École polytechnique fédérale de Lausanne

Semester project fall 2024

Solving an elliptic parametric PDE using finite differences and neural networks

Author:
Zoé Weissbaum

Supervisors:
Michele Barucca
Marco Picasso

Contents

1 Introduction

We consider a parametric partial differential equation (PDE) of the form

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

where $\Omega \subset \mathbb{R}^d$ ($d \geq 1$) denotes the physical space, $\mathcal{P} \subset \mathbb{R}^p$ ($p \geq 1$) is the parameter space and $\mathcal{F}$ is a differential operator. The solution u is a scalar function defined as $u : \Omega \times \mathcal{P} \to \mathbb{R}$.

Finite difference and finite element methods can be used to approximate the solution of this equation for a fixed parameter $\mu \in \mathcal{P}$. However, when multiple evaluations are needed, these methods can become computationally expensive, prompting the development of alternative modeling techniques. Neural networks have shown significant potential as an alternative for approximating the solutions of parametric PDEs, particularly in applications that demand a large number of evaluations.

We use feedforward neural networks to build a neural network approximation $u_{\mathcal{N}}$ of u, with training data generated by finite difference simulations. The training phase can be time consuming, but once this is done, the neural network can be efficiently evaluated.

We first develop neural networks that approximate the discretized parameter-to-solution map $\mu \mapsto (u(x_1,\mu), ..., u(x_N,\mu))$ for some $x_1, ..., x_N$ in Ω.

Then we focus on neural networks that give the solution pointwise, that is, we approximate the mapping $(x,\mu) \mapsto u(x,\mu)$.

To assess the accuracy of the neural network approximation, the error is measured using the norm:

$$|||u - u_{\mathcal{N}}|||^2 := \frac{1}{|\mathcal{P}|} \int_{\mathcal{P}} \|u(\cdot,\mu) - u_{\mathcal{N}}(\cdot,\mu)\|^2_{L^2(\Omega)} \, d\mu. \tag{2}$$

For any $\mu \in \mathcal{P}$, let $u_h(\cdot,\mu) : \Omega \to \mathbb{R}$ denote the finite difference approximation of $u(\cdot,\mu)$. The error $|||u - u_{\mathcal{N}}|||$ can then be decomposed as

$$|||u - u_{\mathcal{N}}||| \leq |||u - u_h||| + |||u_h - u_{\mathcal{N}}|||\,. \tag{3}$$

In this work, we focus on a one-dimensional elliptic equation. Our goal is to estimate the two error components on the right-hand side of the inequality and ensure they are of approximately the same order of magnitude. This work is organized as follows: the finite difference approximation framework is discussed in Section 2. Section 3 details the architecture and training of the neural networks. Section 4 presents numerical results to compare the different methods.

2 Finite differences approximation

To build finite difference approximations u_h of u, we use a centered finite differences scheme with N evenly distributed points $x_1, ..., x_N$ in Ω, and obtain an approximation u_i of $u(x_i)$.

Note that in 1D, the numerical computation of the solution is the same with finite differences and finite elements, because we solve the same linear system. We will thus refer to the finite difference approximation as u_h, even though this notation is most commonly used for finite element approximations in the literature. It is possible to obtain a piecewise linear continuous reconstruction via $u_i + (x - x_i)\frac{u_{i+1}-u_i}{h}$, $x_i \leq x \leq x_{i+1}$, $0 \leq i \leq N$. This is not needed in this work, but it could be useful when considering the error estimator with respect to the gradient (see section 2.1).

In order to approximate the finite difference error $|||u - u_h|||$ in (3), we use the Monte-Carlo method. We thus draw M random samples $\{\mu^{(k)}\}_{k=1}^M$ from a uniform distribution in $\mathcal{P}$, and approximate $|||u - u_h|||^2$ by $|||u - u_h|||^2_M := \frac{1}{M}\sum_{k=1}^M \left\|u(\cdot,\mu^{(k)}) - u_h(\cdot,\mu^{(k)})\right\|^2_{L^2(\Omega)}$.

The expected value of the error between $|||u - u_h|||^2$ and $|||u - u_h|||^2_M$ is given by [1]

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

where Var and Std denote the variance and the standard deviation, respectively. The standard deviation can be estimated over the sample of size M. In practice, if the exact solution u is known, the error $\|u - u_h\|_{L^2(\Omega)} = \left(\int_\Omega (u - u_h)^2\right)^{1/2}$ is approximated using a quadrature formula. But since in general the exact solution u is unknown, the error $\|u - u_h\|_{L^2(\Omega)}$ is instead bounded using an *a posteriori* error estimate.

2.1 Error estimator

We show that the error $\|u - u_h\|_{L^2(\Omega)}$ can be bounded from above by a quantity η called *a posteriori* error estimator:

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

which depends on the differential operator $\mathcal{F}$, and where C is a constant that is indepedent of the stepsize h and the solution u.

We use the following definition of element jumps.

Definition 1. Let $\Omega \subset \mathbb{R}^d$ and let $\mathcal{T}_h$ be a triangulation of Ω. Let $K, K' \in \mathcal{T}_h$ be two interior triangles that share an edge. Then we define the element jump between K and K' as

$$[\nabla u_h \cdot \vec{n_K}] = (\nabla\, u_h|_K - \nabla\, u_h|_{K'}) \cdot \vec{n_K},$$

where $\vec{n_K}$ is the outer normal vector of the shared edge, with respect to K. If K is on the boundary, then

$$[\nabla u_h \cdot \vec{n_K}] = 2\nabla\, u_h|_K \cdot \vec{n_K}.$$

We use two different interpolation operators.

Definition 2. Let $\Omega \subset \mathbb{R}^d$. The Lagrange interpolation operator is defined as

$$\mathcal{V}_h : C^0(\bar{\Omega}) \to V_h, v \mapsto \mathcal{V}_h v(x) = \sum_{j=1}^{N} v(p_j)\varphi_j(x).$$

Definition 3. Let $\Omega \subset \mathbb{R}^d$. The Clément interpolation operator is defined as

$$R_h : L^1(\Omega) \to V_h, v \mapsto R_h v(x) = \sum_{j=1}^{N} R_h v(p_j)\varphi_j(x),$$

where $R_h v(p_j) = \dfrac{\sum_{p_j \in K} \int_K v}{\sum_{p_j \in K} \int_K 1}$.

The following lemma is an intermediary bound that will be useful in the proof of the error estimator.

Lemma 1. *If the mesh is shape regular, there exists $C > 0$ such that for all $h > 0$, for all $K \in \mathcal{T}_h$ and for all $v \in H^1(\Omega)$, we have*

$$\|v - R_h v\|^2_{L^2(K)} + h_K \|v - R_h v\|^2_{L^2(\partial K)} \leq C h_K^2 \|\nabla v\|^2_{L^2(\triangle K)},$$

where $\triangle K$ is the union of triangles that share a vertex with K.

Note that this estimate is incorrect for $d = 2, 3$ with the Lagrange interpolant, because while in one dimension the embedding $H^1(\Omega) \hookrightarrow C(\bar{\Omega})$ holds, in higher dimensions $H^1(\Omega)$-functions embed in $L^q(\Omega)$, but not necessarily in $C(\bar{\Omega})$.

First, we show an upper bound on the L^2 norm of the gradient of the error. Let $\Omega \subset \mathbb{R}^d$. Consider for simplicity of explanation the Laplace equation

$$\begin{cases} -\Delta u = f \text{ in } \Omega \\ u = 0 \text{ on } \partial\Omega \end{cases}$$

with the usual weak formulation: find $u \in H^1(\Omega)$ such that $\int_\Omega \nabla u \nabla v = \int_\Omega fv$ for all $v \in H^1(\Omega)$.

Proposition 1. *There exists $C > 0$ and an estimator $\eta^2 = \sum_{K \in \mathcal{T}_h} \eta_K^2$ such that*

$$\|\nabla(u - u_h)\|^2_{L_2(\Omega)} \leq C \sum_{K \in \mathcal{T}_h} \eta_K^2.$$

Proof. $\|\nabla(u-u_h)\|^2_{L_2(\Omega)} = \int_\Omega \nabla(u-u_h)\nabla(u-u_h)\,dx = \int_\Omega \nabla u\nabla(u-u_h) - \nabla u_h\nabla(u-u_h)\,dx$

$$= \int_\Omega (f(u-u_h) - \nabla u_h\nabla(u-u_h))\,dx$$

$$= \int_\Omega (f(u-u_h-v_h) - \nabla u_h\nabla(u-u_h-v_h))\;dx = \sum_{K\in\mathcal{T}_h}\int_K f(u-u_h-v_h) - \nabla u_h\nabla(u-u_h-v_h)\,dx$$

$$= \sum_{K\in\mathcal{T}_h}\left(\int_K (f+\Delta u_h)(u-u_h-v_h) - \int_{\partial K}\nabla u_h\cdot\vec{n_K}(u-u_h-v_h)\,ds\right)$$

$$= \sum_{K\in\mathcal{T}_h}\left(\int_K (f+\Delta u_h)(u-u_h-v_h) - \frac{1}{2}\int_{\partial K}[\nabla u_h\cdot\vec{n_K}](u-u_h-v_h)\,ds\right)$$

$$\leq \sum_{K\in\mathcal{T}_h}\|f+\Delta u_h\|_{L_2(K)}\|u-u_h-v_h\|_{L^2(K)} + \frac{1}{2}\|[\nabla u_h\cdot\vec{n_K}]\|_{L^2(\partial K)}\|u-u_h-v_h\|_{L^2(\partial K)}$$

Taking $v_h = R_h(u-u_h)$ (Clément interpolant) and using Lemma 1, we get

$$\|\nabla(u-u_h)\|^2_{L_2(\Omega)} \leq C\sum_{K\in\mathcal{T}_h}\left(\|f+\Delta u_h\|_{L_2(K)}\,h_K + \frac{1}{2}\|[\nabla u_h\cdot\vec{n_K}]\|_{L^2(\partial K)}\,h_K^{1/2}\right)\|\nabla(u-u_h)\|_{L^2(\triangle K)}$$

$$\leq C\left(\sum_{K\in\mathcal{T}_h}\eta_K^2\right)^{1/2}\left(\sum_{K\in\mathcal{T}_h}\|\nabla(u-u_h)\|^2_{L^2(\triangle K)}\right)^{1/2} \leq C\left(\sum_{K\in\mathcal{T}_h}\eta_K^2\right)^{1/2}\|\nabla(u-u_h)\|_{L^2(\Omega)}.$$

Hence we get the upper bound $\|\nabla(u-u_h)\|^2_{L_2(\Omega)} \leq C\sum_{K\in\mathcal{T}_h}\eta_K^2$ by defining

$$\eta_K := \|f+\Delta u_h\|_{L_2(K)}\,h_K + \frac{1}{2}\|[\nabla u_h\cdot\vec{n_K}]\|_{L^2(\partial K)}\,h_K^{1/2}.$$

□

We can then show a similar bound on the L^2 norm of the error. Consider the dual problem

$$\begin{cases} -\Delta\varphi = u-u_h \text{ in } \Omega \\ \varphi = 0 \text{ on } \partial\Omega \end{cases}$$

and its weak formulation: $\int_\Omega \nabla\varphi\nabla v = \int_\Omega (u-u_h)v$ for all $v\in H_0^1(\Omega)$.

Proposition 2. *There exists $C>0$ such that*

$$\|u-u_h\|^2_{L^2(\Omega)} \leq C\sum_{K\in\mathcal{T}_h}\eta_K^2 h_K^2.$$

Proof. $\|u-u_h\|^2_{L_2(\Omega} = \int_\Omega \nabla\varphi\nabla(u-u_h)\,dx = \int_\Omega \nabla(\varphi-v_h)\nabla(u-u_h)\,dx$

$$= \int_\Omega f(\varphi-v_h) - \nabla u_h\nabla(\varphi-v_h)\,dx = \int_\Omega (f-\Delta u_h)(\varphi-v_h)\,dx + \int_{\partial\Omega}\nabla u_h\cdot\vec{n}(\varphi-v_h)\,ds$$

$$= \sum_{K\in\mathcal{T}_h}\left(\int_K (f-\Delta u_h)(\varphi-v_h)\,dx + \frac{1}{2}\int_{\partial K}[\nabla u_h\cdot\vec{n_K}](\varphi-v_h)\,ds\right)$$

Since Ω is a convex polygon, we have $\varphi\in H^2(\Omega)$, thus φ is continuous ($\varphi\in C^0(\bar{\Omega})$). Taking $v_h = \mathcal{V}_h\varphi$ (Lagrange interpolant) and using the inequality

$$\frac{1}{h_K^2}\|\varphi-\mathcal{V}_h\varphi\|_{L^2(K)} + \frac{1}{h_K^{3/2}}\|\varphi-\mathcal{V}_h\varphi\|_{L^2(\partial K)} \leq \left\|D^2\varphi\right\|_{L^2(K)}$$

we get

$$\|u-u_h\|^2_{L_2(\Omega} \leq C \sum_{K\in\mathcal{T}_h} \left(\|f+\Delta u_h\|_{L_2(K)} h_K^2 + \frac{1}{2} \|[\nabla u_h \cdot \vec{n_K}]\|_{L^2(\partial K)} h_K^{3/2} \right) \left\|D^2\varphi\right\|_{L^2(K)}$$

$$\leq C \left(\sum_{K\in\mathcal{T}_h} \eta_K^2 h_K^2 \right)^{1/2} \left(\sum_{K\in\mathcal{T}_h} \left\|D^2\varphi\right\|^2_{L^2(K)} \right)^{1/2} \leq C \left(\sum_{K\in\mathcal{T}_h} \eta_K^2 h_K^2 \right)^{1/2} \|u-u_h\|_{L^2(\Omega)}$$

since $\|\varphi\|_{H^2(K)} \leq C \|u-u_h\|^2_{L^2(\Omega)}$. This yields the bound $\|u-u_h\|^2_{L^2(\Omega)} \leq C \sum_{K\in\mathcal{T}_h} \eta_K^2 h_K^2$.

□

When $d=1$, we do not have element jumps, and if we use piecewise linear basis functions for the finite element solution, we get $\Delta u_h = 0$. Thus, in that case, we have $\eta_K^2 = h_K^2 \|f\|^2_{L^2(K)}$. More specifically, when $\Omega = [0,1]$ and $h_K = h_i = x_{i+1} - x_i = \frac{1}{N+1} = h$ for all K, we obtain $\eta_i^2 = h^2 \|f\|^2_{L^2([x_i,x_{i+1}])}$, $i = 1, ..., N$. We can then approximate the error as

$$\|u-u_h\|^2_{L^2(0,1)} = \int_0^1 (u-u_h)^2(x,\mu)\, dx \approx \sum_{i=0}^{N} h^2 \eta_i^2(\mu) = h^4 \sum_{i=0}^{N} \|f\|^2_{L^2([x_i,x_{i+1}])}.$$

3 Neural networks

Linear basis function models for classification take the form

$$y(x,w) = f\left(\sum_{j=1}^{M} w_j \phi_j(x) + w_0 \right) \tag{4}$$

where $f(\cdot)$ is a nonlinear output activation function. Linear models for regression take the same form, but with $f(\cdot)$ replaced by the identity. The set of nonlinear basis functions $\{\phi_i(x)\}$ can be chosen arbitrarily, so that theoretically with this model we could approximate any function. However, in linear models the basis function are chosen ahead of time, and this can be a significant limitation (curse of dimensionality). To solve this problem, neural networks allow for the basis functions to have learnable parameters, and thus to vary during training. One way of doing this is by choosing basis functions that themselves have the same form as (4). In that case, the learnable parameters are the *weights* w_j and the *biases* w_0.

More specifically, we use fully connected feedforward neural networks. A feedforward neural network is made up of an input layer, an output layer, and $L \geq 1$ hidden layers. We let n_j be the number of neurons in the j-th layer, $j = 0, ..., L+1$. We denote by σ_i^j the activation function of the i-th neuron of the j-th layer, $i = 1, ..., n_j$, $j = 0, ..., L+1$. Possible activation functions for neurons in hidden layers are, for example, the Rectified Linear Unit: $ReLU(x) = \max\{x, 0\}$, or the softplus function: $softplus(x) = \ln(1+e^x)$. The activation function for neurons in the output layer is the identity. We let z_i^j be the value associated with the i-th neuron of the j-th layer. These values are given recursively by

$$z_i^j = \sigma_i^j \left(\sum_{k=1}^{n_{j-1}} w_{ik}^{(j)} z_k^{j-1} + w_{i0}^{(j)} \right). \tag{5}$$

For example, for a network with an input layer $x = (x_1, ..., x_{n_0})$, one hidden layer, and an output layer, we would get

$$y_k(x,w) = \sum_{j=1}^{n_1} w_{kj}^{(2)} \sigma_j^1 \left(\sum_{i=1}^{n_0} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \tag{6}$$

for each of the K outputs, $k = 1, ..., K$.

To build feedforward neural networks that approximate the solution to (1), we consider two approaches. In the first approach, the goal is to approximate the discretized parameter-to-solution map, that is, we directly build the full solution. The second approach attempts to approximate the map $(x,\mu) \mapsto u(x,\mu)$, so that we construct the solution pointwise.

3.1 Full solution map

Let $\{x_1, ..., x_N\}$ be a discretization of Ω, and $u_1(\mu), ..., u_N(\mu)$ be the finite difference approximations of $u(x_1, \mu), ..., u(x_N, \mu)$. In this first approach, the goal is to approximate the mapping $\mu \mapsto (u_1(\mu), ..., u_N(\mu))^\top$.

To do this, we first select the training parameters $\{\tilde{\mu}^{(j)}\}_{j=1}^{N_{train}} \subset \mathcal{P}$. Then, for each $\tilde{\mu}^{(j)}$, we compute the finite difference approximation $u_h(x, \tilde{\mu}^{(j)})$ given by $u_1(\tilde{\mu}^{(j)}), ..., u_N(\tilde{\mu}^{(j)})$. Finally, we can set the architecture of the neural network, that is, choose L, W and σ, and find the learnable parameters that minimize a chosen loss function.

For the full solution approach, we use as the training set the parameters $\{\mu^{(k)}\}_{k=1}^{N_{train}}$ with the corresponding solutions $\{\big(u_1(\tilde{\mu}^{(j)}), ..., u_N(\tilde{\mu}^{(j)})\big)\}_{j=1}^{N_{train}}$. Then, the output layer has size N, which is the number of nodes used in the finite differences approximations. As the loss function, we use the classical Mean Squared Error (MSE):

$$\frac{1}{N_{train}N} \sum_{j=1}^{N_{train}} \sum_{i=1}^{N} (u_{\mathcal{N}}(x_i, \tilde{\mu}^{(j)}) - u_h(x_i, \tilde{\mu}^{(j)}))^2.$$

3.2 One node map

The second approach attempts to create a map $(x, \mu) \mapsto u_{\mathcal{N}}(x, \mu)$ for $x \in \{x_1, ..., x_N\}$, so that we construct the solution pointwise, or node by node. This would allow to further develop a method with adapted grid such that the error is close to a preset tolerance, because the grids are different for each parameter.

We proceed in the exact same way as for the first approach: we choose the training parameters $\{\tilde{\mu}^{(j)}\}_{j=1}^{N_{train}} \subset \mathcal{P}$, and compute the finite difference approximations $u_h(x, \tilde{\mu}^{(j)})$ for each $\tilde{\mu}^{(j)}$. We finally choose L, W and σ, and find the learnable parameters that minimize a chosen loss function.

This time, the training set is composed of $\{(x_i, \tilde{\mu}^{(j)})_{i=1}^{N}\}_{j=1}^{N_{train}}$ with corresponding solutions $\{(u_i(\tilde{\mu}^{(j)})_{i=1}^{N}\}_{j=1}^{N_{train}}$. The output layer then has size 1. We again use the MSE as the loss function.

Now that all methods have been presented, we can finally test them and compare them on a specific elliptic problem.

4 Numerical experiments

As a model problem, we consider the parametric Poisson equation in 1D over the interval $\Omega = [0, 1]$, and set $p = 3$. Given $\mu = (\mu_1, \mu_2, \mu_3) \in \mathcal{P}$, the problem consists in finding $u : [0, 1] \to \mathbb{R}$ such that

$$\begin{cases} -u''(x, \mu) = f(x, \mu), \quad 0 < x < 1, \\ u(0, \mu) = a(\mu), \\ u(1, \mu) = b(\mu). \end{cases} \tag{7}$$

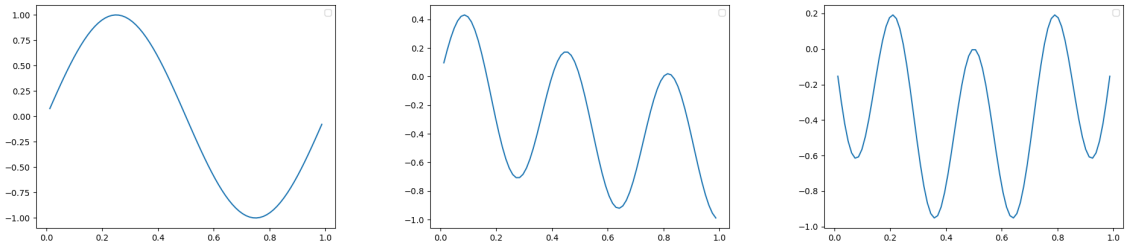
We consider $\mathcal{P} = [-2, 2] \times [1, 4] \times [1, 4]$, and a and b to be such that the exact solution u is given by

$$u(x, \mu) = \frac{\mu_1}{2} \sin(\mu_2 \pi x) \cos(\mu_3 \pi x).$$

Figure 1 shows how the solution can vary depending on the values of the parameters $\mu = (\mu_1, \mu_2, \mu_3)$.

4.1 Finite differences

We start with a fixed parameter $\mu^{(0)} = (2, 2, 0)$, to check the convergence of the error and the error estimator in terms of the discretization size N. The exact solution u is given by $u(x, \mu^{(0)}) = \sin(2\pi x)$. We have $h = \frac{1}{N+1}$ and $x_i = ih$, $i = 1, ..., N$. We get finite difference approximations $u_1(\mu^{(0)}), ..., u_N(\mu^{(0)})$ of $u(x_1, \mu^{(0)}), ..., u(x_N, \mu^{(0)})$ and we approximate $\left\|u(\cdot, \mu^{(0)}) - u_h(\cdot, \mu^{(0)})\right\|_{L^2(0,1)}$ by $(h \sum_{i=1}^{N} |u(x_i, \mu^{(0)}) - u_i(\mu^{(0)})|^2)^{1/2}$. We also compute the error estimator $\eta = (\sum_{i=0}^{N} h^2 \eta_i^2)^{1/2}$, where $\eta_i = h^2 \left\|f\right\|^2_{L^2([x_i, x_{i+1}])}$.



(a) $\mu = (2, 2, 0)$ (b) $\mu = (2, 2.5, 3)$ (c) $\mu = (-2, 4, 3)$

Figure 1: Plot of the solution $u(x, \mu)$, for different values of μ.

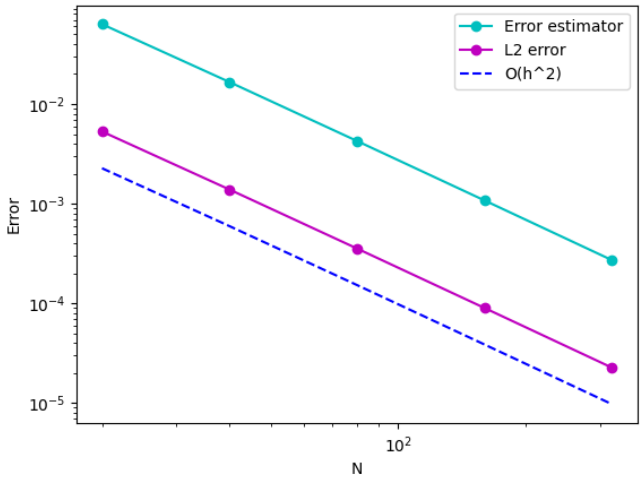


Figure 2: Computed errors from finite difference approximations as a function of N, and their theoretical order (dashed line).

A plot of the errors can be viewed on Figure 2. As expected by theoretical results, the L^2 norm of the error, as well as the error estimator η, behave as $O(h^2)$. We can also observe that the error estimator is indeed an upper bound of $\|u - u_h\|^2_{L^2(0,1)}$.

We then test the method on M random samples $\{\mu^{(k)}\}_{k=1}^M$, as presented in Section 2. We define the effectivity index

$$ei(\mu) := \frac{\left(\sum_{i=0}^N h^2 \eta_i^2(\mu)\right)^{1/2}}{\|u(\cdot, \mu) - u_h(\cdot, \mu)\|_{L^2(0,1)}}.$$

The effectivity index is used to track the sharpness of the error estimator. 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 the set $\{\|u(\cdot, \mu^{(k)}) - u_h(\cdot, \mu^{(k)})\|^2_{L^2(0,1)}\}_{k=1}^M$. We observe that $|||u - u_h|||^2_M$ behaves as $O(h^4)$, as expected. We also see that the mean of the effectivity index is independent of both h and M, but its standard error is large. This indicates that the effectivity index depends on μ. Next, for $M = 4000$, the expected error of the Monte Carlo method $\frac{\mathrm{Std}_M(\|u-u_h\|^2)}{\sqrt{M}}$ is at least one order of magnitude smaller than the error $|||u - u_h|||^2_M$. Consequently, the first significant digit of $|||u - u_h|||^2_M$ is accurate on average. Finally, $\mathrm{Std}_M(\|u - u_h\|^2)$ is relatively large compared to the error, highlighting that the accuracy of the finite element method is highly sensitive to the parameter μ.

Table 1: Computed error of the finite differences solution for various values of h and M.

h	$N+1$	M	$\|\|\|u-u_h\|\|\|_M^2$	$\texttt{Std}_M(\|u-u_h\|^2)$	$\frac{\texttt{Std}_M(\|u-u_h\|^2)}{\sqrt{M}}$	ei_M	$\texttt{Std}_M(ei)$
0.05	20	500	$2.22 \cdot 10^{-4}$	$3.50 \cdot 10^{-4}$	$1.57 \cdot 10^{-5}$	10.22	1.02
		1000	$2.12 \cdot 10^{-4}$	$3.29 \cdot 10^{-4}$	$1.04 \cdot 10^{-5}$	10.21	0.99
		2000	$2.07 \cdot 10^{-4}$	$3.29 \cdot 10^{-4}$	$7.36 \cdot 10^{-6}$	10.24	1.03
		4000	$2.11 \cdot 10^{-4}$	$3.29 \cdot 10^{-4}$	$5.19 \cdot 10^{-6}$	10.24	1.03
0.025	40	500	$1.29 \cdot 10^{-5}$	$1.99 \cdot 10^{-5}$	$8.94 \cdot 10^{-7}$	10.49	1.04
		1000	$1.24 \cdot 10^{-5}$	$1.88 \cdot 10^{-5}$	$5.95 \cdot 10^{-7}$	10.46	1.01
		2000	$1.21 \cdot 10^{-5}$	$1.88 \cdot 10^{-5}$	$4.21 \cdot 10^{-7}$	10.49	1.05
		4000	$1.23 \cdot 10^{-5}$	$1.88 \cdot 10^{-5}$	$2.97 \cdot 10^{-7}$	10.49	1.04
0.0125	80	500	$7.93 \cdot 10^{-7}$	$1.22 \cdot 10^{-6}$	$5.46 \cdot 10^{-8}$	10.55	1.05
		1000	$7.60 \cdot 10^{-7}$	$1.15 \cdot 10^{-6}$	$3.64 \cdot 10^{-8}$	10.53	1.02
		2000	$7.42 \cdot 10^{-7}$	$1.15 \cdot 10^{-6}$	$2.57 \cdot 10^{-8}$	10.56	1.05
		4000	$7.56 \cdot 10^{-7}$	$1.15 \cdot 10^{-6}$	$1.82 \cdot 10^{-8}$	10.55	1.05

4.2 Neural networks

We now approximate the solution of (7) with neural networks, using the two approaches presented in Section 3. All networks are built and trained using the Python library PyTorch. The neural networks are trained with the Adam optimizer, starting from a learning rate of 0.001. The training parameters $\{\tilde{\mu}^{(j)}\}_{j=1}^{N_{train}}$ are drawn randomly from a uniform distribution in $\mathcal{P}$. The parameters are then normalized to have mean zero and standard deviation one.

We take $\sigma = ReLU$ as activation function.

Similarly as for finite differences, we use the Monte-Carlo method to approximate $\|\|u_h - u_\mathcal{N}\|\|^2 = \frac{1}{|\mathcal{P}|}\int_\mathcal{P} \|u_h(\cdot,\mu) - u_\mathcal{N}(\cdot,\mu)\|^2_{L^2(0,1)}\, d\mu$ by $\|\|u_h - u_\mathcal{N}\|\|^2_M := \frac{1}{M}\sum_{k=1}^M \left\|u_h(\cdot,\mu^{(k)}) - u_\mathcal{N}(\cdot,\mu^{(k)})\right\|^2_{L^2(0,1)}$. We use the same test samples $\{\mu^{(k)}\}_{k=1}^M$ as for finite differences.

4.2.1 Full solution map

We use batches of size 256. We start by fixing the training size $N_{train} = 8000$, and vary the number of hidden layers L, the number of neurons per layer W and the size of the discretization N to test the impact of the network architecture on the accuracy.

Table 2: Error of full solution neural networks for various values of L, W and N ($N_{train} = 8000$, $M = 4000$).

L	W	$N+1$	$\|\|\|u_\mathcal{N}-u_h\|\|\|_M^2$	$\texttt{Std}_M(\|u_\mathcal{N}-u_h\|^2)$	$\frac{\texttt{Std}_M(\|u_\mathcal{N}-u_h\|^2)}{\sqrt{M}}$
3	50	40	$1.09{\cdot}10^{-3}$	$1.39{\cdot}10^{-3}$	$2.19{\cdot}10^{-5}$
3	50	80	$1.09{\cdot}10^{-3}$	$1.33{\cdot}10^{-3}$	$2.11{\cdot}10^{-5}$
3	100	40	$4.52{\cdot}10^{-4}$	$5.87{\cdot}10^{-4}$	$9.29{\cdot}10^{-6}$
3	100	80	$4.58{\cdot}10^{-4}$	$5.74{\cdot}10^{-4}$	$9.08{\cdot}10^{-6}$
4	50	40	$6.29{\cdot}10^{-4}$	$7.47{\cdot}10^{-4}$	$1.18{\cdot}10^{-5}$
4	50	80	$7.80{\cdot}10^{-4}$	$1.01{\cdot}10^{-3}$	$1.60{\cdot}10^{-5}$
4	100	40	$2.91{\cdot}10^{-4}$	$3.12{\cdot}10^{-4}$	$4.93{\cdot}10^{-6}$
4	100	80	$3.31{\cdot}10^{-4}$	$3.49{\cdot}10^{-4}$	$5.52{\cdot}10^{-6}$

Table 2 presents the error of neural networks with varying widths, depths, and output sizes. As required, the expected error of the Monte Carlo method $\frac{\texttt{Std}_M(\|u_\mathcal{N}-u_h\|^2)}{\sqrt{M}}$ is small compared to the computed value of $\|\|u_\mathcal{N} - u_h\|\|^2_M$. Notably, the standard deviation of $\|u_\mathcal{N} - u_h\|^2$ is slightly larger than $\|\|u_\mathcal{N} - u_h\|\|^2_M$, indicating that the network's accuracy varies across the parameter space.

For a fixed width (or depth), increasing the depth (or width) improves the accuracy of the networks. However, comparing the results in Tables 1 and 2 reveals that, for this training set size, the errors of all neural networks remain higher than those of the finite difference method.

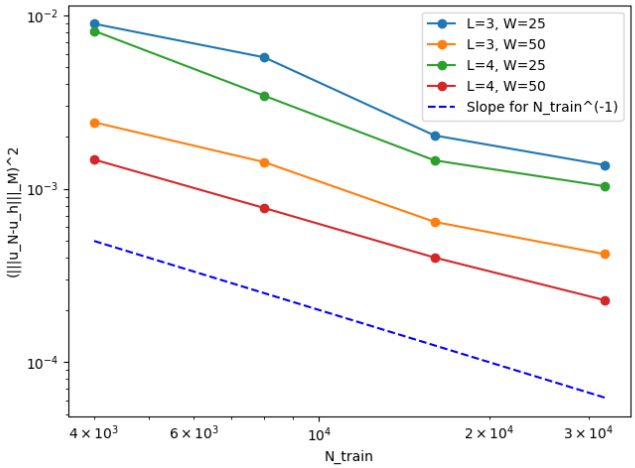


Figure 3: Computed error of full solution neural networks as a function of N_{train} for $h = 0.05$ and $M = 4000$.

We next set $h = 0.05$ ($N = 19$) and test the impact of the training size on the accuracy of the networks. In Figure 3, we can see the results for various architectures. We observe that $|||u_h - u_{\mathcal{N}}|||_M^2$ behaves approximately as $O(N_{train}^{-1})$. Even with a significantly large training size, the achieved accuracy remains inferior to that of finite difference approximations.

Table 3: Comparison of $|||u - u_h|||_M^2$, $|||u_h - u_{\mathcal{N}}|||_M^2$,$|||u - u_{\mathcal{N}}|||_M^2$ for various architectures of full solution networks ($N_{train} = 4000$, $M = 2000$).

h	$\|\|\|u - u_h\|\|\|_M^2$	L	W	$\|\|\|u_h - u_{\mathcal{N}}\|\|\|_M^2$	$\|\|\|u - u_{\mathcal{N}}\|\|\|_M^2$
0.05	$2.17{\cdot}10^{-4}$	3	50	$2.36 \cdot 10^{-3}$	$2.38 \cdot 10^{-3}$
		3	100	$9.72 \cdot 10^{-4}$	$1.12 \cdot 10^{-3}$
		4	50	$2.10 \cdot 10^{-3}$	$2.16 \cdot 10^{-3}$
		4	100	$4.24 \cdot 10^{-4}$	$6.10 \cdot 10^{-4}$
0.025	$1.26 \cdot 10^{-5}$	3	50	$2.26 \cdot 10^{-3}$	$2.23 \cdot 10^{-3}$
		3	100	$8.48 \cdot 10^{-4}$	$8.49 \cdot 10^{-4}$
		4	50	$1.24 \cdot 10^{-3}$	$1.23 \cdot 10^{-3}$
		4	100	$6.05 \cdot 10^{-4}$	$6.13 \cdot 10^{-4}$

In Table 3, we can see the comparison of the three errors $|||u - u_h|||_M^2$, $|||u_h - u_{\mathcal{N}}|||_M^2$,$|||u - u_{\mathcal{N}}|||_M^2$ and check that$|||u - u_{\mathcal{N}}|||_M^2 \leq |||u - u_h|||_M^2 + |||u_h - u_{\mathcal{N}}|||_M^2$. The neural network error is somewhat comparable to that of the finite difference method for larger values of L and W, but it remains higher. This highlights the need for an alternative neural network approach.

4.2.2 One node map

With this approach, the number of samples in the training set is equal to $N \cdot N_{train}$ as opposed to N_{train} for the full map approach. Thus, we use larger batches, of size 2048. Nevertheless, we observe that training is still slower than with the first approach.

Again, we start by fixing N_{train} to 8000 and examining the effect of other parameters on the accuracy. Table 4 reports the errors of neural networks with varying widths, depths, and discretization sizes. The results are similar to those obtained with the first approach, albeit slightly less accurate.

We also evaluate the effect of training size on the error of the neural networks. Figure 4 illustrates the accuracy of various network architectures as a function of the training size N_{train}. The results indicate

Table 4: Error of one node neural networks for various values of L, W and N ($N_{train} = 8000$, $M = 4000$).

L	W	$N+1$	$\|\|\|u_{\mathcal{N}} - u_h\|\|\|_M^2$	$\mathtt{Std}_M(\|u_{\mathcal{N}} - u_h\|^2)$	$\frac{\mathtt{Std}_M(\|u_{\mathcal{N}} - u_h\|^2)}{\sqrt{M}}$
3	25	40	$7.94{\cdot}10^{-3}$	$9.28{\cdot}10^{-3}$	$1.47{\cdot}10^{-4}$
3	25	80	$3.58{\cdot}10^{-3}$	$4.18{\cdot}10^{-3}$	$6.61{\cdot}10^{-5}$
3	50	40	$1.69{\cdot}10^{-3}$	$1.82{\cdot}10^{-3}$	$2.88{\cdot}10^{-5}$
3	50	80	$1.98{\cdot}10^{-3}$	$2.14{\cdot}10^{-3}$	$3.38{\cdot}10^{-5}$
4	25	40	$2.77{\cdot}10^{-3}$	$3.04{\cdot}10^{-3}$	$4.81{\cdot}10^{-5}$
4	25	80	$1.98{\cdot}10^{-3}$	$2.14{\cdot}10^{-3}$	$3.38{\cdot}10^{-5}$
4	50	40	$8.20{\cdot}10^{-4}$	$7.83{\cdot}10^{-4}$	$1.24{\cdot}10^{-5}$
4	50	80	$9.18{\cdot}10^{-4}$	$7.38{\cdot}10^{-4}$	$1.17{\cdot}10^{-5}$

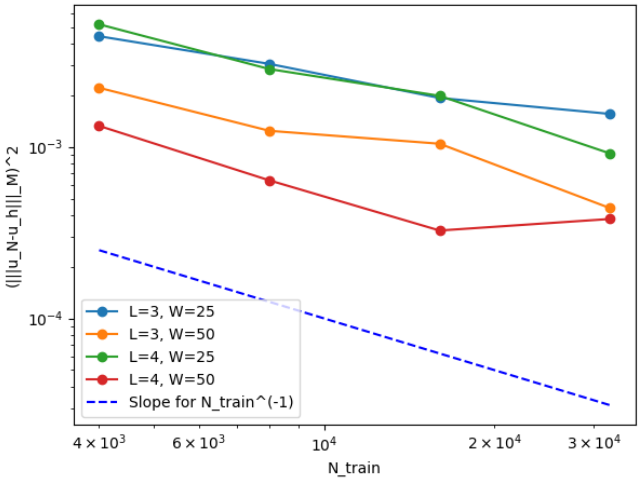


Figure 4: Computed error of one node neural networks as a function of N_{train} for $h = 0.05$ and $M = 4000$.

that this approach performs worse than the first method. We conclude that, without an adapted grid, this approach is inefficient, being both slower and less accurate than the full solution map. It is expected that the results would improve with the use of an adaptive method.

Table 5: Comparison of $|||u - u_h|||_M^2$, $|||u_h - u_{\mathcal{N}}|||_M^2$,$|||u - u_{\mathcal{N}}|||_M^2$ for various architectures of one node networks ($N_{train} = 4000$, $M = 2000$).

h	$\|\|\|u - u_h\|\|\|_M^2$	L	W	$\|\|\|u_h - u_{\mathcal{N}}\|\|\|_M^2$	$\|\|\|u - u_{\mathcal{N}}\|\|\|_M^2$
0.05	$2.17{\cdot}10^{-4}$	3	25	$1.36 \cdot 10^{-2}$	$1.22 \cdot 10^{-2}$
		3	50	$7.49 \cdot 10^{-3}$	$6.74 \cdot 10^{-3}$
		4	25	$7.08 \cdot 10^{-3}$	$6.42 \cdot 10^{-3}$
		4	50	$3.21 \cdot 10^{-3}$	$2.94 \cdot 10^{-3}$
0.025	$1.26 \cdot 10^{-5}$	3	25	$1.11 \cdot 10^{-2}$	$1.08 \cdot 10^{-2}$
		3	50	$3.28 \cdot 10^{-3}$	$3.21 \cdot 10^{-3}$
		4	25	$7.91 \cdot 10^{-3}$	$7.70 \cdot 10^{-3}$
		4	50	$1.87 \cdot 10^{-3}$	$1.83 \cdot 10^{-3}$

We still compare the three errors $|||u - u_h|||_M^2$, $|||u_h - u_{\mathcal{N}}|||_M^2$,$|||u - u_{\mathcal{N}}|||_M^2$ for this approach and check that $|||u - u_{\mathcal{N}}|||_M^2 \leq |||u - u_h|||_M^2 + |||u_h - u_{\mathcal{N}}|||_M^2$. The results are shown in Table 5.

5 Conclusion

In this work, we established a framework combining finite difference/finite element methods and neural networks to approximate the solution of a parametric elliptic PDE. Specifically, we explored two approaches: one targeting the discretized parameter-to-solution map and the other approximating the pointwise solution map. Despite their potential, neither method matched the accuracy of finite difference approximations, underscoring the need for the development of improved neural network methodologies.

One promising direction is to leverage the pointwise solution map to construct an adaptive method, which could provide better error control, as achieved in [1]. Additionally, future studies could extend these methods to tackle more complex PDEs or higher-dimensional problems (e.g., as explored for 2D in [1]). Such advancements would further assess the robustness and applicability of neural network-based approaches in solving parametric PDEs.

References

[1] A. Caboussat, M. Girardin, and M. Picasso, "Error assessment of an adaptive finite elements—neural networks method for an elliptic parametric pde," *Computer Methods in Applied Mechanics and Engineering*, 2024.

[2] C. M. Bishop and H. Bishop, *Deep Learning - Foundations and Concepts*. Springer, 2024.

[3] A. Caboussat, M. Girardin, and M. Picasso, "Error assessment for an adaptive finite elements: Neural networks approach applied to parametric pdes: The transport equation," in *Proceedings of the XIth International Conference on Adaptive Modeling and Simulation (ADMOS 2023)*, 2023.

[4] R. Verfürth, "A posteriori error estimators for convection-diffusion equations," *Numerische Mathematik*, 1998.