Adaptive Deep Learning Method for Solving Partial Differential Equations

Ángel J. Omella\textsuperscript{1} David Pardo\textsuperscript{1,2,3}

\textsuperscript{1}University of the Basque Country (UPV/EHU), Leioa, Spain
\textsuperscript{2}Basque Center for Applied Mathematics (BCAM), Bilbao, Spain
\textsuperscript{3}Ikerbasque, Bilbao, Spain

October 21, 2022

Abstract

We introduce an $r-$adaptive algorithm to solve Partial Differential Equations using a Deep Neural Network. The proposed method restricts to tensor product meshes and optimizes the boundary node locations in one dimension, from which we build two- or three-dimensional meshes. The method allows the definition of fixed interfaces to design conforming meshes, and enables changes in the topology, i.e., some nodes can jump across fixed interfaces. The method simultaneously optimizes the node locations and the PDE solution values over the resulting mesh. To numerically illustrate the performance of our proposed $r-$adaptive method, we apply it in combination with a collocation method, a Least Squares Method, and a Deep Ritz Method. We focus on the latter to solve one- and two-dimensional problems whose solutions are smooth, singular, and/or exhibit strong gradients.

1 Introduction

Deep Learning (DL) \cite{1, 2} is nowadays applied to multiple fields \cite{3}, including biomedical applications \cite{4}, structural health monitoring \cite{5, 6}, and geosteering \cite{7}. Indeed, DL can perform complex tasks with high accuracy without incurring prohibitive computational costs. DL has allowed an essential advance in solving problems where the relationship between input and output data is complex and unknown. For example, merging DL techniques with the Finite Element Method can be used to improve the solution of Partial Differential Equations (PDEs) \cite{8, 9, 10, 11, 12}. In addition, the use of DL to predict PDEs behavior has also raised great interest during the last decade\cite{13, 14, 15}.

To solve a PDE using DL, we define a loss function whose global minimum satisfies the PDE and the boundary conditions (BCs). The selection of the numerical method to solve the PDE, formulated in strong, weak, or ultra-weak form, leads to different definitions of the loss function. Some existing methods used in this context are: the Deep Ritz method (DRM) \cite{16} based on the
minimization of the energy of the PDE solution, Physics-Informed Neural Networks (PINNs) [15] based on collocation methods, the Deep first-order Least-Squares method (DLS) [17], the Deep Galerkin method (DGM) [18], and the \(hp\)-Variational Physical Informed Neural Networks (\(hp\)-VPINNs) [19] based on a Petrov-Galerkin domain decomposition.

The potential of Deep Neural Networks (DNNs) to solve high-dimensional PDEs has been proved, for example, in [20, 21]. For these applications, Monte Carlo methods [22] are used for integration. However, to deal with low-dimensional PDEs, in which the efficiency of Monte Carlo methods is limited compared to other methods, we can resort to standard mesh-based quadrature rules [23]. Since the use of fixed quadrature points may produce overfitting during training [24], we consider piecewise-linear solutions over a mesh.

The accuracy of mesh-based solutions depends on the mesh used to numerically solve the PDE [25, 26, 27]. DNNs can also be employed to adapt the mesh. In particular, [28] proposes a mesh generator based on NNs to grow the number of initial elements to a user-selected one. The algorithm is based on the node probability density function obtained from an error estimate. The authors of [29] propose the use of a DNN to obtain an effective mesh refinement in time-dependent PDEs solved by the finite-element method (FEM). They predict the areas of “interest” at the next time stage from the information of previous times. In the same line, [30] employs recurrent DNNs to refine the mesh. These works use DNNs to adapt the mesh but they resort to traditional methods as FEM or finite differences to find the PDE solution.

Herein, we propose a deep \(r\)-adaptive method to simultaneously optimize the PDE solution and the location of the nodes in the corresponding mesh. In our method, once we have selected the degree of the piecewise-polynomial approximation space and the number of elements in the mesh, we define a set of trainable parameters \(\psi\) that correspond to the location of the mesh nodes. From these variables, we define the mesh \(T_\psi\), which is used as an input of a DNN whose output is the piecewise-polynomial solution \(u_p\). Then, the minimization of the loss function \(L\) automatically and simultaneously adapts the mesh \(T_\psi\) and the corresponding solution \(u_p\). A sketch of this architecture is shown in Figure 1.

![Architecture sketch of our proposed \(r\)-adaptive DNN.](image)

Our proposed method solves the optimal location of the nodes without solving auxiliary non-linear mesh moving PDE (MM-PDE) typical of traditional \(r\)-adaptive methods [31, 32, 33, 34,
The technique of automatic-differentiation [36, 37, 38] used within the back-propagation algorithm [39] allows us to minimize directly the loss function. Another substantial difference with traditional $r-$adaptive methods is that we can modify the topology of the mesh during the adaptation as long as we have access to the derivative of the algorithm that modifies the mesh with respect to each node location. In particular, when restricted to tensor product meshes, it is possible to fix some nodes in the mesh —to ensure conformity with the material properties— while allowing the others to freely move along the mesh.

This work shows the behavior of the $r-$adaptive method in combination with DRM, DLS, and PINNs to solve simple PDEs. For the sake of simplicity, we use piecewise-linear functions over one-dimensional (1D) and two-dimensional (2D) tensor-product meshes to show the potential of the proposed method. The extension to higher-order piecewise-polynomials is straightforward. We can also parametrize more complex geometries applying a mapping from a reference tensor-product mesh to the geometry of the physical space [40], possibly leaving some elements outside the computational domain (e.g., performing techniques similar to the Finite Cell Method [41, 42]). However, if we want to break the tensor product structure of the mesh and allow arbitrary changes in topology, it would require a mesh generator whose derivatives with respect to the node locations are accessible. The computation of the derivatives needed by general mesh-generation algorithms (like Delaunay triangularization [43]) in TensorFlow [44] requires further future study. However, to overcome this restriction, projects like Autograd [45] (and its evolution JAX [46]) are progressing towards automatically differentiate native Python and NumPy functions.

The remainder of the manuscript is organized as follows. Section 2 introduces the model problems and summarizes the numerical methods selected to solve them. Then, section 3 introduces the $r-$adaptive method and explains how we incorporate it within a DNN. Section 4 is devoted to implementation details. Section 5 shows the numerical results. Finally, the last section summarizes the main conclusions and future work.

## 2 Model problems and numerical methods

We consider two problems: a first-order hyperbolic problem and a second-order scalar elliptic problem. For the first (hyperbolic) problem, we introduce two losses, leading to a collocation and a least-squares method. For our second (elliptic) problem, we introduce the Deep Ritz method.

### 2.1 Hyperbolic model problem

Let $\Omega \in \mathbb{R}^d$ be a Lipschitz domain, where $d$ is the spatial dimension, and $\beta(x)$ the velocity field. We define the inflow and outflow boundary parts as $\Gamma_- := \{ x \in \partial \Omega : \beta \cdot n < 0 \}$ and $\Gamma_+ := \{ x \in \partial \Omega : \beta \cdot n > 0 \}$, respectively, with $\partial \Omega = \Gamma_- \cup \Gamma_+$, and $n$ the outward normal vector to $\partial \Omega$. We consider the boundary value problem, governed by the advection-reaction equation:

\[
\begin{aligned}
\nabla \cdot (\beta u) + u(x) &= f & \text{in } \Omega, \\
u &= 0 & \text{on } \Gamma_-,
\end{aligned}
\]  

(1)
where \( f \in L^2(\Omega) \) is the source term and \( u \in H^1_0(\Omega) = \{ u \in H^1(\Omega) \mid u = 0 \text{ on } \Gamma_- \} \) is the solution of the PDE.

Let \( r := \nabla \cdot (\beta u) + u - f \) be the residual. We define the loss for the collocation method as the \( L^1 \)-norm of the residual, i.e.:

\[
L_{\text{col}} := ||r||_{L^1(\Omega)} = \int_{\Omega} |\nabla \cdot (\beta u) + u - f|.
\]

(2)

We also introduce the loss associated to the \( L^2 \)-norm of the residual (least-squares method):

\[
L_{\text{LS}} := ||r||_{L^2(\Omega)} = \sqrt{\int_{\Omega} (\nabla \cdot (\beta u) + u - f)^2}.
\]

(3)

The goal is to solve problem (1) via minimization of the corresponding loss:

\[
u^* = \arg \min_{u \in H^1_0(\Omega)} L(\cdot)(u).
\]

(4)

### 2.2 Elliptic model problem

We consider the open bounded domain \( \Omega \in \mathbb{R}^d \), where \( d \) is the spatial dimension, and we solve the following boundary value problem:

\[
\begin{aligned}
-\nabla \cdot (\sigma \nabla u) &= f & \text{in } \Omega, \\
u &= 0 & \text{on } \Gamma_D, \\
(\sigma \nabla u) \cdot n &= g & \text{on } \Gamma_N,
\end{aligned}
\]

(5)

where \( \sigma \equiv \sigma(x) \in L^\infty(\Omega) \) are material properties, \( u \equiv u(x) \in H^1_0(\Omega) = \{ u \in H^1(\Omega) \mid u = 0 \text{ on } \Gamma_D \} \) is the solution of the second-order elliptic PDE, and \( f \equiv f(x) \in L^2(\Omega) \) is the volumetric source term. \( \Gamma_D \) and \( \Gamma_N \) denote the Dirichlet and Neumann parts of the boundary, respectively, with \( \partial \Omega = \Gamma_D \cup \Gamma_N \), \( g \equiv g(x) \mid_{\Gamma_N} \in H^{-1/2}(\Omega) \), and \( n \) denotes the unitary outward vector to \( \partial \Omega \).

The energy function associated to the symmetric positive definite problem (5) is

\[
L_{\text{Ritz}} = \frac{1}{2} \int_{\Omega} \sigma(\nabla u)^2 - \int_{\Omega} fu - \int_{\Gamma_N} gu.
\]

(6)

The Ritz method [47] finds the solution \( u \) by minimizing \( L_{\text{Ritz}} \) following Equation (4). As proven in [48], the solution \( u \) of the boundary value problem (5) is also the solution of the minimization problem (4).

### 3 Deep \( r \)-adaptive method

We divide the proposed method into three parts, following the sketch shown in Figure 1. The first one determines the location of the points that define the \( r \)-adaptive mesh. The second one uses a DNN to approximate the solution of the PDE at the mesh nodes, imposes the Dirichlet BCs, and interpolates the solution into the selected piecewise-polynomial space. The third part states the minimization problem and show how to compute the loss function. Finally, we explain our minimization algorithm.
3.1 Adaptive mesh

In this work, we discretize the $d$-dimensional domain $\Omega$ by a tensor product mesh $T_\psi$. For each dimension, we build an ordered one-dimensional vector $x$. This vector contains fixed and variable coordinates. Fixed coordinates define the boundary and interfaces to separate materials. The variable coordinates are optimized during the training process to adapt the mesh.

We allow changes in the mesh topology, as illustrated in Figure 2. In these, we observe that the trainable coordinates are allowed to jump from one side to the other of the fixed node $x_3$. The coincidence of coordinates at nodes is also allowed. If this scenario occurs, elements with zero volume are generated and have no contribution to the loss function.

![Figure 2: Representation of a one-dimensional domain, as well as the initial and final meshes. The mesh is built with one interior fixed node and three variables nodes. The movement of the node $x_4$ changes the topology in the final mesh, requiring to reorder the nodes.](image)

3.2 PDE solution approximation

We approximate the solution $u$ of problems (5) and (1) as

$$u \approx u_p := P \circ D \circ u_{NN_\theta}(T_\psi), \quad (7)$$

where $u_{NN_\theta}$ is a DNN with a set of trainable parameters denoted as $\theta$ that maps the mesh $T_\psi$ into a surrogate solution $\tilde{u}$. $D$ is a transformation used to impose Dirichlet BCs in a strong form. Specifically, $D(\tilde{u}) := u_D + \phi_D \tilde{u}$, where $\phi_D$ is a positive function in the interior of the domain and takes value zero on the Dirichlet boundary, and $u_D$ is a function to impose the lift for the case of inhomogeneous Dirichlet BC. Finally, $P$ is the operator that builds the piecewise-linear solution over the mesh $T_\psi$ by interpolating evaluations of the solution at mesh nodes.

3.3 Minimization problem and loss function computation

To find the approximate solution $u_p$ we rewrite the minimization problem from Eq. (4):

$$u \approx u_p(s^*) := \arg \min_{u_p(s), \ s \in S} \mathcal{L}(u_p(s)), \quad (8)$$
where \( s \) denotes the set of all trainable parameters

\[
s := \bigcup_{i,j=1}^{k_s,k_v} \{ \theta_i, \psi_j \},
\]

and \( \theta_i \) are the trainable parameters used to approximate the PDE solution, while \( \psi_j \) are the unknown node locations. Note that by construction, the search space \( \{ u_p(s), s \in S \} \subset H^1_0 \) is not a vector space in general.

We select a Gaussian quadrature rule to compute the integrals that appear in the loss functions —equations (2), (3), and (6)—. The choice of the number of quadrature points per element depends on the degree \( p \) of the piecewise-polynomial function selected to approximate the solution and the source term function \( f \) or the Neumann BC \( g \) when it applies. Integration errors can negatively affect the optimization process, leading to poor approximation of the gradients used to update the DNN trainable parameters.

### 3.4 Guided optimization

The simultaneous minimization of the mesh and solution is a highly non-linear problem and may lead to inaccurate results due to the presence of local minima. For this reason, we decide to guide the optimization process in two stages.

The first stage only optimizes over the values theta, as in (9), which corresponds to optimizing over the values of the DNN at the interpolation points over the fixed, initial mesh. In the second stage, we optimize both the solution and the mesh nodes location. This strategy not only increases the chances of avoiding local minima, but it also often prevents integration problems in the right-hand side \( f \). To distinguish both training phases, we will denote \( u_p \) to the solution after optimization of the solution over a fixed grid. In contrast, \( u_{p,r} \) will denote the solution after simultaneously optimizing both the solution and the node locations.

### 4 Implementation

This section explains the algorithms used to define the tensor product mesh and the deep NN.

#### 4.1 Adaptive mesh implementation

For each spatial dimension, we consider a vector of one-dimensional coordinates \( x \) that will be used to build a tensor product mesh. We consider two types of coordinates: fixed and variable. Among the fixed coordinates, we distinguish between those on the boundary, denoted as \( a, b \in \partial \Omega \), with \( a < b \), and the vector \( x^{fix} \) that stores \( n_{fix} \) fixed coordinates between \( a \) and \( b \). Incorporating interior fixed coordinates allows for the construction of conforming meshes needed to model problems with several materials. On the other hand, the vector of variable coordinates \( x^\delta \) contains a set of \( n_\delta \) coordinates, which are free to move during the training process, allowing for \( r \)-adaptation.

Algorithm 1 computes a one-dimensional ordered vector \( x \) from \( x^{fix} \) and the vector of coordinates \( \psi \) updated during the training. \( \psi := \{ a, x^\delta, b \} \), and we need to map it into the
computational domain after each iteration of the training. The number of entries of \( \mathbf{x} \) denoted as \( n := 2 + n_{f_{x}} + n_{\delta} \) refers to the total number of mesh nodes along one dimension. We start the training from a uniform mesh initializing \( \psi \) as: \( \psi_{i} \leftarrow i, i = 1 \) to \( n_{\delta} + 2 \). The computational cost of the algorithm is \( O(n \log n) \).

Algorithm 1: 1D computation of the coordinate vector: Training coordinates

| Input: | \( \psi, a, b, x^{f_{x}}, n_{f_{x}}, n_{\delta} \) |
| Output: | \( \mathbf{x} (\dim(\mathbf{x}) = n = 2 + n_{f_{x}} + n_{\delta}) \) |
| 1 | \( \psi \leftarrow \text{sort}(\psi) \); \hspace{1cm} // reorder \( \psi \) |
| 2 | \( \psi \leftarrow (\psi - \psi_{1})/(\psi_{n_{\delta}+2} - \psi_{1})(b - a) + a \); \hspace{1cm} // scale \( \psi \) to the computational domain |
| 3 | if \( n_{f_{x}} = 0 \) then |
| 4 | \( \mathbf{x} \leftarrow \psi; \) |
| 5 | else |
| 6 | \( \mathbf{x} \leftarrow \text{sort} (\text{concatenate}(\psi, x^{f_{x}})) \); \hspace{1cm} // add interior fixed coordinates and reorder |
| 7 | return \( \mathbf{x} \); |

Finally, we construct the tensor product mesh \( T_{\psi} \) by the combination of the \( d \) one-dimensional vectors \( \mathbf{x} \). The number of nodes \( n_{T} \) and elements \( n_{K} \) of \( T_{\psi} \) is

\[
n_{T} := \prod_{i=1}^{d} n_{i}, \quad n_{K} := \prod_{i=1}^{d} (n_{i} - 1),
\]

where \( n_{i} \) denotes the dimension of the vector \( \mathbf{x} \) associated to the spatial dimension \( i \).

### 4.2 Deep Neural Network

Once we have the \( n_{T} \) points in the physical space that define the mesh \( T_{\psi} \), we use a feed-forward NN \( u_{NN, \theta}(T_{\psi}) : \mathbb{R}^{d} \rightarrow \mathbb{R} \) composed of \( k \) layers as follows:

\[
u_{NN, \theta} := \ell^{(k)} \circ \ell^{(k-1)} \circ \cdots \circ \ell^{(1)}(T_{\psi}).\tag{11}\]

Each layer \( \ell^{(i)} \) is a non-linear mapping composed of an activation function \( \alpha^{(i)} \) applied component-wise to an affine transformation,

\[
\ell^{(i)}(\cdot) := \alpha^{(i)} \left( W^{(i)\top}(\cdot) + b^{(i)} \right).	ag{12}\]

In this work, we take our activation function to be the sigmoid function \( \alpha^{(i)} = \frac{1}{1 + e^{-x}} \) if \( i \neq k \), and the identity if \( i = k \). \( N_{i} \) is the dimension of the layer output \( \ell^{(i)} \) and is usually understood as the number of neurons in the layer. It is related to the number of trainable parameters, the so-called weights \( W^{(i)} \in \mathbb{R}^{N_{i} \times N_{i-1}} \) and biases \( b^{(i)} \in \mathbb{R}^{N_{i}} \) of the affine transformation. We denote the set of all trainable parameters of the DNN \( u_{NN, \theta} \) as:

\[
\theta := \bigcup_{i=1}^{k} \left\{ (W^{(i)}, b^{(i)}) \right\}.	ag{13}\]
5 Numerical results

We adopt the strategy described in Section 3.4 for training. We first optimize the solution over a fixed mesh (denoted as $u_p$), then we optimize both the mesh and solution to obtain $u_{p,r}$. In both stages, we use ADAM optimizer [49]. The selected feed-forward NN has five hidden layers of ten neurons, each activated with a sigmoid function — see Section 4.2 for details.

Experiment 1 solves the hyperbolic problem with the $r$–adaptive method in combination with residual methods, as explained in Section 2.1. Experiments 2 to 6, summarized in Table 1, focus on solving the elliptic problem introduced in Section 2.2 by the Deep Ritz Method of Eq. 6.

Table 1: Numerical experiments for the elliptic problem of Section 2.2: definition and exact solutions.

| Experiment ID | $u_{exact}$ | $\mathcal{L}_{Ritz}(u_{exact})$ | Domain, material properties, and BCs |
|---------------|-------------|---------------------------------|-----------------------------------|
| Ex. 2         | $x^{0.7}$   | -1.5385                         | $\Gamma_D \bullet \sigma = 1$ $x \in (0, 10)$ $\Gamma_N$ |
| Ex. 3         | $\text{atan}(2x - 10) + \text{atan}(10)$ | -1.5701                         | $\Gamma_D \bullet \sigma = 1$ $x \in (0, 10)$ $\Gamma_N$ |
| Ex. 4         | $\frac{\sin(2\pi x)}{\sigma}$ | -5.8724                         | $\Gamma_D \bullet \sigma = 1$ $x = 0$ $\sigma = 10$ $x = 1$ $\Gamma_D$ |
| Ex. 5         | $x_1^2(x_1 - 1) x_2^2(x_2 - 1)$ | -0.0013                         | $\Gamma_D \bullet \sigma = 1$ $\Gamma_D$ |
| Ex. 6         | $r^2 \sin \left( \frac{\theta}{3} \left( \theta + \frac{\pi}{2} \right) \right) \right)$ | -0.9181                         | $\Gamma_D \bullet \sigma = 1$ $\Gamma_N$ |

[Diagram of domains and material properties]
5.1 Experiment 1: Residual methods

We solve the problem of Equation 1 with \( f = 1 \). The exact solution is \( u = 1 - e^{-x} \). Figure 3 shows the exact and approximated solutions for \( \beta = 10^{-3} \). We consider a mesh of eight elements. The approximated solutions are computed via the minimization of the residuals \( \mathcal{L}_{\text{col}} \) defined in Eq. (2) and \( \mathcal{L}_{\text{LS}} \) in Eq. (3). Figure 3a shows the Gibbs effect for the Least-Square solution \( u_{p,LS} \). The Gibbs phenomenon is not present with the collocation method. When relocating the nodes (see Figure 3b), we obtain superior-quality solutions that are free of Gibbs fluctuations.

![Figure 3: Experiment 1. Exact and approximate solutions for \( \beta = 10^{-3} \) using uniform (left panel) and \( r \)-adaptive (right panel) meshes.](image)

5.2 Experiment 2: Singular solution

The solution of Ex. 2 has a singularity at \( x = 0 \), where the derivative is infinite. Figure 4a shows the solution for a fixed 16-elements mesh, and Figure 4b shows the final solution with our proposed \( r \)-adaptive method. We observe small elements accumulating nearby the singularity, creating a geometrical grid as expected for this type of singularity [50].

Figure 5a shows the error in the loss for a solution over a mesh of 16 elements. We observe a decrease in the loss error values during the second training step, i.e., when we optimize the node locations. This indicates the improvement in accuracy of the solution \( u_{p,r} \) over \( u_p \).

For different number of elements in the mesh, Figure 5b shows the energy-norm error, defined as:

\[
||u - u(\cdot)||_{H^1(\Omega)} = \sqrt{\int_{\Omega} \left( (u_{\text{exact}} - u(\cdot))^2 + (\nabla u_{\text{exact}} - \nabla u(\cdot))^2 \right)}.
\] (14)

We denote as \( u_{\text{FEM}} \) and \( u_{\text{FEM},r} \) the FEM solutions over the uniform and \( r \)-adaptive meshes, respectively. The convergence rate for uniform refinements is 0.2 [51]. Figure 5b confirms this behavior both for the FEM and DL solutions for uniform meshes. When considering \( r \)-adaptive
meshes, the convergence rate dramatically increases (see Figure 5b). However, the convergence rate deteriorates at some point due to either the optimizer and/or considered architecture (see the difference between the FEM and DL solutions on the $r$–adaptive meshes for 16 and 32 elements).

Figure 5: Experiment 2. Loss evolution and convergence of the solutions
5.3 Experiment 3: High-gradient solution

We now consider a solution with a strong gradient. Figure 6a shows the exact and approximated solutions in a 16-element mesh. We observe an accumulation of elements in the zones with high gradients, as expected. Figure 6b shows the loss evolution during the two steps of the training. The decrease in the loss indicates that the solution obtained with the $r-$adaptive mesh is significantly better than the one calculated on a fixed uniform mesh.

![Figure 6a](image1.png)  
(a) Solution on $r-$adaptive mesh.

![Figure 6b](image2.png)  
(b) Loss evolution.

Figure 6: Experiment 3. Exact and approximate solutions and loss evolution.

In this experiment, the method diverges when executing only the one-step optimization without initializing the feed-forward NN on the uniform fixed mesh: The points accumulate in zones with small gradients. The quantity $\int f u$, increases, producing values of the loss $L_{Ritz}$ lower than the best physically possible —the one corresponding to the exact solution—, which indicates an integration error in the right-hand side. Although introducing some fixed points in the mesh solves the problem, we decided to use the two-step training (see Section 3.4) that also resolves the issue.

5.4 Experiment 4: Discontinuous materials

This experiment reveals the capability of the proposed method to solve problems involving different materials. We include a fixed point to separate subdomains with different materials using a conforming mesh. Figure 7 shows the solution and the evolution of the loss during the training process. Since the solution is smooth away from the fixed point, the improvement in the approximated solution accuracy between uniform mesh and the $r-$adaptive solution is lower than that observed in previous experiments.
5.5 **Experiment 5: two-dimensional (2D) smooth solution**

Figure 8 shows the approximated solution on the adapted mesh and the loss evolution for a 2D problem with a smooth solution. We compute the results on a 16x16 elements mesh. We observe that the $r$–adapted mesh is close to symmetric with respect to the line $y = x$. This, together with the low error in the loss, indicate the high quality of the solution.

Figure 8: Experiment 5. Approximate solution and loss evolution.
5.6 Experiment 6: L-shape domain

Figure 9 shows the approximated solution on the adapted mesh and the loss evolution for the so-called L-shape domain problem.

We consider a square reference domain. To properly define the physical L-shape domain, we fix the points $x_1 = 0$ and $x_2 = 0$. In this way, we ensure the conformity of the elements to the L-shape domain. The elements outside the domain have null contribution to the loss integral equal to zero. This example shows that we can consider geometries that go beyond simple tensor-product geometries\(^1\). Figure 9a shows that the final mesh computed over the physical domain is almost symmetric with respect to the line $x = y$ and has fewer elements than the uniform initial one. The reference square domain employs two 16-dimensional vectors of 1D coordinates (producing 192 elements). In contrast, the final adapted mesh only employs 112 elements in the L-shape area of interest (the rest are zero). However, the solution obtained in the adapted mesh constrains lower error than the initial one, as observed by the reduction in the loss value. In the $r$--adapted mesh, the element sizes grow exponentially on both sides from the singularity, increasing the density of elements in the null area of the domain (outside the L-shape).

\[\text{(a) Solution on } r\text{--adaptive mesh.}\]

\[\text{(b) Loss evolution.}\]

Figure 9: Experiment 6. Approximate solution and loss evolution.

6 Conclusions and Future Work

We propose a DL $r$--adaptive method to solve PDEs. The method simultaneously optimizes the mesh and approximated solution. We have implemented the method for one and two spatial dimensions using tensor product meshes. Numerical experiments show promising results for

\(^1\)Nonetheless, we fall short to consider arbitrary geometries
solutions that are smooth, singular, or exhibit strong gradients. The method outperforms the use of uniform meshes.

In this work, we have considered piecewise-linear functions to approximate the solution. The extension to higher-degree piecewise-polynomial functions is straightforward. Moreover, we have restricted meshes with tensor-product topology. The extension to triangular/tetrahedral meshes will be pursued as part of our future effort in the area.

Acknowledgments

David Pardo has received funding from: the European Union’s Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 777778 (MATH-ROCKS); the Spanish Ministry of Science and Innovation projects with references TED2021-132783B-I00, PID2019-108111RB-I00 (FEDER/AEI) and PDC2021-121093-I00 (AEI/Next Generation EU), the “BCAM Severo Ochoa” accreditation of excellence (SEV-2017-0718); and the Basque Government through the BERC 2022-2025 program, the three Elkartek projects 3KIA (KK-2020/00049), EXPERTIA (KK-2021/00048), and SIGZE (KK-2021/00095), and the Consolidated Research Group MATHMODE (IT1456-22) given by the Department of Education.

References

[1] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.

[2] S. Kollmannsberger, D. D’Angella, M. Jokeit, L. Herrmann, Deep Learning in Computational Mechanics: An introductory Course, Springer Cham, 2021. doi:10.1007/978-3-030-76587-3. URL https://doi.org/10.1007/978-3-030-76587-3

[3] L. Deng, D. Yu, Deep Learning: Methods and Applications, Found. Trends Signal Process. 7 (3–4) (2014) 197–387. doi:10.1561/2000000039. URL https://doi.org/10.1561/2000000039

[4] M. Wainberg, D. Merico, A. Delong, B. J. Frey, Deep learning in biomedicine, Nature biotechnology 36 (9) (2018) 829–838. doi:https://doi.org/10.1038/nbt.4233

[5] A. Fernandez-Navamuel, F. Magalhães, D. Zamora-Sánchez, Ángel J Omella, D. Garcia-Sanchez, D. Pardo, Deep learning enhanced principal component analysis for structural health monitoring, Structural Health Monitoring 0 (0) (2022) 14759217211041684. arXiv:https://doi.org/10.1177/14759217211041684, doi:10.1177/14759217211041684. URL https://doi.org/10.1177/14759217211041684

[6] A. Fernandez-Navamuel, D. Zamora-Sánchez, Ángel J. Omella, D. Pardo, D. Garcia-Sanchez, F. Magalhães, Supervised deep learning with finite element simulations for damage identification in bridges, Engineering Structures 257 (2022) 114016.
[7] S. Alyaev, M. Shahriari, D. Pardo, Ángel Javier Omella, D. S. Larsen, N. Jahani, E. Suter, Modeling extra-deep electromagnetic logs using a deep neural network, GEOPHYSICS 86 (3) (2021) E269–E281. arXiv:https://doi.org/10.1190/geo2020-0389.1, doi:10.1190/geo2020-0389.1. URL https://doi.org/10.1190/geo2020-0389.1

[8] I. Brevis, I. Muga, K. G. van der Zee, A machine-learning minimal-residual (ML-MRes) framework for goal-oriented finite element discretizations, Computers & Mathematics with Applications 95 (2021) 186–199, recent Advances in Least-Squares and Discontinuous Petrov–Galerkin Finite Element Methods. doi:https://doi.org/10.1016/j.camwa.2020.08.012. URL https://www.sciencedirect.com/science/article/pii/S0898122120303199

[9] I. Brevis, I. Muga, K. G. van der Zee, Neural control of discrete weak formulations: Galerkin, least-squares and minimal-residual methods with quasi-optimal weights (2022). doi:10.48550/ARXIV.2206.07475. URL https://arxiv.org/abs/2206.07475

[10] M. Paszynski, R. Grzeszczuk, D. Pardo, L. Demkowicz, Deep learning driven self-adaptive hp finite element method, in: M. Paszynski, D. Kranzlmüller, V. V. Krzhiyanovskaya, J. J. Dongarra, P. M. A. Sloot (Eds.), Computational Science – ICCS 2021, Springer International Publishing, Cham, 2021, pp. 114–121.

[11] T. Sluzalec, R. Grzeszczuk, S. Rojas, W. Dżwienel, M. Paszynski, Quasi-optimal hp-finite element refinements towards singularities via deep neural network prediction (2022). doi:10.48550/ARXIV.2209.05844. URL https://arxiv.org/abs/2209.05844

[12] C. Uriarte, D. Pardo, Ángel Javier Omella, A Finite Element based Deep Learning solver for parametric PDEs, Computer Methods in Applied Mechanics and Engineering 391 (2022) 114562. doi:https://doi.org/10.1016/j.cma.2021.114562. URL https://www.sciencedirect.com/science/article/pii/S0045782521007374

[13] C. Beck, M. Hutzenthaler, A. Jentzen, B. Kuckuck, An overview on deep learning-based approximation methods for partial differential equations (2021). arXiv:2012.12348. 1

[14] G. Karniadakis, I. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nature Reviews Physics 3 (6) (2021) 422–440. doi:https://doi.org/10.1038/s42254-021-00314-5.

[15] M. Raissi, P. Perdikaris, G. 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 (2019) 686–707. doi:https://doi.org/10.1016/j.jcp.2018.10.045.
[16] E. Weinan, Y. Bing, The deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems, Communications in Mathematics and Statistics 6 (2018) 1–12. doi:10.1007/s40304-018-0127-z. URL https://doi.org/10.1007/s40304-018-0127-z

[17] Z. Cai, J. Chen, M. Liu, X. Liu, Deep least-squares methods: An unsupervised learning-based numerical method for solving elliptic PDEs, Journal of Computational Physics 420 (2020) 109707. doi:https://doi.org/10.1016/j.jcp.2020.109707. URL https://www.sciencedirect.com/science/article/pii/S0021999120304812

[18] J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, Journal of Computational Physics 375 (2018) 1339–1364. doi:https://doi.org/10.1016/j.jcp.2018.08.029. URL https://www.sciencedirect.com/science/article/pii/S0021999118305527

[19] E. Kharazmi, Z. Zhang, G. E. Karniadakis, hp-VPINNs: Variational physics-informed neural networks with domain decomposition, Computer Methods in Applied Mechanics and Engineering 374 (2021) 113547. doi:https://doi.org/10.1016/j.cma.2020.113547. URL https://www.sciencedirect.com/science/article/pii/S0045782520307325

[20] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, Q. Liao, Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review, International Journal of Automation and Computing 14 (5) (2017) 503–519. doi:10.1007/s11633-017-1054-2. URL https://doi.org/10.1007/s11633-017-1054-2

[21] J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning, Proceedings of the National Academy of Sciences 115 (34) (2018) 8505–8510. arXiv:https://www.pnas.org/content/115/34/8505.full.pdf, doi:10.1073/pnas.1718942115. URL https://www.pnas.org/content/115/34/8505

[22] W. E, J. Han, A. Jentzen, Algorithms for solving high dimensional PDEs: from nonlinear Monte Carlo to machine learning, Nonlinearity 35 (1) (2021) 278–310. doi:10.1088/1361-6544/ac337f. URL https://doi.org/10.1088/1361-6544/ac337f

[23] P. Moin, Fundamentals of Engineering Numerical Analysis, 2nd Edition, Cambridge University Press, 2010. doi:10.1017/CBO9780511781438.

[24] J. A. Rivera, J. M. Taylor, Ángel J. Omella, D. Pardo, On quadrature rules for solving partial differential equations using neural networks, Computer Methods in Applied Mechanics and Engineering 393 (2022) 114710. doi:https://doi.org/10.1016/j.cma.2022.114710. URL https://www.sciencedirect.com/science/article/pii/S0045782522000810
[25] W. Carroll, R. Barker, A theorem for optimum finite-element idealizations, International Journal of Solids and Structures 9 (7) (1973) 883–995. doi:https://doi.org/10.1016/0020-7683(73)90011-5. URL https://www.sciencedirect.com/science/article/pii/0020768373900115

[26] H. G. W. Burchard, D. F. Hale, Piecewise polynomial approximation on optimal meshes, Journal of Approximation Theory 14 (1975) 128–147.

[27] I. Babuška, W. C. Rheinboldt, Analysis of optimal finite-element meshes in $\mathbb{R}^1$, Mathematics of Computation 33 (146) (1979) 435–463. URL http://inis.iaea.org/search/search.aspx?orig_q=RN:10492324

[28] S. Alfonzetti, A finite element mesh generator based on an adaptive neural network, IEEE Transactions on Magnetics 34 (5) (1998) 3363–3366. doi:10.1109/20.717791.

[29] L. Manevitz, A. Bitar, D. Givoli, Neural network time series forecasting of finite-element mesh adaptation, Neurocomputing 63 (2005) 447–463, new Aspects in Neurocomputing: 11th European Symposium on Artificial Neural Networks. doi:https://doi.org/10.1016/j.neucom.2004.06.009. URL https://www.sciencedirect.com/science/article/pii/S0925231204003078

[30] J. Bohn, M. Feischl, Recurrent neural networks as optimal mesh refinement strategies, Computers & Mathematics with Applications 97 (2021) 61–76. doi:https://doi.org/10.1016/j.camwa.2021.05.018. URL https://www.sciencedirect.com/science/article/pii/S089812151200198X

[31] C. J. Budd, W. Huang, R. D. Russell, Adaptivity with moving grids, Acta Numerica 18 (2009) 111–241. doi:10.1017/S096249290900015.

[32] E. Dorfi, L. Drury, Simple adaptive grids for 1-d initial value problems, Journal of Computational Physics 69 (1) (1987) 175–195. doi:https://doi.org/10.1016/0021-9991(87)90161-6. URL https://www.sciencedirect.com/science/article/pii/0021999187901616

[33] W. Huang, R. Russell, Adaptive Moving Mesh Methods, Applied Mathematical Sciences, Springer, New York, NY, 2011. doi:https://doi.org/10.1007/978-1-4419-7916-2.

[34] W. Huang, Y. Ren, R. D. Russell, Moving Mesh Partial Differential Equations (MMPDES) Based on the Equidistribution Principle, SIAM Journal on Numerical Analysis 31 (3) (1994) 709–730. arXiv:https://doi.org/10.1137/0731038, doi:10.1137/0731038. URL https://doi.org/10.1137/0731038

[35] C. J. Budd, A. T. McRae, C. J. Cotter, The scaling and skewness of optimally transported meshes on the sphere, Journal of Computational Physics 375 (2018) 540–564. doi:https://doi.org/10.1016/j.jcp.2018.08.028. URL https://www.sciencedirect.com/science/article/pii/S0021999118305515
[36] B. v. Merriënboer, O. Breuleux, A. Bergeron, P. Lamblin, Automatic Differentiation in ML: Where We Are and Where We Should Be Going, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS’18, Curran Associates Inc., Red Hook, NY, USA, 2018, p. 8771–8781.

[37] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, J. M. Siskind, Automatic Differentiation in Machine Learning: A Survey, J. Mach. Learn. Res. 18 (1) (2017) 5595–5637.

[38] A. Griewank, A. Walther, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, 2nd Edition, no. 105 in Other Titles in Applied Mathematics, SIAM, Philadelphia, PA, 2008.
URL http://bookstore.siam.org/ot105/

[39] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, Nature 323 (1986) 533–536.
doi:https://doi.org/10.1038/323533a0.

[40] P. J. Frey, P. L. George, Mesh Generation, John Wiley & Sons, Ltd, 2008.
doi:https://doi.org/10.1002/9780470611166.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470611166

[41] J. Parvizian, A. Düster, E. Rank, Finite cell method, Computational Mechanics 41 (1) (2007) 121–133.
doi:10.1007/s00466-007-0173-y.
URL https://doi.org/10.1007/s00466-007-0173-y

[42] N. Zander, S. Kollmannsberger, M. Ruess, Z. Yosibash, E. Rank, The finite cell method for linear thermoelasticity, Computers & Mathematics with Applications 64 (11) (2012) 3527–3541.
doi:https://doi.org/10.1016/j.camwa.2012.09.002.
URL https://www.sciencedirect.com/science/article/pii/S0898122112005688

[43] L. Paul Chew, Constrained delaunay triangulations, Algorithmica 4 (1) (1989) 97–108.
doi:10.1007/BF01553881.
URL https://doi.org/10.1007/BF01553881

[44] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL https://www.tensorflow.org/

[45] D. Maclaurin, D. Duvenaud, R. P. Adams, Autograd: Effortless gradients in numpy, in: ICML 2015 AutoML Workshop, Vol. 238, 2015, p. 5.
[46] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, Q. Zhang, JAX: composable transformations of Python+NumPy programs (2022).
URL http://github.com/google/jax

[47] W. Ritz, Über eine neue methode zur lösung gewisser variationsprobleme der mathematischen physik., Journal für die reine und angewandte Mathematik 135 (1909) 1–61.
URL http://eudml.org/doc/149295

[48] C. Johnson, Numerical Solution of Partial Differential Equations by the Finite Element Method, Cambridge U. Press, 1987.

[49] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization (2014).
doi:10.48550/ARXIV.1412.6980.
URL https://arxiv.org/abs/1412.6980

[50] I. Babuška, S. Theofanis, The Finite Element Method and Its Reliability, Numerical mathematics and scientific computation, New York: Oxford University Press, 2001.

[51] W. Gui, I. Babuška, The $h$, $p$ and $h−p$ versions of the finite element method in 1 dimension: Part II. the error analysis of the $h-$ and $h − p$ versions, Numerische Mathematik 49 (6) (1986) 613–657.