. The neural network is trained to learn the density function h which is present in the double layer potential. So the solution on region Ω can be expressed as:

$$\forall x \in \Omega : u[h](x) = \int_{\partial\Omega} h(y) \frac{\partial \phi(x,y)}{\partial n_y} ds_y$$
$$\forall x_0 \in \partial\Omega : \lim_{x \in \Omega \to x_0} u[h](x) = \frac{1}{2} h(x_0) + u[h](x_0)$$

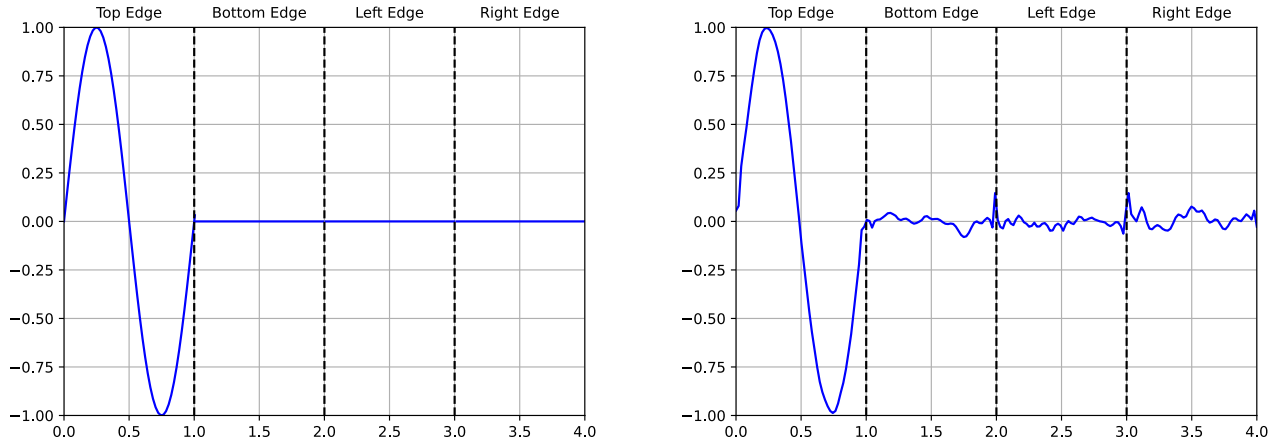
And we train the neural network on the boundary points of the region Ω to find such density function h which satisfies the boundary g for the jump condition at edges:

$$\forall x_0 \in \partial\Omega : u[h](x_0) + \frac{1}{2} h(x_0) = g(x_0)$$

Therefore our error function which we will use to train the neural network is defined by the above expression:

$$E = \frac{1}{N_b} \sum_{i=1}^{N_b} \left[\hat{u}[h](\mathbf{x}_i) + \frac{1}{2} h(\mathbf{x_i}) - g(\mathbf{x_i}) \right]^2$$

Where N_b is the number of points on the boundary and $\mathbf{x_i}$ are the points on the boundary. A more detailed description of the architecture of this neural network is provided in the paper [2]. Applying this approach to our problem means that we only need to train the neural network on the set of boundary points in one dimension. Therefore, we can present the results only at the boundary points Fig. 6. If the model correctly learns the boundary points, it indicates that the model has been trained to accurately approximate the solution to the Dirichlet problem. The disadvantages of this method include the computationally expensive requirement of calculating an integral during the forward pass through the network. Additionally, the method requires knowledge of the fundamental solution, denoted as ϕ, which may not always be accessible for certain problems.



(a) Boundary Condition $g(\mathbf{x})$

(b) BINet learned boundary $\hat{u}$

Figure 6: The graph above (b) shows the outcome of the learned solution for the boundary condition (a) of the BINet model after training for 5000 epochs.

3 Conclusion

In this paper, we have explored the PINNs method and its variations for solving the Helmholtz equation Dirichlet problem. We presented the results of numerical experiments and discussed various challenges encountered when using these methods. As this is ongoing work, our goals are to conduct a more thorough investigation of these methods and determine the optimal neural network architecture for specific problems. Additionally, we aim to explore problems with more complex boundary conditions and source functions, investigating the challenges that may arise and how to address them. Furthermore, we plan to perform similar analyses on various cases of inverse problems to better understand the challenges involved.

Code: github.com/kostabutbaia/pinn

References

[1] Umair Bin Waheed Chao Song, Tariq Alkhalifah. A versatile framework to solve the helmholtz equation using physics-informed neural networks. *Geophysical Journal International*, 2021.

[2] Fukai Chen Xiang Chen Junqing Chen Jun Wang Zuoqiang Shi Guochang Lin, Pipi Hu. Binet: Learning to solve partial differential equations with boundary integral networks. *arXiv:2110.00352*, 2021.

[3] Ulisses Braga-Neto Levi D. McClenny. Self-adaptive physics-informed neural networks using a soft attention mechanism. *arXiv:2009.04544*, 2020.

[4] Zhiping Mao George E. Karniadakis Lu Lu, Xuhui Meng. Deepxde: A deep learning library for solving differential equations. *arXiv:1907.04502*, 2019.

[5] L. Deaconu S. Oliver P. Hatfield D. H.Froula G. Gregori M. Jarvis S. Khatiwala J. Korenaga J. Topp-Mugglestone E. Viezzer S. M. Vinko M. F. Kasim, D. Watson-Parris. Building high accuracy emulators for scientific simulations with deep neural architecture search. *arXiv:2001.08055*, 2020.

[6] Paris Perdikaris Sifan Wang, Yujun Teng. Understanding and mitigating gradient pathologies in physics-informed neural networks. *arXiv:2001.04536*, 2020.

[7] Vincent Sitzmann. Implicit neural representations with periodic activation functions. *arXiv:2009.04544*, 2020.

[8] Yoshua Bengio Xavier Glorot. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of Machine Learning Research*, 2010.

[9] George Em Karniadakis Luca Dal Negro Yuyao Chen, Lu Lu. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *arXiv:1912.01085*, 2019.