Computers and Mathematics with Applications 153 (2024) 33–42

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

r-Adaptive deep learning method for solving partial differential equations

Ángel J. Omella [a,*], David Pardo [a,b,c,*]

[a] University of the Basque Country (UPV/EHU), Leioa, Spain
[b] Basque Center for Applied Mathematics (BCAM), Bilbao, Spain
[c] Ikerbasque, Bilbao, Spain

ARTICLE INFO

Keywords:
Partial differential equations
Adaptive mesh
Neural networks
Deep learning

ABSTRACT

We introduce a Deep Neural Network (DNN) method for solving Partial Differential Equations (PDEs) that simultaneously: (a) constructs an optimal *r*-adapted mesh, i.e., given an initial mesh, it provides optimal node locations, and (b) solves the PDE over the constructed r-adaptive mesh. The node locations are optimized over a set of 1D boundary nodes, and the corresponding 2D quadrilateral meshes are built using tensor product. The method supports the definition of fixed interfaces to create conforming meshes and allows nodes to jump across them, permitting topological variations. To numerically illustrate the performance of our novel *r*-adaptive deep learning method, we apply it in combination with other numerical methods including collocation, Least Squares, and the Deep Ritz method. We solve one- and two-dimensional problems whose solutions are smooth, singular, and/or exhibit strong gradients. Results consistently show the outperformance of employing *r*-adaptivity, while in some cases the improvement is limited by the tensor-product structure of the mesh.

1. Introduction

Deep Learning (DL) [1,2] is nowadays applied to multiple fields [3], including biomedical applications [4], structural health monitoring [5,6], and geosteering [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) [8–12]. In addition, the use of DL to predict PDEs behavior has also raised great interest during the last decade [13–15].

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) [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]. As shown in [24], the use of fixed quadrature points may produce overfitting during training, leading to unacceptable integration errors. This effect may lead to catastrophic results, because the NN will learn to *cheat* a fixed integration rule by building highly nonlinear functions in between quadrature points. To solve this issue, herein, we first select the space over which we want to approximate the solution over a mesh (e.g., piecewise linear), and then we use this information to select an adequate quadrature rule.

The accuracy of mesh-based solutions depends on the mesh used to numerically solve the PDE [25–27]. 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

* Corresponding authors at: University of the Basque Country (UPV/EHU), Leioa, Spain.
E-mail addresses: angeljavier.omella@ehu.eus, ajaome@gmail.com (Á.J. Omella), david.pardo@ehu.eus, dzubiaur@gmail.com (D. Pardo).

https://doi.org/10.1016/j.camwa.2023.11.005

Available online 18 November 2023

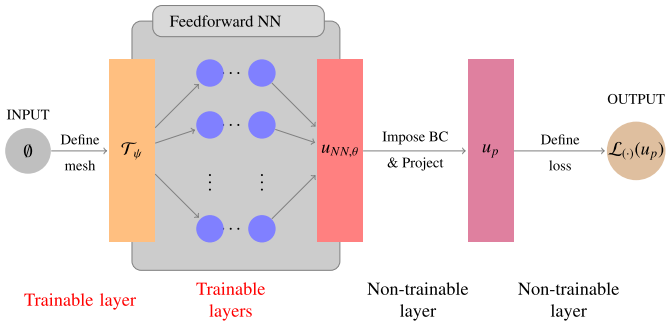


Fig. 1. Architecture sketch of our proposed r-adaptive DNN.

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 proof of concept of a deep r-adaptive method to simultaneously optimize the PDE solution and the location of the nodes in the corresponding mesh. In our novel method, once we have selected the degree of the piecewise-polynomial approximation space and the number of elements in the mesh, we define a simple NN with a set of trainable parameters $\boldsymbol{\psi}$ that correspond to the location of the mesh nodes. From these variables, we define the mesh $\mathcal{T}_\psi$. Then, for each node of $\mathcal{T}_\psi$, we evaluate a DNN to produce $u_{NN,\theta}(\mathcal{T}_\psi)$. The parameters of this feedforward NN, denoted as $\boldsymbol{\theta}$, are independent of the number of elements in $\mathcal{T}_\psi$. The piecewise-polynomial solution u_p is then constructed via interpolation from $u_{NN,\theta}(\mathcal{T}_\psi)$. Finally, the minimization of the loss function $\mathcal{L}$ automatically and simultaneously adapts the mesh $\mathcal{T}_\psi$ through the optimization of $\boldsymbol{\psi}$ and the corresponding solution u_p through the optimization of $\boldsymbol{\theta}$. A sketch of this architecture is shown in Fig. 1. Note that the proposed method does not require input – it is an empty dataset – and we define the initial mesh by initializing the trainable parameters $\boldsymbol{\psi}$.

Our proposed method solves the quasi optimal location of the nodes without solving auxiliary non-linear mesh moving PDE (MMPDE) typical of traditional r-adaptive methods [31–35]. The technique of automatic-differentiation [36–38] 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, this would be insufficient for more complex domains. To break the tensor product structure of the mesh and allow arbitrary changes in topology, we 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. To overcome this restriction, projects like Autograd [45] (and its evolution JAX [46]) are progressing towards automatically differentiate native Python and NumPy functions. We will explore this venue in future works.

While not investigated here, the resulting r-adaptive method could be applied to parametric PDEs. This may that may be specially useful in context of inverse problems. It can also be employed in combination with existing FEM solvers equipped with automatic differentiation.

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 the velocity field $\boldsymbol{\beta} \in [\text{Lip}(\Omega)]^d$, where $\text{Lip}(\Omega)$ denotes the space spanned by Lipschitz continuous functions [47]. We define the inflow and outflow (including flow-aligned) boundary parts as

$$\Gamma_- := \{x \in \partial\Omega : \boldsymbol{\beta} \cdot \boldsymbol{n} < 0\}, \text{ and}$$
$$\Gamma_+ := \{x \in \partial\Omega : \boldsymbol{\beta} \cdot \boldsymbol{n} \geq 0\},$$

respectively, with $\partial\Omega = \Gamma_- \cup \Gamma_+$, and $\boldsymbol{n}$ the outward normal vector to $\partial\Omega$. We consider the boundary value problem, governed by the advection-reaction equation:

$$\begin{cases} \nabla \cdot (\boldsymbol{\beta} u) + u = f & \text{in } \Omega, \\ u = 0 & \text{on } \Gamma_-, \end{cases} \tag{1}$$

where $f \in L^2(\Omega)$ is the source term and $u \in V$ is the solution of the PDE, where V is the so called graph space with homogeneous inflow condition

$$V := \{u \in L^2(\Omega) \mid \beta \cdot \nabla u \in L^2(\Omega) \text{ and } u = 0 \text{ on } \Gamma_-\}.$$

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

$$\mathcal{L}_{col} := ||r||_{L^1(\Omega)} = \int_\Omega |\nabla \cdot (\boldsymbol{\beta} u) + u - f|. \tag{2}$$

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

$$\mathcal{L}_{LS} := ||r||_{L^2(\Omega)} = \sqrt{\int_\Omega (\nabla \cdot (\boldsymbol{\beta} u) + u - f)^2}. \tag{3}$$

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

$$u^* = \arg\min_{u \in V} \mathcal{L}_{(\cdot)}(u). \tag{4}$$

2.2. Elliptic model problem

Let $\Omega \in \mathbb{R}^d$ be an open bounded domain, where d is the spatial dimension. We consider the following boundary value problem:

$$\begin{cases} -\nabla \cdot (\sigma \nabla u) = f & \text{in } \Omega, \\ u = 0 & \text{on } \Gamma_D, \\ (\sigma \nabla u) \cdot \boldsymbol{n} = g & \text{on } \Gamma_N, \end{cases} \tag{5}$$

where $\sigma \equiv \sigma(\boldsymbol{x}) \in L^\infty(\Omega)$ are material properties,

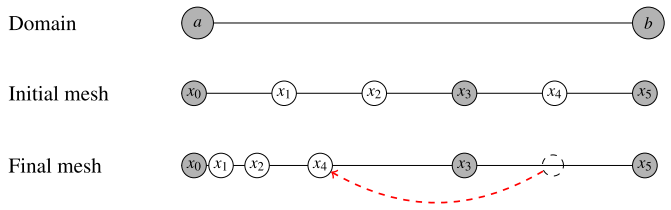


Fig. 2. Representation of a one-dimensional domain, as well as the initial and final meshes. The mesh is built with one interior fixed node and three variables nodes. The movement of the node x_4 changes the topology in the final mesh, requiring to reorder the nodes.

$$u \equiv u(\boldsymbol{x}) \in H^1_{0,D}(\Omega) = \{u \in H^1(\Omega) \,|\, u = 0 \text{ on } \Gamma_D\}$$

is the solution of the second-order elliptic PDE, and $f \equiv f(\boldsymbol{x}) \in L^2(\Omega)$ is the volumetric source term. Γ_D and Γ_N denote the Dirichlet and Neumann parts of the boundary, respectively, with $\partial\Omega = \Gamma_D \cup \Gamma_N$, $g \equiv g(\boldsymbol{x})\Big|_{\Gamma_N} \in H^{-1/2}(\Gamma_N)$, and $\boldsymbol{n}$ denotes the unitary outward vector to $\partial\Omega$. The energy function associated to the symmetric positive definite problem (5) is

$$\mathcal{L}_{Ritz} = \frac{1}{2}\int_\Omega \sigma(\nabla u)^2 - \int_\Omega fu - \int_{\Gamma_N} gu. \tag{6}$$

The Ritz method [48] finds the solution u by solving the minimization problem:

$$u^* = \underset{u \in H^1_{0,D}}{\arg\min}\, \mathcal{L}_{Ritz}(u). \tag{7}$$

As proven in [49], the solution u of the boundary value problem (5) is also the solution of the minimization problem (7).

3. Deep *r*-adaptive method

We divide the proposed method into three parts, following the sketch shown in Fig. 1. The first one does not have an input and 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 shows 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 Ω by a tensor product mesh $\mathcal{T}_\psi$. For each dimension, we build an ordered one-dimensional vector $\boldsymbol{x}$. 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 Fig. 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.

In this training block, the set of trainable parameters $\boldsymbol{\psi}$ defines the location of the mesh nodes. Then, we use a meshing algorithm – trivial for tensor product meshes – to generate the mesh $\mathcal{T}_\psi$.

3.2. PDE solution approximation

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

$$u \approx u_p := P \circ D \circ u_{NN,\theta}(\mathcal{T}_\psi), \tag{8}$$

where $u_{NN,\theta}$ is a DNN[1] with a set of trainable parameters denoted as $\boldsymbol{\theta}$ that maps any point $\boldsymbol{x} \in \mathbb{R}^d$, where d is the spatial dimension of Ω, into a surrogate solution $\tilde{u}$. Notice that the number of trainable parameters $\boldsymbol{\theta}$ is independent of the number of elements considered in the mesh and remains constant in our implementation. We can evaluate $u_{NN,\theta} : \mathbb{R}^d \to \mathbb{R}$ at any point $\boldsymbol{x}$. However we will only use it to evaluate over the degrees of freedom of the mesh. D is a transformation used to impose Dirichlet BCs in a strong form. Specifically, $D(\tilde{u}) := u_D + \phi_D \tilde{u}$, where ϕ_D is a positive function in the interior of the domain that 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 $\mathcal{T}_\psi$ 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 problems from Eq. (4) and Eq. (7):

$$u \approx u_p(\boldsymbol{s}^*) := \underset{\boldsymbol{s} \in S}{\arg\min}\, \mathcal{L}_{(\cdot)}\left(u_p(\boldsymbol{s})\right), \tag{9}$$

where $\boldsymbol{s}$ denotes the set of all trainable parameters, that in general are:

$$\boldsymbol{s} := \bigcup_{i,j=1}^{k_\theta, k_\psi} \{\theta_i, \psi_j\}, \tag{10}$$

and θ_i are the trainable parameters used to approximate the PDE solution, while ψ_j are the unknown node locations. Note that by construction, the space $\{u_p(\boldsymbol{s}) : s \in S\} \subset H^1_{0,D}$ 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. If f or g are highly non linear, we would need to introduce a higher-order quadrature rule to minimize the integration error. As in Finite Element Methods (FEM), integration errors can negatively affect the solution.

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.

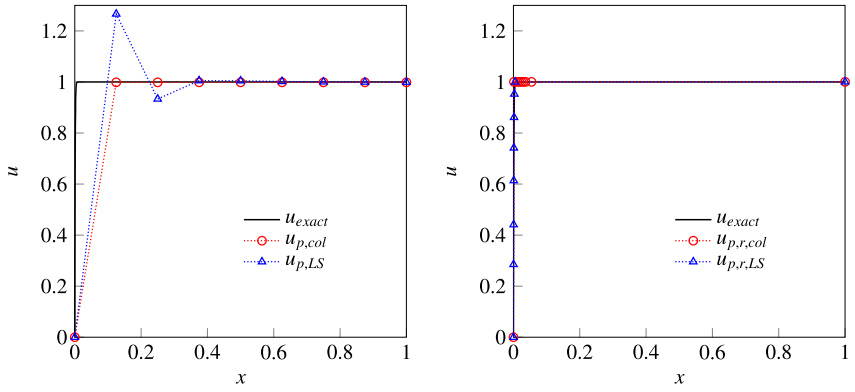
The first stage only optimizes over the values theta, as in (10), which corresponds to optimizing over the values of the DNN at the interpolation points over the fixed, initial mesh. In the second stage, we optimize both the solution and the mesh nodes location. This strategy not only increases the chances 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.

Algorithm 1 computes the location of the adapted mesh nodes $\boldsymbol{\psi}$ and the trainable parameters θ for the DNN $u_{NN,\theta}$ described in Section 4.2 from a given initial mesh $\boldsymbol{\psi}_0$.

4. Implementation

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

[1] Several different architectures can be considered to approximate the solution. For the sake of simplicity, we propose the use of a feedforward NN that is detailed in Section 4.2.



(a) Solution with uniform mesh. (b) Solution with $r-$adaptive mesh.

Fig. 3. Experiment 1. Exact and approximate solutions for $\beta = 10^{-3}$ using uniform (left panel) and r-adaptive (right panel) meshes.

Algorithm 1: Two-stages training.

Input: $\boldsymbol{\psi}_0$
Output: $\boldsymbol{\psi}, \boldsymbol{\theta}$

$\boldsymbol{\psi} \leftarrow \boldsymbol{\psi}_0$; // initialize r-adaptative layer
for *each i-hidden layer in DNN* **do**
  $\boldsymbol{W}^{(i)} \leftarrow$ glorot_uniform_initializer() ; // initialize the weights
  $\boldsymbol{b}^{(i)} \leftarrow$ zeros_initializer() ; // initialize the bias
$\boldsymbol{\theta} \leftarrow \bigcup\{\boldsymbol{W}^{(i)}, \boldsymbol{b}^{(i)}\}$;
$\boldsymbol{\theta} \leftarrow$ train the network solving: $u_p(\boldsymbol{\theta}^*) := \arg\min_{\boldsymbol{\theta}\in\Theta} \mathcal{L}_{(\cdot)}\left(u_p(\boldsymbol{\theta})\right)$;
$\boldsymbol{\psi}, \boldsymbol{\theta} \leftarrow$ train the network solving: $u_p(\boldsymbol{\psi}^*, \boldsymbol{\theta}^*) := \arg\min_{\boldsymbol{\psi}\in\Psi, \boldsymbol{\theta}\in\Theta} \mathcal{L}_{(\cdot)}\left(u_{p,r}(\boldsymbol{\psi}, \boldsymbol{\theta})\right)$;
return $\boldsymbol{\psi}, \boldsymbol{\theta}$;

Algorithm 2: 1D computation of the coordinate vector: Training coordinates.

Input: $\boldsymbol{\psi}, a, b, \boldsymbol{x}^{fix}, n_{fix}, n_\delta$
Output: $\boldsymbol{x}$ ($\dim(\boldsymbol{x}) = n = 2 + n_{fix} + n_\delta$)

$\boldsymbol{\psi} \leftarrow$sort($\boldsymbol{\psi}$) ; // reorder $\boldsymbol{\psi}$
$\boldsymbol{\psi} \leftarrow (\boldsymbol{\psi} - \psi_1)/(\psi_{n_\delta+2} - \psi_1)(b-a) + a$; // scale $\boldsymbol{\psi}$ to the computational domain
if $n_{fix} = 0$ **then**
  $\boldsymbol{x} \leftarrow \boldsymbol{\psi}$;
else
  $\boldsymbol{x} \leftarrow$ sort(concatenate($\boldsymbol{\psi}, \boldsymbol{x}^{fix}$)) ; // add interior fixed coordinates and reorder
return $\boldsymbol{x}$;

4.1. Adaptive mesh implementation

For each spatial dimension, we consider a vector of one-dimensional coordinates $\boldsymbol{x}$ 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 $\boldsymbol{x}^{fix}$ 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, as occurs with experiment 4 in Section 5.3. The fixed points are also helpful to set some geometries with interior corners as that of experiment 6 in Section 5.6, which requires one fixed point per dimension to ensure the L-shape domain. On the other hand, the vector of variable coordinates $\boldsymbol{x}^\delta$ contains a set of n_δ coordinates, which are free to move during the training process, allowing for r-adaptation.

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

Finally, we construct the tensor product mesh $\mathcal{T}_\psi$ by the combination of the d one-dimensional vectors $\boldsymbol{x}$. The number of nodes $n_\mathcal{T}$ and elements n_K of $\mathcal{T}_\psi$ is

$$n_\mathcal{T} := \prod_{i=1}^{d} n_i, \quad n_K := \prod_{i=1}^{d} (n_i - 1), \tag{11}$$

where n_i denotes the dimension of the vector $\boldsymbol{x}$ associated to the spatial dimension i.

4.2. Deep neural network

Let $\boldsymbol{x} \in \mathbb{R}^d$ be a point in a space of dimension d. We define a feed-forward NN $u_{NN,\theta}(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}$ composed of k layers as follows:

$$u_{NN,\theta} := \ell^{(k)} \circ \ell^{(k-1)} \circ \ldots \circ \ell^{(1)}(x). \tag{12}$$

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

$$\ell^{(i)}(\cdot) := \alpha^{(i)}\left(\boldsymbol{W}^{(i)}(\cdot) + \boldsymbol{b}^{(i)}\right). \tag{13}$$

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 $\boldsymbol{W}^{(i)} \in \mathbb{R}^{N_i \times N_{i-1}}$ and biases $\boldsymbol{b}^{(i)} \in \mathbb{R}^{N_i}$ of the affine transformation. We denote the set of all trainable parameters of the DNN $u_{NN,\theta}$ as:

$$\boldsymbol{\theta} := \bigcup_{i=1}^{k} \left\{\left(\boldsymbol{W}^{(i)}, \boldsymbol{b}^{(i)}\right)\right\}. \tag{14}$$

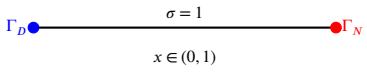
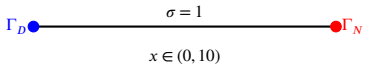
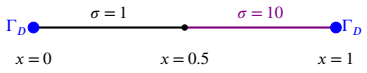
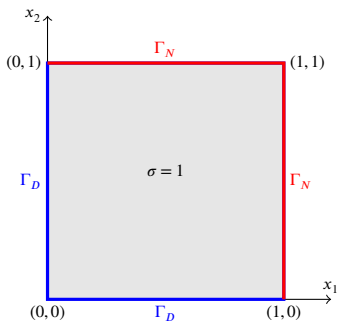
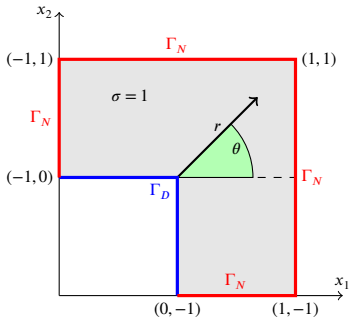
4.3. Optimizer

We use ADAM optimizer, which is a stochastic gradient descent (SGD) method with adaptive learning rate (see [50]).

As an attempt to improve the convergence of the method, and following the ideas exposed in [51,52], we have implemented the option to enrich the optimizer using the damped Least Squares method. This method solves the weights of the last layer using the damped Least Squares method. The rest of the trainable parameters are updated using ADAM, although other optimizers, such as SGD or RMSprop, could be considered.

When we use the damped Least Squares method, we consider architectures with less number of hidden layers; however, the last layer incorporates more neurons. In particular, we pass from a feed-forward

Table 1
Summary of the numerical experiments for the elliptic boundary problem defined in Eq. (5): Identification name, exact solution, exact energy of the loss (6), and a sketch with the domain, boundary conditions, and material properties.

Experiment ID	u_{exact}	$\mathcal{L}_{Ritz}(u_{exact})$	Domain, material properties, and BCs
Ex. 2	$x^{0.7}$	-0.6125	
Ex. 3	$\text{atan}(2x-10)+\text{atan}(10)$	-1.5701	
Ex. 4	$\frac{\sin(2\pi x)}{\sigma}$	-5.8724	
Ex. 5	$(\text{atan}(10x-0.5)+\text{atan}(0.5))\cdot$ $(\text{atan}(10y-0.5)+\text{atan}(0.5))$	-34.3027	
Ex. 6	$r^{\frac{2}{3}}\sin\left(\frac{2}{3}\left(\theta+\frac{\pi}{2}\right)\right)$	-0.9181	

NN with five hidden layers of ten neurons each activated with a sigmoid function – see Section 4.2 for details – to a last layer with 100 neurons preceded by two hidden layers with 10 and 20 neurons, respectively. As we comment in Section 3.2, other architectures can be considered.

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 $u_{p,r}$.

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).

5.1. Experiment 1: residual methods

Let $\Omega = (0,1)$. We solve the problem of Equation (1) with $f(x) = 1$. The exact solution is $u(x) = 1 - e^{\frac{-x}{\beta}}$. Fig. 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 $\mathcal{L}_{col}$ defined in Eq. (2) and $\mathcal{L}_{LS}$ in Eq. (3). Fig. 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 Fig. 3b), we obtain superior-quality solutions that are free of Gibbs fluctuations.

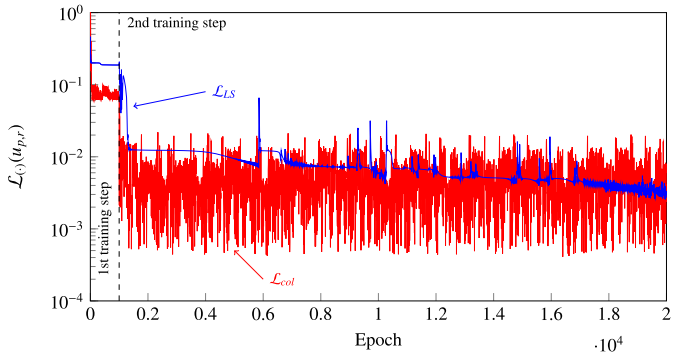
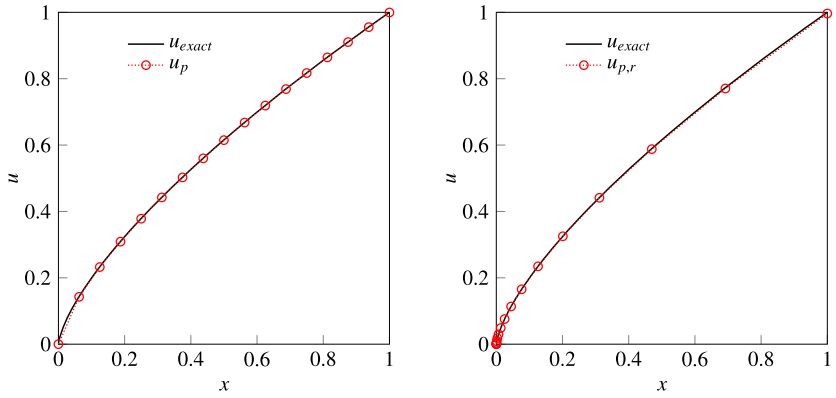


Fig. 4. Experiment 1. Loss evolution of $\mathcal{L}_{col}$ and $\mathcal{L}_{LS}$.

Table 2 lists the L_1 and L_2 relative errors in percentage for the solutions obtained with the two methods in the uniform and r-adapted meshes. The sharp drop in errors experienced when r-adapting the mesh is noticeable (Fig. 4). In addition, we observe better performance in the LS method than in the collocation one.

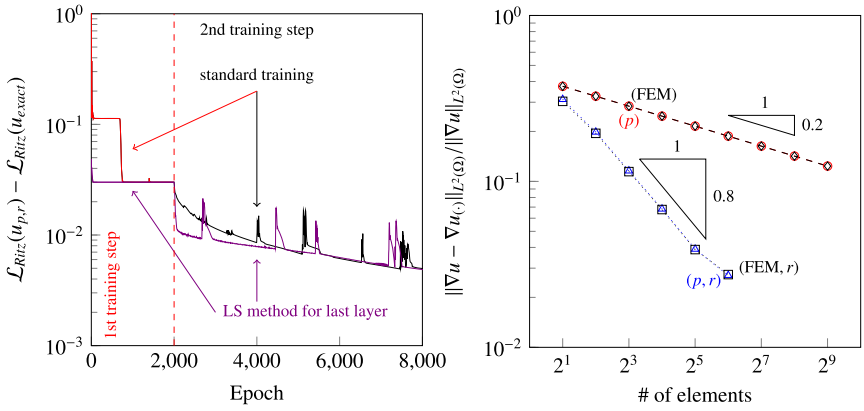
5.2. Experiment 2: singular solution

The solution of Ex. 2 (see Table 1) to the elliptic boundary problem of Eq. (5) has a singularity at $x = 0$, where the derivative is infinite. Fig. 5a shows the solution for a fixed 16-elements mesh, and Fig. 5b



(a) Solution with uniform mesh. (b) Solution with $r-$adaptive mesh.

Fig. 5. Experiment 2. Solutions exact and approximate at the end of each training step.



(a) Loss evolution for a mesh with 32 elements. (b) Relative error in the H^1-seminorm.

Fig. 6. Experiment 2. Loss evolution and convergence of the solutions.

Table 2
Relative errors in norm L_1 and L_2 for the collocation and Least Square solutions in the uniform and r-adapted meshes.

	L_1 relative error in percentage $100\% \times \frac{\|\|u_{(\cdot)} - u_{exact}\|\|_{L^1(\Omega)}}{\|\|u_{exact}\|\|_{L^1(\Omega)}}$	L_2relative error in percentage $100\% \times \frac{\|\|u_{(\cdot)} - u_{exact}\|\|_{L^2(\Omega)}}{\|\|u_{exact}\|\|_{L^2(\Omega)}}$
$u_{p,col}$	6.29%	20.08%
$u_{p,LS}$	6.2%	18.6%
$u_{p,r,col}$	0.05%	0.56%
$u_{p,r,LS}$	0.02%	0.04%

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 [53].

Fig. 6a compares the error in the loss for a solution over a mesh of 32 elements using a standard training and the damped Least squares method used to compute the last layer weights, as described in Section 4.3. 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.

The error in the H^1-seminorm is given by:

$$||\nabla u - \nabla u_{(\cdot)}||_{L^2(\Omega)} := \sqrt{\int_\Omega |\nabla u_{exact} - \nabla u_{(\cdot)}|^2} = \sqrt{2(\mathcal{L}_{Ritz}(u_{(\cdot)}) - \mathcal{L}_{Ritz}(u_{exact}))}. \tag{15}$$

Fig. 6b shows the relative error in the H^1-seminorm for different number of elements in the mesh.

We denote as u_{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 [54]. Fig. 6b confirms this behavior both for the FEM and DL solutions for uniform meshes. When considering r-adaptive meshes, the convergence rate dramatically increases (see Fig. 6b).

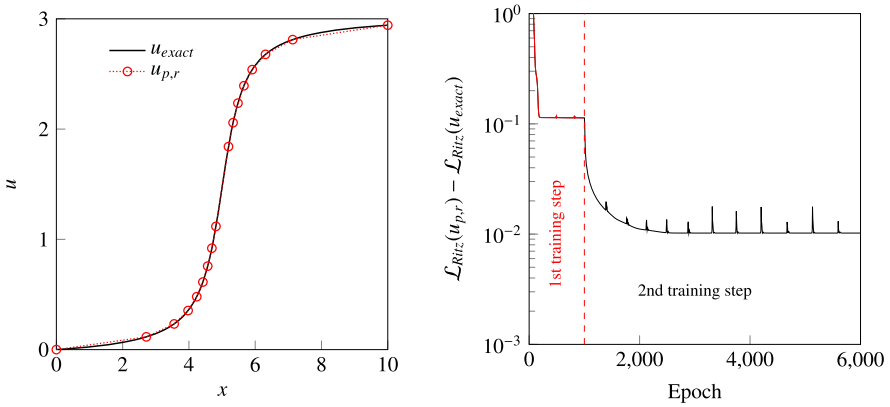
5.3. Experiment 3: high-gradient solution

We now consider a solution with a strong gradient. Fig. 7a 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. Fig. 7b 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. We have obtained for the uniform mesh a 26.8% of relative error in the H^1-seminorm that decreases to 8.1% in the r-adapted mesh.

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 $\int_\Omega fu$, increases, producing values of the loss $\mathcal{L}_{Ritz}$ 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.

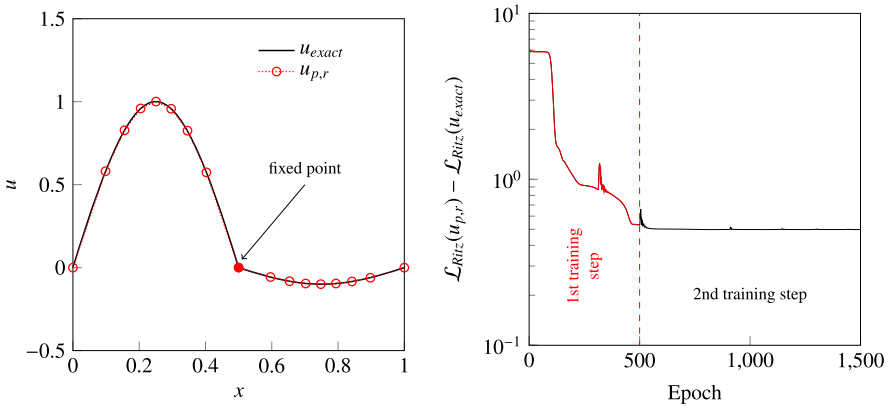
5.4. Experiment 4: discontinuous materials

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. Fig. 8 shows the solution and the evolution of the loss during the train-



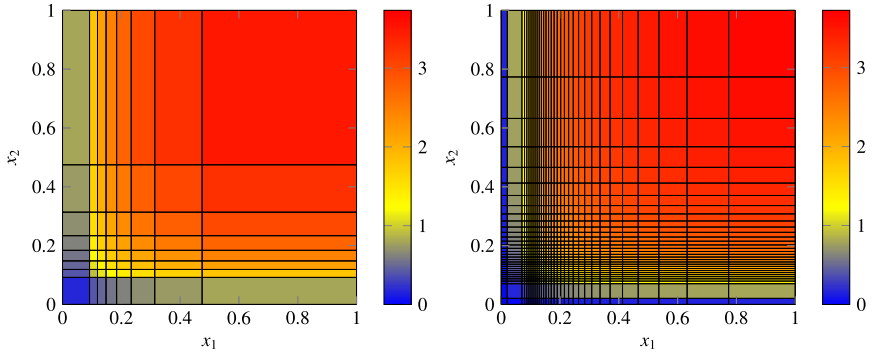
(a) Solution on $r-$adaptive mesh. (b) Loss evolution.

Fig. 7. Experiment 3. Exact and approximate solutions and loss evolution.



(a) Solution on $r-$adaptive mesh. (b) Loss evolution.

Fig. 8. Experiment 4. Exact and approximate solutions and loss evolution.



(a) Solution on $r-$adaptive mesh of $8x8$. (b) Solution on $r-$adaptive mesh of $32x32$.

Fig. 9. Experiment 5. Approximate solution on r-adaptive meshes of $8x8$ and $32x32$ elements.

ing 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.

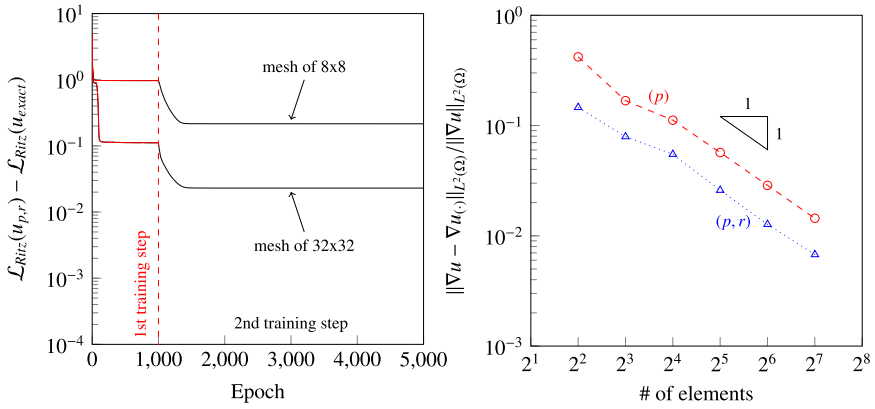
5.5. Experiment 5: two-dimensional (2D) high-gradient solution

In this experiment, we have used the damped Least squares method described in Section 4.2. Fig. 9a and Fig. 9b show the approximated solution on the adapted meshes of 8x8 and 32x32 elements, respectively. We observe that the r-adapted meshes are close (almost) to symmetric with respect to the line $y = x$. As in Ex. 3 (see section 5.3), we observe, as expected, an accumulation of small elements in the zones with high gradients.

Fig. 10a shows the evolution of the loss for the two different meshes. We observe how the loss decreases in the second training step with respect to the value obtained in the first training step. Fig. 10b reveals that the convergence of the method in the energy norm is adequate with a slope equal to one, since it matches with the degree of the piecewise polynomial used for the approximation of the PDE solution. The r-adapted meshes attain similar energy values with half the number of elements per dimension than the uniform mesh.

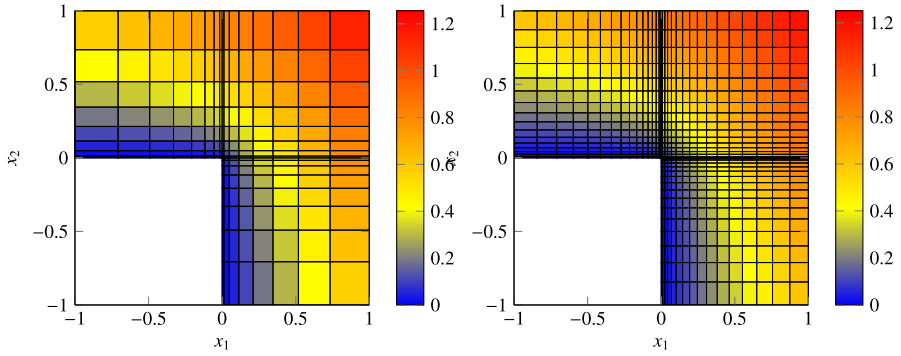
5.6. Experiment 6: L-shape domain

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



(a) Loss evolution for two different meshes. (b) Relative error in the H^1-seminorm.

Fig. 10. Experiment 5. Loss evolution and convergence of the solutions.



(a) Solution on $r-$adaptive mesh (initial 16x16). (b) Solution on $r-$adaptive mesh (initial 32x32).

Fig. 11. Experiment 6. Approximate solution on r-adaptive mesh computed from initial meshes of 16x16 and 32x32 elements.

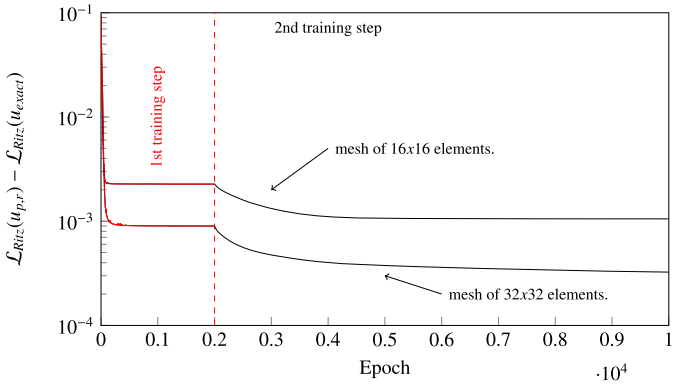


Fig. 12. Experiment 6. Loss evolution for two different meshes.

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.[2] Fig. 11 shows the solutions in the r-adapted mesh obtained for an initial mesh of $16x16$ elements and for $32x32$ elements. The final mesh computed over the physical domain is almost symmetric with respect to the line $x = y$ and remains constant the number of elements inside the L-shape domain than the initial mesh. In both r-adapted meshes, the element sizes grow exponentially on both sides from the singularity.

Fig. 12 shows the loss evolution for the two different meshes. We observe how the loss decreases in the second training step. However, this improvement is small as revealed by the errors in Table 3. This occurs because, in the L-shape problem, the r-adapted mesh should be graded in radial direction toward the singularity at the corner (see [53]). Unfortunately, due to the rigid structure of the employed tensor product mesh, we are unable to adapt radially towards the singularity. As future work, we shall explore other meshes that allow greater freedom in adaptation, e.g., triangular meshes.

Table 3
Absolute error and relative error in H_1-seminorm for the initial meshes of $16x16$ and $32x32$ elements and for its corresponding r-adapted meshes.

	Absolute error in H_1-seminorm	Relative error in H_1-seminorm
initial mesh $16x16$	0.067	4.97%
r-adapted mesh	0.046	3.39%
initial mesh $32x32$	0.042	3.13%
r-adapted mesh	0.025	1.88%

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 solutions that are smooth, singular, or exhibit strong gradients. The method outperforms the use of uniform meshes. However, for some problems, we observe the need for more complex meshes than tensor-product ones to further minimize the error.

The extension to triangular/tetrahedral meshes will be pursued as part of our future effort in the area. In this work, we have considered

[2] Nonetheless, we fall short to consider arbitrary geometries.

piecewise-linear functions to approximate the solution. The extension to higher-degree piecewise-polynomial functions is straightforward. In addition, we shall also explore the use of this method for solving parametric PDEs in context of inverse problems.

Data availability

No data was used for the research described in the article.

Acknowledgements

This work has received funding from: the Spanish Ministry of Science and Innovation projects with references TED2021-132783B-I00, PID2019-108111RB-I00 (FEDER/AEI) and PDC2021-121093-I00 (MCIN/AEI/10.13039/501100011033/Next Generation EU), the "BCAM Severo Ochoa" accreditation of excellence CEX2021-001142-S/MICIN/AEI/10.13039/501100011033; the Spanish Ministry of Economic and Digital Transformation with Misiones Project IA4TES (MIA.2021.M04.008/NextGenerationEU PRTR); 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 Itroductory Course, Springer, Cham, 2021.
[3] L. Deng, D. Yu, Deep learning: methods and applications, Found. Trends Signal Process. 7 (3–4) (2014) 197–387, https://doi.org/10.1561/2000000039.
[4] M. Wainberg, D. Merico, A. Delong, B.J. Frey, Deep learning in biomedicine, Nat. Biotechnol. 36 (9) (2018) 829–838, 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, Struct. Health Monit. 21 (4) (2022) 1710–1722, 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, Eng. Struct. 257 (2022) 114016, https://doi.org/10.1016/j.engstruct.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, 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, Comput. Math. Appl. 95 (2021) 186–199, https://doi.org/10.1016/j.camwa.2020.08.012, Recent Advances in Least-Squares and Discontinuous Petrov–Galerkin Finite Element Methods.
[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, https://doi.org/10.48550/ARXIV.2206.07475, https://arxiv.org/abs/2206.07475, 2022.
[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. Krzhizhanovskaya, 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. Dzwinel, M. Paszynski, Quasi-optimal *hp*-finite element refinements towards singularities via deep neural network prediction, https://doi.org/10.48550/ARXIV.2209.05844, https://arxiv.org/abs/2209.05844, 2022.
[12] C. Uriarte, D. Pardo, A. Omella, A finite element based deep learning solver for parametric PDEs, Comput. Methods Appl. Mech. Eng. 391 (2022) 114562, https://doi.org/10.1016/j.cma.2021.114562.
[13] C. Beck, M. Hutzenthaler, A. Jentzen, B. Kuckuck, An overview on deep learning-based approximation methods for partial differential equations, ArXiv, arXiv:2012.12348 [abs], 2020.
[14] G. Karniadakis, I. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (6) (2021) 422–440, 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, J. Comput. Phys. 378 (2019) 686–707, 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, Commun. Math. Stat. 6 (2018) 1–12, 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, J. Comput. Phys. 420 (2020) 109707, https://doi.org/10.1016/j.jcp.2020.109707.
[18] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364, https://doi.org/10.1016/j.jcp.2018.08.029.
[19] E. Kharazmi, Z. Zhang, G.E. Karniadakis, hp-VPINNs: variational physics-informed neural networks with domain decomposition, Comput. Methods Appl. Mech. Eng. 374 (2021) 113547, https://doi.org/10.1016/j.cma.2020.113547.
[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, Int. J. Autom. Comput. 14 (5) (2017) 503–519, 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, Proc. Natl. Acad. Sci. 115 (34) (2018) 8505–8510, https://doi.org/10.1073/pnas.1718942115, 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, https://doi.org/10.1088/1361-6544/ac337f.
[23] P. Moin, Fundamentals of Engineering Numerical Analysis, 2nd edition, Cambridge University Press, 2010.
[24] J.A. Rivera, J.M. Taylor, Ángel J. Omella, D. Pardo, On quadrature rules for solving partial differential equations using neural networks, Comput. Methods Appl. Mech. Eng. 393 (2022) 114710, https://doi.org/10.1016/j.cma.2022.114710.
[25] W. Carroll, R. Barker, A theorem for optimum finite-element idealizations, Int. J. Solids Struct. 9 (7) (1973) 883–895, https://doi.org/10.1016/0020-7683(73)90011-5.
[26] H.G.W. Burchard, D.F. Hale, Piecewise polynomial approximation on optimal meshes, J. Approx. Theory 14 (1975) 128–147.
[27] I. Babuška, W.C. Rheinboldt, Analysis of optimal finite-element meshes in R^1, Math. Comput. 33 (146) (1979) 435–463, 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 Trans. Magn. 34 (5) (1998) 3363–3366, https://doi.org/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, https://doi.org/10.1016/j.neucom.2004.06.009, New Aspects in Neurocomputing: 11th European Symposium on Artificial Neural Networks.
[30] J. Bohn, M. Feischl, Recurrent neural networks as optimal mesh refinement strategies, Comput. Math. Appl. 97 (2021) 61–76, https://doi.org/10.1016/j.camwa.2021.05.018.
[31] C.J. Budd, W. Huang, R.D. Russell, Adaptivity with moving grids, Acta Numer. 18 (2009) 111–241, https://doi.org/10.1017/S0962492906400015.
[32] E. Dorfi, L. Drury, Simple adaptive grids for 1 - D initial value problems, J. Comput. Phys. 69 (1) (1987) 175–195, https://doi.org/10.1016/0021-9991(87)90161-6.
[33] W. Huang, R. Russell, Adaptive Moving Mesh Methods, Applied Mathematical Sciences, Springer, New York, NY, 2011.
[34] W. Huang, Y. Ren, R.D. Russell, Moving mesh partial differential equations (MMPDES) based on the equidistribution principle, SIAM J. Numer. Anal. 31 (3) (1994) 709–730, 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, J. Comput. Phys. 375 (2018) 540–564, https://doi.org/10.1016/j.jcp.2018.08.028.
[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, pp. 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, Other Titles in Applied Mathematics, vol. 105, SIAM, Philadelphia, PA, 2008, 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, https://doi.org/10.1038/323533a0.
[40] P.J. Frey, P.L. George, Mesh Generation, John Wiley & Sons, Ltd, 2008.
[41] J. Parvizian, A. Düster, E. Rank, Finite cell method, Comput. Mech. 41 (1) (2007) 121–133, 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, Comput. Math. Appl. 64 (11) (2012) 3527–3541, https://doi.org/10.1016/j.camwa.2012.09.002.
[43] L. Paul Chew, Constrained Delaunay triangulations, Algorithmica 4 (1) (1989) 97–108, 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, https://www.tensorflow.org/, 2015.
[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, http://github.com/google/jax, 2022.
[47] D. Di Pietro, A. Ern, Mathematical Aspects of Discontinuous Galerkin Methods, Mathématiques et Applications, Springer Berlin Heidelberg, 2011.
[48] W. Ritz, Über eine neue methode zur lösung gewisser variationsprobleme der mathematischen physik, J. Reine Angew. Math. 135 (1909) 1–61, http://eudml.org/doc/149295.
[49] C. Johnson, Numerical Solution of Partial Differential Equations by the Finite Element Method, Cambridge U. Press, 1987.
[50] D.P. Kingma, J. Ba Adam, A method for stochastic optimization, https://doi.org/10.48550/ARXIV.1412.6980, https://arxiv.org/abs/1412.6980, 2014.
[51] E.C. Cyr, M.A. Gulian, R.G. Patel, M. Perego, N.A. Trask, Robust training and initialization of deep neural networks: an adaptive basis viewpoint, arXiv:1912.04862, 2019.
[52] E. Schiassi, R. Furfaro, C. Leake, M. De Florio, H. Johnston, D. Mortari, Extreme theory of functional connections: a fast physics-informed neural network method for solving ordinary and partial differential equations, Neurocomputing 457 (2021) 334–356, https://doi.org/10.1016/j.neucom.2021.06.015.
[53] I. Babuška, S. Theofanis, The Finite Element Method and Its Reliability, Numerical Mathematics and Scientific Computation, Oxford University Press, New York, 2001.
[54] 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, Numer. Math. 49 (6) (1986) 613–657.