Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 391 (2022) 114562
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

A Finite Element based Deep Learning solver for parametric PDEs

Carlos Uriarte[a,b,*], David Pardo[b,a,c], Ángel Javier Omella[b]

[a] *Basque Center for Applied Mathematics (BCAM), Mazarredo 14, E48009 Bilbao, Spain*
[b] *University of the Basque Country (UPV/EHU), Barrio Sarriena, E48940 Leioa, Spain*
[c] *Basque Foundation for Science (Ikerbasque), Plaza Euskadi 5, E48009 Bilbao, Spain*

Received 22 July 2021; received in revised form 24 November 2021; accepted 29 December 2021
Available online 24 January 2022

Abstract

We introduce a dynamic Deep Learning (DL) architecture based on the Finite Element Method (FEM) to solve linear parametric Partial Differential Equations (PDEs). The connections between neurons in the architecture mimic the Finite Element connectivity graph when applying mesh refinements. We select and discuss several losses employing preconditioners and different norms to enhance convergence. For simplicity, we implement the resulting Deep-FEM in one spatial domain (1D), although its extension to 2D and 3D problems is straightforward. Extensive numerical experiments show in general good approximations for both symmetric positive definite (SPD) and indefinite problems in parametric and non-parametric problems. However, in some cases, lack of convexity prevents us from obtaining high-accuracy solutions.

MSC: 68T07; 65N30; 65N50

Keywords: Deep Learning; Neural Networks; Partial Differential Equations; Finite Element Method

1. Introduction

Inverse problems are of great importance to our society [1,2]. They appear, for example, in imaging [3], electromagnetics [4], non-destructive evaluations [5], and geophysics [6]. There exist different methods to solve them, including gradient-based and statistical-based methods [7,8]. These traditional methods evaluate the inverse solution pointwise (i.e., for a given set of measurements), but they rarely provide a global representation of the inverse operator. To overcome this problem and approximate the full inverse function, it is possible to use Deep Learning (DL) methods (see, e.g., [9–14]), which allow to approximate complex mappings via a composition of linear and non-linear functions.

Training of inverse problems is typically performed in GPUs, especially when using large databases and/or complex Neural Network (NN) architectures. In these cases, it is critical to either have a large labeled database for training or a parametric PDE solver efficiently implemented in a GPU. It is desirable for this parametric PDE solver to return the solution in a fraction of a second to rapidly iterate over distinct parameter candidates during training. In particular, a pretrained NN is suitable to act as a real-time parametric PDE solver.

* Corresponding author at: Basque Center for Applied Mathematics (BCAM), Mazarredo 14, E48009 Bilbao, Spain.
E-mail addresses: carlos.uribar@gmail.com (C. Uriarte), dzubiaur@gmail.com (D. Pardo), ajaome@gmail.com (Á.J. Omella).

https://doi.org/10.1016/j.cma.2021.114562

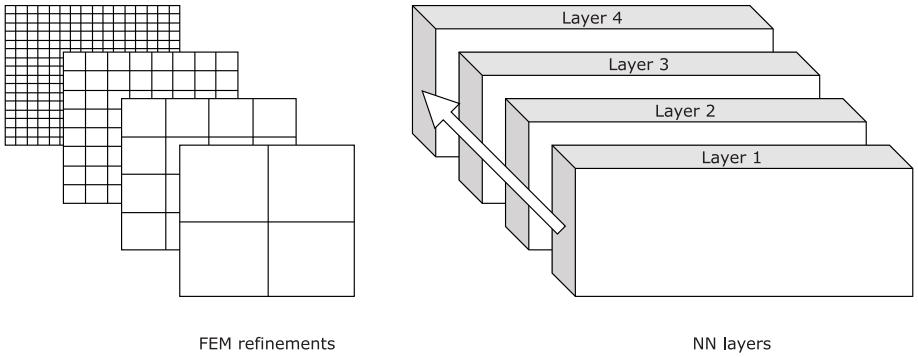


Fig. 1. FEM refinements vs. Deep-FEM layers.

Multiple approaches exist to solve PDEs using DL. For example, [15] proposes a VarNet model based on the variational formulation of PDEs; [16] proposes a Deep Galerkin Method; [17] describes the Deep Ritz Method, which minimizes the energy function using the Ritz method; and [18] proposes a Deep Least-Squares method. There also exist Physics-Informed Neural Networks (PINNs) —originally proposed in [19]— and its multiple variants and applications (see, e.g., [20–24]). There also exist NN approaches for improving the performance of traditional PDE solvers (see, e.g., [25,26]). Regarding parametric PDEs, we encounter several recent works. To mention a few: [27] uses NNs to parameterize the physical quantity of interest as a function of input coefficients; [28] analyzes theoretically the approximability of parametric maps by NNs for parametric PDEs; [29] combines NNs with model reduction; and [30] proposes a multi-level graph NN framework.

All of the above works address the idea of finding a continuous function (the NN) that approximates the solution of a (parametric) PDE. Generally, these designs allow evaluating the NN at any point in the domain, i.e., they have a mesh-free structure. However, they also present some limitations. We highlight the following two: (a) the resulting DL architectures lack from the so-called explainability [31,32], and (b) numerical integration rules are challenging to design within the loss, as mentioned in [20] and further explained in [33].

Herein, we propose a DL method for solving parametric PDEs that resembles the Finite Element Method (FEM) [34]. The NN architecture aims to act as a solver of the parametric system of linear equations arising in the FEM and to mimic the Finite Element connectivity graph when applying mesh refinements: we associate each NN layer with a mesh refinement. Each NN layer has a ResNet [35] design and extends coarse solutions to finer meshes. Fig. 1 illustrates the relation between FEM refinements and NN layers. In this way, the architecture provides certain degree of explainability, being the output the vector of nodal values corresponding to the finest mesh. In addition, this discrete approach enables exact numerical integration during training because the NN prediction lies on a piecewise-polynomial space.

The developed Deep-FEM first sets an initial architecture that produces coarse solutions after training. Then, we iteratively and dynamically insert layers to the architecture, maintaining the previously trained variables and adding new ones. Subsequently, we retrain the variables of the new model and we repeat this process until we achieve a desired degree of accuracy in the solution. The proposed NN allows to: (a) select a priori the number of neurons in the final layer based on experience and existing analytical results, and (b) perform a mesh-convergence study.

Our implementation restricts to 1D problems with piecewise-linear approximations and uniform refinements. The extension to higher dimensional problems, higher order polynomial approximations, and/or adaptive meshes is straightforward; but it requires a more elaborated implementation to deal with geometric criteria and node numbering between refinements, which we do not delve into in this work. We show numerical results of model problems with constant and piecewise-constant parameters.

The main contribution of the presented technology lies in the NN's ability to solve parametric problems. For illustration, we first introduce it for the non-parametric case before considering the parametric problem. We do the same when describing the numerical results, since the comprehensibility and limitations of the method for the non-parametric setting are extrapolable to the parametric one.

The remainder of this work is as follows. Section 2 introduces our problem of interest and the corresponding variational formulation. Section 3 describes our selected Deep-FEM solver architecture and Section 4 defines the loss

when employing several preconditioners. Section 5 shows implementation details and the limitations we encountered when using the widely known NN library Tensorflow (TF) [36]. Section 6 discusses the obtained numerical results and Section 7 summarizes and concludes the work.

2. Model problem

In this manuscript, we focus on a particular parametric Boundary Value Problem (BVP), although the presented approach applies to other problems that can be solved using the FEM.

2.1. Parametric linear PDE model problem

Let Ω be a domain. We consider the following parametric linear BVP,

$$\begin{cases} -\nabla \cdot \sigma \nabla u + \alpha u = f, & \text{in } \Omega, \\ u = u_D, & \text{in } \Gamma_D, \\ -\sigma \frac{\partial u}{\partial n} = g, & \text{in } \Gamma_N, \end{cases} \tag{1}$$

where parameters $\sigma > 0$ and $\alpha \in \mathbb{R}$ are piecewise-constant functions, f is the source, and u_D and g are the Dirichlet and Neumann data, respectively. Γ_D and Γ_N are the Dirichlet and Neumann boundaries, respectively, and with Γ_N possibly being empty. n denotes the outer normal vector at each point of Γ_N and $\partial u/\partial n = \nabla u \cdot n$. This model problem covers Poisson's equation ($\alpha = 0$), Helmholtz's equation ($\alpha < 0; \sigma = 1$), and Reaction–Diffusion's equation ($\alpha > 0; \sigma = 1$).

2.2. Finite Element formulation

A variational formulation of the above BVP reads as follows:

$$\left\| \begin{array}{l} \text{Find } u = u_0 + u_D \text{ with } u_0 \in H_0^1(\Omega) \text{ such that} \\ (\sigma \nabla u, \nabla v)_\Omega + (\alpha u, v)_\Omega = (f, v)_\Omega - (g, v)_{\Gamma_N}, \quad \forall v \in H_0^1(\Omega), \end{array} \right. \tag{2}$$

where

$$(u, v)_\Omega := \int_\Omega u \cdot v. \tag{3}$$

Using a FEM, we look for a solution $u = \sum_j u_j \phi_j$, where u_j are the unknown coefficients and ϕ_j are piecewise-linear basis functions. Discretizing (2), we arrive at the system of linear equations

$$\mathbf{A}\mathbf{u} = \mathbf{f}, \tag{4}$$

where $\mathbf{u}$ is the vector of unknown coefficients, $\mathbf{A}$ is the corresponding matrix, and $\mathbf{f}$ is the load vector.

3. Deep-FEM architecture

We first describe our proposed architecture for the non-parametric problem (i.e., a one-sample parameter problem) and then we extend it to the parametric case. Finally, we consider both constant and piecewise-constant parameters alternatives.

Fig. 2 shows our selected node numbering when performing uniform mesh refinements. Accordingly, the extension operator $\mathbf{E}$ is given by a sparse matrix filled with ones and halves depending on the contribution with which each node propagates from the coarse to the fine mesh. Note that similar extension operators exist for 2D and 3D problems as well as for higher-order elements, and for $H(\text{div})$, $H(\text{curl})$, and L^2 discretizations [37].

3.1. Non-parametric Deep-FEM architecture for constant PDE coefficients

Let σ and α be real-valued constants. For a one-element mesh, we propose the following two-layer depth architecture:

$$\mathbf{u}_1 = (\mathcal{L}_1^2 \circ \mathcal{L}_1^1)(\sigma, \alpha), \tag{5}$$

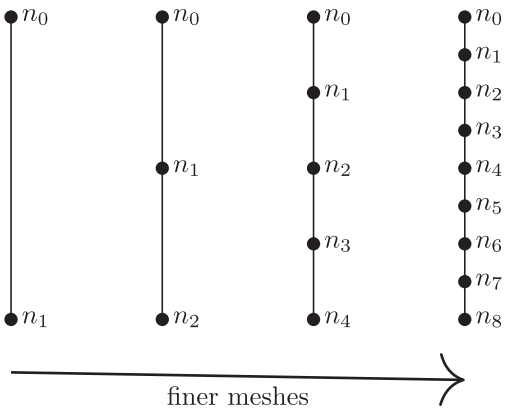


Fig. 2. 1D uniform mesh refinements and node numbering.

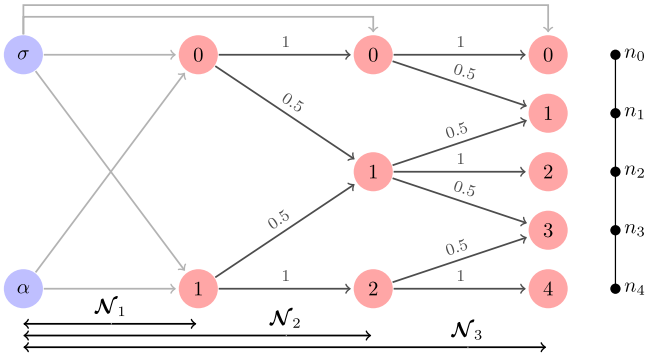


Fig. 3. Non-parametric Deep-FEM dynamic architecture.

$$\mathcal{L}_1^1 = \varphi(\omega_1^\sigma \sigma + b_1^\sigma) + \varphi(\omega_1^\alpha \alpha + b_1^\alpha), \tag{6}$$
$$\mathcal{L}_1^2 = \mathbf{W}_1^2 \mathcal{L}_1^1 + \mathbf{b}_1^2, \tag{7}$$

where φ is an activation function, and $\mathbf{u}_1$ is a vector of dimension two. The set of learnable variables consists of $\theta_1 = \{\omega_1^\sigma, \omega_1^\alpha, b_1^\sigma, b_1^\alpha, \mathbf{W}_1^2, \mathbf{b}_1^2\}$.

We train our first NN $\mathcal{N}_1 := \mathcal{L}_1^2 \circ \mathcal{L}_1^1$ so $\mathbf{u}_1$ approximates the FEM solution in the one-element mesh. Then, we refine and obtain a two-element mesh. We add an input-dependent ResNet [35] to $\mathcal{N}_1$ to define $\mathcal{N}_2$:

$$\mathbf{u}_2 = \mathcal{N}_2(\sigma, \alpha) = \mathbf{E}_1 \mathbf{u}_1 + \mathbf{r}_2, \tag{8}$$
$$\mathbf{r}_2 = (\mathcal{L}_2^2 \circ \mathcal{L}_2^1)(\sigma, \alpha) = \mathbf{W}_2^2 \left\{\varphi(\omega_2^\sigma \sigma + b_2^\sigma) + \varphi(\omega_2^\alpha \alpha + b_2^\alpha)\right\} + \mathbf{b}_2^2, \tag{9}$$

with $\mathbf{r}_2$ and $\mathbf{u}_2$ being vectors of dimension three, and $\mathbf{E}_1$ denoting the extension matrix of $\mathbf{u}_1$ on the fine (two-element) mesh. The learnable set of variables θ_2 is now $\{\omega_2^\sigma, \omega_2^\alpha, b_2^\sigma, b_2^\alpha, \mathbf{W}_2^2, \mathbf{b}_2^2\}$ or $\theta_1 \cup \{\omega_2^\sigma, \omega_2^\alpha, b_2^\sigma, b_2^\alpha, \mathbf{W}_2^2, \mathbf{b}_2^2\}$ depending on if we perform a *layer-by-layer* or an *end-to-end* training, respectively. We initialize the new variables of $\mathcal{N}_2$ so that the residual term $\mathbf{r}_2$ is zero at the beginning of the retraining (e.g., initializing $\mathbf{W}_2^2, \mathbf{b}_2^2 = 0$), and we maintain the learned values in θ_1 in the previous step at the beginning of the retraining in the current step. We retrain $\mathcal{N}_2$ so $\mathbf{u}_2$ approximates the FEM solution on the (two-element) fine mesh.

We repeat this process iteratively, increasing the depth of our NN, until the number of elements is sufficient to accurately approximate the analytic solution. Fig. 3 shows a graph of this dynamic architecture.

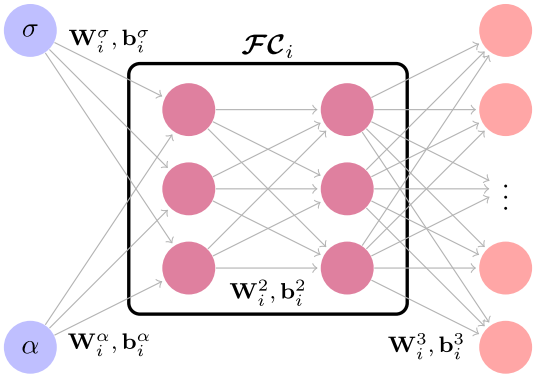


Fig. 4. Trainable block architecture with three-neuron width and two-layer depth.

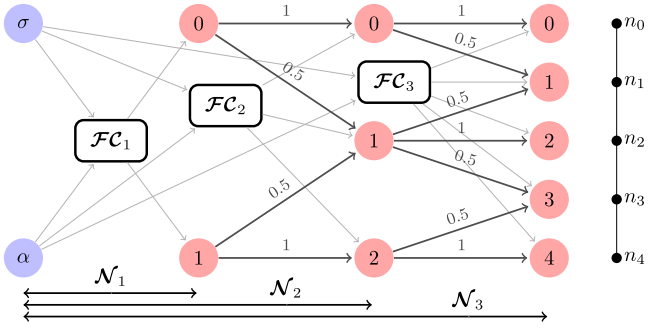


Fig. 5. Parametric Deep-FEM dynamic architecture.

3.2. Parametric Deep-FEM architecture for constant PDE coefficients

For the parametric problem, we add depth and width to the trainable parts of the dynamic architecture, namely,

$$\mathbf{u}_1 = \mathcal{N}_1(\sigma, \alpha) = \mathcal{FC}_1(\sigma, \alpha), \tag{10}$$

$$\mathbf{u}_i = \mathcal{N}_i(\sigma, \alpha) = \mathbf{E}_{i-1}\mathbf{u}_{i-1} + \mathbf{r}_i, \qquad i \geq 2, \tag{11}$$

$$\mathbf{r}_i = \mathcal{FC}_i(\sigma, \alpha), \qquad i \geq 2. \tag{12}$$

$\mathcal{FC}_i$ is a Fully Connected NN with non-activated last layer and depth $d_i \geq 2$,

$$\mathcal{FC}_i = (\mathcal{L}_i^{d_i} \circ \cdots \circ \mathcal{L}_i^2 \circ \mathcal{L}_i^1)(\sigma, \alpha) \tag{13}$$

$$\mathcal{L}_i^1 = \varphi(\mathbf{W}_i^\sigma \sigma + \mathbf{b}_i^\sigma) + \varphi(\mathbf{W}_i^\alpha \alpha + \mathbf{b}_i^\alpha), \tag{14}$$

$$\mathcal{L}_i^j = \varphi(\mathbf{W}_i^j \mathcal{L}_i^{j-1} + \mathbf{b}_i^j), \qquad 1 < j < d_i, \tag{15}$$

$$\mathcal{L}_i^{d_i} = \mathbf{W}_i^{d_i} \mathcal{L}_i^{d_i - 1} + \mathbf{b}_i^{d_i}. \tag{16}$$

We call *trainable block* to the $\mathcal{FC}_i$ NN in both the non-parametric and parametric problems. Fig. 4 illustrates a trainable block architecture and Fig. 5 depicts the appearance of the parametric architecture.

3.3. Parametric Deep-FEM architecture for piecewise-constant PDE coefficients

Finally, we add the piecewise-constant behavior to the parameters. We assume that the parameters take constant values on each element of the initial mesh. Thus, the architecture is affected only in the input layer by replacing the σ and α values by vectors of sizes equal to the number of elements of the initial mesh. Fig. 6 illustrates this appearance of a dynamic and parametric graph for an initial two-element mesh with piecewise-constant parameters.

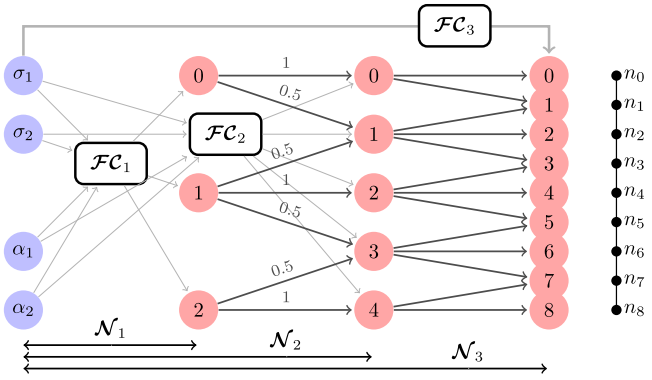


Fig. 6. Parametric Deep-FEM dynamic architecture for piecewise-constant parameters.

3.4. Number of trainable variables

In the layer-by-layer training, we associate the final layer with the final Finite Element mesh. Then, the number of trainable variables is linear with respect to the number of nodes in the finest mesh, up to a constant that depends on the intrinsic prescribed architecture of the trainable block related to the final layer. In the end-to-end training, in addition to the final mesh, we associate each of the previous layers of the NN architecture with coarser meshes: as we move away from the final layer, each subsequent layer contains in its corresponding trainable block approximately half as many trainable variables as in the previous layer. Adding up, the overall number of trainable variables is still linear with respect to the number of nodes in the final mesh.

4. Loss function and step-by-step optimization

We propose a step-by-step training. At each step, we aim $\boldsymbol{\mathcal{N}}_i$ to approximate the solution to the parametric system of linear equations arising in the FEM for an ith mesh-refinement level. Denoting by $\mathbf{A}_i$ and $\mathbf{f}_i$ to the corresponding matrix and load vector, we employ the loss

$$\ell_i(\sigma, \alpha; \theta_i) = \|\mathbf{f}_i - \mathbf{A}_i \mathbf{u}_i\|, \tag{17}$$

where $\mathbf{u}_i = \boldsymbol{\mathcal{N}}_i(\sigma, \alpha; \theta_i)$ is the NN prediction, $\mathbf{f}_i - \mathbf{A}_i \mathbf{u}_i$ is the residual, and $\|\cdot\|$ is a discrete vector norm (e.g., the l^2-norm). Then, considering a data D of samples of parameters, for each ith step, we train $\boldsymbol{\mathcal{N}}_i$ via the mean loss indicator

$$\theta_i^* = \arg\min_{\theta_i} \frac{1}{|D|} \sum_{(\sigma,\alpha)\in D} \ell_i(\sigma, \alpha; \theta_i). \tag{18}$$

4.1. Gradient-descent based training

At the beginning of each step, we apply the Adam optimizer [38]. If the number of iterations attains an established maximum, we stop its execution. Alternatively, to control stagnation, we monitor and compare the loss obtained at any given iteration with the lowest loss obtained few iterations before. If the loss improvement is insignificant after a prescribed number of iterations, we stop the execution. Since the loss may increase or decrease during the Adam performance, we monitor the loss so as to return the best trainable variables configuration at the end of its execution. Thereafter, we apply a customized gradient-descent based optimizer with a loss-dependent adaptive learning rate (GD loss-lr). We maintain the same stopping and stagnation criteria than in Adam. The GD loss-lr optimizer readjusts the trainable variables to prevent a loss increase. Algorithm 1 shows the learning rate adaptivity and trainable variables acceptance–rejection criteria. In the numerical results, we compare both optimizers: Adam and GD loss-lr. While Adam rapidly decreases the loss but only reaches a suboptimal threshold in which it oscillates and it does not further improve, GD loss-lr decreases the loss monotonically. However, sometimes this convergence is slower than with Adam.

Algorithm 1: Loss-dependent adaptive learning rate in gradient-descent based optimization

Input: θ^0, η^0 ; // Initial variables and learning rate
Output: θ^*, ℓ^* ; // Final variables and optimal loss value
$\ell^0 \leftarrow \frac{1}{|D|}\sum_{(\sigma,\alpha)\in D}\ell(\sigma,\alpha;\theta^0)$; $\theta^1 \leftarrow \theta^0 - \eta^0\frac{\partial \ell}{\partial \theta}(\theta^0)$;
$\theta^* \leftarrow \theta^0$; $\ell^* \leftarrow \ell^0$; $t \leftarrow 1$;
while not STOP **do**
  $\ell^t \leftarrow \frac{1}{|D|}\sum_{(\sigma,\alpha)\in D}\ell(\sigma,\alpha;\theta^t)$; $\theta^{t+1} \leftarrow \theta^t - \eta^t\frac{\partial \ell}{\partial \theta}(\theta^t)$;
  if $\ell^t > \ell^*$ **then**
    $\eta^{t+1} \leftarrow$ `Decrease`(η^t); $\theta^{t+1} \leftarrow \theta^*$;
  else
    $\theta^* \leftarrow \theta^t$;
    if convergence is slow **and** no variables rejection in $t-1$ **then**
      $\eta^{t+1} \leftarrow$ `Increase`(η^t);
    else
      $\eta^{t+1} \leftarrow \eta^t$;
    $\ell^* \leftarrow \ell^t$;
  $t \leftarrow t+1$;
return θ^*, ℓ^*

4.2. Norm selection: preconditioning

We want to select a loss with a similar behavior than the *energy-norm error*. In a SPD problem, the energy norm is given by

$$\|\mathbf{v}\|_{\mathbf{A}} := \sqrt{\mathbf{v}^T\mathbf{A}\mathbf{v}}, \tag{19}$$

where $\mathbf{v}^T$ is the transpose of vector $\mathbf{v}$, and $\mathbf{A}$ denotes the SPD matrix of the system of linear equations. Writing the error vector as $\mathbf{e}_i = \mathbf{A}_i^{-1}\mathbf{f}_i - \mathbf{u}_i$ for each ith mesh-refinement level, we arrive at the following norm relation with the residual:

$$\|\mathbf{e}_i\|_{\mathbf{A}_i} = \|\mathbf{f}_i - \mathbf{A}_i\mathbf{u}_i\|_{\mathbf{A}_i^{-1}}. \tag{20}$$

In practice, the inverse of $\mathbf{A}_i$ is non-computable and thus the error is never known. Hence, as in iterative methods for solving a system of linear equations, we select a preconditioner to mimic the inverse operator and thus decrease the condition number of the system [39]. Thus, we define the loss as

$$\ell_i(\sigma,\alpha;\theta_i) := \|\mathbf{f}_i - \mathbf{A}_i\mathbf{u}_i\|_{\mathbf{P}_i} = \sqrt{(\mathbf{f}_i - \mathbf{A}_i\mathbf{u}_i)^T\mathbf{P}_i(\mathbf{f}_i - \mathbf{A}_i\mathbf{u}_i)}, \tag{21}$$

where $\mathbf{P}_i$ is a preconditioner for $\mathbf{A}_i$. We select for $\mathbf{P}_i$ block Jacobi preconditioners of different block sizes and with one-element overlap [40]. If $\mathbf{P}_i$ is the identity matrix, the loss corresponds to the discrete l^2-norm of the residual.

In case the problem is indefinite, we select a positive definite operator $\mathbf{B}_i$ and measure the error on the corresponding norm. The relation between the error and the residual in this new norm is given by

$$\|\mathbf{e}_i\|_{\mathbf{B}_i} = \|\mathbf{A}_i^{-1}(\mathbf{f}_i - \mathbf{A}_i\mathbf{u}_i)\|_{\mathbf{B}_i}, \tag{22}$$

where following the above reasoning leads us to define the loss as

$$\ell_i(\sigma,\alpha;\theta_i) := \|\mathbf{P}_i(\mathbf{f}_i - \mathbf{A}_i\mathbf{u}_i)\|_{\mathbf{B}_i}. \tag{23}$$

Again, if $\mathbf{P}_i$ and $\mathbf{B}_i$ are the identity operators, the above equation reduces to the discrete l^2-norm of the residual.

In the numerical results, in addition to using the energy-norm for positive definite problems, we employ the continuous L^2 and H^1 norms both for testing and for monitoring the (preconditioned) residual and the error. These continuous norms have their equivalent discrete definitions:

$$\|u_i\|_{L^2} = \sqrt{\mathbf{u}_i^T\mathbf{M}_i\mathbf{u}_i} = \|\mathbf{u}_i\|_{\mathbf{M}_i}, \tag{24}$$

$$\|u_i\|_{H^1} = \sqrt{\mathbf{u}_i^T \mathbf{M}_i \mathbf{u}_i + \mathbf{u}_i^T \mathbf{K}_i \mathbf{u}_i} = \|\mathbf{u}_i\|_{\mathbf{M}_i+\mathbf{K}_i}, \tag{25}$$

where $\mathbf{u}_i = [u_{ij}]$ is the vector of evaluations of $u_i = \sum_j u_{ij}\phi_{ij}$ at the nodal points of the mesh, ϕ_{ij} is the basis function centered at the jth node for the ith mesh-refinement level, and $\mathbf{M}_i$ and $\mathbf{K}_i$ are the mass and stiffness matrices defined by $(\phi_{is}, \phi_{ir})_\Omega$ and $(\nabla\phi_{is}, \nabla\phi_{ir})_\Omega$ in the (r, s)-th entry, respectively. For convenience, we call the discrete norms (24) and (25) by the names inherited from their continuous norms, namely, L^2 and H^1 norms, respectively.

5. Implementation

We implement the code in the Python programming language, and we use the libraries Tensorflow 2 (TF2) [36], NumPy [41], and SciPy [42] to build the NN models and to generate and manage the parameters and FEM data. We create the layers by redefining the corresponding base classes in Keras inside TF2 (`tf.keras`). We use a dedicated sparse library within TF2 (`tf.sparse`) to handle the extension operators and the FEM matrices to save memory and achieve high-performance. We use double precision (float64) instead of the default single precision (float32). We calculate the loss in an auxiliary non-trainable layer that is applied after the main model $\mathcal{N}_i$ at each step. Below, we describe the three main difficulties encountered during our implementation.

5.1. Reshape of the batch sparse tensors

Tensors flowing through Keras models are prepared to maintain their first dimension (a.k.a axis) for the batch of samples. As a consequence, matrix–vector multiplications in the loss transform into an operation between a 3D sparse tensor (batch of sparse matrices) and a 2D tensor (batch of vectors) in the Keras model. If both tensors were dense, we could define our operation via Einstein summation. However, there is still not an equivalent TF2 function supporting sparse tensors [43]. We overcome this by reshaping the 3D sparse tensor as a sparse non-overlapping block 2D matrix, and the 2D tensor as a long 1D vector. This leads us to miss the batch flow behavior of the model and we are forced to avoid using the Keras training function. We thus perform a low-level optimization via automatic differentiation in *eager execution*, which is significantly slower than training via *graph execution* within Keras [44].

5.2. FEM data generation

We utilize SciPy as the FEM environment to build the extension operators and the matrices and preconditioners of the system of linear equations. We also utilize it to solve the sparse systems via its built-in solver to compare model predictions. For more complicated geometries, there exist other more efficient pieces of software to assemble the matrices, preconditioners, and extension operators (e.g., FEniCS [45]). We recommend their use for more complex FE systems (e.g., in 2D and 3D problems). In any case, these offline operations are calculated prior to the training of the NNs, and therefore they do not affect to the optimization time of the model.

5.3. Preconditioners assembly and action

Ideally, we should manage preconditioners as LU block decompositions of the matrices of the system, and calculate their actions at each loss evaluation using a forward–backward substitution algorithm [46,47]. Again, TF2 lacks an equivalent built-in function to evaluate these actions. For this reason, we manage the preconditioners as already assembled tensors.

6. Numerical results

To analyze the Deep-FEM performance, we carry out a series of experiments. Sections 6.1, 6.2, and 6.3 contemplate non-parametric problems, while Section 6.4 addresses parametric problems. All experiments are carried out in the spatial domain (0, 1) with Dirichlet and Neumann boundary conditions at 0 and 1, respectively.

6.1. A single PDE example

Let

$$\begin{cases} -u'' = -20x^3, \\ u(0) = 0, u'(1) = 5. \end{cases} \tag{26}$$

Its analytic solution is $u(x) = x^5$. We solve it employing the Deep-FEM starting from a one-element mesh. We perform ten mesh refinements (eleven steps) to finish with a 1024-element overkill mesh. In this first experiment, we analyze our proposed method when performing multiple refinements. All the training blocks of the DL model consist of one-neuron width and one-layer depth (see Section 3.2). We employ ReLU activation functions on all trainable blocks.

Fig. 7 illustrates the NN predictions in the first four steps. At each step, the model extends the coarse prediction to the fine mesh, and then trains the resulting NN to obtain a proper fine grid approximation. Results show a superb accuracy in these first steps.

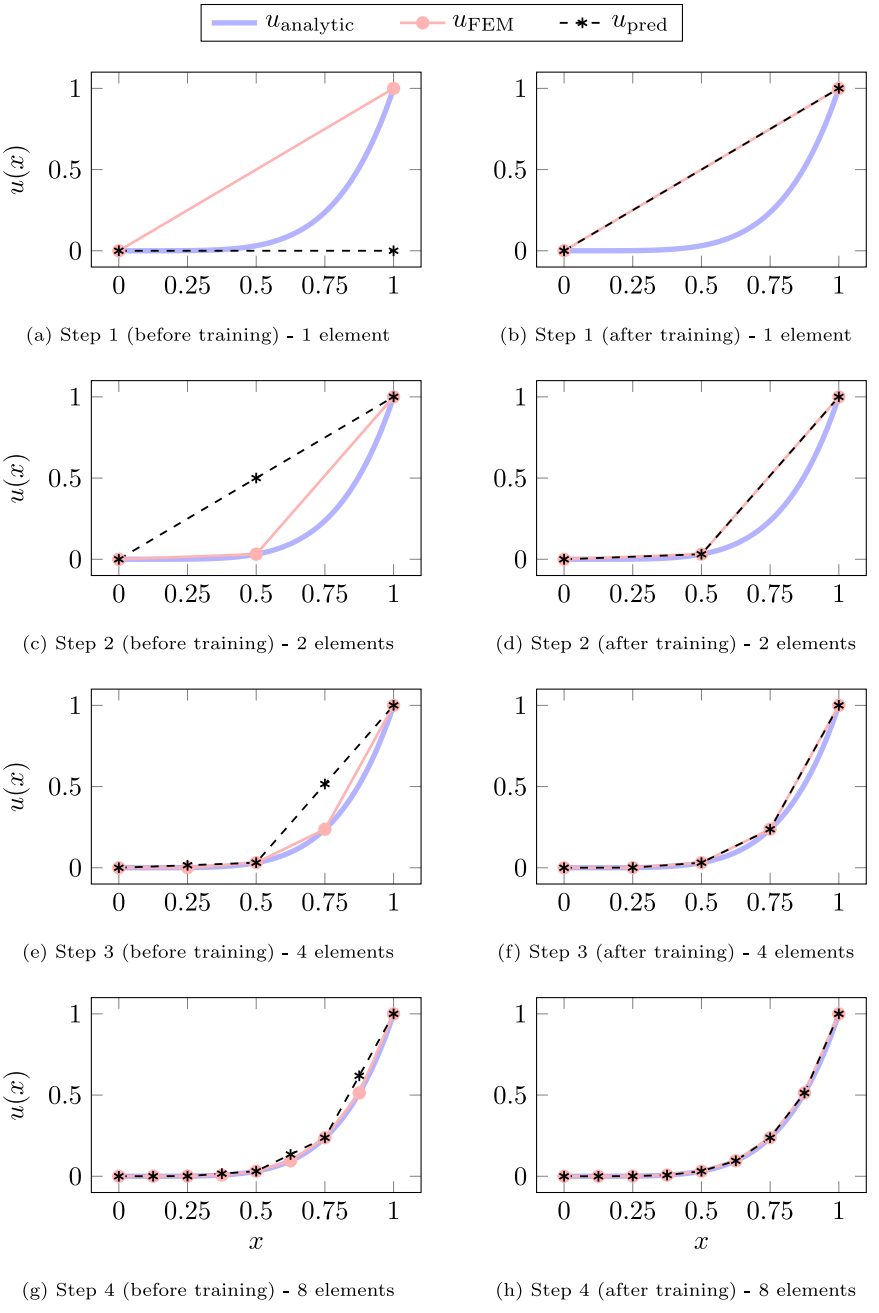
Fig. 8(a) shows the loss evolution along the first four steps when the loss coincides with the energy-norm error. Fig. 8(b) shows all eleven steps of the training process. This example illustrates the convergence behavior and limitations of DL under optimal conditions, i.e., when the inverse of the matrix is already part of the loss. The vertical jumps observed in the loss correspond to the extension from one grid to the next one. After each jump, we observe a noisy convergence of the loss. This phase corresponds to the use of the Adam optimizer, which works as an initial aggressive loss descender that gets stunned quickly. We then switch to the GD loss-lr optimizer, which exhibits a monotonic loss decrease. At each phase, we select an initial learning rate equal to the first loss evaluation multiplied by 10^{-3} (for Adam) or by 10^{-2} (for GD loss-lr). We set a maximum of 2000 (Adam) and 4000 (GD loss-lr) iterations for each optimizer performance at each step. Moreover, if we attain a loss below 10^{-12}, we stop the optimization at each step. We carry out an end-to-end training of the Deep-FEM model. We observe a convergence deterioration as we increase the step number (and grid size). Nonetheless, the final error remains below 10^{-8} (see Fig. 8(c)), which is a superb accuracy for DL algorithms.

6.1.1. End-to-end vs. layer-by-layer training

By construction, each layer of Deep-FEM produces the coefficients related to the basis functions associated to each Finite Element mesh. The finer the mesh, the more local are the supports of the basis functions associated to the coefficients. While an end-to-end training allows adjustments in the entire hierarchy of coefficients associated to basis functions of coarse and fine meshes, a layer-by-layer training only adjusts coefficients associated to the finest mesh. Fig. 9(a) shows the loss convergence under the same conditions as above but performing a layer-by-layer training. We observe that convergence deteriorates as the number of iterations increases compared to performing an end-to-end training (recall Fig. 8(b)). This occurs because the loss involves the gradient of the error when employing the energy-norm. Fig. 9(b) shows the error function, which is almost constant. Thus, the derivative of the error is almost zero and it is challenging to minimize the energy-norm error by adjusting individual coefficients associated to local-support basis functions, since this often implies an increase in the derivative of the error. This convergence accelerates using both global-support and local-support basis functions, as in multigrid methods [48].

If we select a norm for the loss that ignores gradients, e.g., the L^2-norm, training of local-support basis functions provide outstanding results, as shown in Fig. 10(a). However, optimizing with respect to the L^2-norm is discouraged for solving differential equations.

Even if the end-to-end training is the best alternative when dealing with losses involving the gradient of the residual/error, we observe a convergence deterioration of the loss function as iterations move forward (recall Fig. 8(b)). We suspect that this is independent of the norm selection, but it occurs because of the conflicting coexistence of many trainable variables. To illustrate this fact, we consider the above L^2-norm case where layer-by-layer training suffices to achieve an outstanding convergence, and we perform an end-to-end training. Fig. 10(b) shows the convergence deterioration of the loss function. In the following, we only consider end-to-end training cases of study.



(a) Step 1 (before training) - 1 element

(b) Step 1 (after training) - 1 element

(c) Step 2 (before training) - 2 elements

(d) Step 2 (after training) - 2 elements

(e) Step 3 (before training) - 4 elements

(f) Step 3 (after training) - 4 elements

(g) Step 4 (before training) - 8 elements

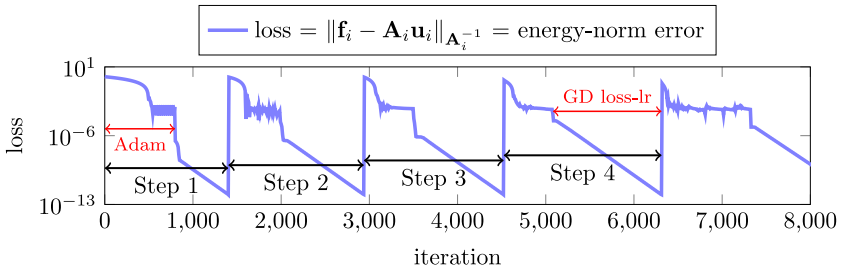
(h) Step 4 (after training) - 8 elements

Fig. 7. First four steps of Deep-FEM predictions for problem (26). u_{analytic} is the analytic solution, u_{FEM} is the finite element solution, and u_{pred} is the Deep-FEM prediction.

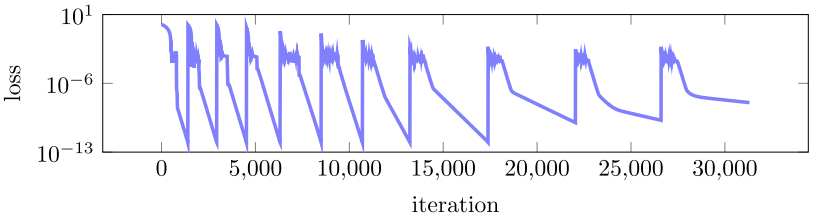
6.1.2. Preconditioners action

We now consider the loss given by Eq. (21) with three different preconditioners: (a) $\mathbf{P}_i$ being the identity matrix (Fig. 11(a)); (b) $\mathbf{P}_i$ being a block Jacobi preconditioner with blocks of size two (Fig. 11(b)); and (c) $\mathbf{P}_i$ being a block Jacobi preconditioner with size-adaptive blocks equal to half of the number of elements in the mesh (Fig. 11(c)). In all the cases, we show the loss evolution along with the energy-norm error evolution.

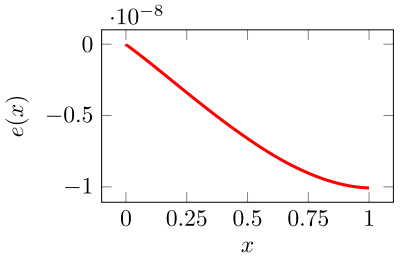
We observe some differences between the energy-norm error and the loss that increases with the mesh size, as expected. We also observe that the larger the size of the block-Jacobi, the smaller the discrepancy between the loss



(a) Loss evolution during the first four steps



(b) Loss evolution during eleven steps



(c) Error function at the end of the training.
$e(x) = u_{\text{FEM}}(x) - u_{\text{pred}}(x)$.

Fig. 8. End-to-end training of the Deep-FEM for problem (26). The loss is the energy-norm error.

and the norm error. In addition, the further the loss is from the energy-norm error, the earlier the loss stagnates convergence —compare errors at Figs. 11(d) and 11(e). This suggests that the loss induced by the energy-norm error is more convex with respect to the variables than other simplified variants such as the loss induced by the l^2-norm of the residual vector.

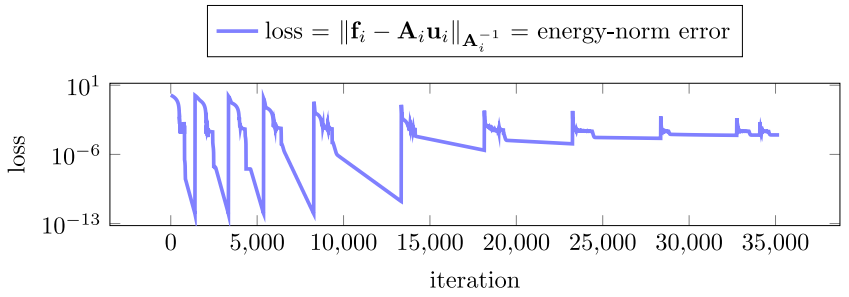
6.1.3. Norm interchange during training

Depending on the norm we select, we deal with different convexity shapes of the loss with respect to the variables of the NN. Within certain portion of the learnable variables domain, it often happens that a given loss is more convex than others, which has a direct impact on the optimizer convergence. With the aim of avoiding stagnation, we propose to change the norm during optimization, expecting to improve convexity. In this way, we define

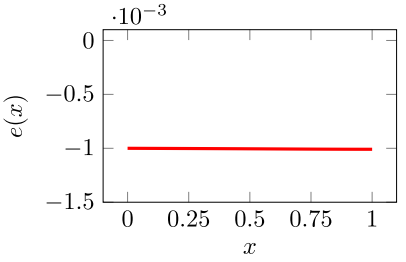
$$\ell_i(\sigma, \alpha; \theta_i) = C_E \|\mathbf{f}_i - \mathbf{A}_i\mathbf{u}_i\|_{\mathbf{P}_i} + C_{L^2} \|\mathbf{P}_i(\mathbf{f}_i - \mathbf{A}_i\mathbf{u}_i)\|_{\mathbf{M}_i}, \tag{27}$$

where $C_E, C_{L^2} \in \{0, 1\}$ are distinct values that interchange if the convergence stagnates.

To illustrate the above idea, we consider the case of Fig. 11(b). We maintain the loss at Eq. (21) for the first four steps. Then, we consider two variants for the loss in the fifth step when employing the GD loss-lr optimizer: (a) same loss as in the previous steps —maintaining $C_E = 1$ in (27)— but with a maximum of 12,000 iterations; and (b) the loss at Eq. (27) with $C_E = 1$ for 2000 iterations, changing to $C_{L^2} = 1$ for another 8,000 iterations, and returning back to $C_E = 1$ for 2000 additional iterations. Although the total number of iterations is the same in all situations, we obtain a loss at the end of the fifth step that is lower when performing the norm change (around

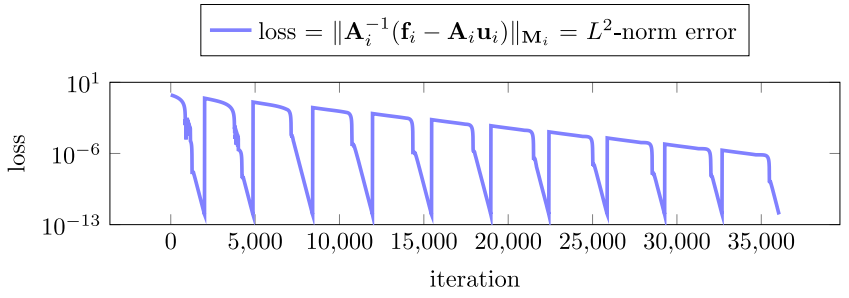


(a) Loss evolution

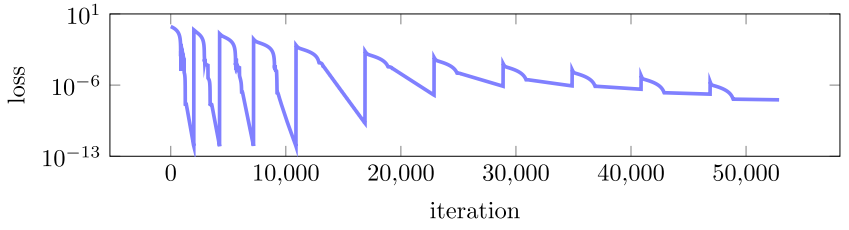


(b) Error function at the end of the training.
$e(x) = u_{\text{FEM}}(x) - u_{\text{pred}}(x)$.

Fig. 9. Layer-by-layer training of the Deep-FEM model for problem (26) employing the energy-norm for the loss.



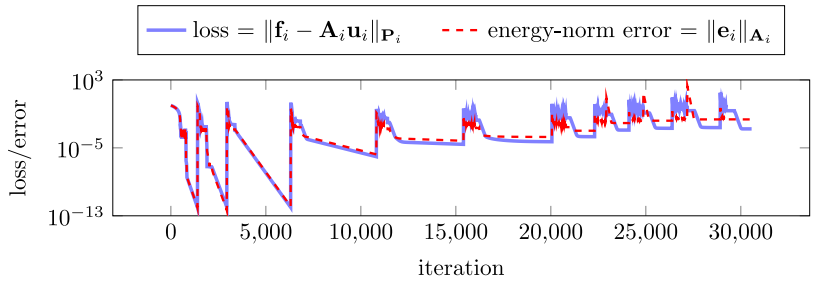
(a) Loss evolution when perfoming a layer-by-layer training



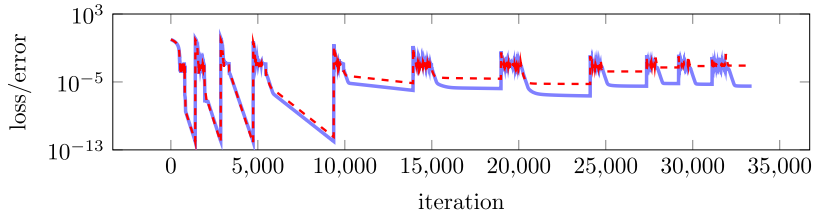
(b) Loss evolution when performing an end-to-end training

Fig. 10. Training of the Deep-FEM model for problem (26) using the L^2-norm for the loss.

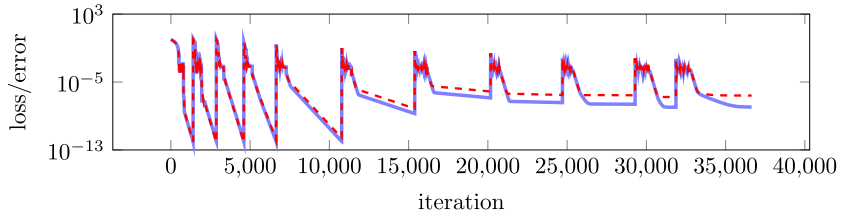
10^{-10}) than when maintaining the energy-norm (around 10^{-8}) —see Fig. 12. The slope of the loss evolution is higher when employing the L^2-norm, which allows us to start from a lower loss value when returning to the energy-norm compared to maintaining it during the entire training step.



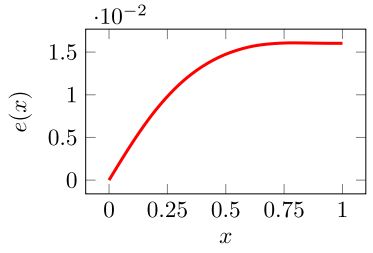
(a) Employing no preconditioner (i.e., the loss coincides with the l^2-norm of the residual)



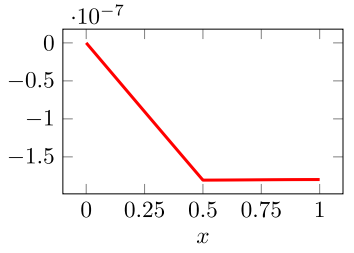
(b) Employing blocks of size two in the block Jacobi preconditioner



(c) Employing blocks of size equal to the half of the size of the mesh in the block Jacobi preconditioner



(d) Error function of case (a) at the end of the training. $e(x) = u_{\text{FEM}}(x) - u_{\text{pred}}(x)$.



(e) Error function of case (c) at the end of the training. $e(x) = u_{\text{FEM}}(x) - u_{\text{pred}}(x)$.

Fig. 11. End-to-end training of the Deep-FEM for problem (26) using different block-Jacobi sizes for the preconditioner.

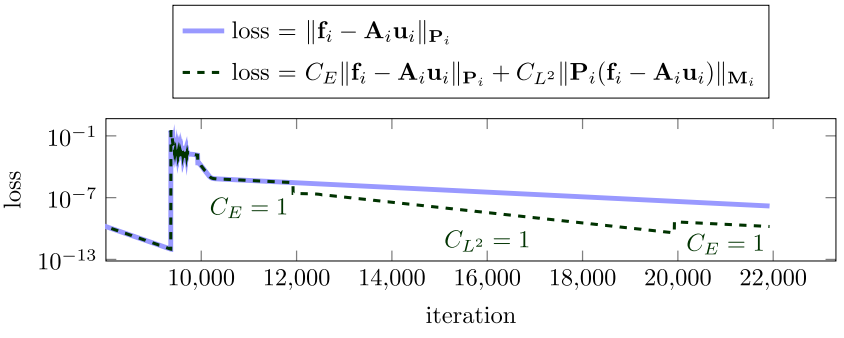


Fig. 12. Energy-norm long training vs. energy- and L^2-norm interchange training at step five of Fig. 11(b).

6.2. Sine solution in the Poisson and Helmholtz equations

We consider two different BVPs whose solutions are $u(x) = \sin(10\pi x)$:

$$\begin{cases} -u'' = 100\pi^2 \sin(10\pi x), \\ u(0) = 0, u'(1) = 10\pi, \end{cases} \qquad \begin{cases} -u'' - 100\pi^2 u = 0, \\ u(0) = 0, u'(1) = 10\pi. \end{cases} \tag{28}$$

The former is a SPD problem, while the second is indefinite. We solve both employing the Deep-FEM selecting the loss as the H^1-norm of the preconditioned residual,

$$\ell_i(\sigma, \alpha; \theta_i) = \|\mathbf{P}_i(\mathbf{f}_i - \mathbf{A}_i \mathbf{u}_i)\|_{\mathbf{M}_i + \mathbf{K}_i}. \tag{29}$$

We start from 32 elements and perform three mesh refinements. We select $\mathbf{P}_i$ with blocks of size 32 for all $1 \leq i \leq 4$. The optimizers and NN architecture are those of Section 6.1.

Figs. 13 and 14 show the NN predictions of the Poisson and Helmholtz problems, respectively. In Poisson, FEM nodal solutions coincide with the analytical ones. This is not the case for Helmholtz equation.

When training with the preconditioner being the inverse matrix, the NN converges perfectly with a monotonic decrease. However, when considering non-inverse preconditioning, convergence stagnates after decreasing a couple of orders of magnitude (see Figs. 15(a) and 15(b)). In the GD loss-lr optimizer training phases, the loss decreases abruptly in the early iterations, but then remains flat. The H^1-norm error reduction is small. In both experiments, we obtain errors around 10^{-3} (see Figs. 15(c) and 15(d)). These results illustrate that it is possible to obtain approximate solutions with certain degree of accuracy; however, lack of convexity of the loss prevents us from obtaining high-accuracy solutions. These observations agree with known facts of DL algorithms and prevent us from developing a convergence theory (see, e.g., [49–51]), where they assume that the DL optimizer converges, which unfortunately is not always the case when dealing with non-convex losses.

6.3. A sinusoidal problem with piecewise-constant parameters

Following the same sinusoidal example above, we consider an indefinite PDE with varying frequencies along the propagation domain. For this purpose, we select piecewise-constant coefficients:

$$\begin{cases} -(\sigma u')' + \alpha u = 0, \\ u(0) = 0, u'(1) = 10\pi, \end{cases} \tag{30}$$

with

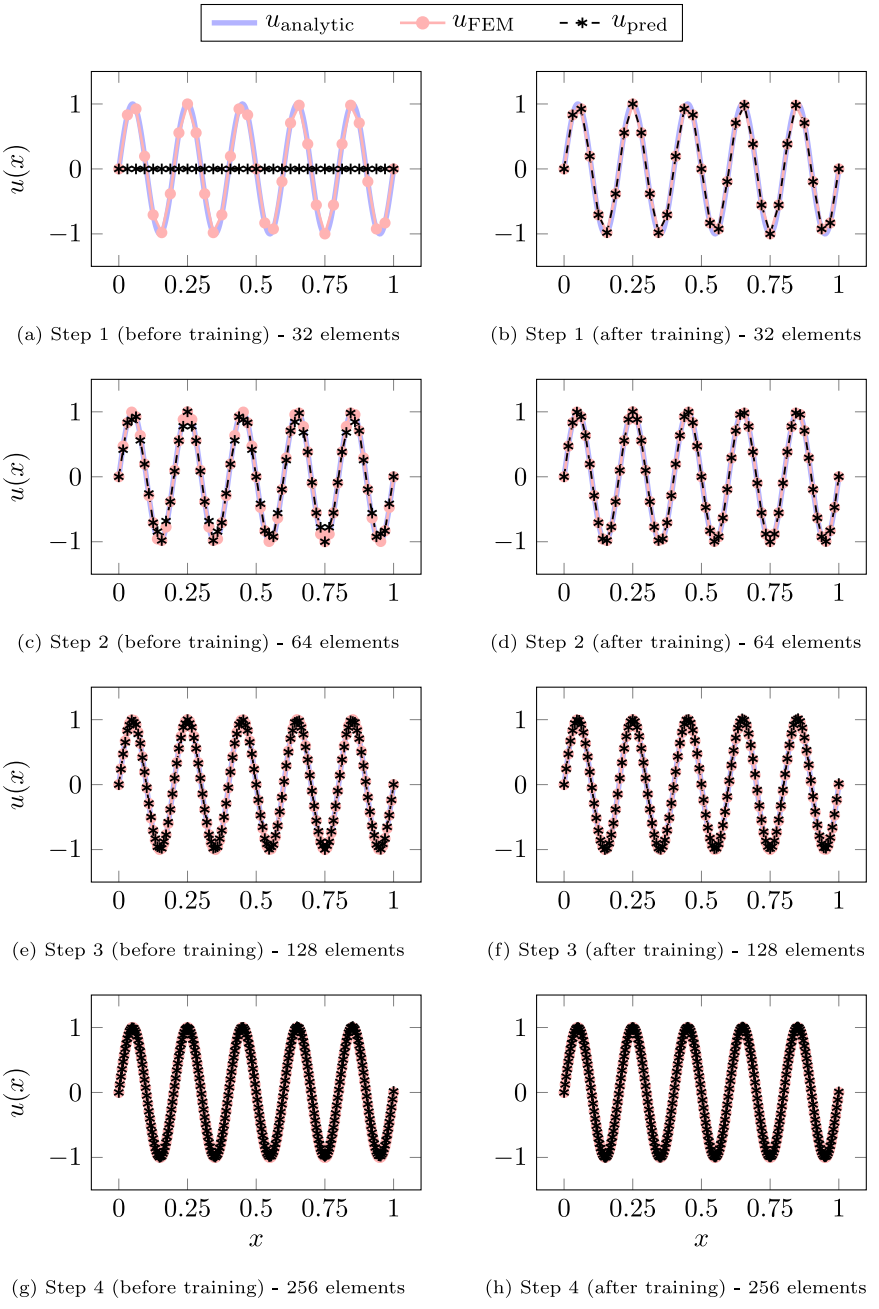
$$\sigma = \begin{cases} 1, & \text{if } 0 < x < 1/3, \\ 2, & \text{if } 1/3 < x < 2/3, \\ 3, & \text{if } 2/3 < x < 1, \end{cases} \qquad \alpha = \begin{cases} -3000, & \text{if } 0 < x < 1/3, \\ -2000, & \text{if } 1/3 < x < 2/3, \\ -1000, & \text{if } 2/3 < x < 1. \end{cases} \tag{31}$$

We solve it employing the H^1-norm for the loss. We start with 48 elements and perform three uniform refinements. We select $\mathbf{P}_i$ with blocks of sizes 48 in the first two grids, and 96 in the last two grids. The optimizers and NN architecture are those described in Section 6.1.

Fig. 16 shows the NN predictions and Fig. 17(a) shows the loss evolution during training. The NN nicely converges when employing the inverse as preconditioner, but in the subsequent steps we observe a stagnation of the loss, as was previously the case. In addition, there is a decreasing influence of the loss on the H^1-norm error, which remains constant throughout the last training step, while the loss decreases by two orders of magnitude. Figs. 17(b) and 17(c) show the errors at the end of steps three and four, respectively.

6.4. Parametric boundary value problems

In this section, we naturally extend the Deep-FEM in its parametric variant: we train the NN so as to learn the FEM solutions from one mesh to another for a family of PDE coefficients with fixed boundary conditions. We consider experiments varying only the α coefficient with a constant parametric behavior.



(a) Step 1 (before training) - 32 elements

(b) Step 1 (after training) - 32 elements

(c) Step 2 (before training) - 64 elements

(d) Step 2 (after training) - 64 elements

(e) Step 3 (before training) - 128 elements

(f) Step 3 (after training) - 128 elements

(g) Step 4 (before training) - 256 elements

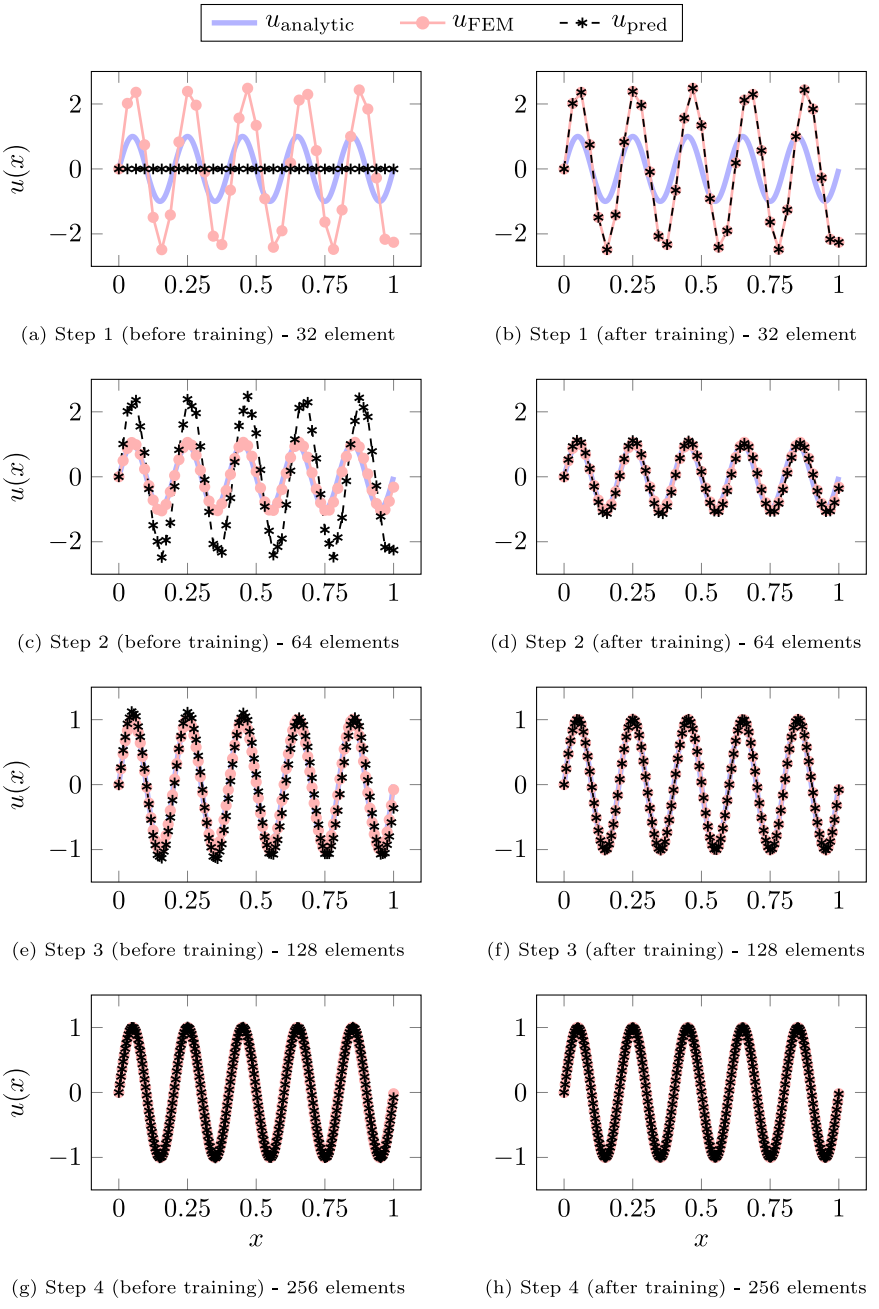
(h) Step 4 (after training) - 256 elements

Fig. 13. Four steps of Deep-FEM predictions for Poisson's problem (28). u_{analytic} is the analytic solution, u_{FEM} is the finite element solution, and u_{pred} is the Deep-FEM prediction.

Let

$$\begin{cases} -u'' + \alpha u = 0, \\ u(0) = 0, u'(1) = 2\pi. \end{cases} \tag{32}$$

We solve it performing three uniform refinements. For the training, we select databases of randomly selected samples of α coefficients. We train the NN on the entire database without batch partitioning. Our loss computes the norm



(a) Step 1 (before training) - 32 element

(b) Step 1 (after training) - 32 element

(c) Step 2 (before training) - 64 elements

(d) Step 2 (after training) - 64 elements

(e) Step 3 (before training) - 128 elements

(f) Step 3 (after training) - 128 elements

(g) Step 4 (before training) - 256 elements

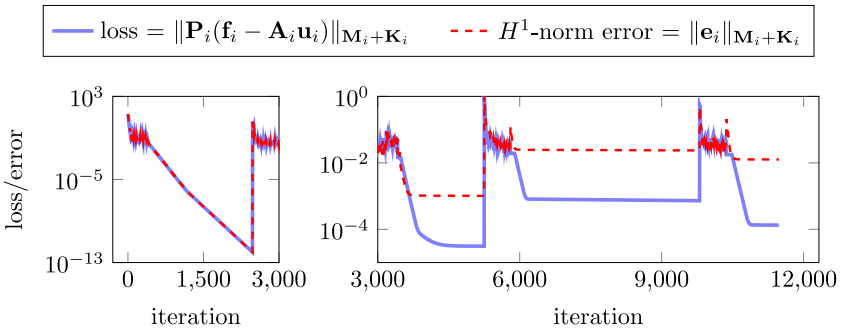
(h) Step 4 (after training) - 256 elements

Fig. 14. Four steps of Deep-FEM predictions for Helmholtz's problem (28). u_{analytic} is the analytic solution, u_{FEM} is the finite element solution, and u_{pred} is the Deep-FEM prediction.

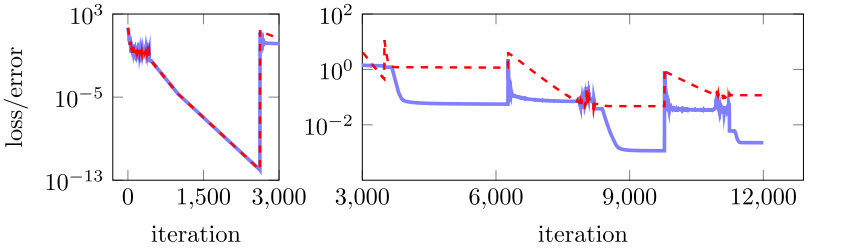
over the preconditioned residual that depend on each sample. Specifically, we optimize with respect to an averaged sum of losses (norms) —see Eq. (18).

6.4.1. Reaction–Diffusion parametric equation: $0 < \alpha < 200$

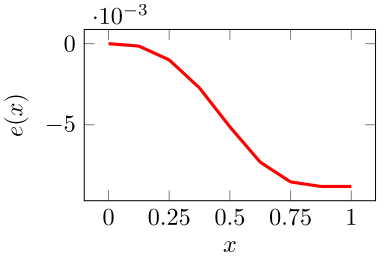
In this example, the analytic solution is $u(x) = C(e^{\sqrt{\alpha}x} - e^{-\sqrt{\alpha}x})$ with $C = \frac{2\pi e^{\sqrt{\alpha}}}{\sqrt{\alpha}(e^{2\sqrt{\alpha}}+1)}$ and $u(x) \to 2\pi x$ as $\alpha \to 0$. We establish one-layer depth and 20-neuron width training block architecture for the NN at each step.



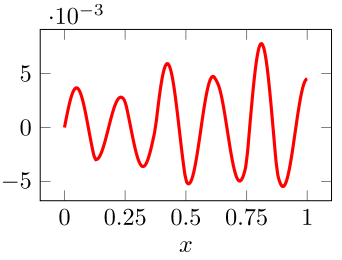
(a) Loss evolution along with the H^1-norm error for Poisson problem (28)



(b) Loss evolution along with the H^1-norm error for Helmholtz problem (28)



(c) Error function at the end of the training. $e(x) = u_{\text{FEM}}(x) - u_{\text{pred}}(x)$.



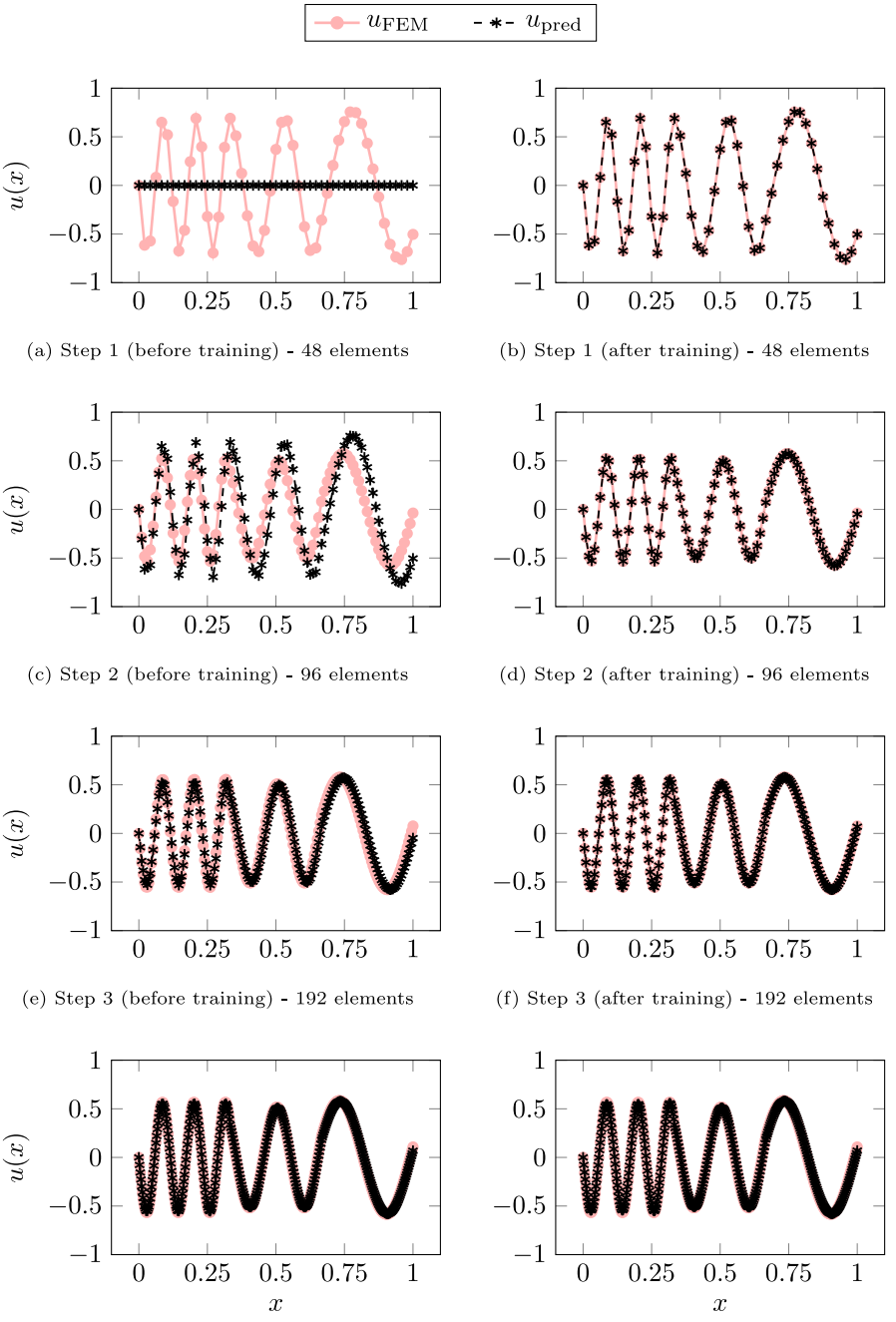
(d) Error function at the end of the training. $e(x) = u_{\text{FEM}}(x) - u_{\text{pred}}(x)$.

Fig. 15. End-to-end training of the Deep-FEM for problem (28) along three steps with block Jacobi preconditioner with blocks of size 32.

We start the Deep-FEM by a uniform eight-element mesh and we finish with a 64-element mesh. At each step, we employ preconditioners with blocks of size eight. Since the parametric problem is SPD, we consider the norm induced by the preconditioner for the optimization.

We select a database of 100 samples randomly and logarithmically distributed to train the NN. To visualize the NN performance after training, we select the test data $\{0, 3, 15, 50, 200\}$, which does not take part in the training. Fig. 18 shows the parametric mesh-by-mesh adaptivity of the NN over these test data. We observe a good behavior of the NN for all values of α. Table 1 displays the losses and energy-norm errors evaluated on the test data points at the end of the training.

Fig. 19 shows the error functions on the test data at the end of each training step. The largest errors occur for the extreme values of α, namely, $\alpha = 0$ and $\alpha = 200$. Analyzing the evolution of the training loss in Fig. 20(a), we observe that the loss decrease is larger than the energy-norm decrease, as expected. If we increase the size of the blocks in the preconditioners step by step (e.g., with sizes 8, 8, 16 and 32 at steps one, two, three, and four, respectively), the energy-norm error does decrease in consistency with the loss (see Fig. 20(b)). Table 2 displays the losses and energy-norm errors evaluated on the test data at the end of this training with an enhanced preconditioner.



(a) Step 1 (before training) - 48 elements

(b) Step 1 (after training) - 48 elements

(c) Step 2 (before training) - 96 elements

(d) Step 2 (after training) - 96 elements

(e) Step 3 (before training) - 192 elements

(f) Step 3 (after training) - 192 elements

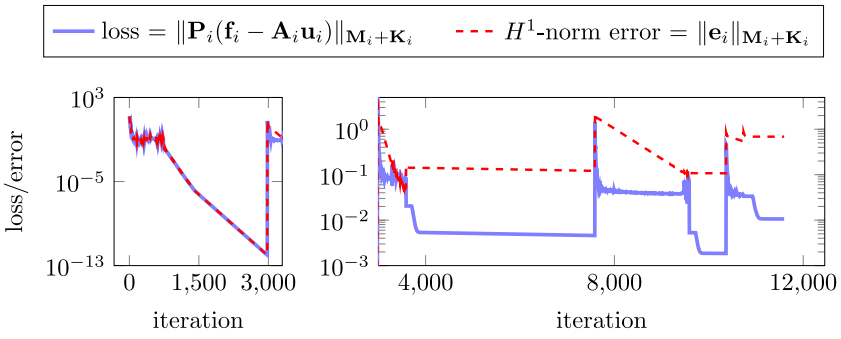
(g) Step 4 (before training) - 384 elements

(h) Step 4 (after training) - 384 elements

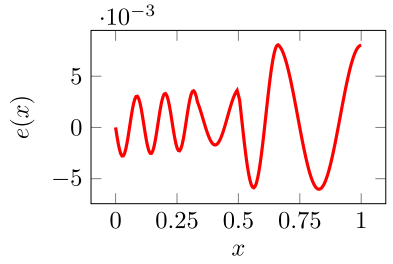
Fig. 16. Four steps of Deep-FEM predictions for Helmholtz problem (30). u_{FEM} is the finite element solution and u_{pred} is the Deep-FEM prediction.

6.4.2. Helmholtz parametric equation: $-50 < \alpha < -30$

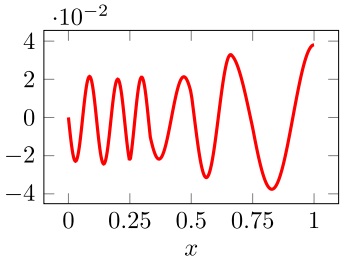
In this example, the analytic solution is $u(x) = C\sin(\sqrt{\alpha}x)$ with $C = \frac{2\pi}{\sqrt{\alpha}\cos(\sqrt{\alpha})}$. We select 100 samples uniformly distributed for the training data and we select $\{-50, -45, -40, -35, -30\}$ as the test data. We consider the H^1-norm for the loss and we maintain the same trainable blocks architecture as above. If we train the NN employing the



(a) Loss evolution along with the H^1-norm error for Helmholtz problem (28)



(b) Error function at the end of step three.
$e(x) = u_{\text{FEM}}(x) - u_{\text{pred}}(x)$.



(c) Error function at the end of step four.
$e(x) = u_{\text{FEM}}(x) - u_{\text{pred}}(x)$.

Fig. 17. End-to-end training of the Deep-FEM for problem (30) along three steps with block-Jacobi preconditioner of sizes 48, 48, 96, and 96, respectively.

Table 1
Losses and energy-norm errors of the test samples at the end of the fourth step for the reaction–diffusion problem (32) with $0 < \alpha < 200$ when employing eight-size blocks in the preconditioners.

α	0	3	15	50	200
Loss	0.0015	0.013	0.0046	0.012	0.082
Energy-norm error	0.0160	0.014	0.0061	0.014	0.095

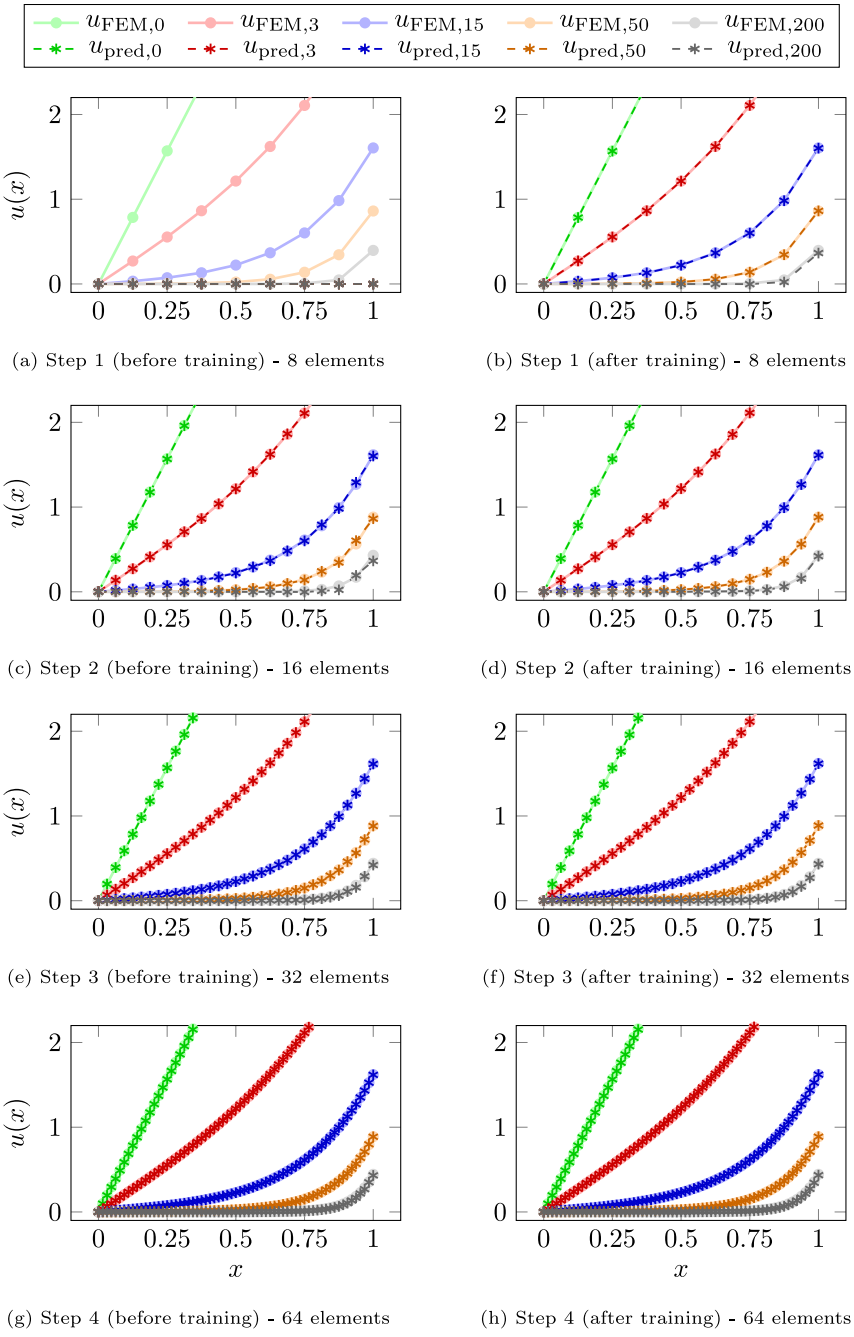
Table 2
Losses and energy-norm errors of the test samples at the end of the fourth step for the reaction–diffusion problem (32) with $0 < \alpha < 200$ when employing size-increasing blocks in the preconditioners.

α	0	3	15	50	200
Loss	0.0022	0.0059	0.0043	0.0086	0.011
Energy-norm error	0.0053	0.0075	0.0058	0.0101	0.013

same increasing block-size criterion for the preconditioners as above, we observe that in this occasion the loss does not decrease the H^1-norm error (Fig. 21(a)). If we employ the inverses as the preconditioners, the NN converges sufficiently (see Fig. 21(b)) to show adequate results. Fig. 22 shows the step-by-step predictions for the test data when the training is performed according to Fig. 21(b).

Note that utilizing the inverses is equivalent to applying the loss:

$$\ell_i(D;\theta_i) = \frac{1}{|D|}\sum_{\alpha\in D}\|\mathbf{u}_{\text{FEM},i}(\alpha) - \mathbf{u}_i(\alpha)\|_{\mathbf{M}_i+\mathbf{K}_i}, \tag{33}$$



(a) Step 1 (before training) - 8 elements

(b) Step 1 (after training) - 8 elements

(c) Step 2 (before training) - 16 elements

(d) Step 2 (after training) - 16 elements

(e) Step 3 (before training) - 32 elements

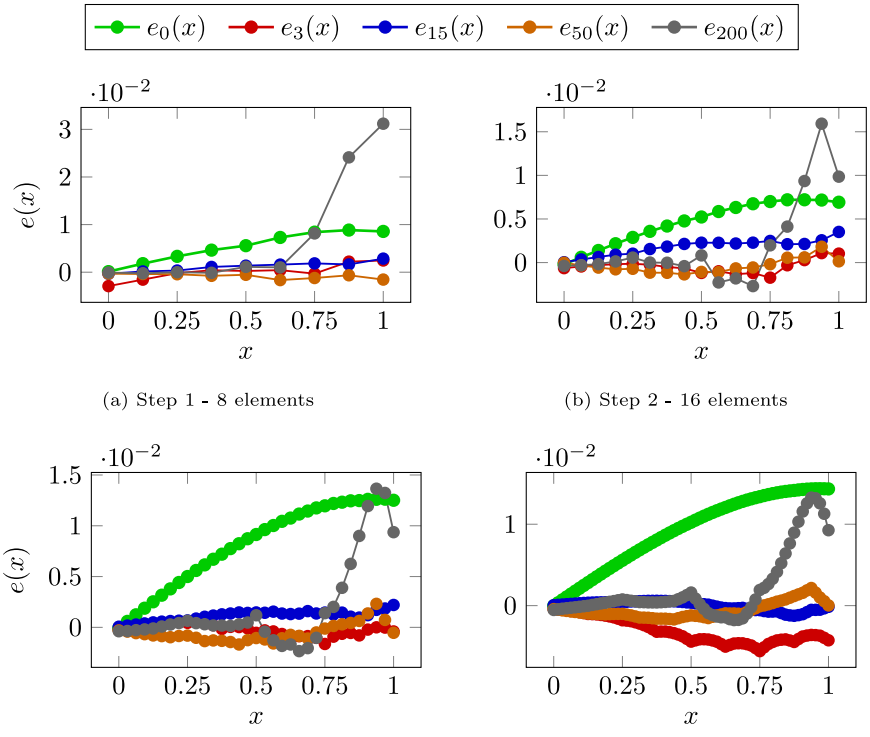
(f) Step 3 (after training) - 32 elements

(g) Step 4 (before training) - 64 elements

(h) Step 4 (after training) - 64 elements

Fig. 18. Four steps of Deep-FEM predictions on test samples for parametric Reaction–Diffusion problem (32) with $0 < \alpha < 200$. $u_{\text{FEM},\alpha} = u_{\text{FEM},\alpha}(x)$ and $u_{\text{pred},\alpha} = u_{\text{pred},\alpha}(x)$ are the FEM solution and Deep-FEM prediction evaluated at the α parameter coefficient, respectively.

where $\mathbf{u}_{\text{FEM},i}(\alpha)$ denotes the FEM solution vector for the α parameter. Precalculating all these vectors and employing (33) as the loss during training could be significantly less expensive than computing/dealing with the inverse matrices. For simplicity, along the manuscript, we computed inverses to be consistent with the presented reasoning and we refrained from reporting times, since an efficient solver should never invert a matrix explicitly (see [34,40,46,47]).



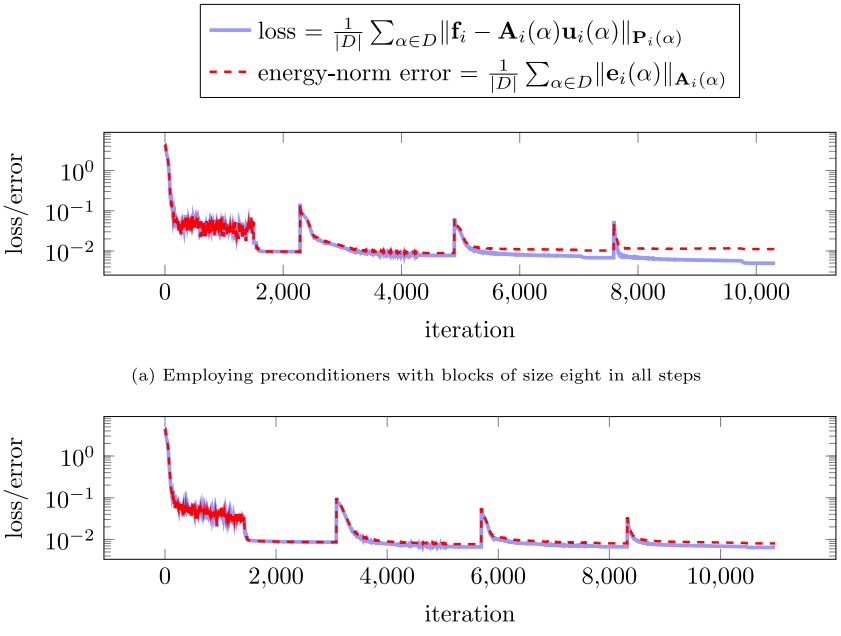
(a) Step 1 - 8 elements

(b) Step 2 - 16 elements

(c) Step 3 - 32 elements

(d) Step 4 - 64 elements

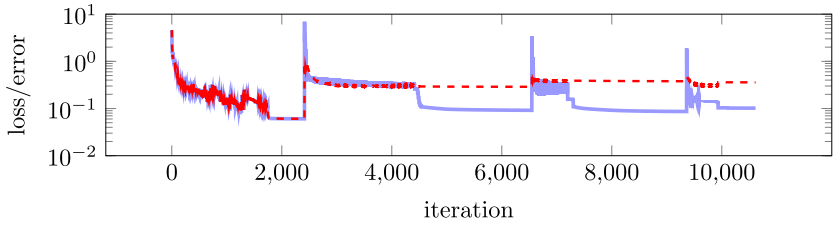
Fig. 19. Error functions at the end of each training step. $e_\alpha(x)$ is the error function evaluated at the α parameter.



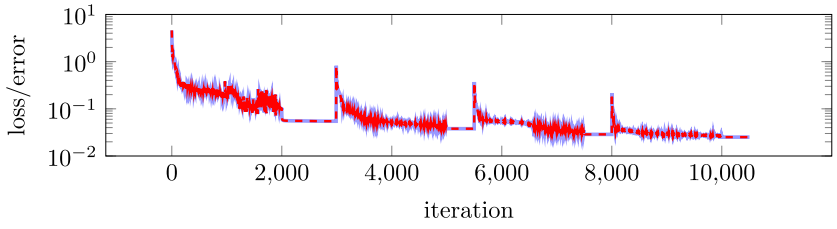
(a) Employing preconditioners with blocks of size eight in all steps

(b) Employing preconditioners with blocks of sizes 8, 8, 16 and 32 at steps one, two, three, and four, respectively

Fig. 20. Loss evolution along energy-norm error during the four-step training at problem (32) with $0 \leq \alpha \leq 200$.



(a) Employing preconditioners of block-sizes equal to 8, 8, 16 and 32 at the four steps, respectively



(b) Employing the inverses as preconditioners

Fig. 21. Loss evolution along H^1-norm error during the four-step training at problem (32) with $-50 \leq \alpha \leq -30$.

7. Conclusions and Future Work

We designed a novel DL architecture to solve PDEs that mimics a FEM. Starting from a low-dimensional solution, we dynamically scale to higher dimensional spaces starting from the solutions found in the previous subspaces. The method solves both parametric and non-parametric PDEs, depending on the number of variables we establish in the architecture. While solving non-parametric problems with Deep-FEM is computationally inefficient, the execution times when solving parametric problems are comparable to non-parametric ones (i.e., the number of employed iterations is similar), which is where the true power of the method holds. We developed and implemented the Deep-FEM in one spatial dimension with piecewise-linear approximations and uniform refinements. The extension to more complicated 2D or 3D geometries, with adaptive meshes, and with higher-order polynomials for the approximations is straightforward.

The learning approach minimizes the residual of the FEM. Since our Deep-FEM architecture is partially explainable, we identified relations between the type of training (end-to-end vs. layer-by-layer), the unknowns arising in the FEM when performing mesh-refinements, and the selected preconditioners and applied norms. Indeed, the convexity of the loss with respect to the variables is critical and prevents us from obtaining a robust method. Nonetheless, in almost all shown problems we achieve an adequate level of accuracy for engineering purposes.

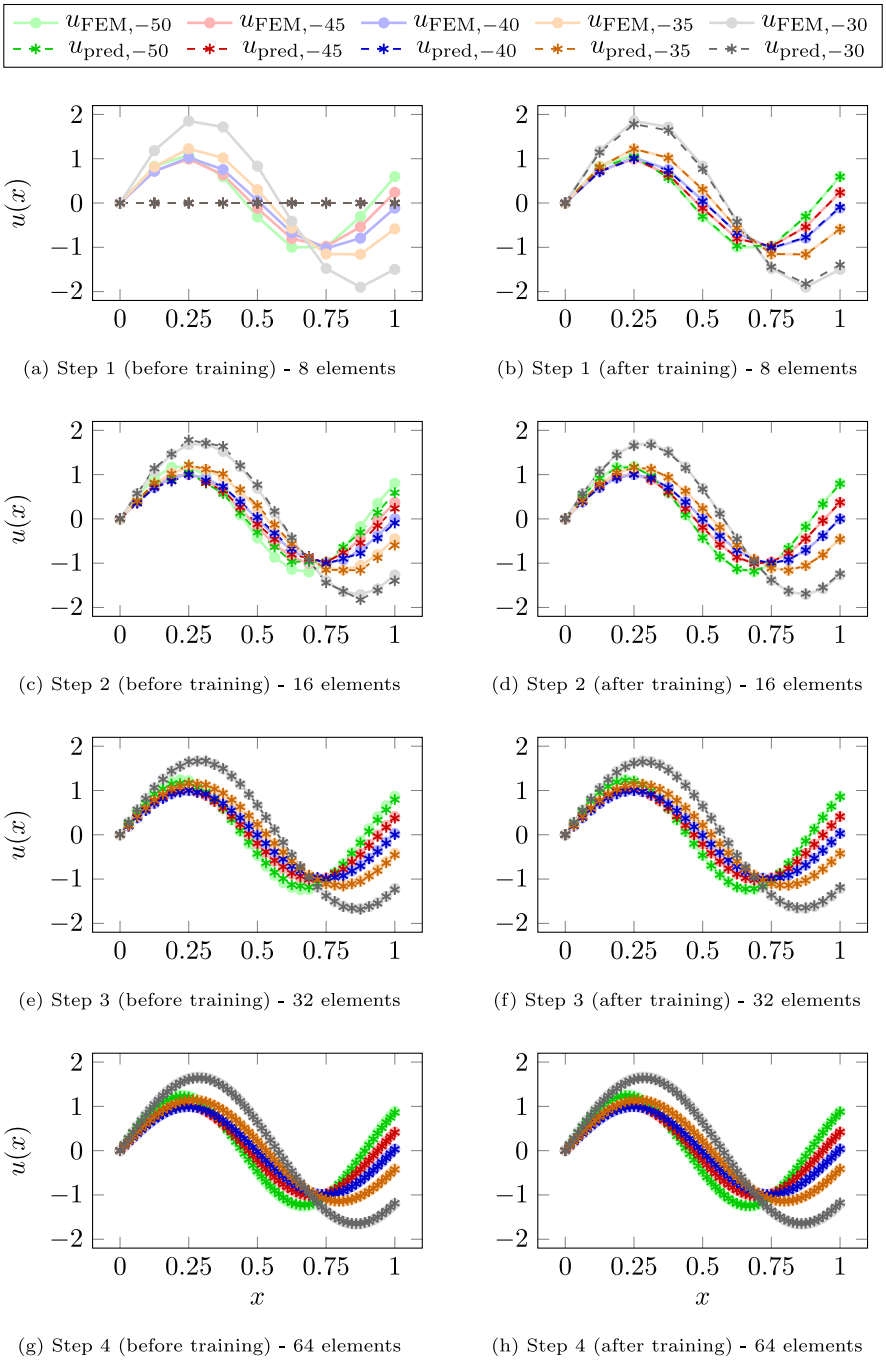
We leave as future work to compare the performance of Deep-FEM vs. other methods in context of higher-dimensional problems and/or specific applications.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has received funding from: the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement no. 777778 (MATHROCKS); the European Regional Development Fund (ERDF) through the Interreg V-A Spain-France-Andorra program POCTEFA 2014–2020 Project PIXIL (EFA362/19); the Spanish Ministry of Science and Innovation projects with references PID2019-108111RB-I00 (FEDER/AEI) and PDC2021-121093-I00, the "BCAM Severo Ochoa" accreditation of excellence (SEV-2017-0718); and the Basque Government through the BERC 2018–2021 program, the three Elkartek projects 3KIA (KK-2020/00049), EXPERTIA (KK-2021/00048), and SIGZE (KK-2021/00095), and the Consolidated Research Group MATHMODE (IT1294-19) given by the Department of Education.



(a) Step 1 (before training) - 8 elements

(b) Step 1 (after training) - 8 elements

(c) Step 2 (before training) - 16 elements

(d) Step 2 (after training) - 16 elements

(e) Step 3 (before training) - 32 elements

(f) Step 3 (after training) - 32 elements

(g) Step 4 (before training) - 64 elements

(h) Step 4 (after training) - 64 elements

Fig. 22. Four-step predictions on test samples for parametric Helmholtz problem (32) with $-50 < \alpha < -30$.

References

[1] D. La Torre, H. Kunze, F. Mendivil, M. Ruiz Galan, R. Zaki, Inverse Problems: Theory and Application to Science and Engineering 2015, Math. Problems Eng. 2015 (2015) 1–3, http://dx.doi.org/10.1155/2015/796094.

[2] F. Yaman, V.G. Yakhno, R. Potthast, Recent Theory and Applications on Inverse Problems, Math. Problems Eng. 2013 (2013) 303154, http://dx.doi.org/10.1155/2013/303154.

[3] M. Bertero, P. Boccacci, Introduction to Inverse Problems in Imaging, CRC Press, 2020, http://dx.doi.org/10.1201/9780367806941.

[4] J. Alvarez-Aramberri, D. Pardo, H. Barucq, Inversion of Magnetotelluric Measurements Using Multigoal Oriented hp-adaptivity, Procedia Comput. Sci. 18 (2013) 1564–1573, http://dx.doi.org/10.1016/j.procs.2013.05.324.
[5] A.J. Omella, R. Celorrio, D. Pardo, Sensitivity and uncertainty analysis by discontinuous Galerkin of lock-in thermography for crack characterization, Comput. Methods Appl. Mech. Eng. 373 (2021) 113523, http://dx.doi.org/10.1016/j.cma.2020.113523.
[6] S. Rojas, I. Muga, D. Pardo, A quadrature-free method for simulation and inversion of 1.5D direct current (DC) borehole measurements, Comput. Geosci. 20 (6) (2016) 1301–1318, http://dx.doi.org/10.1007/s10596-016-9592-1.
[7] A. Tarantola, Inverse Problem Theory and Methods for Model Parameter Estimation, Society for Industrial and Applied Mathematics, USA, 2004.
[8] R.C. Aster, B. Borchers, C.H. Thurber, Parameter Estimation and Inverse Problems, Elsevier, 2019, http://dx.doi.org/10.1016/C2015-0-02458-3.
[9] U. Iturrarán-Viveros, J.O. Parra, Artificial Neural Networks applied to estimate permeability, porosity and intrinsic attenuation using seismic attributes and well-log data, J. Appl. Geophys. 107 (2014) 45–54, http://dx.doi.org/10.1016/j.jappgeo.2014.05.010.
[10] J.C. Ye, Y. Han, E. Cha, Deep Convolutional Framelets: A General Deep Learning Framework for Inverse Problems, SIAM J. Imag. Sci. 11 (2) (2018) 991–1048, http://dx.doi.org/10.1137/17M1141771.
[11] G. Ongie, A. Jalal, C.A. Metzler, R.G. Baraniuk, A.G. Dimakis, R. Willett, Deep Learning Techniques for Inverse Problems in Imaging, IEEE J. Selected Areas Inf. Theory 1 (1) (2020) 39–56, http://dx.doi.org/10.1109/JSAIT.2020.2991563.
[12] M. Shahriari, D. Pardo, A. Picon, A. Galdran, J. Del Ser, C. Torres-Verdín, A deep learning approach to the inversion of borehole resistivity measurements, Comput. Geosci. 24 (3) (2020) 971–994, http://dx.doi.org/10.1007/s10596-019-09859-y.
[13] S. Alyaev, M. Shahriari, D. Pardo, A.J. Omella, D.S. Larsen, N. Jahani, E. Suter, Modeling extra-deep electromagnetic logs using a deep neural network, Geophysics 86 (3) (2021) E269–E281, http://dx.doi.org/10.1190/geo2020-0389.1.
[14] M. Shahriari, D. Pardo, J.A. Rivera, C. Torres-Verdín, A. Picon, J. Del Ser, S. Ossandón, V.M. Calo, Error control and loss functions for the deep learning inversion of borehole resistivity measurements, Int. J. Numer. Methods Eng. 122 (6) (2021) 1629–1657, http://dx.doi.org/10.1002/nme.6593.
[15] R. Khodayi-Mehr, M. Zavlanos, VarNet: Variational Neural Networks for the Solution of Partial Differential Equations, in: A.M. Bayen, A. Jadbabaie, G. Pappas, P.A. Parrilo, B. Recht, C. Tomlin, M. Zeilinger (Eds.), Proceedings Of The 2nd Conference on Learning for Dynamics and Control, in: Proceedings of Machine Learning Research, Vol. 120, PMLR, The Cloud, 2020, pp. 298–307.
[16] J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364, http://dx.doi.org/10.1016/j.jcp.2018.08.029.
[17] W. E, B. Yu, The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems, Commun. Math. Statist. 6 (1) (2018) 1–12, http://dx.doi.org/10.1007/s40304-018-0127-z.
[18] 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, http://dx.doi.org/10.1016/j.jcp.2020.109707.
[19] 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, http://dx.doi.org/10.1016/j.jcp.2018.10.045.
[20] E. Kharazmi, Z. Zhang, G.E. Karniadakis, VPINNs: Variational physics-informed neural networks for solving partial differential equations, 2019, pp. 1–24, ArXiv http://arxiv.org/abs/1912.00873.
[21] 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, http://dx.doi.org/10.1016/j.cma.2020.113547.
[22] A.D. Jagtap, G.E. Karniadakis, Extended Physics-Informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition Based Deep Learning Framework for Nonlinear Partial Differential Equations, Commun. Comput. Phys. 28 (5) (2020) 2002–2041, http://dx.doi.org/10.4208/cicp.OA-2020-0164.
[23] Z. Mao, A.D. Jagtap, G.E. Karniadakis, Physics-informed neural networks for high-speed flows, Comput. Methods Appl. Mech. Eng. 360 (2020) 112789, http://dx.doi.org/10.1016/j.cma.2019.112789.
[24] K. Shukla, P.C. Di Leoni, J. Blackshire, D. Sparkman, G.E. Karniadakis, Physics-Informed Neural Network for Ultrasound Nondestructive Quantification of Surface Breaking Cracks, J. Nondestruct. Eval. 39 (3) (2020) 61, http://dx.doi.org/10.1007/s10921-020-00705-1.
[25] 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, http://dx.doi.org/10.1016/j.camwa.2020.08.012.
[26] M. Paszyński, R. Grzeszczuk, D. Pardo, L. Demkowicz, Deep Learning Driven Self-adaptive Hp Finite Element Method, in: Computational Science – ICCS 2021, Springer International Publishing, 2021, pp. 114–121, http://dx.doi.org/10.1007/978-3-030-77961-0_11.
[27] Y. Khoo, J. Lu, L. Ying, Solving parametric PDE problems with artificial neural networks, Eur. J. Appl. Math. 32 (3) (2021) 421–435, http://dx.doi.org/10.1017/S0956792520000182.
[28] G. Kutyniok, P. Petersen, M. Raslan, R. Schneider, A Theoretical Analysis of Deep Neural Networks and Parametric PDEs, Constr. Approx. (2021) http://dx.doi.org/10.1007/s00365-021-09551-4.
[29] K. Bhattacharya, B. Hosseini, N.B. Kovachki, A.M. Stuart, Model Reduction and Neural Networks for Parametric PDEs, SMAI J. Comput. Math. 7 (2021) 121–157, http://dx.doi.org/10.5802/smai-jcm.74.
[30] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Multipole Graph Neural Operator for Parametric Partial Differential Equations, (NeurIPS 2020) 2020, http://arxiv.org/abs/2006.09535.
[31] W. Samek, T. Wiegand, K.-R. Müller, Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models, (1) 2017, pp. 39–48, http://arxiv.org/abs/1708.08296.

[32] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, F. Herrera, Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI, Inf. Fusion 58 (2020) 82–115, http://dx.doi.org/10.1016/j.inffus.2019.12.012.

[33] J.A. Rivera, J.M. Taylor, A.J. Omella, D. Pardo, On quadrature rules for solving Partial Differential Equations using Neural Networks, 2021, pp. 1–28, http://arxiv.org/abs/2111.00217.

[34] L. Sabat, C.K. Kundu, History of Finite Element Method: A Review, Springer, Singapore, 2021, pp. 395–404, http://dx.doi.org/10.1007/978-981-15-4577-1_32.

[35] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2016, pp. 770–778, http://dx.doi.org/10.1109/CVPR.2016.90.

[36] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. 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 Distributed Systems, 2015, URL http://download.tensorflow.org/paper/whitepaper2015.pdf.

[37] L. Demkowicz, P. Monk, L. Vardapetyan, W. Rachowicz, De Rham diagram for hp finite element spaces, Comput. Math. Appl. 39 (7–8) (2000) 29–38, http://dx.doi.org/10.1016/S0898-1221(00)00062-6.

[38] D.P. Kingma, J.L. Ba, Adam: A method for stochastic optimization, in: 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, 2015, pp. 1–15, http://arxiv.org/abs/1412.6980.

[39] L.K. Muller, L.M. de Lima, L. Catabriga, A comparative study of local and global preconditioners for finite element analysis, in: Proceedings of The XXXVIII Iberian Latin American Congress on Computational Methods in Engineering, 2017, http://dx.doi.org/10.20906/cps/cilamce2017-0937.

[40] D. Pardo, Integration of hp-adaptivity with a two grid solver: applications to electromagnetics, (Ph.D. thesis), University of Texas at Austin, 2004.

[41] C.R. Harris, K.J. Millman, S.J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith, R. Kern, M. Picus, S. Hoyer, M.H. van Kerkwijk, M. Brett, A. Haldane, J.F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T.E. Oliphant, Array programming with NumPy, in: Nature, 585, (7825) 2020, pp. 357–362, http://dx.doi.org/10.1038/s41586-020-2649-2.

[42] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, I. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0: fundamental algorithms for scientific computing in Python, in: Nature Methods, 17, (3) 2020, pp. 261–272, http://dx.doi.org/10.1038/s41592-019-0686-2.

[43] C. Uriarte, einsum for SparseTensors #43497, viewed 25 May 2021, URL https://github.com/tensorflow/tensorflow/issues/43497.

[44] O.G. Yalcin, Eager execution vs. Graph execution in TensorFlow: Which is better?, viewed 12 June 2021, URL https://towardsdatascience.com/eager-execution-vs-graph-execution-which-is-better-38162ea4dbf6.

[45] M.S. Alnaes, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, E. Rognes, G.N. Wells, The FEniCS Project Version 1 . 5, 3, (100) 2015, pp. 9–23, http://dx.doi.org/10.11588/ans.2015.100.20553.

[46] J. Moshfegh, M.N. Vouvakis, Direct solution of FEM models: Are sparse direct solvers the best strategy? in: 2017 International Conference on Electromagnetics In Advanced Applications (ICEAA), IEEE, 2017, pp. 1636–1638, http://dx.doi.org/10.1109/ICEAA.2017.8065603.

[47] A. Abur, A parallel scheme for the forward/backward substitutions in solving sparse linear equations, IEEE Trans. Power Syst. 3 (4) (1988) 1471–1478, http://dx.doi.org/10.1109/59.192955.

[48] J.H. Bramble, Multigrid Methods, Chapman and Hall/CRC, 2019, http://dx.doi.org/10.1201/9780203746332.

[49] Z. Allen-Zhu, Y. Li, Z. Song, A convergence theory for deep learning via over-parameterization, in: 36th International Conference on Machine Learning, ICML 2019, 2019-June, 2019, pp. 362–372, http://arxiv.org/abs/1811.03962.

[50] B. Amos, L. Xu, J.Z. Kolter, Input convex neural networks: Supplementary material, in: 34th International Conference on Machine Learning, ICML 2017, Vol. 1, 2017, pp. 192–206.

[51] S. Mishra, R. Molinaro, Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs, IMA J. Numer. Anal. (2021) http://dx.doi.org/10.1093/imanum/drab032.