Computer Methods in Applied Mechanics and Engineering 421 (2024) 116784

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

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

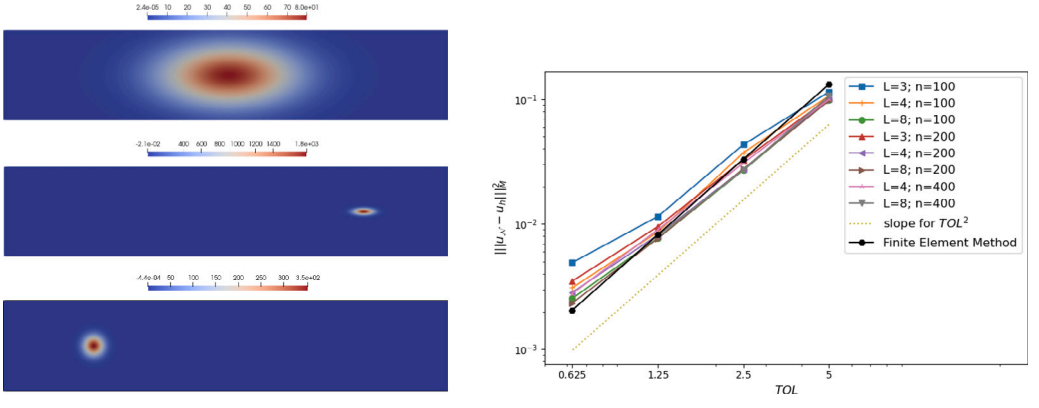
Error assessment of an adaptive finite elements—neural networks method for an elliptic parametric PDE

Alexandre Caboussat [a], Maude Girardin [a,b,*], Marco Picasso [b]

[a] Geneva School of Business Administration (HEG), University of Applied Sciences and Arts Western Switzerland (HES-SO), Rue de la Tambourine 17, 1227, Carouge, Switzerland

[b] Institute of Mathematics, EPFL, Station 8, 1015, Lausanne, Switzerland

GRAPHICAL ABSTRACT



ARTICLE INFO

MSC:
65N15
65N30
65N50
68T07
65C05

Keywords:
Error estimates
Adaptive finite element method
Parametric PDEs
Neural networks
Adaptive mesh refinement

ABSTRACT

We present a finite elements—neural network approach for the numerical approximation of parametric partial differential equations. The algorithm generates training data from finite element simulations, and uses a data-driven (supervised) feedforward neural network for the online approximation of the solution. The objective is to ensure that the overall error of the method is below some preset tolerance, and we thus control and balance the error coming from the finite element method, and the one introduced by the neural network approximation.

Two finite element methods are considered and compared; a fixed grid approach uses the same mesh for all values of the parameters, while an adaptive finite elements approach enforces the same discretization error uniformly in the parameters space.

Numerical results are presented for an elliptic model problem. The fixed grid approach shows limitations in terms of error balancing and control, while the adaptive approach allows a better accuracy and more flexibility of the method. We conclude by proposing an adaptive

* Corresponding author at: Institute of Mathematics, EPFL, Station 8, 1015, Lausanne, Switzerland.
E-mail addresses: alexandre.caboussat@hesge.ch (A. Caboussat), maude.girardin@hesge.ch, maude.girardin@epfl.ch (M. Girardin), marco.picasso@epfl.ch (M. Picasso).

https://doi.org/10.1016/j.cma.2024.116784
Received 31 October 2023; Received in revised form 22 December 2023; Accepted 15 January 2024
Available online 27 January 2024

algorithm to control the size of the training set given a network architecture and ensure that the overall error of the method is below a given tolerance.

1. Introduction

Neural networks have shown to be very efficient to approximate high dimensional functions. From the approximation point of view, the universal approximation property, as well as various convergence rates with respect to the architecture of the networks have been proved in the literature, see, e.g., [1–6]. However, most of these results are existence results, and do not discuss how to build the neural networks. The non-convex optimization problem corresponding to the training of the networks is discussed, e.g., in [7,8] and is in a less advanced state of knowledge. Theoretical results concerning the approximation of solutions of (parametric) PDEs using neural networks can be found, e.g., in [9,10] and references therein. In most cases, neural networks are trained in a so-called supervised setting, i.e., with a *data-driven* objective function [2,11,12]. If the function of interest is solution of a partial differential equation (PDE), the networks can also be trained in an unsupervised manner, leading to *Physically Informed Neural Networks* (PINNs). In this case, the objective function contains the differential equation, in strong form [13–16] or in variational form [17–19].

In the context of parametric PDEs, neural networks are generally used to approximate the parameter-to-solution map, either as a map between Hilbert spaces [20–22], or in a discretized version [9,23–26].

The underlying motivation behind this work is the numerical approximation of strongly nonlinear parametric PDEs arising in multi-physics problems, for instance laser melting processes that couple free surface flows with heat transfer and solidification [27]. The goal is to approximate the solution of such parametric PDEs efficiently, in real time, for various values of the parameters. This goal is unreachable for time-consuming finite element simulations. Since complex multi-physics problems are ultimately targeted, implementing the PDEs in the objective function of the neural network is not an easy task, preventing the use of PINNs. Moreover, as the targeted problem includes transport phenomena, classical reduced modeling approaches relying on a linear subspace, such as, e.g., reduced basis methods [28,29] or proper orthogonal decomposition [30–32], are not suitable [33,34].

We consider a generic parametric partial differential equation: find $u : \Omega \times \mathcal{P} \to \mathbb{R}$ satisfying

$$\mathcal{F}(u(x;\mu);\mu) = 0 \quad x \in \Omega, \quad \mu \in \mathcal{P}, \tag{1}$$

where $\Omega \subseteq \mathbb{R}^d$ ($d \geq 1$) is the physical space, $\mathcal{P} \subset \mathbb{R}^p$ (with $p \geq 1$ possibly large) is the parameters space and $\mathcal{F}$ is some differential operator.

We advocate a *data-driven feedforward neural network* approach, with training data generated by finite element simulations, in order to build a neural network approximation $u_{\mathcal{N}}$ of u. The numerical simulations, as well as the training of the neural network, are done during an *offline* phase, which can be time consuming but is done once and for all. Once trained, the network can be efficiently evaluated during the *online* phase.

Our objective is to control the error between u and $u_{\mathcal{N}}$ in order to ensure that it is below some preset tolerance ϵ. For this purpose, we need to control and balance two sources of error: the error coming from the finite element method, embedded in the training data, and the one coming from the neural network approximation. We thus start by discussing how these two errors can be estimated a posteriori. We then investigate their dependence with respect to different parameters and study to which extend they can be balanced with each other.

We first focus on the usage of neural networks to approximate the discretized parameter-to-solution mapping, as in [24], using a fixed finite element grid to perform all the numerical simulations. When the solution exhibits strong gradients for some values of the parameters, a very fine mesh is needed in order to reach a sufficient accuracy for all values of the parameters. Furthermore, it turns out that, with this approach, the accuracy of the neural network is lower than the accuracy of the finite element method. This implies in particular that we cannot ensure that the overall error between u and $u_{\mathcal{N}}$ is bounded by the preset tolerance.

In order to overcome these issues, we advocate a novel *adaptive finite elements—neural network* method. The finite element simulations used to generate the training data are performed using a classical isotropic adaptive mesh refinement algorithm, see [35] for instance. This implies that the structure of the network has only a scalar output; the network thus does not approximate the discretized parameter-to-solution map, but the function $(x;\mu) \mapsto u(x;\mu)$. Similarly as in [36] where a PINN is used to infer the solution of a parametric Navier–Stokes equation, the space coordinates are therefore given as an input to the network. The key ingredients of the method are the adaptive mesh refinement approach, which allows to control the accuracy of the training data uniformly in the parameters space, and an adaptive algorithm to control the size of the training set, which ensures that the error of the overall method is below the given tolerance ϵ. Numerical results on an elliptic model problem with smooth solutions confirm that the finite element error can be uniformly controlled across the parameters space, and that it can be balanced with the neural network approximation error.

This work is organized as follows: Section 2 details the error assessment of the finite element—neural network method. In Section 3, the architecture and training of the neural networks are discussed. Section 4 details numerical results for an elliptic model problem, when using the fixed and adaptive approaches. In Section 5, the adaptive algorithm to control the size of the training set is presented and discussed. Finally, the generation of a parameter-dependent mesh to evaluate the neural network solution is discussed in Section 6.

2. Error assessment

We start by detailing the error estimates of the coupled finite element-neural network approach. In order to evaluate the accuracy of the final approximation given by the neural network, we assess the $L^2(\Omega \times \mathcal{P})$-error between the exact solution u and the approximation $u_{\mathcal{N}}$. More precisely, we estimate $|||u - u_{\mathcal{N}}|||$, where

$$|||u - u_{\mathcal{N}}|||^2 := \frac{1}{|\mathcal{P}|} \int_{\mathcal{P}} \|u(\cdot;\mu) - u_{\mathcal{N}}(\cdot;\mu)\|^2 d\mu$$

and

$$\|u(\cdot;\mu) - u_{\mathcal{N}}(\cdot;\mu)\|^2 := \frac{1}{|\Omega|} \int_{\Omega} |u(x;\mu) - u_{\mathcal{N}}(x;\mu)|^2 dx.$$

For $\mu \in \mathcal{P}$, let $u_h(\cdot;\mu)$ be the finite element approximation of $u(\cdot;\mu)$ obtained with a discretization of typical size h. The error between u and $u_{\mathcal{N}}$ is decomposed as

$$|||u - u_{\mathcal{N}}||| \leq |||u - u_h||| + |||u_h - u_{\mathcal{N}}|||. \tag{2}$$

The two terms in the right-hand side of (2) correspond respectively to the error of the finite element method and the error of the neural network approximation. We start by estimating the finite element error $|||u - u_h|||$. The Monte-Carlo method is used to approximate the integral over the parameter domain, see, e.g., [37, Chapter 4]. We thus consider M parameters $\{\mu_k\}_{k=1}^M$ randomly drawn with a uniform distribution in $\mathcal{P}$, and approximate $|||u - u_h|||^2$ by

$$|||u - u_h|||_M^2 := \frac{1}{M} \sum_{k=1}^{M} \|u(\cdot;\mu_k) - u_h(\cdot;\mu_k)\|^2. \tag{3}$$

The expected value of the error between $|||u - u_h|||^2$ and $|||u - u_h|||_M^2$ is given by [37, Section 4.1]

$$\mathbb{E}\left[\left| |||u - u_h|||^2 - |||u - u_h|||_M^2 \right|\right] = \sqrt{\frac{\mathrm{Var}\left(\|u(\cdot;\mu) - u_h(\cdot;\mu)\|^2\right)}{M}} = \frac{\mathrm{Std}\left(\|u(\cdot;\mu) - u_h(\cdot;\mu)\|^2\right)}{\sqrt{M}},$$

where Var denotes the variance and Std the standard deviation. The standard deviation can then be estimated over the sample of size M, which can be chosen in such a way that the computed expected error is smaller than a given tolerance. Since the exact solution u is not known in general, the error $\|u(\cdot;\mu) - u_h(\cdot;\mu)\|^2$ can be bounded above by an *a posteriori* error estimate

$$\|u(\cdot;\mu) - u_h(\cdot;\mu)\|^2 \leq \frac{C}{|\Omega|} \eta^2(u_h(\cdot;\mu);\mu),$$

where the error estimator η depends on the differential operator $\mathcal{F}$ and C is a constant (independent of h and of solution u) [35]. Then

$$|||u - u_h|||_M^2 \leq C \eta_M^2(u_h),$$

with

$$\eta_M^2(u_h) := \frac{1}{M} \frac{1}{|\Omega|} \sum_{k=1}^{M} \eta^2(u_h(\cdot;\mu_k);\mu_k).$$

To estimate the error of the neural network approximation, the Monte-Carlo method is used again to approximate $|||u_{\mathcal{N}} - u_h|||^2$ by

$$|||u_{\mathcal{N}} - u_h|||_M^2 := \frac{1}{M} \sum_{k=1}^{M} \|u_h(\cdot;\mu_k) - u_{\mathcal{N}}(\cdot;\mu_k)\|^2. \tag{4}$$

The integrals over Ω are then computed using an overkill quadrature formula (of sufficiently high order). For ease of notation, the corresponding result is again denoted by $|||u_{\mathcal{N}} - u_h|||_M^2$.

3. Fully connected feedforward neural networks

To approximate u_h by $u_{\mathcal{N}}$, we use fully connected feedforward neural networks. We recall hereafter the main characteristics of such networks (see, e.g., [38] for a complete description).

A feedforward neural network is made up of an input layer, an output layer and $L \geq 1$ hidden layers. We denote by n_j the number of neurons of the j^{th} layer, $j = 0, \ldots, L+1$. We let σ_i^j and z_i^j be respectively the activation function and the value associated to the i^{th} neuron of the j^{th} layer, $i = 1, \ldots, n_j$, $j = 0, \ldots, L+1$. Possible activation functions for neurons in hidden layers are the hyperbolic

tangent, the Rectified Linear Unit – defined by $ReLU(x) = \max\{x, 0\}$ – or the softplus function – given by $softplus(x) = \ln(1 + e^x)$. For neurons in the output layer, the activation function is the identity. The value associated to a neuron is recursively given by

$$z_i^j = \sigma_i^j\Big(\sum_{k=1}^{n_{j-1}} a_{ik}^j z_k^{j-1} + b_i^j\Big),\ i = 1, \dots, n_j,\ j = 1, \dots, L+1,$$

where a_{ik}^j and b_i^j are respectively the weights and biases of the neural network, and z^0 is the input. We denote by θ the set of trainable parameters of the network, i.e. the set of all a_{ik}^j and b_i^j. Similarly as in [2], we denote by $\mathcal{Y}^{W,L}(\sigma; d_{in}, d_{out})$ the set of fully-connected feedforward neural networks with input dimension d_{in}, output dimension d_{out}, and L hidden layers, each constituted of W neurons having σ as activation function. Note that a neural network $\mathcal{N} \in \mathcal{Y}^{W,L}(\sigma; d_{in}, d_{out})$ has $N_\theta = (d_{in}+1)W + W(W+1)(L-1) + d_{out}(W+1)$ trainable parameters. Once these parameters have been set, the network provides a function $S_{\mathcal{N}}(\cdot;\theta) : \mathbb{R}^{d_{in}} \mapsto \mathbb{R}^{d_{out}}$. To build feedforward neural networks that approximate the solution to (1), we consider two approaches which are described in the following sections.

3.1. Fixed grid approach

Let $\mathcal{T}_h$ be a discretization of Ω with elements of size less than h. Let N_h denote the number of vertices of the discretization, and consider the continuous, piecewise linear basis functions $\{\varphi^i\}_{i=1}^{N_h}$ associated to the vertices $\{x^i\}_{i=1}^{N_h}$ (we do not discuss the implementation of boundary conditions here in order to simplify the presentation). Let $u_h(x;\mu)$ be the finite element approximation of $u(x;\mu)$, written as

$$u_h(x;\mu) = \sum_{i=1}^{N_h} U_h^i(\mu)\varphi^i(x).$$

As in [24], the goal is thus to approximate the mapping

$$\mu \mapsto U_h(\mu) = \begin{pmatrix} U_h^1(\mu) \\ \vdots \\ U_h^{N_h}(\mu) \end{pmatrix}.$$

We therefore consider a neural network $\mathcal{N} \in \mathcal{Y}^{W,L}(\sigma; p, N_h)$. Once its trainable parameters θ are set and given $\mu \in \mathbb{R}^p$, the neural network provides an output $U_{\mathcal{N}}(\mu;\theta) \in \mathbb{R}^{N_h}$. The training procedure is summarized as follows:

1. Set the discretization $\mathcal{T}_h$.
2. Select the training parameters $\{\tilde{\mu}_j\}_{j=1}^{N_{train}} \subseteq \mathcal{P}$.
3. For each $\tilde{\mu}_j$, compute a piecewise linear finite element approximation $u_h(x;\tilde{\mu}_j) = \sum_{i=1}^{N_h} U_h^i(\tilde{\mu}_j)\varphi^i(x)$ of $u(x;\tilde{\mu}_j)$.
4. Set the architecture of the neural network, i.e. L, W and σ. Then choose the trainable parameters θ of $\mathcal{N} \in \mathcal{Y}^{W,L}(\sigma; p, N_h)$ in order to minimize $\Phi(\theta) := \mathcal{L}_{N_{train}}(U_{\mathcal{N}}(\cdot;\theta); U_h)$. Here $\mathcal{L}_{N_{train}}$ is defined as

$$\mathcal{L}_{N_{train}}(U_{\mathcal{N}}(\cdot;\theta); U_h) := \frac{1}{N_{train}}\frac{1}{N_h}\sum_{j=1}^{N_{train}}\sum_{i=1}^{N_h} c_j^i |U_{\mathcal{N}}^i(\tilde{\mu}_j;\theta) - U_h^i(\tilde{\mu}_j)|^2, \tag{5}$$

   where the coefficients c_j^i will be defined later. In practice, the minimization problem is solved using a gradient descent algorithm, or one of its variants [39,40]. Let θ^* denote the set of parameters returned by the optimization algorithm after step 4. For ease of notation, we denote $U_{\mathcal{N}}(\mu)$ instead of $U_{\mathcal{N}}(\mu;\theta^*)$ in the sequel.

For the fixed grid approach, the training set is thus composed of the parameters $\{\tilde{\mu}_j\}_{j=1}^{N_{train}}$ and of the corresponding solutions $\{U_h(\tilde{\mu}_j)\}_{j=1}^{N_{train}}$. Choosing $c_j^i = 1$ in (5) defines the objective function

$$\mathcal{L}^{\ell^2}_{N_{train}}(U_{\mathcal{N}}(\cdot;\theta); U_h) := \frac{1}{N_{train}}\frac{1}{N_h}\sum_{j=1}^{N_{train}}\sum_{i=1}^{N_h} |U_{\mathcal{N}}^i(\tilde{\mu}_j;\theta) - U_h^i(\tilde{\mu}_j)|^2, \tag{6}$$

which corresponds to the classical Mean Squared Error. Choosing

$$c_j^i = \frac{|\Omega(x^i)|}{d+1}, \qquad \text{where} \qquad |\Omega(x^i)| = \sum_{\substack{K\in\mathcal{T}_h \\ x^i\in K}} |K|$$

defines the objective function

$$\mathcal{L}^{L^2}_{N_{train}}(U_{\mathcal{N}}(\cdot;\theta); U_h) := \frac{1}{N_{train}}\frac{1}{N_h}\sum_{j=1}^{N_{train}}\sum_{i=1}^{N_h} \frac{|\Omega(x^i)|}{d+1} |U_{\mathcal{N}}^i(\tilde{\mu}_j;\theta) - U_h^i(\tilde{\mu}_j)|^2. \tag{7}$$

Once the network has been trained, a function $u_{\mathcal{N}} : \Omega \times \mathcal{P} \to \mathbb{R}$ can be recovered by setting

$$u_{\mathcal{N}}(x;\mu) := \sum_{i=1}^{N_h} U_{\mathcal{N}}^i(\mu)\varphi^i(x).$$

Then $\sum_{i=1}^{N_h} \frac{|\Omega(x^i)|}{d+1} |U^i_{\mathcal{N}}(\mu;\theta) - U^i_h(\mu)|^2$ corresponds to $\|u_h(\cdot;\mu) - u_{\mathcal{N}}(\cdot;\mu)\|_{L^2(\Omega)}$ approximated with the trapeze quadrature formula.

3.2. Adapted grid approach

For this second approach, a mesh adaptation algorithm is used to compute the numerical solutions, to ensure that all the latter have an accuracy close to a preset tolerance. The finite element meshes are thus different for each parameter. We therefore cannot approximate the discretized parameter-to-solution map $\mu \mapsto U(\mu) \in \mathbb{R}^{N_h}$, without interpolating all the finite element solutions on a common, sufficiently fine, mesh. In order to avoid interpolation procedures, the mapping approximated by the network must be changed. In particular, its input and output dimensions must be modified: we now consider a network that takes as input both the parameter μ and the space variable x, and outputs an approximation of $u(x;\mu) \in \mathbb{R}$. The input and output dimensions of the network are thus respectively given by $d_{in} = p+d$ and $d_{out} = 1$, which amounts to considering a neural network $\mathcal{N} \in \Upsilon^{W,L}(\sigma; d+p, 1)$. Remark that in this case, the networks are independent of the finite element meshes; any $x \in \Omega$ could thus in theory be chosen as a training point. Nevertheless, since the adaptive mesh algorithm is designed to add vertices in region of interest of the solution, we decided to consider only vertices of the adapted meshes as training points. One advantage is that any interpolation issues are avoided.

Once its trainable parameters θ have been set, the network provides, for $(x;\mu) \in \mathbb{R}^d \times \mathbb{R}^p$, $u_{\mathcal{N}}(x;\mu;\theta) \in \mathbb{R}$. The training procedure is as follows:

1. Select the training parameters $\{\tilde{\mu}_j\}_{j=1}^{N_{train}} \subseteq \mathcal{P}$.
2. For each $\tilde{\mu}_j$, set an appropriate discretization $\mathcal{T}_j$ of Ω with piecewise linear basis functions $\{\varphi^i_j\}_{i=1}^{N_j}$ associated to the vertices $\{x^i_j\}_{i=1}^{N_j}$. Compute a piecewise linear finite element approximation $u_h(x;\tilde{\mu}_j) = \sum_{i=1}^{N_j} U^i_h(\tilde{\mu}_j)\varphi^i_j(x)$ of $u(x;\tilde{\mu}_j)$.
3. Set the architecture of the neural network, that is L, W and σ. Then choose the trainable parameters θ of $\mathcal{N} \in \Upsilon^{W,L}(\sigma; p+d, 1)$ in order to minimize $\Phi(\theta) := \mathcal{L}_{N_{train}}(u_{\mathcal{N}}(\cdot;\theta); u_h)$. Here $\mathcal{L}_{N_{train}}$ is defined as:

$$\mathcal{L}_{N_{train}}(u_{\mathcal{N}}(\cdot;\theta); u_h) := \frac{1}{\sum_{j=1}^{N_{train}} N_j} \sum_{j=1}^{N_{train}} \sum_{i=1}^{N_j} c^i_j |u_{\mathcal{N}}(x^i_j;\tilde{\mu}_j;\theta) - u_h(x^i_j;\tilde{\mu}_j)|^2,$$

with the coefficients c^i_j corresponding to those of (6) or (7). As before, we let θ^* be the set of parameters obtained by some gradient descent type algorithm and denote $u_{\mathcal{N}}(x;\mu;\theta^*)$ simply by $u_{\mathcal{N}}(x;\mu)$ in what follows.

In this case, the training set is composed by $\{(x^i_j;\tilde{\mu}_j)_{i=1}^{N_j}\}_{j=1}^{N_{train}}$, with the corresponding solutions $\{(u_h(x^i_j;\tilde{\mu}_j))_{i=1}^{N_j}\}_{j=1}^{N_{train}}$.

In the next section we consider a model problem to test both the accuracy of the finite element method and of the neural network, and to compare the fixed and adapted grid approaches presented above.

4. Numerical experiments

As a model problem, we consider the 2D parametric Poisson problem

$$\begin{cases} -\Delta u(x;\mu) &= f(x;\mu) \quad x \in \Omega,\ \mu \in \mathcal{P}, \\ u(x;\mu) &= g(x;\mu) \quad x \in \partial\Omega,\ \mu \in \mathcal{P}, \end{cases} \tag{8}$$

with $d = 2$, $\Omega = [0,10] \times [0,2]$, $p = 4$, $\mathcal{P} = [1.5, 8.5] \times [1, 10] \times [100, 1000] \times [0.3, 2]$. Here f and g are such that the exact solution u of (8) is given by

$$u(x;\mu) = \frac{\mu_3}{\pi\mu_4^2} \exp\{-2\mu_4^{-2}((x_1-\mu_1)^2 + \mu_2(x_2-1)^2)\}.$$

Fig. 1 visualizes snapshots of the solutions for several values of the parameters $\mu = (\mu_1, \mu_2, \mu_3, \mu_4)$.

This test case has been chosen since its solutions are reminiscent of the temperature fields that can be observed during laser polishing processes. Furthermore, even if the solutions are smooth, the fact that their support is localized and that they are translated in Ω make the solution manifold hard to approximate when using linear approximation spaces, such as, e.g., reduced basis [33,34].

4.1. Fixed grid approach

4.1.1. Finite element method

The classical (isotropic) residual-based *a posteriori* error estimator is given, for each $\mu \in \mathcal{P}$, by

$$\eta^2(u_h(\cdot;\mu);\mu) = \sum_{K \in \mathcal{T}_h} \eta^2_K(u_h(\cdot;\mu);\mu),$$

see, e.g., [41], with

$$\eta_K(u_h(\cdot;\mu);\mu) = h^2_K \|(\Delta u_h + f)(\cdot;\mu)\|^2_{L^2(K)} + \frac{1}{2} h_K^{\frac{3}{2}} \|[\nabla u_h(\cdot;\mu) \cdot n_K]\|^2_{L^2(\partial K)}, \tag{9}$$

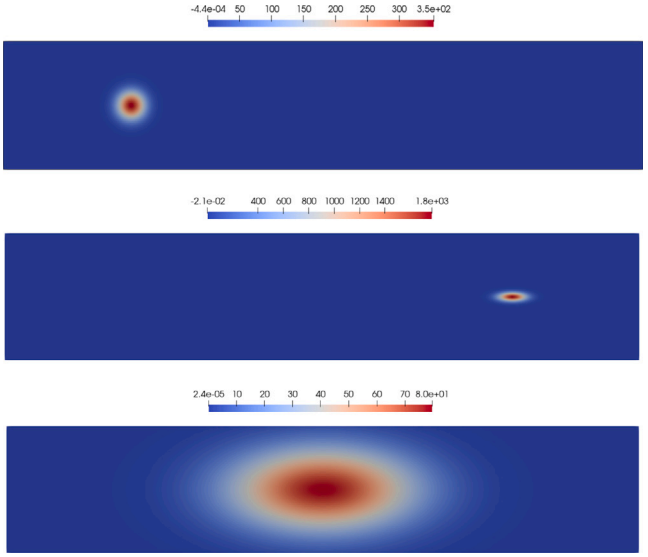


Fig. 1. Finite element solutions $u_h(\cdot;\mu)$ computed on a uniform mesh of size $h = 0.0125$. From top to bottom : $\mu = (2, 1, 100, 0.3)$, $\mu = (8, 10, 500, 0.3)$, $\mu = (5, 5, 1000, 2)$.

Table 1
Computed error of the finite element method for various values of h and M.

h	M	$\|u-u_h\|_M^2$	$\mathrm{Std}_M(\|u-u_h\|^2)$	$\frac{\mathrm{Std}_M(\|u-u_h\|^2)}{\sqrt{M}}$	ei_M	$\mathrm{Std}_M(ei)$
0.05	500	$5.51 \cdot 10^{-1}$	2.30	$1.03 \cdot 10^{-1}$	11	$2.93 \cdot 10^{-1}$
	1000	$3.83 \cdot 10^{-1}$	1.75	$5.53 \cdot 10^{-2}$	11	$2.73 \cdot 10^{-1}$
	2000	$7.14 \cdot 10^{-1}$	3.40	$7.60 \cdot 10^{-2}$	11	$2.89 \cdot 10^{-1}$
	4000	$5.78 \cdot 10^{-1}$	2.8	$4.43 \cdot 10^{-2}$	11	$2.85 \cdot 10^{-1}$
0.025	500	$3.82 \cdot 10^{-2}$	$1.58 \cdot 10^{-1}$	$7.10 \cdot 10^{-3}$	10.9	$2.21 \cdot 10^{-1}$
	1000	$4.20 \cdot 10^{-2}$	$2.07 \cdot 10^{-1}$	$6.55 \cdot 10^{-3}$	10.9	$2.00 \cdot 10^{-1}$
	2000	$4.65 \cdot 10^{-2}$	$2.41 \cdot 10^{-1}$	$5.40 \cdot 10^{-3}$	10.9	$2.06 \cdot 10^{-1}$
	4000	$3.83 \cdot 10^{-2}$	$1.96 \cdot 10^{-1}$	$3.10 \cdot 10^{-3}$	10.9	$2.12 \cdot 10^{-1}$
0.0125	500	$2.06 \cdot 10^{-3}$	$1.06 \cdot 10^{-2}$	$4.76 \cdot 10^{-4}$	10.9	$1.80 \cdot 10^{-1}$
	1000	$2.26 \cdot 10^{-3}$	$1.17 \cdot 10^{-2}$	$3.69 \cdot 10^{-4}$	10.9	$1.75 \cdot 10^{-1}$
	2000	$2.06 \cdot 10^{-3}$	$9.22 \cdot 10^{-3}$	$2.06 \cdot 10^{-4}$	10.9	$1.72 \cdot 10^{-1}$
	4000	$2.46 \cdot 10^{-3}$	$1.17 \cdot 10^{-2}$	$1.86 \cdot 10^{-4}$	10.9	$1.77 \cdot 10^{-1}$

where $[\nabla u_h(\cdot;\mu) \cdot n_K]$ is the jump of the normal derivative across the edges of ∂K. In order to track the sharpness of the error estimator, we use the effectivity index

$$ei(\mu) := \frac{\left(\sum_{K\in\mathcal{T}_h} \eta_K^2(u_h(\cdot;\mu);\mu)\right)^{1/2}}{\|u(\cdot;\mu) - u_h(\cdot;\mu)\|_{L^2(\Omega)}}. \tag{10}$$

The error of the finite element method is estimated with the setting presented in Section 2 using test sets $\{\mu_k\}_{k=1}^M$. Numerical results are shown in Table 1, where ei_M and $\mathrm{Std}_M(ei)$ denote respectively the mean and the standard deviation of the set $\{ei(\mu_k)\}_{k=1}^M$; similarly $\mathrm{Std}_M(\|u-u_h\|^2)$ denotes the standard deviation of $\{\|u(\cdot;\mu_k) - u_h(\cdot;\mu_k)\|^2\}_{k=1}^M$.

First, we note that $\|\!|u-u_h|\!\|_M^2$ behave as $\mathcal{O}(h^4)$, as expected. Second, the mean of the effectivity index does not depend on h or M, and its standard deviation is small; this suggest that the effectivity index depends only weakly on μ. Next, for $M = 4000$, the expected error of the Monte-Carlo method (fifth column) is always one order of magnitude smaller than $\|\!|u-u_h|\!\|_M^2$; the first significant digit of the latter is thus correct, in expectation. Finally, $\mathrm{Std}_M(\|u-u_h\|^2)$ is large compared to the computed quantity $\|\!|u-u_h|\!\|_M^2$; the accuracy of the finite element method thus highly depends on μ.

Table 2 visualizes the time needed to solve the linear system during the finite element simulations; WT corresponds to the wall time (*Intel Core, 3.5 GHz*) needed to solve the system, averaged over 2000 resolutions. As expected, WT behaves roughly as $\mathcal{O}(N_h^{\frac{3}{2}}) = \mathcal{O}(h^{-3})$.

4.1.2. Neural networks

We next turn to the construction of neural networks to approximate the solution of (8), using the first of the two approaches presented in Section 3, and therefore networks belonging to $\mathcal{Y}^{W,L}(\sigma; 4, N_h)$.

Table 2
Wall time to solve the linear system for various values of h.

h	N_h	WT [s]
0.05	9626	0.10
0.025	38 524	1.33
0.0125	154 349	10.4

Table 3
Wall time needed to evaluate neural networks of different widths, depths and output dimensions.

L	W	N_h	WT [s]
2	100	9626	$3.48 \cdot 10^{-2}$
		38 524	$3.53 \cdot 10^{-2}$
		154 349	$3.62 \cdot 10^{-2}$
4	800	9626	$3.51 \cdot 10^{-2}$
		38 524	$3.60 \cdot 10^{-2}$
8	100	9626	$3.68 \cdot 10^{-2}$
		38 524	$3.67 \cdot 10^{-2}$

All the neural networks are built and trained using the opensource library `Keras` [40]. For this test case, we take $\sigma = softplus$ as activation function. The initial weights of the networks are chosen using the Glorot Normal initialization [42], which amounts to choosing them with a normal distribution centered at 0 and whose standard deviation depends on the number of input and output units of the layer. The neural networks are trained with the Nadam optimizer [43], starting from a learning rate of 0.001 which is decreased when a plateau is reached, using batches of size 32 and early stopping. The training parameters $\{\tilde{\mu}_j\}_{j=1}^{N_{train}}$ are chosen randomly with a uniform distribution in $\mathcal{P}$. As advocated in [40], the parameters are then normalized so that all the four components have mean zero and standard deviation one. Since we consider uniform meshes, the two objective functions $\mathcal{L}^{L^2}$ and $\mathcal{L}^{\ell^2}$, defined by (6) and (7) respectively, are similar. As we want to track the L^2 error between $u_{\mathcal{N}}$ and u_h, we use $\mathcal{L}^{L^2}$ to train the networks for consistency. To estimate the error of the neural networks, we use the same test sets (that is the same μ_k, $k = 1, \dots, M$) that we used to estimate the error of the finite element method $|||u - u_h|||_M$.

One of the main advantages of this fixed grid approach is that, for a given $\mu \in \mathcal{P}$, an approximation of $u_h(\cdot;\mu)$ is obtained at all vertices of the finite element grid in a single evaluation of the neural network. The time needed to evaluate neural networks (averaged over 2000 evaluations) of different sizes using the graphical card *Gigabyte GeForce RTX 3080* is reported in Table 3. By comparing the results in Tables 2 and 3, we note that, for the network architectures and the mesh sizes considered here, the network evaluation is faster than solving the linear system to obtain u_h.

We next test the impact of different parameters on the accuracy of the neural networks. Note that, when training several times a neural network $\mathcal{N} \in Y^{W,L}(\sigma; 4, N_h)$, with fixed W, L and σ, the variability of $|||u_{\mathcal{N}} - u_h|||_M$ is typically smaller than 10% of the value of the error itself. All the results in the sequel are thus the result of a single training phase for all neural networks.

We start by letting the architecture of the networks vary, while keeping $N_{train} = 8000$ constant. The structure of the networks depends on h and the number of trainable parameters is given by

$$\begin{aligned} N_\theta &= (d_{in}+1)W + W(W+1)(L-1) + d_{out}(W+1) \\ &= 5W + W(W+1)(L-1) + N_h(W+1) \\ &=: N'_\theta + N_h(W+1), \end{aligned}$$

where N'_θ corresponds to the number of trainable parameters of the hidden layers. Table 4 shows the error of neural networks with different widths, depths and output sizes. As required, the expected error of the Monte-Carlo method (last column) is small compared to the computed value of $|||u_h - u_{\mathcal{N}}|||_M$. We also remark that the standard deviation of $\|u_h - u_{\mathcal{N}}\|^2$ is large compared to $|||u_{\mathcal{N}} - u_h|||_M^2$; the accuracy of the networks thus varies widely across the parameters space. For a fixed width (resp. depth) increasing the depth (resp. width) allows to improve the accuracy of the networks. Nevertheless, comparing the values in Tables 1 and 4, the error of all neural networks remains much higher than the error of the finite element method for this training set' size.

We next set $h = 0.05$ ($N_h = 9626$) and test the impact of N_{train} on the accuracy of the networks. Fig. 2 illustrates the numerical results obtained. We observe that $|||u_h - u_{\mathcal{N}}|||_M^2$ behaves roughly as $\mathcal{O}(N_{train}^{-1})$ for the neural networks tested here. However, even for the largest number of training examples considered here, the error of all the neural networks considered remains higher than the error of the finite element method.

The use of neural networks $\mathcal{N} \in Y^{W,L}(\sigma; p, N_h)$ has thus two main drawbacks. First, all the finite element simulations must be performed on the same grid; this implies in particular that the standard deviation of the finite element error in the parameter domain remains large. Second, even for a large number of training examples and the larger mesh size h, we are not able to reach the same accuracy for the neural network approximation than for the finite element method. In order to overcome these drawbacks, we propose, in the next section, an adapted grid approach to build the neural networks.

Table 4
Error of neural networks for various values of L, W and N_h. $N_{train} = 8000$, $M = 4000$.

L	W	N'_θ	N_h	N_θ	$\|u_\mathcal{N} - u_h\|^2_M$	$\mathrm{Std}_M(\|u_h - u_\mathcal{N}\|^2)$	$\frac{\mathrm{Std}_M(\|u_h - u_\mathcal{N}\|^2)}{\sqrt{M}}$
2	100	10 600	9626	982 826	93.3	379	6.00
			38 524	3 901 524	108	450	7.12
2	200	41 200	9626	1 976 026	71.0	320	5.06
			38 524	7 784 524	83.0	391	6.18
2	400	162 400	9626	4 022 426	46.4	268	4.23
			38 524	15 610 524	56.7	316	4.99
4	100	30 800	9626	1 003 026	33.6	162	2.56
			38 524	3 921 724	32.6	174	2.76
			154 349	15 620 049	30	121	1.91
4	200	121 600	9626	2 056 426	18.6	125	1.98
			38 524	7 864 924	15.8	116	1.83
			154 349	31 145 749	14.9	66.6	1.05
4	400	483 200	9626	4 343 226	8.76	67.9	1.07
			38 524	15 931 324	10.7	95.8	1.51
			154 349	61 893 949	6.07	29.1	$4.61 \cdot 10^{-1}$
4	800	1 926 400	9626	9 636 826	5.50	53.4	$8.44 \cdot 10^{-1}$
			38 524	32 784 124	6.23	65.3	1.03
8	100	71 200	9626	1 043 426	13.3	58.9	$9.31 \cdot 10^{-1}$
			38 524	3 962 124	16.1	87.5	1.38
8	200	282 400	9626	2 217 226	7.76	36.4	$5.76 \cdot 10^{-1}$
			38 524	8 025 724	7.01	44.9	$7.10 \cdot 10^{-1}$
8	400	1 124 800	9626	4 984 826	4.79	26.5	$4.19 \cdot 10^{-1}$
			38 524	16 572 924	6.49	53.5	$8.46 \cdot 10^{-1}$
8	800	4 489 600	9626	12 200 026	2.64	20.3	$3.21 \cdot 10^{-1}$
			38 524	35 347 324	3.78	34.2	$5.42 \cdot 10^{-1}$

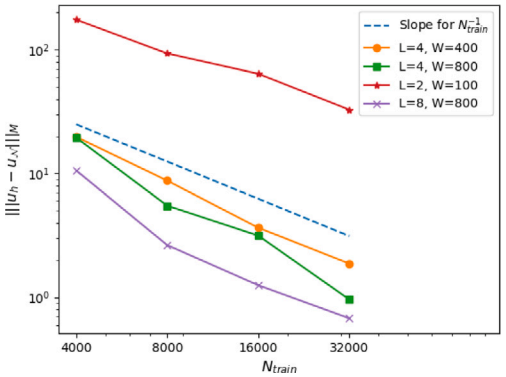


Fig. 2. Computed error of neural networks of different sizes as a function of N_{train} for $h = 0.05$ and $M = 4000$.

4.2. Adapted grid approach

4.2.1. Adaptive finite element method

The goal of the adaptive finite element algorithm is, given a parameter μ, to build a discretization $\mathcal{T}_h$ such that the error is close to a preset tolerance TOL, namely

$$0.75\ TOL \leq \left(\frac{\|u(\cdot;\mu) - u_h(\cdot;\mu)\|^2_{L^2(\Omega)}}{|\Omega|} \right)^{\frac{1}{2}} \leq 1.25\ TOL.$$

In practice, the error is replaced by the error estimator defined in (9) and we require that

$$0.75\ TOL \leq \left(\frac{\sum_{K\in\mathcal{T}_h} \eta^2_K(u_h(\cdot;\mu);\mu)}{|\Omega|} \right)^{\frac{1}{2}} \leq 1.25\ TOL.$$

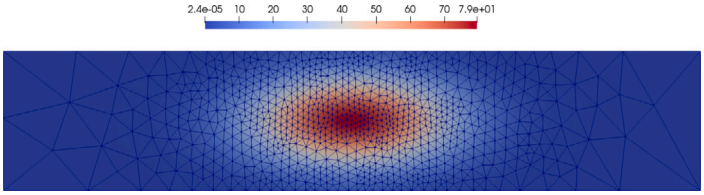


Fig. 3. Finite element solution $u_h(\cdot;\mu)$ and corresponding adapted mesh for $\mu = (5, 5, 1000, 2)$ and $TOL = 1.25$.

Table 5
Computed error of the finite element method for various values of TOL and M.

TOL	M	$\overline{N}_M$	$\|u-u_h\|^2_M$	$\mathrm{Std}_M(\|u-u_h\|^2)$	$\frac{\mathrm{Std}_M(\|u-u_h\|^2)}{\sqrt{M}}$	ei_M	$\mathrm{Std}_M(ei)$
5	500	410	$1.31 \cdot 10^{-1}$	$2.86 \cdot 10^{-2}$	$1.27 \cdot 10^{-3}$	11.3	1.02
	1000	403	$1.31 \cdot 10^{-1}$	$2.92 \cdot 10^{-2}$	$9.23 \cdot 10^{-4}$	11.3	1.03
	2000	409	$1.31 \cdot 10^{-1}$	$2.76 \cdot 10^{-2}$	$6.18 \cdot 10^{-4}$	11.3	1.01
2.5	500	749	$3.25 \cdot 10^{-2}$	$5.97 \cdot 10^{-3}$	$2.67 \cdot 10^{-4}$	11.4	$9.10 \cdot 10^{-1}$
	1000	768	$3.29 \cdot 10^{-2}$	$6.04 \cdot 10^{-3}$	$1.92 \cdot 10^{-4}$	11.4	$9.12 \cdot 10^{-1}$
	2000	767	$3.31 \cdot 10^{-2}$	$6.60 \cdot 10^{-3}$	$1.48 \cdot 10^{-4}$	11.3	$9.39 \cdot 10^{-1}$
1.25	500	1455	$8.20 \cdot 10^{-3}$	$1.39 \cdot 10^{-3}$	$6.21 \cdot 10^{-5}$	11.4	$8.53 \cdot 10^{-1}$
	1000	1520	$8.25 \cdot 10^{-3}$	$1.49 \cdot 10^{-3}$	$4.74 \cdot 10^{-5}$	11.4	$8.66 \cdot 10^{-1}$
	2000	1465	$8.23 \cdot 10^{-3}$	$1.40 \cdot 10^{-3}$	$3.13 \cdot 10^{-5}$	11.4	$8.62 \cdot 10^{-1}$
0.625	500	2746	$2.03 \cdot 10^{-3}$	$3.00 \cdot 10^{-4}$	$1.34 \cdot 10^{-5}$	11.5	$7.89 \cdot 10^{-1}$
	1000	2813	$2.03 \cdot 10^{-3}$	$3.02 \cdot 10^{-4}$	$9.55 \cdot 10^{-6}$	11.5	$7.94 \cdot 10^{-1}$
	2000	2901	$2.05 \cdot 10^{-3}$	$2.99 \cdot 10^{-4}$	$6.68 \cdot 10^{-6}$	11.5	$8.00 \cdot 10^{-1}$

The above conditions are met when, for all $K \in \mathcal{T}_h$,

$$0.75^2\ TOL^2\ \frac{|\Omega|}{N_K^2} \leq \eta_K^2(u_h(\cdot;\mu);\mu) \leq 1.25^2\ TOL^2\ \frac{|\Omega|}{N_K^2}, \tag{11}$$

where N_K is the number of triangles of $\mathcal{T}_h$. The mesh is adapted according to (11), with the BL2D mesh generator [44]. We use a continuation algorithm for the tolerance, as described in [45]: if the desired tolerance is TOL, we start the algorithm with an initial tolerance larger than TOL, and we decrease it regularly until the targeted tolerance is reached. Fig. 3 visualizes an example of adapted mesh.

To test the accuracy of the finite element method, we randomly draw M parameters $\{\mu_k\}_{k=1}^M$ according to a uniform distribution, as described in Section 2. For each μ_k, we adapt the mesh according to (11) and denote by N_k the number of vertices in the resulting grid. As in Section 4.1, the effectivity index (10) is used to track the sharpness of the estimator. Table 5 shows the average number of vertices

$$\overline{N}_M := \frac{1}{M}\sum_{k=1}^{M} N_k,$$

together with the computed finite element error $|||u-u_h|||^2_M$ and the effectivity index, for various values of TOL and M. As expected, $|||u-u_h|||^2_M$ behaves as $\mathcal{O}(TOL^2)$ and since $d = 2$, we observe that $TOL = \mathcal{O}(h^{-2}) = \mathcal{O}(\overline{N}_M)$. Furthermore, the mean of the effectivity index does not depend on h and M, and its standard deviation is small across the parameters space. Also, the standard deviation $\mathrm{Std}_M(\|u(\cdot;\mu) - u_h(\cdot;\mu)\|^2)$ is small compared to the computed quantity $|||u-u_h|||^2_M$, unlike when using fixed meshes (see Table 1). Finally, note that the expected error of the Monte-Carlo method (sixth column) is negligible compared to the computed quantity $|||u-u_h|||^2_M$.

Table 6 visualizes the time needed to solve the linear system corresponding to the adapted mesh, for different tolerances and different parameters μ. Hereafter, N denotes the number of vertices in the adapted mesh and WT denotes the wall time needed to solve the linear system using one CPU *Intel Core, 3.5 GHz*, averaged over 2000 resolutions of the system. As for the first test case, we observe that $WT = \mathcal{O}(N^{\frac{3}{2}})$. Nevertheless, due to the mesh adaptation algorithm, the number of vertices highly depends on μ, and so does the wall time needed to solve the linear system.

4.2.2. Neural networks

We construct neural networks to approximate the solution of (8), with $softplus$ as activation function. We initialize the weights of the networks using the Glorot Normal initialization [42]. All the networks are trained using the Nadam optimizer [43], with an initial learning rate of 0.001, which is decreased when a plateau is reached, and early stopping. With this adapted grid approach, the number of samples to be reviewed during the training process is equal to

$$\sum_{j=1}^{N_{train}} N_j,$$

Table 6
Wall time to solve the linear system in the last mesh iteration.

μ	TOL	N	WT [s]
(5, 10, 1000, 0.3)	5	2337	$7.75 \cdot 10^{-3}$
	2.5	4510	$2.03 \cdot 10^{-2}$
	1.25	8780	$5.25 \cdot 10^{-2}$
	0.625	17 910	$1.54 \cdot 10^{-1}$
(5, 1, 100, 2)	5	23	$4.2 \cdot 10^{-5}$
	2.5	37	$6.5 \cdot 10^{-5}$
	1.25	70	$8 \cdot 10^{-5}$
	0.625	118	$1.2 \cdot 10^{-4}$
(5, 5, 500, 1)	5	358	$5.6 \cdot 10^{-4}$
	2.5	658	$1.24 \cdot 10^{-3}$
	1.25	1259	$3.09 \cdot 10^{-3}$
	0.625	2389	$7.65 \cdot 10^{-3}$

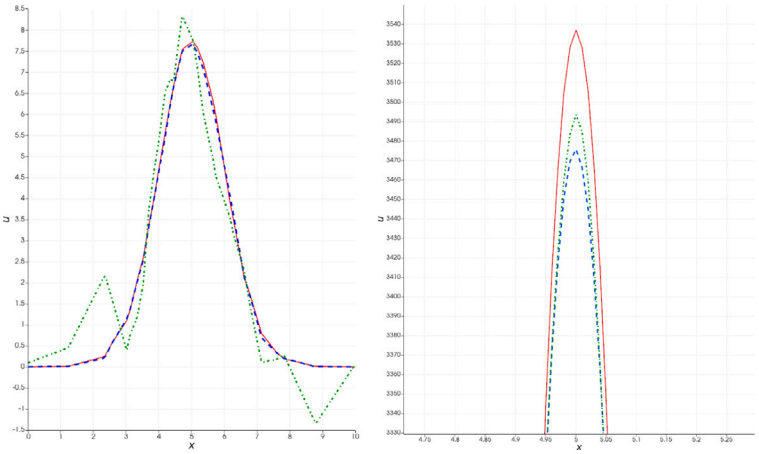


Fig. 4. Comparison along the axis $y = 1$ of u_h (red, plain line) and $u_{\mathcal{N}}$, for a neural network $\mathcal{N} \in Y^{100,4}(\sigma; 6, 1)$ trained with the $\mathcal{L}^{\ell^2}$ objective function (green, dash-dotted line) and the $\mathcal{L}^{L^2}$ objective function (blue, dashed line) ($TOL = 1.25$, $N_{train} = 4000$). Left: $\mu = (5, 1, 100, 2)$. Right: $\mu = (5, 10, 1000, 0.3)$.

as opposed to N_{train} for the fixed grid approach; we thus use larger batches and set $bs = 1024$. Before training, we normalize the set $\{(x_j^i; \tilde{\mu}_j)_{i=1}^{N_j}\}_{j=1}^{N_{train}}$ such that all the components have zero mean and unit standard deviation. To test the accuracy of the network, i.e. to compute $|||u_h - u_{\mathcal{N}}|||_M$, we take the same test set $\{\mu_k\}_{k=1}^{M}$ as the one used to compute $|||u - u_h|||_M$ in the previous paragraph, with $M = 2000$.

Since the meshes considered here are no longer uniform, the use of $\mathcal{L}^{L^2}$ or $\mathcal{L}^{\ell^2}$ as an objective function leads to different results. Since we are interested in minimizing the $L^2(\Omega \times \mathcal{P})$ error between $u_{\mathcal{N}}$ and u_h, we choose $\mathcal{L}^{L^2}$ to train the networks. This choice implies that more weight is given in the objective function to the vertices having shape functions with a large support. The use of $\mathcal{L}^{L^2}$ thus allows to avoid the oscillations observed for small μ_3 and large μ_4 when $\mathcal{L}^{\ell^2}$ is used, see Fig. 4 (left). On the other hand, it tends to decrease slightly the accuracy of the network for parameters with large μ_3 and small μ_4, see Fig. 4 (right).

In what follows, we investigate the impact that the choices of L, W, N_{train} and TOL have on the accuracy of the networks. As for the fixed grid approach, training several times a neural network $\mathcal{N} \in Y^{W,L}(\sigma; 6, 1)$ results in a variability of $|||u_{\mathcal{N}} - u_h|||_M$ that is typically smaller than 10% of the value of the error itself. All the results in the sequel are thus the result of a single training phase for each neural network.

We start by setting $TOL = 1.25$ and we let both N_{train} and the architecture of the networks, L and W, vary. Numerical results are reported in Fig. 5. We first remark that neural networks with depth $L = 2$ show less accuracy than networks with $L = 4, 8$. So we restrict ourselves to deeper networks in what follows. Next, for this tolerance, the error of networks with width $L = 4, 8$ start to plateau between $N_{train} = 2000$ and $N_{train} = 4000$.

Based on these results, we set $N_{train} = 4000$ in the sequel to test further the effect of the tolerance and of the architecture of the networks on the error. This choice will be further discussed with the adaptive algorithm presented in Section 5. Table 7 reports the error of neural networks with various widths and depths, and various values of TOL.

Several comments are in order concerning these results and their comparison with the ones obtained with the fixed grid approach (see Table 4). First, note that – contrary to the fixed grid approach – dividing TOL by two now lets the architecture of the networks unchanged, but multiplies the average number of vertices, and thus the number of training data in Ω, by four. The error of the networks $|||u_h - u_{\mathcal{N}}|||_M$ is thus expected to decrease as TOL decreases, which is confirmed by Fig. 6. Second, for the values of L, W and TOL considered here, increasing the size of the networks does not increase their accuracy. This suggests that the error comes

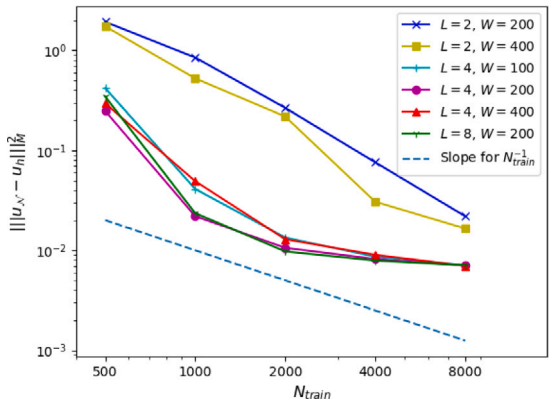


Fig. 5. Error of various networks with different widths and depths, as a function of N_{train} ($TOL = 1.25$, $M = 2000$).

Table 7
Computed error of neural networks for various values of L, W and TOL ($N_{train} = 4000$, $M = 2000$).

L	W	N_θ	TOL	$\|\|u_{\mathcal{N}} - u_h\|\|_M^2$	$\mathrm{Std}_M(\|u_{\mathcal{N}} - u_h\|^2)$	$\frac{\mathrm{Std}_M(\|u_{\mathcal{N}} - u_h\|^2)}{\sqrt{M}}$
4	100	31 101	5	$1.06 \cdot 10^{-1}$	$1.64 \cdot 10^{-1}$	$3.66 \cdot 10^{-3}$
			2.5	$3.72 \cdot 10^{-2}$	$3.90 \cdot 10^{-1}$	$8.73 \cdot 10^{-3}$
			1.25	$8.64 \cdot 10^{-3}$	$1.96 \cdot 10^{-2}$	$4.38 \cdot 10^{-4}$
			0.625	$3.10 \cdot 10^{-3}$	$2.81 \cdot 10^{-3}$	$6.26 \cdot 10^{-5}$
4	200	122 201	5	$9.97 \cdot 10^{-2}$	$1.80 \cdot 10^{-2}$	$4.03 \cdot 10^{-4}$
			2.5	$2.75 \cdot 10^{-2}$	$2.22 \cdot 10^{-2}$	$4.97 \cdot 10^{-4}$
			1.25	$8.18 \cdot 10^{-3}$	$1.30 \cdot 10^{-2}$	$2.91 \cdot 10^{-4}$
			0.625	$2.84 \cdot 10^{-3}$	$2.36 \cdot 10^{-3}$	$5.28 \cdot 10^{-5}$
4	400	484 401	5	$9.93 \cdot 10^{-2}$	$1.98 \cdot 10^{-2}$	$4.44 \cdot 10^{-4}$
			2.5	$3.08 \cdot 10^{-2}$	$7.83 \cdot 10^{-2}$	$1.75 \cdot 10^{-3}$
			1.25	$9.01 \cdot 10^{-3}$	$4.70 \cdot 10^{-2}$	$1.05 \cdot 10^{-3}$
			0.625	$2.78 \cdot 10^{-3}$	$2.16 \cdot 10^{-3}$	$4.83 \cdot 10^{-5}$
8	100	71 501	5	$9.90 \cdot 10^{-2}$	$1.66 \cdot 10^{-2}$	$3.72 \cdot 10^{-4}$
			2.5	$2.70 \cdot 10^{-2}$	$4.24 \cdot 10^{-2}$	$9.48 \cdot 10^{-4}$
			1.25	$7.71 \cdot 10^{-3}$	$2.25 \cdot 10^{-2}$	$5.03 \cdot 10^{-4}$
			0.625	$2.55 \cdot 10^{-3}$	$1.40 \cdot 10^{-3}$	$3.13 \cdot 10^{-5}$
8	200	283 001	5	$9.70 \cdot 10^{-2}$	$1.39 \cdot 10^{-2}$	$3.11 \cdot 10^{-4}$
			2.5	$2.76 \cdot 10^{-2}$	$4.78 \cdot 10^{-2}$	$1.07 \cdot 10^{-3}$
			1.25	$7.68 \cdot 10^{-3}$	$9.70 \cdot 10^{-3}$	$2.17 \cdot 10^{-4}$
			0.625	$2.56 \cdot 10^{-3}$	$1.29 \cdot 10^{-3}$	$2.89 \cdot 10^{-5}$
8	400	1 126 001	5	$1.06 \cdot 10^{-1}$	$2.43 \cdot 10^{-2}$	$5.48 \cdot 10^{-4}$
			2.5	$2.71 \cdot 10^{-2}$	$4.70 \cdot 10^{-2}$	$1.05 \cdot 10^{-3}$
			1.25	$8.08 \cdot 10^{-3}$	$1.01 \cdot 10^{-2}$	$2.26 \cdot 10^{-4}$
			0.625	$2.52 \cdot 10^{-3}$	$2.23 \cdot 10^{-3}$	$4.98 \cdot 10^{-5}$

mainly from the training procedure, and not from the structure of the networks itself. Third, we emphasize that – as illustrated in Fig. 6 – the two errors $|||u - u_h|||_M^2$ and $|||u_h - u_{\mathcal{N}}|||_M^2$ are comparable for large enough neural networks (for all tolerances tested). Nevertheless, the standard deviation of the neural networks error is larger than the standard deviation of the finite element error. Finally, the expected error of the Monte-Carlo method (last column) is one order of magnitude smaller than the computed value of $|||u_h - u_{\mathcal{N}}|||_M^2$, for $TOL = 1.25$, 0.625 and for all the neural networks considered.

We compare now the time needed to evaluate one neural network with the time needed to solve the linear system (see Table 6). We consider the networks $\mathcal{N} \in \Upsilon^{400,4}(\sigma; 6, 1)$ obtained for the different tolerances TOL, and evaluate them by batches of size $bs = 1024$. Table 8 visualizes the results, with a wall time being averaged over 2000 evaluations and N denoting the number of vertices in the finite element mesh.

We note that evaluating the network in all the grid points takes more time than solving the linear system for this test case. Nevertheless, this evaluation time does not depend on the underlying PDE, and, when considering more complex PDEs, we expect that evaluating the neural network would be faster than to solve the (possibly nonlinear) system.

Since the analytical solution u is known for this test case, $|||u - u_{\mathcal{N}}|||_M$, $|||u - u_h|||_M$ and $|||u_h - u_{\mathcal{N}}|||_M$ can also be directly compared. Numerical results for networks with different architectures and $TOL = 1.25, 0.625$ are reported in Table 9.

Table 8
Time to evaluate $\mathcal{N} \in Y^{400,4}(\sigma; 6, 1)$ on the finite element mesh.

μ	TOL	N	WT [s]
(5, 10, 1000, 0.3)	5	2337	$4.72 \cdot 10^{-2}$
	2.5	4510	$5.25 \cdot 10^{-2}$
	1.25	8780	$6.2 \cdot 10^{-2}$
	0.625	17 910	$8.85 \cdot 10^{-2}$
(5, 1, 100, 2)	5	23	$4 \cdot 10^{-2}$
	2.5	37	$4 \cdot 10^{-2}$
	1.25	70	$4 \cdot 10^{-2}$
	0.625	118	$4 \cdot 10^{-2}$
(5, 5, 500, 1)	5	358	$4 \cdot 10^{-2}$
	2.5	658	$4.04 \cdot 10^{-2}$
	1.25	1259	$4.47 \cdot 10^{-2}$
	0.625	2389	$4.79 \cdot 10^{-2}$

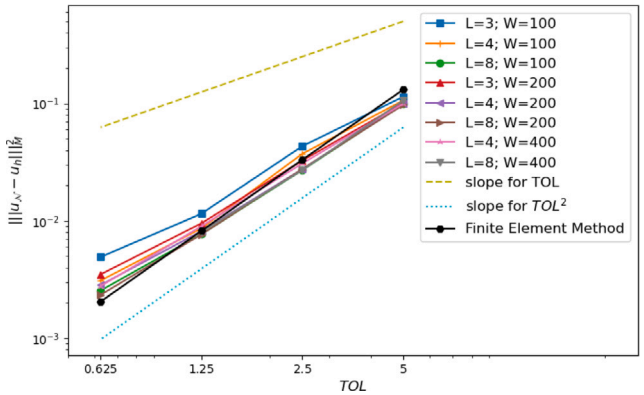


Fig. 6. Error of neural networks with different widths and depths as a function of TOL, for $N_{train} = 4000$ and $M = 2000$.

Table 9
Comparison of $\|u - u_h\|_M$, $\|u_h - u_{\mathcal{N}}\|_M$, $\|u - u_{\mathcal{N}}\|_M$ for various architectures of networks and different tolerances TOL ($M = 2000$).

TOL	$\|u - u_h\|_M$	L	W	$\|u_h - u_{\mathcal{N}}\|_M$	$\|u - u_{\mathcal{N}}\|_M$
1.25	0.0907	4	100	0.0929	0.0649
		4	200	0.0904	0.0606
		4	400	0.0948	0.0680
		8	100	0.0878	0.0594
		8	200	0.0876	0.0514
		8	400	0.0899	0.0562
0.625	0.0453	4	100	0.0557	0.0440
		4	200	0.0533	0.0406
		4	400	0.0527	0.0400
		8	100	0.0505	0.0368
		8	200	0.0506	0.0373
		8	400	0.0502	0.0370

First, we note that $|||u - u_{\mathcal{N}}|||_M$ decreases as the tolerance decreases, which is a consequence of having both more precise and more numerous training data. Next, we have in any cases $|||u - u_{\mathcal{N}}|||_M \leq |||u - u_h|||_M + |||u_h - u_{\mathcal{N}}|||_M$, as expected. Finally, we remark that $u_{\mathcal{N}}$ seems to act as a post-processing of u_h, yielding on average a slightly better approximation of u than u_h itself.

Numerical results in Table 9 give insights only on the average accuracy of $u_{\mathcal{N}}$ across the parameters space. In Fig. 7, we thus compare $u(\cdot; \mu)$, $u_h(\cdot; \mu)$ and $u_{\mathcal{N}}(\cdot; \mu)$ for given parameters μ, in particular for extreme parameters that lie on the boundary of $\mathcal{P}$. We note that, in all cases, $u_{\mathcal{N}}(\cdot; \mu)$ gives an accurate approximation of the exact solution $u(\cdot; \mu)$.

5. An adaptive algorithm for the number of training samples

The previous numerical experiments estimate *a posteriori* the error of a neural network trained with a given number of samples. In what follows, we discuss an algorithm that, given a (small) initial training set, incrementally increases the number of training samples in order to ensure that the overall error $|||u - u_{\mathcal{N}}|||^2$ is smaller than some preset tolerance ϵ^2. Using $|||u - u_{\mathcal{N}}|||^2 \leq 2(|||u - u_h|||^2 + |||u_h - u_{\mathcal{N}}|||^2)$, the idea is, on the one hand, to choose the tolerance TOL of the mesh adaptation algorithm in such a way that $|||u - u_h|||^2 \leq \epsilon^2/4$

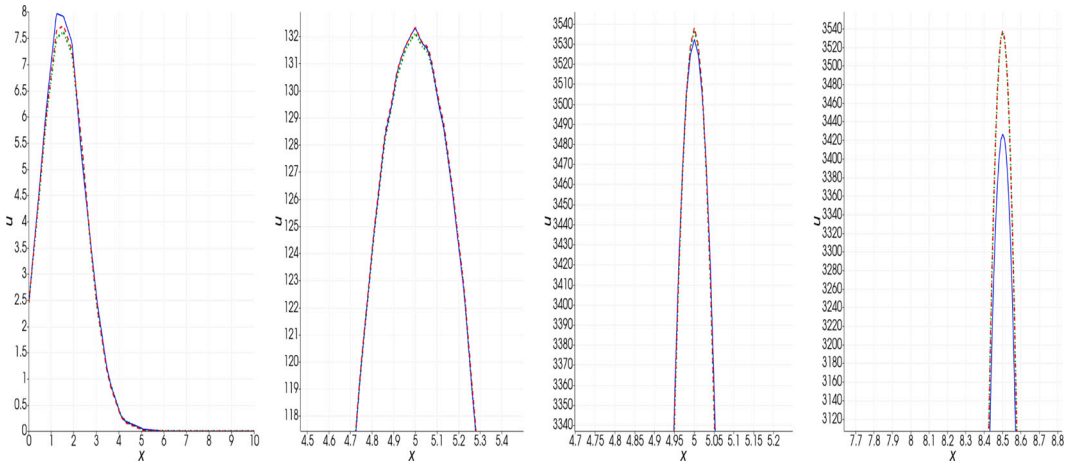


Fig. 7. Comparison of u (red, dashed line), u_h (green, dash-dotted line) and $u_{\mathcal{N}}$ (blue, plain line) along the axis $y = 1$ for a neural network $\mathcal{N} \in Y^{400,4}(\sigma; 6, 1)$ trained with $N_{train} = 4000$ examples and $TOL = 1.25$. From left to right: $\mu = (1.5, 1, 100, 2)$, $(5, 5.5, 550, 1.15)$, $(5, 10, 1000, 0.3)$, $(8.5, 1, 1000, 0.3)$.

and, on the other hand, to increase the size of the training set until $|||u_h - u_{\mathcal{N}}|||^2 \leq \epsilon^2/4$. Note that, in order to ensure $|||u - u_h|||^2 \leq \epsilon^2/4$ using the adaptive algorithm given in Section 4.2.1, one needs to choose $TOL^2 = \epsilon^2 ei^2/4$, where ei is the effectivity index defined in (10). According to Table 5, we take $ei = 11.4$ in what follows. Given ϵ, an initial number of samples N^1_{train}, a maximum number of iterations k_{max}, and the number of samples to be added at each iterations N_{add}, the algorithm is fully described in Algorithm 1.

Algorithm 1 An adaptive algorithm to control N_{train}

Require: ϵ, N^1_{train}, k_{max}, N_{add}, a test set $\{\mu_k\}_{k=1}^M$, and a given architecture of the neural network.

Set $k = 1$, err $= \epsilon^2$, $N^0_{train} = 0$
while err $> \dfrac{\epsilon^2}{4}$ and $k \leq k_{max}$ **do**
  Select randomly the parameters $\tilde{\mu}_j$, $j = N^{k-1}_{train}+1, \ldots, N^k_{train}$
  Compute the FE solutions $u_h(\cdot; \tilde{\mu}_j)$, $j = N^{k-1}_{train}+1, \ldots, N^k_{train}$ using (11) with $TOL^2 = \dfrac{\epsilon^2 ei^2}{4}$
  Train the neural network in order to obtain $u^k_{\mathcal{N}}$
  Compute err $= |||u^k_{\mathcal{N}} - u_h|||^2_M$
  Set $N^{k+1}_{train} = N^k_{train} + N_{add}$
  Set $k = k + 1$
end while

Note that the neural network is re-trained at each iteration of the algorithm (for a network $\mathcal{N} \in Y^{200,4}(\sigma; 6, 1)$ and $TOL = 1.25$, the training typically takes between around 30 minutes ($N_{train} = 500$) and 2.5 h ($N_{train} = 4000$)). This algorithm performs thus well when the training times are negligible compared to the time needed to perform new numerical simulations, which is for example the case in the context of laser melting [27]. In this situation, an additive increase of the training parameters is beneficial, in order to perform as few numerical simulations as possible. We test here the algorithm for the Poisson problem as a proof of concept.

Table 10 visualizes the evolution of $|||u_h - u^k_{\mathcal{N}}|||^2_M$ during the iterations of Algorithm 1 for $\epsilon = 0.2$, when different values of N_{add} and N^1_{train} are chosen ($k_{max} = 15$). Note that, for the sake of consistency, we have computed $|||u^k_{\mathcal{N}} - u_h|||^2_M$ using the same test set of size $M = 2000$ as in Section 4.2. We first remark that the algorithm is able to ensure the given accuracy before reaching the maximum number of iterations, and this for all starting points N^1_{train} and increments N_{add}. Furthermore, we note that the final number of training samples is between 2000 and 4000 in all cases, which is coherent with the numerical results obtained in Section 4.2 (see Table 7 and Fig. 5).

We next set $N_{add} = N^1_{train} = 500$ and test the algorithm for different values of ϵ. Table 11 visualizes the numerical results. For validation purposes, we use here the same test set of size $M = 2000$ to check the stopping criterion. Note however that we obtain similar results and behavior when using a smaller, cheaper, test set with $M = 500$ for instance.

For the three values of ϵ tested here, the adaptive algorithm stops before the maximum number of iterations and is able to ensure that $|||u_h - u_{\mathcal{N}}|||^2_M < \epsilon^2/4$, as desired.

Table 10
Error of a neural network $\mathcal{N} \in Y^{200,4}(\sigma; 6, 1)$ at each iteration of the continuation algorithm, for different values of N_{add}, N^1_{train} and $\epsilon = 0.2$. $M = 2000$.

$N^1_{train} = 1000,\ N_{add} = 1000$						
	$k = 1$		$k = 2$			$k = 3$
N^k_{train}	1000		2000			3000
$\|u^k_{\mathcal{N}} - u_h\|^2_M$	$2.64 \cdot 10^{-2}$		$1.11 \cdot 10^{-2}$			$9.76 \cdot 10^{-3}$
$\epsilon^2/4$			$1 \cdot 10^{-2}$			
TOL			1.25			
$\|u - u_h\|^2_M$			$8.23 \cdot 10^{-3}$			
$N^1_{train} = 500,\ N_{add} = 500$						
	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
N^k_{train}	500	1000	1500	2000	2500	3000
$\|u^k_{\mathcal{N}} - u_h\|^2_M$	$2.05 \cdot 10^{-1}$	$1.93 \cdot 10^{-2}$	$1.43 \cdot 10^{-2}$	$1.21 \cdot 10^{-2}$	$1.03 \cdot 10^{-2}$	$9.25 \cdot 10^{-3}$
$\epsilon^2/4$				$1 \cdot 10^{-2}$		
TOL				1.25		
$\|u - u_h\|^2_M$				$8.23 \cdot 10^{-3}$		
$N^1_{train} = 500,\ N_{add} = 250$						
	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	
N^k_{train}	500	750	1000	1250	1500	
$\|u^k_{\mathcal{N}} - u_h\|^2_M$	$3.48 \cdot 10^{-1}$	$6.05 \cdot 10^{-2}$	$2.41 \cdot 10^{-2}$	$1.72 \cdot 10^{-2}$	$1.45 \cdot 10^{-2}$	
	$k = 6$	$k = 7$	$k = 8$	$k = 9$	$k = 10$	
N^k_{train}	1750	2000	2250	2500	2750	
$\|u^k_{\mathcal{N}} - u_h\|^2_M$	$1.24 \cdot 10^{-2}$	$1.16 \cdot 10^{-2}$	$1.05 \cdot 10^{-2}$	$1.02 \cdot 10^{-2}$	$9.80 \cdot 10^{-3}$	
$\epsilon^2/4$			$1 \cdot 10^{-2}$			
TOL			1.25			
$\|u - u_h\|^2_M$			$8.23 \cdot 10^{-3}$			

Table 11
Error of a neural network $\mathcal{N} \in Y^{200,4}(\sigma; 6, 1)$ at each iteration of the continuation algorithm, for different values of ϵ. $N^1_{train} = N_{add} = 500$.

$\epsilon = 0.4$						
	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
N^k_{train}	500	1000	1500	2000	2500	3000
$\|u^k_{\mathcal{N}} - u_h\|^2_M$	$3.64 \cdot 10^{-1}$	$6.71 \cdot 10^{-2}$	$5.51 \cdot 10^{-2}$	$5.11 \cdot 10^{-2}$	$4.70 \cdot 10^{-2}$	$3.65 \cdot 10^{-2}$
$\epsilon^2/4$				$4 \cdot 10^{-2}$		
TOL				2.5		
$\|u - u_h\|^2_M$				$3.31 \cdot 10^{-2}$		
$\epsilon = 0.2$						
	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	
N^k_{train}	500	1000	1500	2000	2500	
$\|u^k_{\mathcal{N}} - u_h\|^2_M$	$2.32 \cdot 10^{-1}$	$1.80 \cdot 10^{-2}$	$1.29 \cdot 10^{-2}$	$1.01 \cdot 10^{-2}$	$9.45 \cdot 10^{-3}$	
$\epsilon^2/4$			$1 \cdot 10^{-2}$			
TOL			1.25			
$\|u - u_h\|^2_M$			$8.23 \cdot 10^{-3}$			
$\epsilon = 0.1$						
	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$
N^k_{train}	500	1000	1500	2000	2500	3000	3500
$\|u^k_{\mathcal{N}} - u_h\|^2_M$	$9.47 \cdot 10^{-2}$	$1.02 \cdot 10^{-2}$	$7.53 \cdot 10^{-3}$	$5.30 \cdot 10^{-3}$	$4.46 \cdot 10^{-3}$	$4.09 \cdot 10^{-3}$	$3.17 \cdot 10^{-3}$
	$k = 8$	$k = 9$	$k = 10$	$k = 11$	$k = 12$	$k = 13$	
N^k_{train}	4000	4500	5000	5500	6000	6500	
$\|u^k_{\mathcal{N}} - u_h\|^2_M$	$2.90 \cdot 10^{-3}$	$2.76 \cdot 10^{-3}$	$2.65 \cdot 10^{-3}$	$2.61 \cdot 10^{-3}$	$2.51 \cdot 10^{-3}$	$2.43 \cdot 10^{-3}$	
$\epsilon^2/4$				$2.5 \cdot 10^{-3}$			
TOL				0.625			
$\|u - u_h\|^2_M$				$2.05 \cdot 10^{-3}$			

6. Construction of a parameter-dependent mesh

Finally, in order to complete the implementation of the proposed algorithm, one has to evaluate the mapping $(x, \mu) \rightarrow u_{\mathcal{N}}(x; \mu)$ for several values of x in order to compute the solution $u_{\mathcal{N}}(\cdot; \mu)$ in the whole domain Ω. For a given value of the parameter μ, we thus need to construct a discretization on which to evaluate the neural network, and this without computing the corresponding finite element solution $u_h(\cdot; \mu)$. For this purpose, we advocate hereafter an algorithm based on $L^2(\Omega)$-projections.

Consider a discretization $\mathcal{T}_h$ of Ω and let V_h be the usual finite element space of continuous, piecewise linear functions on $\mathcal{T}_h$. The $L^2(\Omega)$-projection onto V_h of a function $g \in L^2(\Omega)$ is denoted by $\Pi_h g \in V_h$, and defined by

$$\int_\Omega (\Pi_h g) v_h = \int_\Omega g v_h \quad \forall v_h \in V_h.$$

For every $w_h \in V_h$, we have

$$\|g - \Pi_h g\|^2_{L^2(\Omega)} = \int_\Omega (g - \Pi_h g)(g - w_h) \leq \|g - \Pi_h g\|_{L^2(\Omega)} \|g - w_h\|_{L^2(\Omega)},$$

so that

$$\|g - \Pi_h g\|_{L^2(\Omega)} \leq \|g - w_h\|_{L^2(\Omega)} \quad \forall w_h \in V_h.$$

In particular, taking $w_h = \Pi_h g + R_h(g - \Pi_h g)$, where R_h denotes the Clément interpolant, we obtain [46]

$$\begin{aligned} \|g - \Pi_h g\|^2_{L^2(\Omega)} &\leq \|g - \Pi_h g - R_h(g - \Pi_h g)\|^2_{L^2(\Omega)} \\ &\leq \sum_{K \in \mathcal{T}} \|g - \Pi_h g - R_h(g - \Pi_h g)\|^2_{L^2(K)} \\ &\leq C \sum_{K \in \mathcal{T}} h_K^2 \|\nabla(g - \Pi_h g)\|^2_{L^2(\Delta K)}, \end{aligned}$$

where ΔK is the set of triangles sharing a vertex with K and C is a constant independent of h and g (but depending on the mesh aspect ratio). We apply a Zienkiewicz–Zhu post-processing [47–49] to approximate ∇g by $G^{ZZ} g \in V_h$, which is defined, for any vertex $x \in \mathcal{T}_h$ by

$$G^{ZZ} g(x) := \frac{\sum_{K \in \mathcal{T}_h, x \in K} |K| \left(\nabla \Pi_h g\right)_{|K}}{\sum_{K \in \mathcal{T}_h, x \in K} |K|}.$$

For a given tolerance tol, we then want to build a mesh $\mathcal{T}_h$ such that

$$0.75\ tol \leq \left(\frac{\sum_{K \in \mathcal{T}_h} h_K^2 \|G^{ZZ} g - \nabla \Pi_h g\|_{L^2(K)}}{|\Omega|} \right)^{\frac{1}{2}} \leq 1.25\ tol.$$

The above conditions are met when, for all $K \in \mathcal{T}_h$,

$$0.75^2\ tol^2 \frac{|\Omega|}{N_K^2} \leq h_K^2 \|G^{ZZ} g - \nabla \Pi_h g\|_{L^2(K)} \leq 1.25^2\ tol^2 \frac{|\Omega|}{N_K^2}, \tag{12}$$

where N_K is the number of triangles of $\mathcal{T}_h$. For a given μ, the condition (12) is used to build a discretization adapted to $g = u_{\mathcal{N}}(\cdot; \mu)$. This discretization depends only on the solution g to display, but not on the underlying PDE anymore.

Let us consider the neural network $\mathcal{N} \in Y^{400,4}(\sigma; 6, 1)$ trained with finite element simulations performed with $TOL = 1.25$, and let us use the above algorithm with $g = u_{\mathcal{N}}(\cdot; \mu)$, for various values of μ. Table 12 visualizes results obtained with $tol = 0.3125$, and compare them to the ones obtained with the finite element method. Note that we take two different tolerances: TOL for computing u_h satisfying (11), and tol to build a mesh satisfying (12) (with $g = u_{\mathcal{N}}$), since the effectivity index of the two error indicators are different. The times reported correspond to wall times obtained to perform the whole adaptation algorithms (as opposed to Table 6, where only the last iteration is considered) when solving the linear systems using a CPU *Intel Core, 3.5 GHz* and evaluating the networks on a graphical card *Gigabyte GeForce RTX 3080*. Fig. 8 illustrates the comparison of meshes obtained with the finite element and the $L^2(\Omega)$– projection algorithms.

In this case, solving the underlying PDE is relatively simple and fast. Thus, generating the mesh using the $L^2(\Omega)$-projection takes more time than using the finite element method. However, the $L^2(\Omega)$-projection presented here can be used indifferently for any differential equation, once a network has been trained. We thus expect it to perform well when a more involved and possibly non-linear PDE is considered.

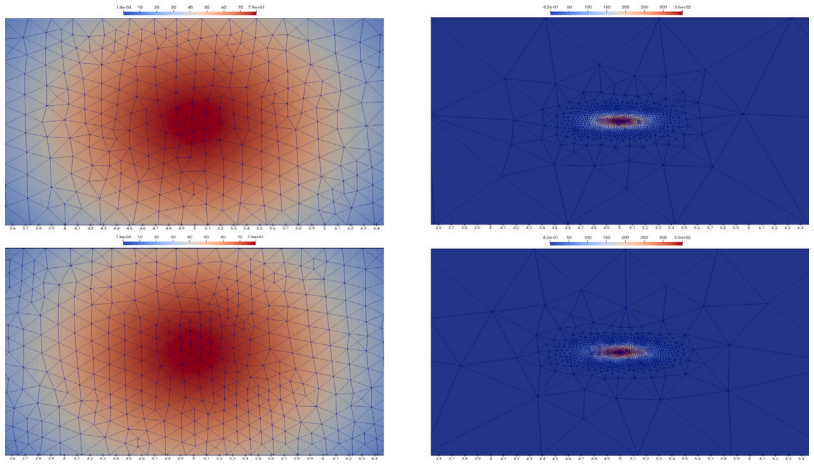
7. Conclusion

We have proposed a data-driven, adaptive finite element—neural network method to approximate the solution of parametric PDEs. The method uses data from finite element simulations to train a deep neural network, that can then be used to approximate the solution of the PDE.

By considering first a fixed finite-element grid approach combined with a neural network used to approximate the discretized parameter-to-solution map, we have observed that this method shows limitation in term of error control. By considering an adaptive

Table 12
Comparison of meshes adapted according to (11) and (12) for various values of μ. The times reported correspond to the wall-times needed to perform the whole adaptation algorithms.

μ	Finite elements			$L^2(\Omega)$ projection with $u_{\mathcal{N}}$		
	N_h	$\|u-u_h\|^2$	WT [s]	N_h	$\|u-u_{\mathcal{N}}\|^2$	WT [s]
(5, 10, 1000, 0.3)	8780	$1.90 \cdot 10^{-2}$	10.3	10 654	$8.92 \cdot 10^{-2}$	18.6
(5, 1, 100, 2)	70	$8.73 \cdot 10^{-3}$	2.16	70	$9.41 \cdot 10^{-3}$	5.62
(5, 1, 1000, 2)	485	$1.09 \cdot 10^{-2}$	2.71	625	$1.92 \cdot 10^{-2}$	6.21
(5, 10, 100, 0.3)	942	$2.77 \cdot 10^{-2}$	3.52	1179	$4.90 \cdot 10^{-2}$	7.19
(5, 5, 500, 1)	1259	$7.11 \cdot 10^{-3}$	3.68	1473	$5.80 \cdot 10^{-3}$	7.54
(5, 5, 1000, 1)	2356	$7.64 \cdot 10^{-3}$	4.61	2852	$9.13 \cdot 10^{-3}$	9.48
(5, 5, 500, 0.3)	4074	$1.13 \cdot 10^{-2}$	6.18	5047	$1.91 \cdot 10^{-2}$	11.7



(a) $\mu = (5, 1, 1000, 2)$ (b) $\mu = (5, 10, 100, 0.3)$

Fig. 8. Comparison of meshes obtained with the finite element method and the $L^2(\Omega)$-projection algorithm. Top: meshes adapted according to (11). Bottom: meshes adapted according to (12).

finite-element method combined with a neural network used to approximate the function $(x; \mu) \mapsto u(x; \mu)$, we have concluded that this new approach allows to balance the neural network error and the error of the finite element method. Finally, an adaptive algorithm to control the size of the training set has been proposed, in order to ensure that the overall error is below a given tolerance ϵ. A parameter-dependent grid reconstruction allowing the reconstruction of the final neural network solution independently of the underlying PDE has also been proposed.

Perspectives for future work include the application of the present method to more complex PDEs, with possibly non smooth solutions, as well as the study of a criterion to better choose the parameters in the training set.

CRediT authorship contribution statement

Alexandre Caboussat: Conceptualization, Funding acquisition, Methodology, Supervision, Writing – original draft, Writing – review & editing. **Maude Girardin:** Conceptualization, Methodology, Software, Writing – original draft, Writing – review & editing, Visualization. **Marco Picasso:** Conceptualization, Funding acquisition, Methodology, Supervision, Writing – original draft, Writing – review & editing.

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.

Data availability

Data will be made available on request.

Acknowledgments

The authors thank Paride Passelli (EPFL) for fruitful discussions.

References

[1] A.R. Barron, Approximation and estimation bounds for artificial neural networks, Mach. Learn. 14 (1994) 115–133.
[2] R. DeVore, B. Hanin, G. Petrova, Neural network approximation, Acta Numer. 30 (2021) 327–444.
[3] W. E, C. Ma, L. Wu, Barron spaces and the compositional function spaces for neural network models, 2019, arXiv preprint arXiv:1906.08039.
[4] P. Petersen, F. Voigtlaender, Optimal approximation of piecewise smooth functions using deep relu neural networks, Neural Netw. 108 (2018) 296–330.
[5] U. Shaham, A. Cloninger, R. Coifman, Provable approximation properties for deep neural networks, Appl. Comput. Harmon. Anal. 44 (2018) 537–557.
[6] D. Yarotsky, Optimal approximation of continuous functions by very deep ReLU networks, in: Conference on Learning Theory, PMLR, 2018, pp. 639–649.
[7] F. Bach, L. Chizat, Gradient descent on infinitely wide neural networks: Global convergence and generalization, 2021, arXiv preprint arXiv:2110.08084.
[8] L. Bottou, F. Curtis, J. Nocedal, Optimization methods for large-scale machine learning, SIAM Rev. 60 (2018) 223–311.
[9] G. Kutyniok, P. Petersen, M. Raslan, R. Schneider, A theoretical analysis of deep neural networks and parametric pdes, Constr. Approx. 55 (2022) 73–125.
[10] C. Schwab, J. Zech, Deep learning in high dimension: Neural network expression rates for generalized polynomial chaos expansions in uq, Anal. Appl. 17 (2019) 19–55.
[11] C. Ma, S. Wojtowytsch, L. Wu, et al., Towards a mathematical understanding of neural network-based machine learning: what we know and what we don't, 2020, arXiv preprint arXiv:2009.10713.
[12] J. Schmidt-Hieber, Nonparametric regression using deep neural networks with relu activation function, Ann. Statist. 48 (2020) http://dx.doi.org/10.1214/19-aos1875, URL: http://dx.doi.org/10.1214/19-AOS1875.
[13] G. Bai, U. Koley, S. Mishra, R. Molinaro, Physics informed neural networks (pinns) for approximating nonlinear dispersive pdes, J. Comput. Math. 39 (2021) 816.
[14] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (2021) 422–440.
[15] M. Raissi, P. Perdikaris, G. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, 2017, arXiv preprint arXiv:1711.10561.
[16] M. Raissi, P. Perdikaris, G.E. 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.
[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.
[18] E. Samaniego, C. Anitescu, S. Goswami, V. Nguyen-Thanh, H. Guo, K. Hamdia, X. Zhuang, T. Rabczuk, An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications, Comput. Methods Appl. Mech. Engrg. 362 (2020) 112790.
[19] B. Yu, et al., The deep ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (2018) 1–12.
[20] S. Lanthaler, S. Mishra, G.E. Karniadakis, Error estimates for deeponets: A deep learning framework in infinite dimensions, Trans. Math. Appl. 6 (2022) tnac001.
[21] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (2021) 218–229.
[22] S. Wang, H. Wang, P. Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed deeponets, Sci. Adv. 7 (2021) eabi8605.
[23] N. Dal Santo, S. Deparis, L. Pegolotti, Data driven approximation of parametrized pdes by reduced basis and neural networks, J. Comput. Phys. 416 (2020) 109550.
[24] M. Geist, P. Petersen, M. Raslan, R. Schneider, G. Kutyniok, Numerical solution of the parametric diffusion equation by deep neural networks, J. Sci. Comput. 88 (2021) 1–37.
[25] B. Haasdonk, H. Kleikamp, M. Ohlberger, F. Schindler, T. Wenzel, A new certified hierarchical and adaptive rb-ml-rom surrogate model for parametrized pdes, SIAM J. Sci. Comput. 45 (2023) A1039–A1065.
[26] J.S. Hesthaven, S. Ubbiali, Non-intrusive reduced order modeling of nonlinear problems using neural networks, J. Comput. Phys. 363 (2018) 55–78.
[27] A. Caboussat, J. Hess, A. Masserey, M. Picasso, Numerical simulation of temperature-driven free surface flows, with application to laser melting and polishing, J. Comput. Phys.: X 17 (2023) 100127.
[28] J.S. Hesthaven, G. Rozza, B. Stamm, et al., Certified Reduced Basis Methods for Parametrized Partial Differential Equations, vol. 590, Springer, 2016.
[29] A. Quarteroni, A. Manzoni, F. Negri, Reduced Basis Methods for Partial Differential Equations: An Introduction, vol. 92, Springer, 2015.
[30] A. Chatterjee, An introduction to the proper orthogonal decomposition, Curr. Sci. (2000) 808–817.
[31] Y. Liang, H. Lee, S. Lim, W. Lin, K. Lee, C. Wu, Proper orthogonal decomposition and its applications—part i: Theory, J. Sound Vib. 252 (2002) 527–544.
[32] S. Volkwein, Model Reduction using Proper Orthogonal Decomposition, in: Lecture Notes, vol. 1025, Institute of Mathematics and Scientific Computing, University of Graz, 2011, see http://www.uni-graz.at/imawww/volkwein/POD.pdf.
[33] K. Lee, K.T. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, J. Comput. Phys. 404 (2020) 108973.
[34] M. Ohlberger, S. Rave, Reduced basis methods: Success, limitations and future challenges, 2015, arXiv preprint arXiv:1511.02021.
[35] R. Verfürth, A Posteriori Error Estimation Techniques for Finite Element Methods, OUP Oxford, 2013.
[36] C.J. Arthurs, A.P. King, Active training of physics-informed neural networks to aggregate and interpolate parametric solutions to the navier-stokes equations, J. Comput. Phys. 438 (2021) 110364.
[37] W. E, T. Li, E. Vanden-Eijnden, Applied Stochastic Analysis, vol. 199, American Mathematical Soc, 2021.
[38] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–444.
[39] Y. Bengio, I. Goodfellow, A. Courville, Deep Learning, vol. 1, MIT press Cambridge, MA, USA, 2017.
[40] F. Chollet, et al., Keras, 2015, https://keras.io.
[41] R. Verfürth, A posteriori error estimators for convection–diffusion equations, Numer. Math. 80 (1998) 641–663.
[42] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, in: JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.
[43] S. Ruder, An overview of gradient descent optimization algorithms, 2016, arXiv preprint arXiv:1609.04747.
[44] H. Borouchaki, P. Laug, The bl2d mesh generator: Beginner's guide, user's and programmer's manual, 1996.
[45] P. Passelli, M. Picasso, Daptive finite elements with large aspect ratio for aluminium electrolysis, in: Proceedings of the 11th Edition of the International Conference on Adaptive Modeling and Simulation, (ADMOS), 2023.
[46] P. Clément, Approximation by finite element functions using local regularization, Rev. Française d'Autom. Inform. Rech. Opér. Anal. Numér. 9 (1975) 77–84.
[47] M. Ainsworth, J.T. Oden, A posteriori error estimation in finite element analysis, Comput. Methods Appl. Mech. Engrg. 142 (1997) 1–88.
[48] O.C. Zienkiewicz, P.B. Morice, The Finite Element Method in Engineering Science, vol. 1977, McGraw-hill London, 1971.
[49] O.C. Zienkiewicz, J.Z. Zhu, A simple error estimator and adaptive procedure for practical engineering analysis, Int. J. Numer. Mehtods Eng. 24 (1987) 337–357.