Trabalho apresentado no XLIII CNMAC, Centro de Convenções do Armação Resort - Porto de Galinhas - PE, 2024

Convergence Analysis of Physics-Informed Neural Networks and Comparison with Finite Difference Methods of the Two-Dimensional Heat Equation

Vitor Bueno[1]
PPG-Ccomp/UERJ e PESC/UFRJ, Rio de Janeiro, RJ
Cristiane Faria[2] Karla Figueiredo[3]
IME, UERJ, Rio de Janeiro, RJ
Fabio dos Santos[4]
COPPE/UFRJ, Rio de Janeiro, RJ

Resumo. Partial differential equations (PDEs) are used in mathematics to describe physical phenomena. However, analytical solutions are often unavailable, requiring approximations. The physics-informed neural network (PINN) is a recent technique that combines deep neural networks with physical knowledge leveraging automatic differentiation techniques, to provide accurate approximations. This work analyzes the convergence of the PINN method, considering the discretizations and architecture of the networks. PINN approximations were compared with Finite Difference Methods (FDM) in the case of the two-dimensional heat equation. The experiments carried out suggested that there is an optimal number of points and architectural parameters to be used, which can lead to an improvement in the generalization and estimation error. It was possible to see an advantage over FDM as it approximates the solution of all the time steps at once, without propagating the error from one step to the next.

Palavras-chave. PINN, Convergence Analysis, Comparative Study, FDM, Two-Dimensional Heat Equation.

1 Introduction

Partial Differential Equations (PDE) are an essential area of mathematics used to describe physical and natural phenomena that involve continuous/discontinuous variations and changes in several independent or dependent variables. In most cases, analytical solutions are unavailable, so numerical methods are used to obtain approximate solutions for these complex systems.

Among all new proposals, approaches using machine learning (ML) methods are increasingly used in all research areas. In simple terms, the standard ML approach is to obtain knowledge of the problem at hand by extracting and integrating the pattern and behavior of previously acquired data to predict the outcome of new data. When analyzing these problems through this scope, it was noticed that limitations remain when using purely data-driven models in real-world applications. These arise as a need for robustness, interpretability, and adherence to physical constraints.

Researchers have proposed various methods to incorporate physical knowledge into machine learning, termed Physics-Informed Machine Learning. Deep neural networks became a natural

[1]vitorb@cos.ufrj.br
[2]cofaria@ime.uerj.br
[3]karlafigueiredo@ime.uerj.br
[4]fsantos@cos.ufrj.br

choice for handling PDE due to their universal function approximation property. While early attempts [1] used multilayer perceptrons, they lacked the accuracy of established numerical methods. The concept was later revitalized by Raissi, Perdikaris, and Karniadakis in 2019, leading to the Physically Informed Neural Networks (PINN) method [2]. The PINN method addresses data scarcity by leveraging physical properties and minimizing generalization errors in regions with limited data. The contribution of the PINN was a simple yet powerful way to implement prior physical knowledge in training these networks using developments in automatic differentiation [3] to provide solutions for various types of PDE.

This work aims to evaluate the approximation solution of the two-dimensional heat equation, from the perspective of the estimation and the approximation errors. Previous works on this topic by the authors can be found at [4], [5]. The estimation error is analyzed to understand how the approximation of the solution is affected by the discretization used to train the network. In contrast, the approximation error is analyzed to understand how the approximation of the solution is affected by the class of neural networks chosen. An optimal number of points and architectural parameters are found through the tests carried out. In addition, comparative studies of the accuracy of the optimum PINN approximation with some Finite Difference Methods (FDM) are shown, confirming the efficiency of the PINN.

2 PINN

Multilayer perceptron is the neural network used in this work to approximate the solution in the PINN framework. The set θ represents the trainable parameters (weights and bias), which are optimized through the minimization of a loss function $\mathcal{L}(\theta)$ in the training process by some gradient descend optimization technique. The gradients are calculated using the backpropagation algorithm; each update of the parameters in some epoch of training i is given by $\theta^i = \theta^{i-1} - \eta \bigtriangledown \mathcal{L}$, where η is the learning rate. The goal is to find $\theta^* = argmin_\theta \mathcal{L}(\theta)$.

The PINN approach considers the solution to a PDE to be in the form of a neural network $u_\theta(x)$; this network is used to calculate the approximate solution of a generic differential equation of the form:

$$\begin{cases} F(t, x, u_\theta(t, x); \gamma) & = & 0 & t \in [0, T],\ x \in \Omega \\ I(x, u_\theta(0, x)) & = & 0 & x \in \Omega \\ B(t, x, u_\theta(t, x)) & = & 0 & t \in [0, T],\ x \in \partial\Omega \end{cases} \tag{1}$$

In the equation (1), θ represents the network parameters, F is a general differential operator that can consist of temporal derivatives, spatial derivatives, linear and non-linear terms, x is the position vector in the domain Ω, with boundary $\partial\Omega$, I and B denote, respectively, the initial and boundary conditions and can also consist of differential, linear or non-linear operators.

PINN's incorporation of the prior physical knowledge is done using the property of automatic differentiation [3] to differentiate neural networks output with respect to it's input, calculating the residual of the differential equation and incorporating it as a soft constraint in the loss function being minimized during the training. That way the approximation adheres to the physical condition of the problem described by the same differential equation [2]. The main idea of automatic differentiation is that all numerical calculations are ultimately compositions of a finite set of elementary operations for which the derivatives are known; combining the derivatives of the constituent operations via the chain rule gives the derivative of the overall composition [3]. Expressing the entire domain during training is impossible, making discretizations necessary. Therefore, the implemented loss function is the equation 2 [2], [6], [7]. The value of each term is obtained as equation 2, where $\{0, x^i_{in}\}_{i=1}^{N_{ic}}$, $\{t^i_{bc} x^i_{bc}\}_{i=1}^{N_{bc}}$ and $\{t^i_r x^i_r\}_{i=1}^{N_r}$ represents respectively the set of points sampled in the initial condition, boundary condition and the collocation points, where the residual of the differential equation is calculated, each containing N_{ic},N_{bc} and N_r points respectively. For simplicity, x represents all the dimensions of the spatial domain, and t is the time domain.

$$L(\theta) = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} (I(x_{ic}^i, u_\theta(0, x_{ic}^i)))^2 + \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} (B(t_{bc}^i, x_{bc}^i, u_\theta(t_{bc}^i, x_{bc}^i)))^2 + \frac{1}{N_r} \sum_{i=1}^{N_r} (F(t_r^i, x_r^i, u_\theta(t_r^i, x_r^i)))^2 \tag{2}$$

The process of approximating the solution to the PDE is reduced to an optimization problem, where the initial and boundary conditions and the differential equation's residual can be seen as constraints.

Backpropagation is used to update neural network parameters during training; it calculates gradients of the loss function concerning the network parameters. A more general form of Automatic Differentiation in reverse mode also calculates the derivatives of the network output with respect to its inputs to obtain the residual of the differential equation. This approach ensures that physical knowledge is incorporated into the network, and it learns the underlying physics of the problem.

3 Methodology

A possible approach to understanding the error in the approximations to solutions is to divide it into three components, as illustrated in Figure 1. The objective is to examine the influence of estimation and approximation errors independently.

The estimation error is intrinsic to any approximation of a function defined in the continuous domain through a finite amount of data. The influence of this portion was analyzed using different discretizations with regular spacing to train the same neural network (same architecture and initial parameters) and obtain approximations in predefined points. The objective is to independently verify the influence of a finer discretization in each domain on the generalization capacity of the network.

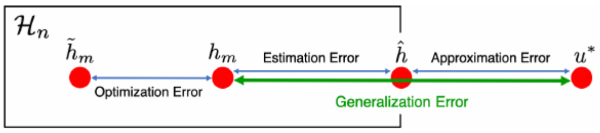


Figure 1: Illustration of the total errors from [8].

The approximation error aims to measure the expressiveness of the neural network used to solve the problem. This influence was analyzed using different architectural parameters of the network and verify their influence. The parameters tested were the number of neurons in the hidden layer and the number of hidden layers. Each modification was made independently, and the error was obtained by averaging five models compiled with initial parameters drawn from the same distribution to mitigate the randomness of parameter initialization.

PINN can generate an approximation of the solution at any point in the domain, regardless of whether this point was used to train the network. However, to compare approximations under similar conditions, the discretizations used in implementing the finite difference methods were also used to train the network and generate the solutions.

All applications of PINN were made in a TensorFlow environment [9]. Two finite difference methods were chosen to compare the results obtained with PINN: the forward time-centered space method (FTCS) and the alternating-direction implicit (ADI) methods [10], [11].

4 Computational Simulations

4.1 Discretization Influence

The network used in all discretizations consisted of ten densely connected hidden layers with thirty neurons each; the activation function used was the Gaussian Error Linear Unit (GELU) [12], trained with the Adam optimizer for 5000 epochs. For this problem, the input layer had three dimensions, two for space and one for time, and the output layer had one dimension. This network had a total of 2941 trainable parameters.

The results of the experiments can be seen in Figure 2, where each point represents a particular discretization. The x-axis shows the total number of points in the discretization, and the y-axis shows the relative L_2 norm of the approximation with the analytical solution, both on a logarithmic scale. The relative L_2 norm was calculated using the solution obtained with fifteen points in each space domain and fifty in the time domain for a total of eleven thousand two hundred and fifty points. The values provided by the networks were then compared with the analytical solution on the same points. The blue and orange lines represent the time and spatial domain refinement experiment, respectively, and the green line for both domains simultaneously. We can see that the relative L_2 error tends to converge for all situations.

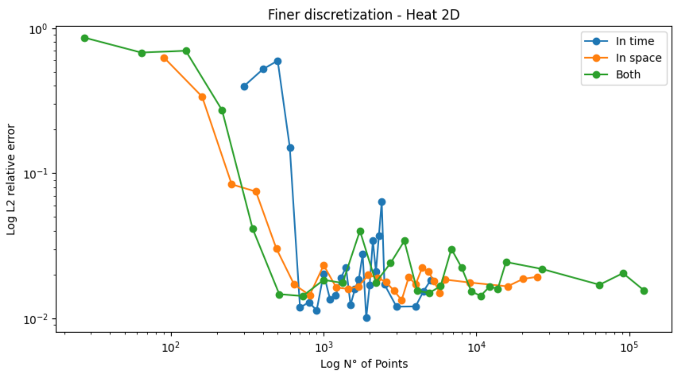
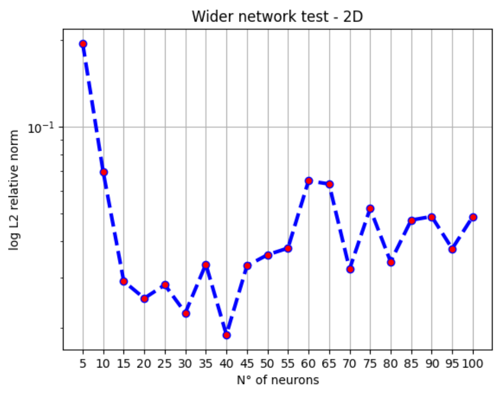


Figure 2: The orange line is refining the space domain, the blue line the time domain, and the green line both. Each experiment has a total of 27 different discretizations. Source: from the authors.

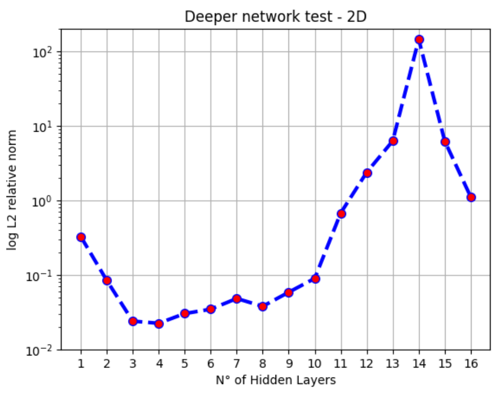
4.2 Network Architecture Influence

To discover the best architecture for the network used to approximate the solution, a multilayer perceptron with the Gaussian Error Linear Unit (GELU) [12] as its activation functions was initially used. For the test aimed at finding the optimal number of neurons in the hidden layers, the number of hidden layers was initially fixed at four, and the number of neurons in these layers was varied; this initial parameter was chosen since it would provide the network with enough expressibility while not making the training impractical. It is possible to see the results in Figure 3a.

The second test, aiming to discover the impact of the number of hidden layers, was performed with a fixed number of thirty neurons in each hidden layer while varying the number of layers. The results are displayed in the Figure 3b.



(a) Experiment where the number of neurons in the hidden layer was varied from 1 to 100 neurons in the hidden layers, adding 5 each time.

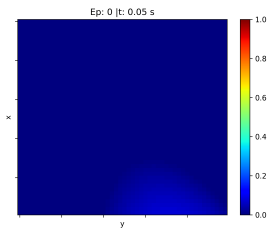


(b) Experiment where the number of hidden layers of the network was varied from varied from 1 to 16, adding 1 at a time.

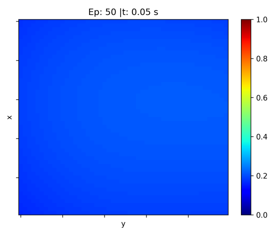
Figure 3: Experiments varying the number of trainable parameters in the networks by independently adding neurons in the hidden layers and adding hidden layers. Source: from the authors

4.3 Approximate Solution Visualization

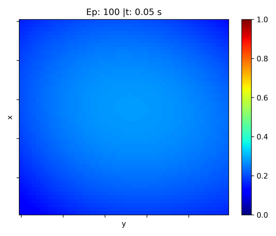
Figure 4 shows the approximations of the network after some training epochs for the specific time steps 0.05 second. The displayed approximations were produced after 0, 50, 100, and 1000.The presented solutions were obtained from a grid with 50 points in each dimension of the space domain and 25 points in the time domain.



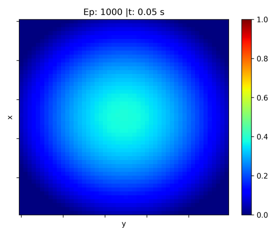
(a) 0 epochs.



(b) 50 epochs.



(c) 100 epochs.



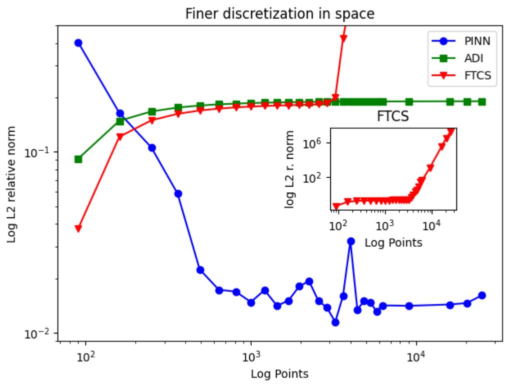
(d) 1000 epochs.

Figure 4: Solutions obtained during the training process of physics-informed neural networks at the time 0.05 seconds after 0, 50, 100 and 1000 epochs. Source: from the authors.

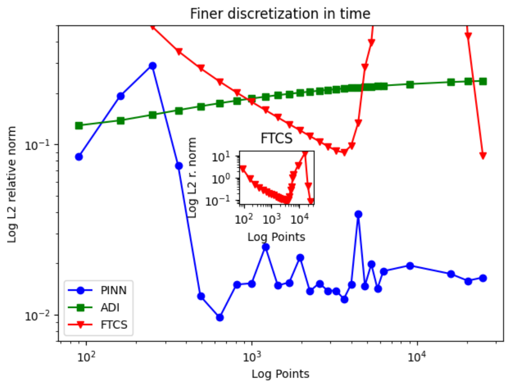
The network was composed of four hidden layers with forty neurons each, totaling 2941 trainable parameters, each with GELU as an activation function. The network was trained for 5000 epochs with a learning rate of 10^{-2} to 1000 epochs, 10^{-3} from 1000 to 3000 epochs, and 5.10^{-4} from 3000 to 5000 epochs.

4.4 Comparative Tests

Two finite difference methods were chosen to compare the results obtained with PINN: one explicit with the Forward Time-Centered Space method (FTCS) and one implicit using the Alternating Direction Implicit (ADI) method [10], [11]. The FTCS method has to obey a condition related to the step size in each dimension to produce a solution; the ADI method, on the other hand, is made in such a way as to be able to produce a solution regardless of the chosen steps [11]. The discretizations used in implementing the finite difference methods were also used to train and evaluate the network. All experiments are displayed in Figures 5a and 5b.



(a) The time domain was fixed at 10 points while both dimensions of the space domain ranged from 3 points to 25, adding 1 point, and then from 25 points to 50, adding 5 points each time.



(b) Both dimensions of the spatial domain were fixed with 10 points, while the time domain varied from 3 points to 25, adding 1 point, and after 25 points to 50, adding 5 points each time.

Figure 5: Each plot relates the number of points to the L_2 relative norm, axes in logarithmic scale. Each point is a distinct discretization, totaling 27 different discretizations in each experiment. Source: from the authors

5 Conclusions

It can be suggested that a minimum number of points is required to accurately represent a domain when addressing estimation errors. However, beyond a certain threshold, the solution's approximation does not significantly improve.

Wider neural networks pose less significant training challenges. Increasing neurons produces relatively closer approximations. However, there is a limit at which approximations stop improving, suggesting an ideal number of neurons exists.

Increasing the hidden layers in neural networks can make training more challenging. The ideal number of hidden layers incorporates the complexity of the solution without encountering difficulties in the training process.

PINN outperformed the two finite difference methods in almost all scenarios. While ADI solved all cases, it could not match the performance of the PINN approximations as the FTCS method had strict convergence constraints.

Visualizing the training process shows that PINN approximates the PDE solution at all points simultaneously, giving it an advantage. Unlike FDM, PINN does not propagate errors from previous time steps to the next ones, which makes it more stable. The independence of the approximate solutions at each time step gives PINN a unique advantage in terms of stability, which results in better performance in the two-dimensional problem where error propagation plays a crucial role.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001

References

[1] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," **IEEE transactions on neural networks**, vol. 9, no. 5, pp. 987–1000, 1998. DOI: 10.1109/72.712178.

[2] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," **Journal of Computational physics**, vol. 378, pp. 686–707, 2019. DOI: 10.1007/s10915-022-01939-z.

[3] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," **Journal of Marchine Learning Research**, vol. 18, pp. 1–43, 2018. [Online]. Available: http://jmlr.org/papers/v18/17-468.html.

[4] V. Souza, "Convergence analysis of physics-informed neural networks and comparison with finite difference methods," Master dissertation, UERJ, 2024.

[5] V. Souza, C. Faria, and K. Leite, "Pinn convergence with regular discretizations and comparison with fdm," in **Anais do Encontro Nacional de Modelagem Computacional e Encontro de Ciência e Tecnologia dos Materiais**, 2023. [Online]. Available: https://www.even3.com.br/anais/xxvi-encontro-nacional-de-modelagem-computacional-xiv-encontro-de-ciencia-e-tecnologia-dos-materiais-338941/705486-pinn-convergence-with-regular-discretizations-and-comparison-with-fdm/.

[6] Z. Hao, S. Liu, Y. Zhang, C. Ying, Y. Feng, H. Su, and J. Zhu, "Physics-informed machine learning: A survey on problems, methods and applications," **arXiv preprint:2211.08064**, 2022.

[7] M. A. Nabian, R. J. Gladstone, and H. Meidani, "Efficient training of physics-informed neural networks via importance sampling," **Computer-Aided Civil and Infrastructure Engineering**, vol. 36, no. 8, pp. 962–977, 2021. DOI: 10.1111/mice.12685.

[8] Y. Shin, J. Darbon, and G. E. Karniadakis, "On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes," **Communications in Computational Physics**, vol. 28, no. 5, pp. 2042–2074, 2020. DOI: 10.4208/cicp.OA-2020-0193.

[9] M. A. et al, **TensorFlow: Large-scale machine learning on heterogeneous systems**, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[10] K. A. Hoffmann and S. T. Chiang, **Computational Fluid Dynamics Volume 1.** Engineering Education System, 1998, ISBN: 978-0962373107.

[11] R. J. LeVeque, **Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems**. SIAM, 2007, ISBN: 978-0898716290.

[12] D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)," **arXiv preprint: 1606.08415**, 2016.