An Efficient PINNs Approach Using Hard Constraints Boundary Conditions for Solving Electromagnetic Problems

Rodrigo Silva Rezende and Rolf Schuhmann

Abstract – Physics-Informed Neural Networks (PINNs) might be a possible solution where mesh-based approaches, such as the finite element method (FEM) fail. High dimensional problems with strong non-linearities are good examples, where the mesh-based methods have difficulties. PINNs can solve these problems since they are capable of modeling non-linearities within high-dimensional geometries if the right architecture is used. However, training PINNs is usually cumbersome since the required architecture's size is normally large, even for small, i.e., 2-dimensional problems. We propose using hard constraint boundary conditions built with suitable functions for training PINNs. This results in faster training times and better accuracy compared to the usual soft constraint-based PINNs.

1. Introduction

Mesh-based numerical approaches such as FEM or finite difference (FD) method are the most used approaches to numerically solve partial differential equations (PDEs) with suitable boundary conditions. They have in most cases high precision, high efficiency, and stable convergence rates.

However, in some situations, these methods have a couple of drawbacks. For example, the final resolution of a solution might be numerically expensive, especially for high wavenumbers in propagation problems [1]. Further, the computational costs for higher-dimensional cases, concerning geometry, usually scale badly. Finally, the precision of those methods generally decreases with non-linear behaviors.

In recent years, neural networks (NNs) have been used to solve PDEs numerically. In this context, the NNs are called Physics-Informed Neural Networks (PINNs) and were first introduced in [2]. The main idea here is to use physical laws' information, i.e., the PDE and the boundary conditions (BCs), and add them to the loss function of the network. Hence, the PDE is converted into an optimization problem, which can be approximated with the help of automatic differentiation and backpropagation approaches. The computation of electromagnetic problems using PINNs is promising since they are mesh-free and can model high-dimensional and geometrically complex problems with strong non-linearities [3].

Manuscript received 6 March 2025.

First and Second Authors are with Theoretische Elektrotechnik, Technische Universität Berlin, Berlin, Germany; e-mail: rodrigo.silvarezende@tu-berlin.de and rolf.schuhmann@tu-berlin.de.

Nevertheless, training PINNs can be cumbersome. Within this approach, one tries to find the best sum of weighted functions to approximate the solution to the problem at hand. The difficulty is that the functions, i.e., neurons are all interconnected, which increases the number of model parameters exponentially. Therefore, even for simpler problems, such as the ones with 2-dimensional geometries, the number of functions (parameters) required to reach a low error is generally big. Hence, the PINNs's solutions tend to be computationally costly.

In this paper, we propose increasing the PINN's architecture by adding a function to fulfill all the BCs even before training it. This approach allows the model to solve the problem using simpler architectures (fewer parameters), facilitating the training process. Instead of meeting the BCs softly, i.e., training the network to approximate the BCs' values, we employ a model (hard-constrained PINN) that inherently matches the correct values at the boundary.

Note that our work here intends to demonstrate and benchmark a more robust way to enforce the physical constraints in initially simpler cases, where the key weakness of the standard PINN framework is already present. This is then a pre-step for enabling PINN to reliably and efficiently tackle more complex problems. Then, in the future, a benchmark between hard-constrained PINNs and grid-based methods will be a valuable and meaningful approach.

To achieve this goal in an application example, we solve the 2D inhomogeneous Helmholtz Equation for the vector potential $\mathbf{A}$ with an imposed current density $\mathbf{J}$ as an excitation source. The BCs are set to be Dirichlet ones, i.e., rather simpler BCs since the focus here is the comparison of soft and hard-constrained PINNs within canonical examples. We show that by using a hard-constrained PINN, one can train the model faster and achieve more accurate results than those computed with a soft-constrained PINN, even for simpler models and BCs.

2. PINNs

We consider $\mathbf{x} \in \mathbb{R}^d$ to be the input of the function we want to approximate, defined as $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^m : = \mathbf{x} \mapsto \mathbf{y}$. For that, we use an NN with L layers and n_i neurons in the i-th layer, with $i = 1, 2, \cdots L$. This network can be described as $\mathbf{f}^{[L]} : \mathbb{R}^d \rightarrow \mathbb{R}^m : = \mathbf{x} \mapsto \mathbf{y}^{[L]}$, with [3]:

$$\begin{cases} \mathbf{y}^{[0]} = \mathbf{x}, \\ \mathbf{y}^{[\ell]} = \sigma(\mathbf{W}^{[\ell]}\mathbf{y}^{[\ell-1]} + \mathbf{b}^{[\ell]}) \ \text{ for } \ \ell = 1, 2, 3, \ldots, L \end{cases} \tag{1}$$

where $\mathbf{W}^{[\ell]} \in \mathbb{R}^{n_{l+1} \times n_l}$ and $\mathbf{b}^{[\ell]} \in \mathbb{R}^{n_{l+1}}$ are weights and biases of the l-th layer, respectively. The σ represents the sigmoid function applied pointwise to the matrix-vector product. Then, for a given L and n_i we compute the best $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ such that:

$$\underset{\mathbf{W}^{[\ell]}, \mathbf{b}^{[\ell]}}{\arg\min} \, (\mathbf{y}^{[L]} - \mathbf{y})^2 \tag{2}$$

By sufficiently minimizing this expression, one would build a model capable of approximating $\mathbf{f}$. Now, consider we have further information about $\mathbf{f}$, which is interpreted as a physical law of the problem being solved. We want to use this knowledge in the cost function described in (2). This information is actually a PDE involving $\mathbf{f}$, as follows:

$$\begin{aligned} \mathscr{N}[\mathbf{f}] &= \mathbf{u} \text{ in } D \\ \mathscr{B}[\mathbf{f}] &= \mathbf{w} \text{ on } \partial D \end{aligned} \tag{3}$$

where $\mathscr{N}$ is the linear or nonlinear differential operator, $\mathscr{B}$ is the boundary operator, $\mathbf{u} : \mathbb{R}^d \to \mathbb{R}^m$ and $\mathbf{w} : \mathbb{R}^d \to \mathbb{R}^m$, D and ∂D are the computation domain and its boundaries, respectively. Hence, we want to use the network described in (1) to approximate the function in (3). We then build the PDE and BC residuals as:

$$\begin{aligned} L_{PDE} &:= \mathscr{N}[\mathbf{f}^{[L]}] - \mathbf{u} \\ L_{BC} &:= \mathscr{B}[\mathbf{f}^{[L]}] - \mathbf{w} \end{aligned} \tag{4}$$

Finally, these residuals are added into (2):

$$\underset{\mathbf{W}^{[\ell]}, \mathbf{b}^{[\ell]}}{\arg\min} \left((\mathbf{y}^{[L]} - \mathbf{y})^2 + L_{PDE}^2 + L_{BC}^2 \right) \tag{5}$$

This final equation is a mathematical description of a PINN. Note that, this architecture is called a soft-constrained PINN since the term for the BC is explicitly a part of the loss function.

More specifically, if we want to solve the inhomogeneous Helmholtz equation in the context of electromagnetism, the input dimension is $d = 3$ and output $m = 3$. The function that we consider here in the $\mathbf{f} := \mathbf{A}$, i.e., the vector potential. Then the PDE looks like:

$$\begin{aligned} \mathscr{N}[\mathbf{A}] &:= \Delta\mathbf{A} + \omega^2 \boldsymbol{\epsilon}_0 \mu_0 \mathbf{A} = -\mu_0 \mathbf{J} \\ \mathscr{B}[\mathbf{A}] &:= \mathbf{A} \,|_{\,\partial D} = \mathbf{0} \end{aligned} \tag{6}$$

where $\mathbf{J}$ is the imposed current density. Note that, in (6) we define the BC as a Dirichlet BC equal to zero since it is the only BC we use in this work. Our minimization problem becomes then:

$$\underset{\mathbf{W}^{[\ell]}, \mathbf{b}^{[\ell]}}{\arg\min} \Big((\mathbf{A}^{[L]} - \mathbf{A})^2 + (\Delta\mathbf{A}^{[L]} + \omega^2 \boldsymbol{\epsilon}_0 \mu_0 \mathbf{A}^{[L]} + \mu_0 \mathbf{J})^2 + (\mathbf{A}^{[L]} \,|_{\,\partial D})^2 \Big) \tag{7}$$

Now by minimizing this equation, we find an approximate solution for the inhomogeneous Helmholtz equation stated at (6). However, as already stated, minimizing (7) is often computationally costly since the number of parameters required to reduce all three loss terms is high.

In order to, reduce the complexity of this loss function and fasten the process, we employ the hard-constrained PINNs.

3. Hard-constrained PINNs

The goal here is to satisfy the BC stated in (6), i.e. the Dirichlet BC, perfectly without the need for the third term in (7). Therefore, we search a function $\mathbf{A}^{[}L]*$ such that:

$$\forall \mathbf{W}^{[\ell]}, \mathbf{b}^{[\ell]}, \forall \mathbf{x} \in \partial D : (\mathbf{A}^{[L]*}(\mathbf{x}))^2 = \mathbf{0} \tag{8}$$

To achieve that, we define a transformation function $\mathbf{tr} : \mathbb{R}^d \to \mathbb{R}^m := \mathbf{x} \mapsto \mathbf{y}$, where $\forall \mathbf{x} \in \partial D : \mathbf{tr}(\mathbf{x}) = \mathbf{0}$. Then we define $\mathbf{A}^{[L]*}$ as follows:

$$\mathbf{A}^{[L]*}(\mathbf{x}) := \mathbf{A}^{[L]}(\mathbf{x}) \cdot \mathbf{tr}(\mathbf{x}) \tag{9}$$

where $\mathbf{A}^{[L]}$ is a soft-constrained PINN. Hence, with the new function we meet the BC automatically and can simplify the loss term. Additionally, we seek to train the PINN without any training points, i.e., we construct the PINN exclusively with the PDE information. Considering this, we can also exclude the first term of (7). Then, the final loss term for the hard-constrained PINN is:

$$\underset{\mathbf{W}^{[\ell]}, \mathbf{b}^{[\ell]}}{\arg\min} \, ((\Delta\mathbf{A}^{[L]*} + \omega^2 \boldsymbol{\epsilon}_0 \mu_0 \mathbf{A}^{[L]*} + \mu_0 \mathbf{J})^2) \tag{10}$$

By minimizing this new expression, we calculate an approximate solution for (6) with BC residual equal to zero. This operation facilitates the building of the PINN once the complexity of the loss function is extremely reduced before training. Hence, the number of neurons and layers required for the PINN will be reduced, thereby accelerating training and improving accuracy.

Note that in this work, we chose to implement Dirichlet BCs in a hard-constrained way. However, other complex boundary conditions can also be implemented in this way, making PINNs, and specifically hard-constrained PINNs, a topic of interest in the community [4]. As mentioned before, we chose to tackle the hard-constrained Dirichlet BCs, since this is a direct and clear way of comparing both PINNs in a canonical

example. Nevertheless, in recent years, the implementation of more complex BCs such as absorbing boundary conditions (ABCs) or PMLs has been achieved in preliminary stages [3].

In the next section, we show how the hard-constrained PINN can solve an inhomogeneous Helmholtz equation in a 2-D problem with Dirichlet BCs more efficiently than usual PINNs.

4. Application Example

As an application example, we solve the inhomogeneous Helmholtz equation for the vector potential $\mathbf{A} = A_z \mathbf{e}_z$, with an imposed current density $\mathbf{J} = J_z \mathbf{e}_z$ as a source term, stated in (6). The computation domain is a square $D = [-0.5, 0.5] \times [-0.6, 0.6]$ m. Within this domain, we set the medium as vacuum (ϵ_0 and μ_0), and on the boundary ∂D we have perfect electric conductive (PEC) walls. In the middle, we define a source area on a circle with a radius $r = 0.02$ m. In this source area, we impose $\mathbf{J} = J_z \mathbf{e}_z$ such that we have a total current $I = 1$ A. For this problem, we have a Dirichlet BC that matches that of (6), i.e., $A_z \,|_{\partial D} = 0$.

As a benchmark, we solve this problem for A_z first with a hard-constrained PINN (HCPINN), as described in the last section. We also solve it with a soft-constrained PINN (SCPINN1) trained with similar resources, i.e., the same architecture (# layers and neurons) and training time as for the HCPINN. Further, we compute A_z with a second soft-constrained PINN (SCPINN2), which allows for arbitrary big architectures and training times. In the end, we compare all solutions with the one computed with an FD frequency domain in-house code. All models are implemented in Python, and the PINNs are coded within the DeepXDE package [4]. The computer used here has 32 GB RAM and a processor Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz.

We implement the HCPINN by first building a usual PINN with $L = 6$ layers and $n_i = 120$ neurons per layer and with tanh as the activation function. The inputs of this PINN are the coordinates of the domain (x, y). The PINN output is $A_z^{[6]}(x, y)$ and is then transformed, with a function $\mathbf{tr}$ as stated in (9), with

$$A_z^{[6]*}(x,y) = A_z^{[6]}(x,y) \cdot \mathbf{tr}(x,y) := A_z^{[6]}(x,y) \cdot ((0.5+x) \cdot (x-0.5) \cdot (0.6+y) \cdot (y-0.6)) \tag{11}$$

which transforms this soft-constrained PINN into a hard-constrained one. In other words, by composing the initial PINN with $\mathbf{tr}$, we impose the $A_z \,|_{\partial D} = 0$ on the PEC walls. Therefore, by its nature, the HCPINN has zero loss on the boundary, and to train it we can indeed compute (10) instead of (7). To do so, we employ an Adamax optimizer with a decaying learning rate implemented within a loop [5]. We use 1000 (x, y) points randomly distributed on the square domain as training points and 100 as testing points. The total minimization time, i.e., the training time of this HCPINN takes 2256 s. After building the HCPINN we compute its approximation for A_z within the domain D. We also calculate the FD solution for A_z and build the pointwise absolute error of the HCPINN. This is shown in Fig. 1. Note that, the HCPINN has an extremely low error, except for a small area near the source in the middle. We also plotted on this figure a red line on which we can compare all the methods later. This is most probably due to the special bias of PINNs, i.e., their ability to learn more easily low-frequency components than higher ones of the solution. Furthermore, the choice of the colocation points that are randomly chosen for our model plays a crucial role here. Since the PINNs do not solve the PDE on the cable area (source domain), only

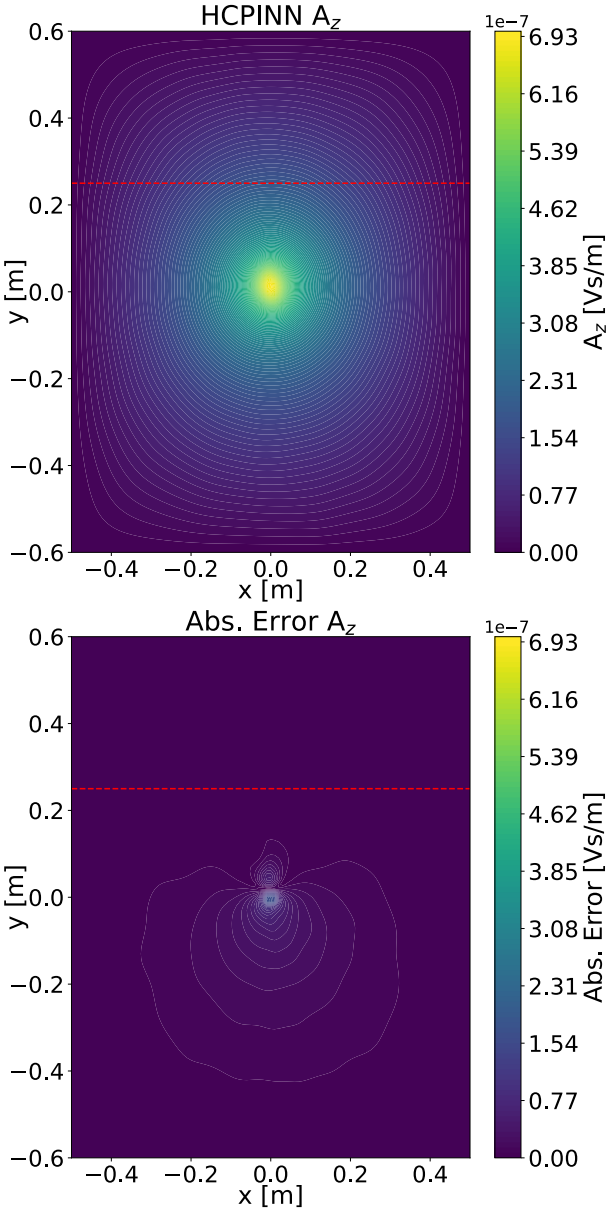

Figure 1. Top: HCPINN Solution for A_z. Bottom: Absolute Error for A_z between HCPINN and FD.

around it, the points located near the cable are the ones that will introduce the source information into the model. If fewer points are located there, the error might be higher, since it's a sensible area for the solution (source information). To tackle this problem, one can improve the sampling of the domain by increasing the density of collocation points in the vicinity of the source.

After this step, we construct a usual soft-constrained PINN (SCPINN1) with the same resources as for the HCPINN, i.e., $L = 6$ layers and $n_i = 120$ neurons per layer. Since here we have a BC loss, we must then minimize (7). For this, in addition to the 1000 training and 100 test points in the domain, we use 250 on the boundary to minimize the BC loss. Here the selection of these parameters was made heuristically, i.e., without using a hyperparameter optimization. The choice was achieved after testing fewer points, which generated extremely inaccurate results. Conversely, more points caused the training of the models to take too long, such that they had to be aborted. Hence, this selection was a good compromise between accuracy and computational costs. The training is done in 2691 s (similar to HCPINN). The second soft-constraining PINN (SCPINN2) is constructed with more resources. This model is built with $L = 10$ layers and $n_i = 120$ neurons per layer. The same number of points are used at the domain and boundary as for SCPINN1. The training here takes a total time of 8293 s. Finally, the FD implementation uses a mesh with 12000 points (100×120), with which sufficient accuracy is achieved for the PINNs' benchmark. Here, the current density is also imposed on a circle with radius $r = 0.02$ m, such that the total current is $I = 1$ A.

For a better visual comparison between the methods, we take the approximation of A_z of each of them and calculate the H field with:

$$\mathbf{H} = H_x \mathbf{e}_x + H_y \mathbf{e}_y = \frac{1}{\mu_0} \nabla \times (A_z \mathbf{e}_z) \tag{12}$$

We then choose the line at $y = 0.25$ m, shown in Fig. 1 (red line), read the values of H_x and H_y on this line for all the methods, and plot them together with the solutions from the FD method. The results of this procedure are shown in Fig. 2 and Fig. 3. From these figures, it becomes clear that the HCPINN and SCPINN2 have a low error for both components of the $\mathbf{H}$ field. Yet the computational costs used to achieve these solutions with similar errors are much smaller for the HCPINN. It is also visible that the SCPINN1 has high errors, especially for H_x. Here, the resources (# layers, neurons, training time) are too scarce to perform the training. This is a strong indication that simplifying the loss function of a PINN, i.e., reducing it by its BC term, makes the training procedure much easier. In other words, using exactly the same architecture and similar training times HCPINN is capable of achieving low errors, while SCPINN1 suffers with the BCs. Finally, we compute the normalized root mean squared errors for each of the PINNs in comparison with the FD

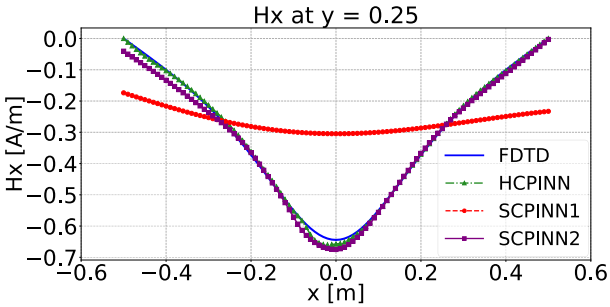

Figure 2. Hx solutions plotted at the red line of Fig. 1 ($y = 0.25$) for FD, HCPINN, SCPINN1, and SCPINN2.

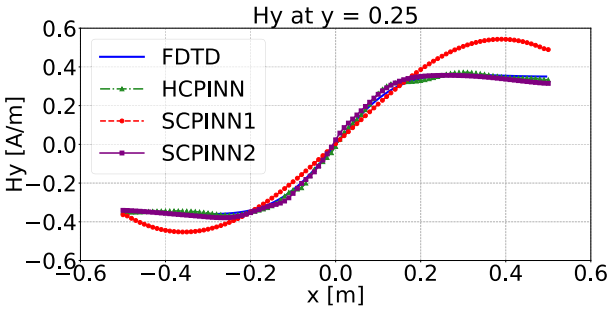

Figure 3. H_y solutions plotted at the red line of Fig. 1 ($y = 0.25$) for FD, HCPINN, SCPINN1, and SCPINN2.

solution for H_x and H_y. These can be seen in Table 1 together with the architecture size and training times of each model. Here the difference between the HCPINN and the SCPINNs becomes even more visible. We achieve with the former slightly better results (lower errors) than the big model SCPINN2, being almost 3.7 times faster. Hence, excluding the loss term with the right transformation function **tr** turned the minimization process into a simpler process (faster) without forcing the model to have higher errors. Contrarily, the accuracy is also increased as shown in the images and in the table.

5. Conclusion

In this work, we present a modified PINN to solve electromagnetic problems more efficiently and with better accuracy. We exclude the BCs' term of the usual loss function of a PINN. For that, we employ a transformation function coupled together with the PINN. This transformation enforces the BC from the beginning. Hence, the BC loss term is set to zero already before training, i.e., naturally. The new architecture is called a hard-constrained PINN. We employ hard-constrained PINN to solve a 2-dimensional inhomogeneous Helmholtz equation for the vector potential in a rectangular domain with Dirichlet boundary conditions and compare its results with usual PINNs and with FD computations. We achieved more accurate results with the hard-constrained PINN even against much

Table 1. Comparison of HCPINN and SCPINN Models

Metric	HCPINN	SCPINN1	SCPINN2
Training Time [s]	**2256**	2691	8293
# Layers	6	6	10
# Neurons	120	120	120
NRMSE H_x	**0.012**	0.282	0.031
NRMSE H_y	**0.019**	0.58	0.022

larger soft-constrained PINN models. Further, we could achieve lower errors while being faster at the same time. This indicates the major capability of the hard-constrained PINN for solving partial differential equations using lower computational costs. Additionally, these results strongly suggest that this methodology could be used for more complex geometries (many interfaces) and boundary conditions (multiple boundary conditions) efficiently and with good accuracy for three main reasons. Firstly, its simplified loss term allows for the direct elimination of each boundary and interface loss term, which within standard PINNs should all be minimized. Secondly, hard-constrained PINNs improve stability and convergence by forcing the output to strictly satisfy the BC before construction, which reduces the search space of the optimizer. Finally, hard-constrained PINNs guarantee the physical underlying laws, whereas soft-constrained PINNs satisfy them approximately. In the end, this is critical for complex problems where high accuracy is desired.

6. References

1. P. Escapil-Inchauspé and G. A. Ruz, *Hyper-parameter tuning of physics-informed neural networks: Application to Helmholtz problems*, Neurocomputing, vol. **561**, p. 126826, Dec. 2023.
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 Comp. Physics, vol. **378**, pp. 686-707, 2019.
3. X. Li, J. Deng, J. Wu, S. Zhang, W. Li, J. Deng, and Y.-G. Wang, *Physical informed neural networks with soft and hard boundary constraints for solving advection-diffusion equations using Fourier expansions*, Computers and Mathematics with Applications vol. **159**, 60-75, 2024.
4. L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, *DeepXDE: A deep learning library for solving differential equations*, SIAM Review, vol. **63**, no. 1, pp. 208-228, 2021, https://github.com/lululxvi/deepxde.
5. D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*. International Conference on Learning Representations (ICLR), 2015.