Physics-Informed Graph Convolutional Networks: Towards a generalized framework for complex geometries

M. Chenaud[1,2], F. Magoulès[1,3] and J. Alves[2]

[1]MICS, CentraleSupélec
Université Paris Saclay, Gif-sur-Yvette, France
[2]Transvalor S.A. E-Golf Park, Biot, France
[3]Faculty of Engineering and Information Technology, University of Pécs, Hungary

Abstract

Since the seminal work of [9] and their *Physics-Informed neural networks* (PINNs), many efforts have been conducted towards solving partial differential equations (PDEs) with Deep Learning models. However, some challenges remain, for instance the extension of such models to complex three-dimensional geometries, and a study on how such approaches could be combined to classical numerical solvers. In this work, we justify the use of graph neural networks for these problems, based on the similarity between these architectures and the meshes used in traditional numerical techniques for solving partial differential equations. After proving an issue with the Physics-Informed framework for complex geometries, during the computation of PDE residuals, an alternative procedure is proposed, by combining classical numerical solvers and the Physics-Informed framework. Finally, we propose an implementation of this approach, that we test on a three-dimensional problem on an irregular geometry.

Keywords: partial differential equations, finite element method, scientific machine learning, graph neural networks, physics-informed neural networks

1 Introduction

Partial differential equations (PDEs) arise in almost every known physical systems and at every scale, whether looking at macro scales with continuous mechanics (fluid or solid), the very small with quantum physics or the very large with general relativity. Differential equations constitute also the basis of other domains such as finance or population growth in biology. In this context, and since the exact solutions of these equations are generally not reachable, an efficient and accurate technique to estimate these solutions is critical. In the last century, the theory and numerical procedures tackling these problems have been developed thoroughly.

One of the most successful techniques is called *finite element method* (FEM). The principle of this method is to discretize the domain in which one needs to solve the PDE, and to approximate the solution as a linear combination of known, truncated polynomials. The idea is then to find the best combination of these functions, a problem which is usually reduced to a linear system of equations, that is tackled efficiently numerically. However, there still are some unresolved issues with these approaches. With large-scale, or high dimensional problems, the numerical cost of classical methods is prohibitive. In industrial applications, with complex geometries and various physics interacting with each other, some efforts need to be made in order to balance model complexity and computational requirements. Hence, with the rise of Deep Learning and the broad range of problems that can be dealt with these techniques efficiently, it seems natural to try and develop new Deep Learning solutions for physical applications. In this context, getting real data or accurate simulations is expensive and time-consuming, thus available datasets are rare and scarse. Classical Deep Learning frameworks may thus be innapropriate for this range of problems, therefore, the objective is to develop new techniques combining physical knowledge and the scarse available data to create accurate and trustworthy models.

Many works have focused on this range of problems. With this increasing interest, some works have been naturally focusing on the dataless, direct solving of partial differential equations with numerical surrogates using Deep Learning frameworks. Some early works on this topic trace back to the end of the last century, see, for instance [7], but the subject has really gained attention since a few years ago, with the seminal work of [9]. Following this regain in interest, many different techniques and variations of the original approach have been developed, often combining knowledge from numerical analysis and Deep Learning techniques. However, the applications of such techniques are often limited to 2D cases with regular geometries, settings for which classical numerical methods are more efficient and accurate than PINN-based approaches.

1.1 Partial differential equations and PINNs

For the sake of simplicity, the following section will focus on steady state formalism. A steady PDE can be written in a general form like Equation (1).

$$\mathcal{E}(u)(x) = 0 \quad \forall x \in \Omega, \quad \mathcal{E} \text{ being any differential operator,} \tag{1a}$$

$$\mathcal{B}(u)(x) = 0 \quad \forall x \in \partial\Omega \quad \text{(Boundary conditions, noted (B.C)).} \tag{1b}$$

Here, $\mathcal{E}$ represents the differential operator of the equation, it can be linear or nonlinear, an elliptic operator for instance. Ω is the domain in which the equation has to be solved (usually, an open, connex set of $\mathbb{R}^d$ for any integer $d \geq 1$). The solution we look for is u, function of the variable space $x \in \Omega$. Finally, $\mathcal{B}$ is any boundary operator (Dirichlet, Neumann or mixed usually, but it can be more general).

The original idea, as developed in [7], updated and improved by [9] to tackle the resolution of steady PDEs such as (1) with neural networks was to use a rather simple network architecture, the Multi-Layer Perceptron (MLP). This model produces a predicted solution $\hat{u}$, and in order to ensure that the PDE is respected in the domain by this solution, some *collocation points* $(x_i)_{i=1,...,N}$ are used. These are points chosen, randomly or not, in the domain Ω, for which the residuals $\mathcal{E}(\hat{u})(x_i)$ are computed. The loss function that needs to be minimized during training is hence:

$$\mathcal{L}(\hat{u}) = \frac{1}{N} \sum_{i=1}^{N} ||\mathcal{E}(\hat{u})(x_i)||^2 := \mathcal{L}_r(\hat{u}). \tag{2}$$

A neural network with smooth activation functions is differentiable, so $\mathcal{E}(\hat{u})$ can be computed, either, for simple cases such as in [7], analytically, or with auto differentiation techniques provided by frameworks such as Pytorch [8]. The minimization procedure of the loss is typically done by generalized gradient descent techniques. To ensure that the boundary conditions are respected as well, some collocation points are selected on the border as well, to compute the gap between the predicted and expected BC value. The loss is hence:

$$\mathcal{L} = \lambda_r \mathcal{L}_r(\hat{u}) + \lambda_{u_b} \mathcal{L}_{u_b}(\hat{u}), \tag{3}$$

with λ_r, λ_{u_b} some hyperparameters that need to be tuned. Here, supposing there are M data points located on the border $\partial\Omega$ for which the expected value u_b is known, the loss term $\mathcal{L}_{u_b}$ is:

$$\mathcal{L}_{u_b}(\hat{u}) = \frac{1}{M} \sum_{i=1}^{M} ||\hat{u}(x_i) - u_b(x_i)||^2. \tag{4}$$

Models trained in this framework are known as *Physics-Informed neural networks* (PINNs).

1.2 Deep Learning on graphs

Graph-based approaches have been trending in Deep-Learning, in many fields such as chemistry, social interactions modelization or biology. For an overview and some examples of these approaches, see [10, 13]. This framework yield state-of-the-art results, which is a consequence of the *inductive biases* of these models. This concept has been studied thoroughly in [2]. With this structure, the main idea is to propagate information between nodes and edges, step by step, from closer nodes to more distant ones. For this, *graph convolutions* [6] are used.

A graph convolution corresponds to a Fourier transform of the graph laplacian matrix, which necessitates the computation of its eigenvalues. However, in order to circumvent this high computational cost, the authors of [6] decided to use a first-order polynomial approximation of this transformation. For the sake of brievty, we will not detail the mathematical formulation of this operation. The underlying idea is to only compute, at each convolutional layer, the influence of the direct neighbours of each node in order to update the predicted value at this node. This operation appears to be very efficient, and to accurately propagate information through the graph. This framework has therefore been widely used in graph-based prediction tasks.

These convolutions are based on the nodes of the graph, but other approaches focus on edge-based information propagation. In [12], the authors propose EdgeConv, a convolution based on the edges features of the graph. This architecture has the same inductive biases as nodes convolutions, but also includes the ability to exploit the relative difference of node features. It is particularly suitable for tasks in which the relative position of nodes between themselves, rather than their absolute coordinates, is relevant. It is the case for physical diffusion processes for instance. The authors prove the ability of the model to diffuse local information over the graph, but also to capture more global, long-distance phenomena.

2 Physics-Informed graph neural networks

In [4], the message-passing step over a graph is explained as a diffusion process, and the authors prove that the different convolutions can be seen as different techniques to solve a discretized heat diffusion PDE. This strong link between graph convolutional architectures and physical diffusion processes, along with the natural inductive biases explained in [2], motivates the use of this framework to simulate physical processes. As a result, several recent works have been focusing on this task.

In [5], the authors take advantage of the natural link between graphs and meshes to tackle the problem of solving static PDEs. A mesh is seen as a graph, and the neural network learns the discretized solution of the variational problem in each point of the graph. However, only 2 dimensional cases are tackled. [3] incorporated a numerical

solver inside a graph model. The result of the solver, which was computed on a coarse mesh, are fed into the graph network to help convergence.

3   Building PDE residuals on complex geometries

Many works using the Physics-Informed framework have been focusing on one or two dimensional problems, for which classical numerical techniques already offer fast and accurate simulations. Hence, it seems natural to extend Physics-Informed models to complex three-dimensional geometries, for which classical techniques are slow. However, for these irregular problems, there is an underlying issue with the computation of the PDE residuals, that are necessary to train Physics-Informed models. To prove this issue, let us assume that we have a linear model M, which takes as input a field φ and the position $x = (x_1, x_2, x_3)$ on a given geometry $\Omega \in \mathbb{R}^3$, and which outputs a field $M(\varphi, x)$. Suppose that the input field φ, which can be any physical information on the problem, such as the boundary conditions, depends on the position: $\varphi = \varphi(x)$. Since M is linear, there exist scalar numbers α, β, γ such that $M(\varphi, x) = \alpha\varphi(x) + \beta x + \gamma$. Thus, the corresponding derivative of the output with respect to the variable x_1 is:

$$\frac{\partial M}{\partial x_1} = \alpha\frac{\partial \varphi}{\partial x_1} + \beta. \tag{5}$$

For simple, two dimensional cases, or regular three dimensional cases, φ can usually be analytically constructed from x. But for more complex cases, when dealing with irregular geometries for example, this may not be possible. Hence, the field φ may be an input of the model in the form of a spatial field, with no analytical expression. Autodifferentiation [8] enables automatic computation of derivatives by applying the chain rule using a recorded computational graph. Every operation on a tensor will be recorded and traced back afterwards, and the actual derivative of this operation will be constructed from these steps, by the chain rule. The derivative of the output with respect to x_1 will thus be computed once again as $\frac{\partial M}{\partial x_1}\underset{\text{autodiff}}{=} \alpha\frac{\partial \varphi}{\partial x_1} + \beta$. However, φ being a direct input to the model, there is no recorded link in the computational graph between φ and x_1: therefore, $\frac{\partial \varphi}{\partial x_1}\underset{\text{autodiff}}{=} 0$. Hence, the derivative of φ with respect to x_1 will be computed as:

$$\frac{\partial \varphi}{\partial x_1}\underset{\text{autodiff}}{=} \alpha. \tag{6}$$

This issue prevents the use of the Physics-Informed framework for complex geometries, this is why there is a clear lack of work in this direction. There are ways to overcome this issue, for instance, [14] approximated the spatial derivatives with finite differences, allowing to tackle three dimensional problems. However, this approach does not generalize well for irregular problems, since it is known from classical numerical analysis that finite differences are not accurate enough. Instead, finite element

approaches have the flexibility and robustness to allow the computation of approximate spatial derivatives on very irregular classes of geometries. The remaining issue is that these computations are often made inside numerical solvers, and operate complex operations that are not reachable by the Deep Learning autodifferentiation methods. Hence, if an external numerical solver is used to compute the loss residuals, the autodifferentiation will not trace back this computation step, preventing the model to be trained. The missing link is thus the derivative of the numerical gradient operator with respect to the field, i.e the second derivative of the field, the hessian. This second-order derivative could also be computed by a numerical solver. Therefore, in this work, we will be combining the Physics-Informed workflow and a finite element solver to compute the PDE residuals. The corresponding workflow is presented in Figure 1. Note that in this workflow, the model M_θ could be any machine learning model, and is not restricted to a graph network or a neural network.
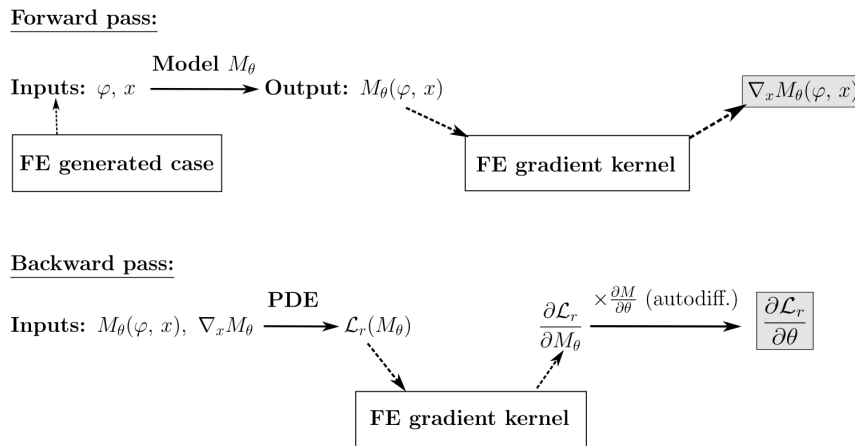

Figure 1: Workflow of the proposed process. The dashed lines represent the exchange of information between the Deep Learning process and the finite element solver, outside of the autodifferentiation framework.

4   Proposed approach and results

4.1   Problem tackled and chosen model

In order to test the framework presented in the previous section, along with a graph-based approach, an electrostatic physical problem has been chosen. The unknown is the electric potential V on a given, three dimensional, irregular geometry. Two portions of the boundary are fixed, as boundary conditions, and the PDE that needs to be solved for V is simply:

$$\Delta V = 0. \tag{7}$$

This problem has been solved by the commercial finite element solver FORGE ® [1]. In order to combine the classical FE solver with the Physics-Informed workflow two

key steps are distinguished: first, mesh and field data created by the software are used as input to contruct the graph-based model. The gradient kernel of the FE solver is used throughout the learning process, as detailed in Section 3. The target solution, along with the boundary conditions field, can be seen in Figure 2. This graph structure is then fed to the Deep Learning model.
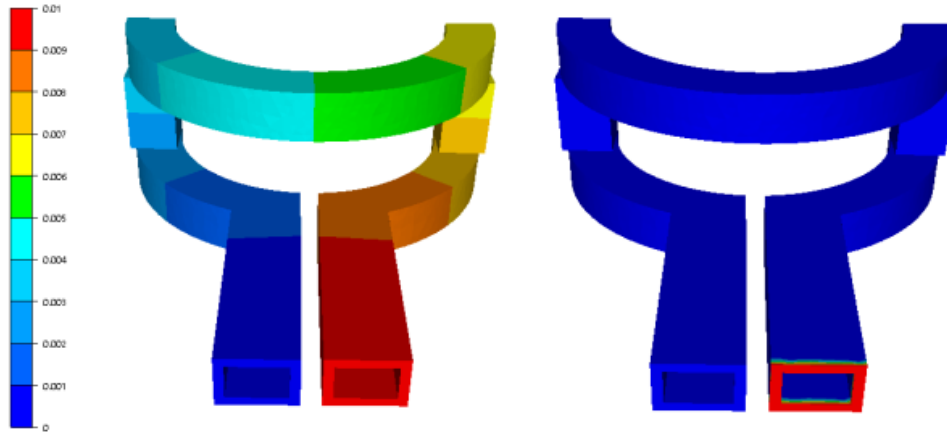

Figure 2: (left) Target field, solution computed by the finite element solver. (right) Boundary conditions field, given as input to the model.

The model is based on the Edge Convolution described in section 1. The network is composed of two edge convolution layers, each one being a 2 hidden layers MLP with ReLU activation function, with a width of 128. The model is then trained with the Adam optimizer, with the learning parameters (number of epochs, learning rate, weight decay...) depending on the problem. For the results gathered in table 1, our model is denoted as PIECN, for *Physics-Informed Edge Convolutional Network*.

For the loss function, several options have been explored, to validate the results made in the previous sections. The idea being to exploit at its best a single numerical simulation, our approach slightly differs from a pure Physics-Informed approach. The PDE residuals are used for the loss function, along with the mean absolute error (MAE) between the model's prediction and the target field. When two loss terms are used, for the weights between the two terms, several options have been explored. An implementation of [11] has been tried, but there was no significative improvement over fixed weights during training. Hence, for the results presented after, the weights have been fixed to 1 for both terms, except for the PDE residuals computed with the finite element solver, where the weight is set to 0.01. This value is in agreement with the results in the bibliography, since the gradients of this term tend to be larger than for the classical terms.

4.2 Results and discussions

In order to test the different aspects of the proposed approach, several models have been tested. Three performance indicators have been monitored: the Mean Absolute Error (MAE), which is the L_1 error between the predicted and the actual field, the PDE residuals as computed by the custom spatial gradients, with the procedure described in section 3, and the PDE residuals as computed by the autodifferentiation module used (here, Pytorch). "PINN" means that the tested model was a MLP, in the same fashion as [9], and "PIECN" refers to our models, as described in 4.1. The results are gathered in table 1.

	PDE residuals	MAE	Autodiff. Residuals
PINN with autodiff. residuals	0.111	0.950	0.0
PIECN with autodiff. residuals	0.078	0.769	$9.38.10^{-5}$
PINN trained on MAE	0.042	0.155	-
PIECN trained on MAE	$2.66.10^{-4}$	0.0082	-
PINN, spatial residuals + MAE	0.034	0.180	-
PIECN, spatial residuals + MAE	$\mathbf{2.19.10^{-4}}$	**0.0070**	-
Target solution	0.0	0.0	-

Table 1: Results of the tested models on the electrostatic problem.

First, with the two first lines, it is clear that the residuals computed by autodifferentiation are not suitable. With this framework, a PINN achieves a zero residual, without actually converging to the true solution, and same goes for the PIECN. On the other hand, as seen in the last line, the target solution has a zero residual when computed with our own gradient implementation. As explained in section 3, our approach circumvents the issue of autodifferentiation.

Overall, it is clear that a simple neural network is less efficient than graph-based approaches. The inputs of the PINN are the absolute positions and fields, although for such physical processes, the relative positions between points are more significant. The diffusion process happens inside the geometry considered, and is well captured by the mesh-based geometry of our approach, while the PINN is not able to capture this phenomenon. Finally, for the graph model, a training on the field and the residuals of the equations yields slightly more accurate results than a training on the field only. The difference is not spectacular, but this may be due to the smoothness of the equation considered, and the fact that the geometry is not very complex. Some further work on more complex physical problems will be conducted in order to quantify the benefit of this approach.

The prediction made by the most accurate model, the PIECN trained on both the field and the PDE residuals, has been reproduced in Figure 3, along with the corresponding MAE. The peak errors are located farthest from the non-zero boundary con-

dition, in agreement with the way diffusion happens on a graph convolutional model.
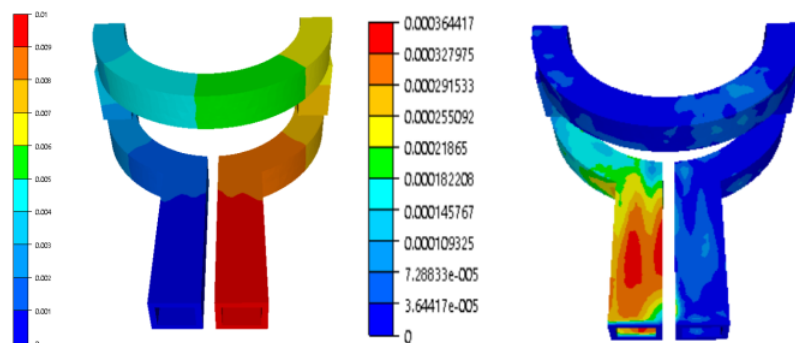

Figure 3: (left) Prediction of our model. (right) Corresponding MAE.

5 Concluding remarks

In this work, after an overview of the works tackling Physics-Informed Deep Learning for physical problems, we provided insight on why graph-based learning are well suited for these applications. We then proved a limitation of autodifferentiation techniques for Physics-Informed learning, and we proposed a new workflow combining a finite element solver and Deep Learning to tackle this issue. Finally, we proved the benefit of this approach on a three dimensional geometry. This new framework will allow us to tackle more challenging cases in the future. First, a quantification of the benefit of combining physial knowledge and numerical data should be conducted. Then, the generalization capability of such models should be investigated. Finally, the proposed approach should be tested on more complex equations such as non linear, time-dependent problems.

References

[1] J. Alves, S. Acevedo, S. Marie, B. Adams, K. Mocellin, and F. Bay. Numerical modeling of electrical upsetting manufacturing processes based on forge® environment. In *AIP Conference Proceedings*, volume 1896, page 120003. AIP Publishing LLC, 2017.

[2] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[3] F. D. A. Belbute-Peres, T. Economon, and Z. Kolter. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *International Conference on Machine Learning*, pages 2402–2411. PMLR, 2020.

[4] B. Chamberlain, J. Rowbottom, M. I. Gorinova, M. Bronstein, S. Webb, and E. Rossi. Grand: Graph neural diffusion. In *International Conference on Machine Learning*, pages 1407–1418. PMLR, 2021.

[5] H. Gao, M. J. Zahr, and J.-X. Wang. Physics-informed graph neural galerkin networks: A unified framework for solving pde-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 390:114502, 2022.

[6] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[7] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[8] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[9] 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*, 378:686–707, 2019.

[10] M. V. Shivaditya, J. Alves, F. Bugiotti, and F. Magoulès. Graph neural network-based surrogate models for finite element analysis. In *2022 21st International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, pages 54–57. IEEE, 2022.

[11] S. Wang, Y. Teng, and P. Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[12] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[13] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[14] Z. Xiang, W. Peng, and W. Yao. Rbf-mgn: Solving spatiotemporal pdes with physics-informed graph neural network. *arXiv preprint arXiv:2212.02861*, 2022.