Journal of Scientific Computing (2022) 92:100
https://doi.org/10.1007/s10915-022-01950-4

Variational Physics Informed Neural Networks: the Role of Quadratures and Test Functions

Stefano Berrone[1] · Claudio Canuto[1] · Moreno Pintore[1]

Received: 4 September 2021 / Revised: 3 June 2022 / Accepted: 6 July 2022 /
Published online: 1 August 2022

Abstract
In this work we analyze how quadrature rules of different precisions and piecewise polynomial test functions of different degrees affect the convergence rate of Variational Physics Informed Neural Networks (VPINN) with respect to mesh refinement, while solving elliptic boundary-value problems. Using a Petrov-Galerkin framework relying on an inf-sup condition, we derive an a priori error estimate in the energy norm between the exact solution and a suitable high-order piecewise interpolant of a computed neural network. Numerical experiments confirm the theoretical predictions and highlight the importance of the inf-sup condition. Our results suggest, somehow counterintuitively, that for smooth solutions the best strategy to achieve a high decay rate of the error consists in choosing test functions of the lowest polynomial degree, while using quadrature formulas of suitably high precision.

Keywords Variational Physics Informed Neural Networks · Quadrature formulas · Inf-sup condition · A priori error estimate · Convergence rates · Elliptic problems

Mathematics Subject Classification 35B45 · 35J20 · 35Q93 · 65K10 · 65N20 · 68T07

1 Introduction

Exploiting the recent advances in artificial intelligence and, in particular, in deep learning, several innovative numerical techniques have been developed in the last few years to compute numerical solutions of partial differential equations (PDEs). In such methods, the solution is approximated by a neural network that is trained by taking advantage of the knowledge of the underlying differential equation. One of the earliest models involving a neural network

✉ Claudio Canuto
claudio.canuto@polito.it

Stefano Berrone
stefano.berrone@polito.it

Moreno Pintore
moreno.pintore@polito.it

1 Dipartimento di Scienze Matematiche, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy

was described in [1]: it is based on the concept of Physics Informed Neural Networks (PINN) and it inspired further works such as e.g. [2] or [3], until the recent paper [4] which presents a very general framework for the solution of operator equations by deep neural networks.

In such papers, given an arbitrary PDE coupled with proper boundary conditions, the training of the PINN aims at finding the weights $\mathbf{w}$ of a neural network such that the associated function $u^{\mathcal{NN}}(\mathbf{x}; \mathbf{w})$ minimizes some functional of the equation residual while satisfying as much as possible the imposed boundary conditions. To do so, the neural network is trained to minimize the residual only at a finite set of collocation points and additional terms are added to the loss function in order to force the network to approximately satisfy the boundary conditions. Thanks to the good approximation properties of neural networks, formally proved e.g. in [5–10] under suitable assumptions, the PINN approach looks very promising because it is able to efficiently and accurately compute approximate solutions of arbitrary PDEs encoding their structures in the loss function.

Subsequently, the PINN paradigm has been further developed in [11] to obtain the so-called Variational Physics Informed Neural Networks (VPINN). The main differences with respect to the PINN are that the weak formulation of the PDE is exploited, the collocation points are replaced by test functions, and quadrature points are used to compute the integrals involved in the variational residuals. In such a method the solution is still approximated by a neural network, but the test functions are represented by a finite set of known functions or by a second neural network (see [12]); therefore, the technique can be seen as a Petrov-Galerkin method. The method is more flexible than the standard PINN because the integration by parts, involved in the weak formulation, decreases the required regularity of the approximate solution. Furthermore, the fact that the dataset used in the training phase consists of quadrature points significantly reduces the computational cost of the training phase. Indeed, quadrature points are, in general, much fewer than collocation points.

Combining the VPINN with the Finite Element Method (FEM), the authors of [13] developed VarNet, a VPINN that exploits the test functions of the $\mathbb{P}_1$-FEM. Such a work has been then extended in [14] to consider arbitrary high-order polynomials as test functions, as in the *hp* version of the FEM. Although the authors of the cited works empirically observed that both PINNs and VPINNs are able to efficiently approximate the desired solution, no proof of a priori error estimates with convergence rates is provided for VPINNs. On the contrary, rigorous a posteriori error analyses are already available (see, for instance, [15]). Recently (see [16]) we derived a posteriori error estimates for the discretization setting considered in this paper.

The purpose of this paper is to investigate how the choice of piecewise polynomial test functions and quadrature formulas influence the accuracy of the resulting VPINN approximation of second-order elliptic problems. One might think that test functions of high polynomial degree are needed to get a high order of accuracy; we prove that this is not the case, actually we indicate that precisely the opposite is true: it is more convenient to keep the degree of the test functions as low as possible, while using quadrature formulas of precision as high as possible. Indeed, for sufficiently smooth solutions, the error decay rate is given by

$$q + 2 - k_{\text{test}} \,,$$

where q is the precision of the quadrature formula and k_{test} is the degree of the test functions.

Using a Petrov–Galerkin framework, we derive an a priori error estimate in the energy norm between the exact solution and a suitable piecewise polynomial interpolant of the computed neural network; we assume that the architecture of the neural network is fixed and sufficiently rich, and we explore the behaviour of the error versus the size of the mesh supporting the test functions. Our analysis relies upon the validity of an inf-sup condition between the spaces of test functions and the space in which the neural network is interpolated.

Numerical experiments confirm the theoretical prediction. Interestingly, in our experiments the error between the exact solution and the computed neural network decays asymptotically with the same rate as predicted by our theory for the interpolant of the network; however, this behaviour cannot be rigorously guaranteed, since in general the minimization problem which defines the computed neural network is underdetermined, and the computed neural network may be affected by spurious components. Indeed, we show that for a problem with zero data the minimization of the loss function may yield non-vanishing neural networks. With the method proposed in this paper, we combine the efficiency of the VPINN approach with the availability of a sound and certified convergence analysis.

The paper is organized as follows. In Sect. 2 we introduce the elliptic problem we are focusing on, and we also present the way in which the Dirichlet boundary conditions are exactly imposed, which is uncommon in PINNs and VPINNs but can be generalized as in [17]. In Sect. 3 we focus on the numerical discretization; in particular, the involved neural network architecture is described in Sect. 3.1, while the problem discretization and the corresponding loss function are described in Sect. 3.2. Here we also introduce an interpolation operator $\mathcal{I}_H$ applied to the neural networks. Sect. 4 is the key theoretical section: through a series of preliminary results, we formally derive the a priori error estimate, the main result being Theorem 6. In Sect. 5 we specify the parameters of the neural network used for the numerical tests and the training phase details. We also analyse the consequences of fulfilling the inf-sup condition in connection with the VPINN efficiency. Various numerical tests are presented and discussed in Sect. 6 for two-dimensional elliptic problems. Such tests empirically confirm the validity of the a priori estimate in different scenarios. Furthermore, we compare the accuracy of the proposed method with that of a standard PINN and the non-interpolated VPINN. We also analyse the relationship between the neural network hyperparameters and the VPINN accuracy, and we highlight, with numerical experiments and analytical examples, the importance of the inf-sup condition. In Sect. 7, we show that our VPINN can be easily adapted to solve a parametric nonlinear PDE, with accurate results for the whole range of parameters. Finally, in Sect. 8, we draw some conclusions and highlight the future perspective of the current work.

2 The Model Boundary-value Problem

Let $\Omega \subset \mathbb{R}^n$ be a bounded polygonal/polyhedral domain with boundary $\Gamma = \partial\Omega$, partitioned into $\Gamma = \Gamma_D \cup \Gamma_N$ with $\Gamma_D \cap \Gamma_N = \emptyset$ and $\text{meas}_{n-1}(\Gamma_D) > 0$.

Let us consider the model elliptic boundary-value problem

$$\begin{cases} Lu := -\nabla \cdot (\mu \nabla u) + \boldsymbol{\beta} \cdot \nabla u + \sigma u = f & \text{in } \Omega\,, \\ u = g & \text{on } \Gamma_D\,, \\ \mu \frac{\partial u}{\partial n} = \psi & \text{on } \Gamma_N\,, \end{cases} \tag{2.1}$$

where $\mu, \sigma \in \mathrm{L}^\infty(\Omega)$, $\boldsymbol{\beta} \in (\mathrm{W}^{1,\infty}(\Omega))^n$ satisfy $\mu \geq \mu_0$, $\sigma - \frac{1}{2}\nabla \cdot \boldsymbol{\beta} \geq 0$ in Ω for some constant $\mu_0 > 0$, whereas $f \in L^2(\Omega)$, $g = \bar{u}_{|\Gamma_D}$ for some $\bar{u} \in \mathrm{H}^1(\Omega)$, and $\psi \in \mathrm{L}^2(\Gamma_N)$.

Define the spaces $U = \mathrm{H}^1(\Omega)$, $V = \mathrm{H}^1_{0,\Gamma_D}(\Omega) := \{v \in U : v_{|\Gamma_D} = 0\}$, the bilinear form $a : U \times V \to \mathbb{R}$ and the linear form $F : V \to \mathbb{R}$ such that

$$a(w, v) = \int_\Omega \mu \nabla w \cdot \nabla v + \boldsymbol{\beta} \cdot \nabla w\, v + \sigma w\, v\,, \qquad F(v) = \int_\Omega f\, v + \int_{\Gamma_N} \psi\, v\,; \tag{2.2}$$

denote by $\alpha \geq \mu_0$ the coercivity constant of the form a, and by $\|a\|$, $\|F\|$ the continuity constants of the forms a and F. Problem (2.1) is formulated variationally as follows: *Find $u \in \bar{u} + V$ such that*

$$a(u, v) = F(v) \qquad \forall v \in V\,. \tag{2.3}$$

We assume that we can represent u in the form

$$u = \bar{u} + \Phi\tilde{u}\,, \tag{2.4}$$

for some (known) smooth function $\Phi \in V$ and some $\tilde{u} \in U$ having the same smoothness of u. Let us introduce the affine mapping

$$B : U \to \bar{u} + V \qquad \text{such that} \quad Bw = \bar{u} + \Phi w \tag{2.5}$$

which enforces the given Dirichlet boundary condition. Then, Problem (2.3) can be equivalently formulated as follows: *Find $\tilde{u} \in U$ such that*

$$a(B\tilde{u}, v) = F(v) \qquad \forall v \in V\,. \tag{2.6}$$

Remark 1 *(Enforcement of the Dirichlet conditions)* The approach we follow to enforce Dirichlet boundary conditions will allow us to deal with a loss function which is built solely by the residuals of the variational equations. Other approaches are obviously possible: for instance, one could augment such loss function by a term penalizing the distance of the numerical solution from the data on Γ_D, or adopt a Nitsche's type variational formulation of the boundary-value problem [18]. Both strategies involve parameters which may need a tuning, whereas in our approach the definition of the loss function is simple and natural, allowing us to focus on the performances of the neural networks.

3 The VPINN-based Numerical Discretization

In this section, we first introduce the class of neural networks used in this paper, then we describe the numerical discretization of the boundary-value problem (2.6), which uses neural networks to represent the discrete solution and piecewise polynomial functions to enforce the variational equations. An inf-sup stable Petrov-Galerkin formulation is introduced which guarantees stability and convergence, as indicated in Sect. 4; this is the main difference between the proposed method and other formulations, such as [11, 14].

3.1 Neural Networks

In this work we only use fully-connected feed-forward neural networks (named also multi-layered perceptrons), therefore the following description is focused on such a class of networks. Since we deal with a scalar equation, a neural network will be a function $w : \mathbb{R}^n \to \mathbb{R}$ defined as follows: for any $\boldsymbol{x} \in \mathbb{R}^n$, the output $w(\boldsymbol{x})$ is computed via the chain of assignments

$$\begin{aligned} &\boldsymbol{x}_0 = \boldsymbol{x}, \\ &\boldsymbol{x}_\ell = \rho(\mathbf{A}_\ell \boldsymbol{x}_{\ell-1} + \mathbf{b}_\ell), \qquad \ell = 1, ..., L-1, \\ &w(\boldsymbol{x}) = \mathbf{A}_L \boldsymbol{x}_{L-1} + b_L\,. \end{aligned} \tag{3.1}$$

Here, $\mathbf{A}_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and $\mathbf{b}_\ell \in \mathbb{R}^{N_\ell}$, $\ell = 1, ..., L$, are matrices and vectors that store the network weights (with $N_0 = n$ and $N_L = 1$); furthermore, L is the number of layers, whereas ρ is the (nonlinear) activation function which acts component-wise (i.e. $\rho(\mathbf{y}) = \left[\rho(y_1), ..., \rho(y_{n_y})\right]$ for any vector $\mathbf{y} \in \mathbb{R}^{n_y}$). It can be noted from Eq. (3.1), that if $\rho \in \mathrm{C}^k(\mathbb{R})$, then w inherits the same regularity because it can be seen as a composition of functions belonging to $\mathrm{C}^k(\mathbb{R})$. Popular choices include the ReLU ($k = 0$) and RePU ($k > 0$ finite) functions, as well as the hyperbolic tangent ($k = \infty$) if one wants to exploit the maximum of regularity in the solution of interest.

The neural network structure $\mathcal{NN}$ is identified by fixing the number of layers L, the integers N_ℓ and the activation function ρ. The entire set of weights that parametrize the network can be logically organized into a single vector $\mathbf{w} \in \mathbb{R}^N$. Thus, the neural network structure $\mathcal{NN}$ induces a mapping

$$\mathcal{F}^{\mathcal{NN}} : \mathbb{R}^N \to \mathrm{C}^\infty(\bar{\Omega}), \qquad \mathbf{w} \mapsto \mathcal{F}^{\mathcal{NN}}(\mathbf{w}) = w, \quad \text{where } w = w(\boldsymbol{x}, \mathbf{w}). \tag{3.2}$$

It is convenient to define the manifold

$$U^{\mathcal{NN}} \subset U, \qquad U^{\mathcal{NN}} = \mathcal{F}^{\mathcal{NN}}(\mathbb{R}^N)$$

containing all functions that can be generated by the neural network structure $\mathcal{NN}$.

3.2 The VPINN Discretization

We aim at approximating the solution of Problem (2.1) by a generalized Petrov-Galerkin strategy. To this end, let us introduce a conforming, shape-regular triangulation $\mathcal{T}_h = \{E\}$ of $\bar{\Omega}$ with meshsize $h > 0$ and, for a fixed integer $k_{\text{test}} \geq 1$, let $V_h \subset V$ be the linear subspace formed by the functions which are piecewise polynomials of degree k_{test} over the triangulation $\mathcal{T}_h$. Furthermore, let us introduce computable approximations of the forms a and F by numerical quadratures. Precisely, for any $E \in \mathcal{T}_h$, let $\{(\xi_\iota^E, \omega_\iota^E) : \iota \in I^E\}$ be the nodes and weights of a quadrature formula of precision

$$q \geq 2k_{\text{test}} \tag{3.3}$$

on E. Assume that Γ_N is the union of a collection $\partial\mathcal{T}_h(\Gamma_N)$ of edges of elements of $\mathcal{T}_h$; for any such edge e, let $\{(\xi_\iota^e, \omega_\iota^e) : \iota \in I^e\}$ be the nodes and weights of a quadrature formula of precision q on e. Then, assuming that all the data μ, $\boldsymbol{\beta}$, σ, f, ψ are continuous in each element of the triangulation, we define the approximate forms

$$a_h(w, v) = \sum_{E \in \mathcal{T}_h} \sum_{\iota \in I^E} [\mu \nabla w \cdot \nabla v + \boldsymbol{\beta} \cdot \nabla w \, v + \sigma w v](\xi_\iota^E)\, \omega_\iota^E, \tag{3.4}$$

$$F_h(v) = \sum_{E \in \mathcal{T}_h} \sum_{\iota \in I^E} [f v](\xi_\iota^E)\, \omega_\iota^E \quad + \sum_{e \in \partial\mathcal{T}_h(\Gamma_N)} \sum_{\iota \in I^e} [\psi v](\xi_\iota^e)\, \omega_\iota^e. \tag{3.5}$$

With these ingredients at hand, we would like to approximate the solution of Problem (2.6) by some $u^{\mathcal{NN}} \in U^{\mathcal{NN}}$ satisfying

$$a_h(Bu^{\mathcal{NN}}, v_h) = F_h(v_h) \qquad \forall v_h \in V_h. \tag{3.6}$$

Such a problem might be ill-posed when, for computational efficiency, the dimension of the test space V_h is chosen smaller than the dimension of the manifold $U^{\mathcal{NN}}$. In this situation, we get an under-determined problem, with obvious difficulties in deriving stability estimates on some norms of the function $Bu^{\mathcal{NN}}$. Actually, Problem (3.6) with zero data (i.e., zero f, g, ψ) could admit non-zero solutions (see Sect. 6.3).

To avoid these difficulties, we adopt the strategy of applying a projection (indeed, an interpolation) to the function $Bu^{\mathcal{NN}}$, mapping it into a finite dimensional space of dimension comparable to that of V_h, and we limit ourselves with estimating some norm of this projection.

To be precise, let us introduce a conforming, shape-regular partition $\mathcal{T}_H = \{G\}$ of $\bar{\Omega}$, which is equal to or coarser than $\mathcal{T}_h$ (i.e., each element $E \in \mathcal{T}_h$ is contained in an element $G \in \mathcal{T}_H$) but compatible with $\mathcal{T}_h$ (i.e., its meshsize $H > 0$ satisfies $H \lesssim h$). Let the integer $k_{\text{int}} \geq 1$ be defined by the condition

$$k_{\text{int}} + k_{\text{test}} = q + 2\,. \tag{3.7}$$

Let $U_H \subset U$ be the linear subspace formed by the functions which are piecewise polynomials of degree k_{int} over the triangulation $\mathcal{T}_H$, and let $U_{H,0} = U_H \cap V$ be the subspace of U_H formed by the functions vanishing on Γ_D. Finally, let $\mathcal{I}_H : \mathrm{C}^0(\bar{\Omega}) \to U_H$ be an interpolation operator, satisfying the condition $\mathcal{I}_H : \mathrm{C}^0(\bar{\Omega}) \cap V \to U_{H,0}$ as well as the following approximation properties: for all $v \in \mathrm{H}^{k+1}(\Omega)$, $1 \leq k \leq k_{\text{int}}$,

$$|v - \mathcal{I}_H v|_{\ell,G} \lesssim H^{k+1-\ell} |v|_{k+1,G}\,, \qquad 0 \leq \ell \leq k+1\,, \quad \forall G \in \mathcal{T}_H\,. \tag{3.8}$$

In this framework, assuming the lifting $\bar{u}$ to be continuous in $\bar{\Omega}$, we replace the target Eq. (3.6) by the following ones:

$$a_h(\mathcal{I}_H Bu^{\mathcal{NN}}, v_h) = F_h(v_h) \qquad \forall v_h \in V_h\,. \tag{3.9}$$

In order to handle this problem with the neural network, let us introduce a basis in V_h, say $V_h = \text{span}\{\varphi_i : i \in I_h\}$, and for any w smooth enough let us define the residuals

$$r_{h,i}(w) = F_h(\varphi_i) - a_h(\mathcal{I}_H Bw, \varphi_i)\,, \qquad i \in I_h\,, \tag{3.10}$$

as well as the loss function

$$R_h^2(w) = \sum_{i \in I_h} r_{h,i}^2(w)\, \gamma_i^{-1}\,, \tag{3.11}$$

where $\gamma_i > 0$ are suitable weights. Then, we search for a global minimum of the loss function in $U^{\mathcal{NN}}$, i.e., we consider the following discretization of Problem (2.6): *Find* $u^{\mathcal{NN}} \in U^{\mathcal{NN}}$ *such that*

$$u^{\mathcal{NN}} \in \arg\min_{w \in U^{\mathcal{NN}}} R_h^2(w)\,. \tag{3.12}$$

Note that the solution $u^{\mathcal{NN}}$ may not be unique; however, a suitable choice of the space U_H may lead to the control of the error $u - \mathcal{I}_H Bu^{\mathcal{NN}}$ in the H^1-norm, as we will see in the sequel.

Remark 2 *(Discretization without interpolation)* For the sake of comparison, we will also consider the optimization problem in which no interpolation is applied to the neural network functions. In other words, the target equations are those in (3.6), which induce the following definition of loss function

$$\hat{R}_h^2(w) = \sum_{i \in I_h} \hat{r}_{h,i}^2(w)\, \gamma_i^{-1}\,, \qquad \text{with} \quad \hat{r}_{h,i}(w) = F_h(\varphi_i) - a_h(Bw, \varphi_i)\,, \tag{3.13}$$

and the following minimization problem: *Find* $\hat{u}^{\mathcal{NN}} \in U^{\mathcal{NN}}$ *such that*

$$\hat{u}^{\mathcal{NN}} \in \arg\min_{w \in U^{\mathcal{NN}}} \hat{R}_h^2(w)\,. \tag{3.14}$$

Note that in this problem the triangulation $\mathcal{T}_H$ and the space U_H play no role. Although we will not provide a rigorous error analysis for such discretization, it will be interesting to numerically compare the behaviour of the approaches (i.e., with or without interpolation). This will be done in Sect. 6.

4 A Priori Error Estimates

Let $u^{\mathcal{NN}} \in U^{\mathcal{NN}}$ be any solution of the minimization problem (3.12); let us set

$$u_H^{\mathcal{NN}} = \mathcal{I}_H B u^{\mathcal{NN}} \in U_H \,. \tag{4.1}$$

Recalling the definition (2.4) of the affine mapping B, it holds

$$u_H^{\mathcal{NN}} = \bar{u}_H + u_H^{\mathcal{NN},0} \,, \quad \text{with} \quad \bar{u}_H = \mathcal{I}_H \bar{u} \quad \text{and} \quad u_{H,0}^{\mathcal{NN}} = \mathcal{I}_H(\Phi u^{\mathcal{NN}}) \in U_{H,0} \,; \tag{4.2}$$

note that $\bar{u}_H$ is a discrete lifting in U_H of the Dirichlet data g.

We aim at estimating the error between u and $u_H^{\mathcal{NN}}$. To accomplish this task, we need several definitions, assumptions, and technical results.

Definition 1 *(norm-equivalence)* Let us denote by $0 < c_h \leq C_h$ the constants in the norm equivalence

$$c_h \|v_h\|_{1,\Omega} \leq \|\boldsymbol{v}\|_\gamma \leq C_h \|v_h\|_{1,\Omega} \qquad \forall v_h \in V_h \,, \tag{4.3}$$

where $\boldsymbol{v} = (v_i)_{i \in I_h}$ is such that $v_h = \sum_{i \in I_h} v_i \varphi_i$, and $\|\boldsymbol{v}\|_\gamma = \left(\sum_{i \in I_h} v_i^2 \gamma_i\right)^{1/2}$.

Next, we introduce the consistency errors due to numerical quadratures

$$E_h^a(w_H, v_h) = a(w_H, v_h) - a_h(w_H, v_h) \qquad \forall w_H \in U_H,\ \forall v_h \in V_h \,, \tag{4.4}$$

$$E_h^F(v_h) = F(v_h) - F_h(v_h) \qquad \forall v_h \in V_h \,, \tag{4.5}$$

and we provide a bound on these errors. To this end, let us assume that the quadrature rules used in the elements in $\mathcal{T}_h$ are obtained by affine transformations from a quadrature rule $\{(\hat{\xi}_\iota, \hat{\omega}_\iota) : \iota \in \hat{I}\}$ on a reference element $\hat{E} \subset \mathbb{R}^n$; similarly, let us assume that the quadrature rules used in the edges on $\partial\mathcal{T}_h(\Gamma_N)$ are obtained by affine transformations from a quadrature rule $\{(\check{\xi}_\iota, \check{\omega}_\iota) : \iota \in \check{I}\}$ on a reference element $\check{e} \subset \mathbb{R}^{n-1}$.

Assumption 1 *(Data smoothness)* Let us assume the following smoothness of data:

$$\mu,\ \sigma,\ f \in \mathrm{W}^{k,\infty}(\Omega) \,, \qquad \boldsymbol{\beta} \in (\mathrm{W}^{k,\infty}(\Omega))^n \,, \qquad \psi \in \mathrm{W}^{k,\infty}(\Gamma_N) \,, \tag{4.6}$$

where k is an integer satisfying

$$1 \leq k \leq k_{\text{int}} = q + 2 - k_{\text{test}} \,. \tag{4.7}$$

Consequently, let us introduce the following notation

$$\mathcal{N}_k(\mu, \boldsymbol{\beta}, \sigma) = \|\mu\|_{\mathrm{W}^{k,\infty}(\Omega)} + \|\boldsymbol{\beta}\|_{(\mathrm{W}^{k,\infty}(\Omega))^n} + \|\sigma\|_{\mathrm{W}^{k,\infty}(\Omega)} \,, \tag{4.8}$$

$$\mathcal{N}_k(f, \psi) = \|f\|_{\mathrm{W}^{k,\infty}(\Omega)} + \|\psi\|_{\mathrm{W}^{k,\infty}(\Gamma_N)} \,, \tag{4.9}$$

$$\|w_H\|_{k,\mathcal{T}_H} = \left(\sum_{G \in \mathcal{T}_H} \|w_{H|G}\|_{\mathrm{H}^k(G)} \right)^{1/2} \qquad \forall w_H \in U_H \,. \tag{4.10}$$

Property 2 (approximation of the forms a and F) *Under Assumption 1, it holds*

$$|E_h^a(w_H, v_h)| \lesssim h^k \mathcal{N}_k(\mu, \boldsymbol{\beta}, \sigma)\|w_H\|_{k,\mathcal{T}_H}\|v_h\|_{1,\Omega} \qquad \forall w_H \in U_H,\ \forall v_h \in V_h\,, \tag{4.11}$$

$$|E_h^F(v_h)| \lesssim h^k \mathcal{N}_k(f, \psi)\|v_h\|_{1,\Omega} \qquad \forall v_h \in V_h\,, \tag{4.12}$$

Proof Both estimates are classical in the theory of finite elements (see, e.g., [19]). As far as (4.11) is concerned, the standard proof given for the case in which the polynomial degree is the same for both arguments, i.e., $k = k_{\text{test}} \geq 1$ and $q = 2(k-1)$, can be easily adapted to the present situation $k + k_{\text{test}} \leq q + 2$. In this way, one gets $|E_h^a(w_H, v_h)| \lesssim h^k \mathcal{N}_k(\mu, \boldsymbol{\beta}, \sigma)\|w_H\|_{k,\mathcal{T}_h}\|v_h\|_{1,\Omega}$, and one concludes by observing that $\|w_H\|_{k,\mathcal{T}_h} = \|w_H\|_{k,\mathcal{T}_H}$ since $\mathcal{T}_h$ is a refinement of $\mathcal{T}_H$. □

Finally, we pose a fundamental assumption.

Assumption 3 *(inf-sup condition between $U_{H,0}$ and V_h)* The bilinear form a satisfies an inf-sup condition with respect to the spaces $U_{H,0}$ and V_h, namely there exists a constant $\alpha_\star > 0$, independent of the meshsizes h and H, such that

$$\alpha_\star \|w_H\|_{1,\Omega} \leq \sup_{v_h \in V_h} \frac{a(w_H, v_h)}{\|v_h\|_{1,\Omega}} \qquad \forall w_H \in U_{H,0}\,. \tag{4.13}$$

This assumption together with Property 2 yields the following result.

Proposition 4 (discrete inf-sup condition between $U_{H,0}$ and V_h) *Under Assumptions 1 and 3, for all $h \leq h_0$ small enough the bilinear form a_h satisfies an inf-sup condition with respect to the spaces $U_{H,0}$ and V_h, namely there exists a constant $\tilde{\alpha}_\star > 0$ such that*

$$\tilde{\alpha}_\star \|w_H\|_{1,\Omega} \leq \sup_{v_h \in V_h} \frac{a_h(w_H, v_h)}{\|v_h\|_{1,\Omega}} \qquad \forall w_H \in U_{H,0}\,. \tag{4.14}$$

Proof We have $a_h(w_H, v_h) = a(w_H, v_h) - E_h^a(w_H, v_h)$. Using the bound (4.11) with $k = 1$ and observing that $\|w_H\|_{1,\mathcal{T}_H} = \|w_H\|_{1,\Omega}$, one can find $h_0 > 0$ small enough such that, for all $h \leq h_0$, $|E_h^a(w_H, v_h)| \leq \frac{1}{2}\alpha_\star \|w_H\|_{1,\Omega}\|v_h\|_{1,\Omega}$, whence the result with $\tilde{\alpha}_\star = \frac{1}{2}\alpha_\star$ □

We are ready to estimate the error $\|u - u_H^{\mathcal{NN}}\|_{1,\Omega}$. Recalling the decomposition (4.2), we use the triangle inequality

$$\|u - u_H^{\mathcal{NN}}\|_{1,\Omega} \leq \|u - u_H\|_{1,\Omega} + \|u_H - u_H^{\mathcal{NN}}\|_{1,\Omega}\,, \tag{4.15}$$

where u_H is a suitable element in the affine subspace $\bar{u}_H + U_{H,0} \subset U_H$. Writing $u_H = \bar{u}_H + u_{H,0}$ with $u_{H,0} \in U_{H,0}$, one has $u_H - u_H^{\mathcal{NN}} = u_{H,0} - u_{H,0}^{\mathcal{NN}} \in U_{H,0}$; hence, we can apply (4.14) to get

$$\|u_H - u_H^{\mathcal{NN}}\|_{1,\Omega} \leq \frac{1}{\tilde{\alpha}_\star} \sup_{v_h \in V_h} \frac{a_h(u_H - u_H^{\mathcal{NN}}, v_h)}{\|v_h\|_{1,\Omega}}\,. \tag{4.16}$$

Recalling the definitions (4.4) and (4.5), it holds

$$\begin{aligned}
a_h(u_H, v_h) &= a(u_H, v_h) - E_h^a(u_H, v_h) \\
&= a(u, v_h) - a(u - u_H, v_h) - E_h^a(u_H, v_h) \\
&= F(v_h) - a(u - u_H, v_h) - E_h^a(u_H, v_h) \\
&= F_h(v_h) + E_h^F(v_h) - a(u - u_H, v_h) - E_h^a(u_H, v_h)\,.
\end{aligned}$$

Thus, the numerator in (4.16) is given by

$$a_h(u_H - u_H^{\mathcal{NN}}, v_h) = F_h(v_h) - a_h(u_H^{\mathcal{NN}}, v_h) - a(u - u_H, v_h) - E_h^a(u_H, v_h) + E_h^F(v_h)\,.$$

On the other hand, recalling (3.10) we have

$$F_h(v_h) - a_h(u_H^{\mathcal{NN}}, v_h) = F_h(v_h) - a_h(\mathcal{I}_H B u^{\mathcal{NN}}, v_h) = \sum_{i\in I_h} r_{h,i}(u^{\mathcal{NN}})\, v_i\,, \tag{4.17}$$

hence, by (3.11) and (4.3),

$$|F_h(v_h) - a_h(u_H^{\mathcal{NN}}, v_h)| \le R_h(u^{\mathcal{NN}})\|\boldsymbol{v}\|_\gamma \le C_h R_h(u^{\mathcal{NN}})\|v_h\|_{1,\Omega}\,.$$

Using the bounds (4.11) and (4.12), we obtain the following inequality

$$\begin{aligned}\|u - u_H^{\mathcal{NN}}\|_{1,\Omega} \lesssim \left(1 + \frac{1}{\tilde{\alpha}_\star}\right)\Bigg(\inf_{u_H\in\bar{u}_H+U_{H,0}} \left(\|u - u_H\|_{1,\Omega} + h^k\mathcal{N}_k(\mu,\boldsymbol{\beta},\sigma)\|u_H\|_{k,\mathcal{T}_H}\right) \\ + C_h R_h(u^{\mathcal{NN}}) + h^k\mathcal{N}_k(f,\psi)\Bigg)\,.\end{aligned} \tag{4.18}$$

From now on, we assume that $u \in \mathrm{H}^{k+1}(\Omega)$. Then, assumption (3.8) yields the inequalities

$$\|u - \mathcal{I}_H u\|_{1,\Omega} \lesssim H^k|u|_{k+1,\Omega} \lesssim h^k|u|_{k+1,\Omega} \tag{4.19}$$

and

$$\|\mathcal{I}_H u\|_{k,\mathcal{T}_H} \le \|u\|_{k,\Omega} + \|u - \mathcal{I}_H u\|_{k,\mathcal{T}_H} \lesssim \|u\|_{k,\Omega} + H|u|_{k+1,\Omega} \lesssim \|u\|_{k+1,\Omega}\,. \tag{4.20}$$

Choosing $u_H = \mathcal{I}_H u \in \bar{u}_H + U_{H,0}$ in (4.18) and using these estimates, we arrive at the following intermediate result, which can be viewed as a mixed a priori/a posteriori error estimate.

Lemma 4.1 *Under the previous assumptions, it holds*

$$\|u - u_H^{\mathcal{NN}}\|_{1,\Omega} \lesssim h^k(|u|_{k+1,G} + \mathcal{N}_k(\mu,\boldsymbol{\beta},\sigma)\|u\|_{k+1,\Omega} + \mathcal{N}_k(f,\psi)) + C_h R_h(u^{\mathcal{NN}})\,.$$

Our next task will be bounding the term $R_h(u^{\mathcal{NN}})$. To this end, we use the minimality condition (3.12) to get

$$R_h(u^{\mathcal{NN}}) \le R_h(w^{\mathcal{NN}}) \qquad \forall w^{\mathcal{NN}} \in U^{\mathcal{NN}}\,. \tag{4.21}$$

On the other hand, since $R_h(w^{\mathcal{NN}})$ is a weighted ℓ_2-norm in $\mathbb{R}^{|I_h|}$, we can write

$$R_h(w^{\mathcal{NN}}) = \sup_{\mathbf{z}\in\mathbb{R}^{|I_h|}} \frac{1}{\|\mathbf{z}\|_\gamma}\sum_{i\in I_h} r_{h,i}(w^{\mathcal{NN}})z_i\,,$$

where, similarly to (4.17),

$$\sum_{i\in I_h} r_{h,i}(w^{\mathcal{NN}})z_i = F_h(z_h) - a_h(\mathcal{I}_H B w^{\mathcal{NN}}, z_h) \qquad \text{with } z_h = \sum_{i\in I_h} z_i\varphi_i \in V_h\,.$$

For convenience, in analogy with (4.1), let us set

$$w_H^{\mathcal{NN}} = \mathcal{I}_H B w^{\mathcal{NN}} \in U_H\,. \tag{4.22}$$

Thus, recalling (4.3), we obtain

$$R_h(w^{\mathcal{NN}}) \le \frac{1}{c_h}\sup_{z_h\in V_h}\frac{F_h(z_h) - a_h(w_H^{\mathcal{NN}}, z_h)}{\|z_h\|_{1,\Omega}} \qquad \forall w^{\mathcal{NN}} \in U^{\mathcal{NN}}\,. \tag{4.23}$$

The numerator can be manipulated as above, using

$$F_h(z_h) = F(z_h) - E_h^F(z_h) = a(u, z_h) - E_h^F(z_h)$$

and

$$a_h(w_H^{\mathcal{NN}}, z_h) = a(w_H^{\mathcal{NN}}, z_h) - E_h^a(w_H^{\mathcal{NN}}, z_h)\,,$$

whence, using once more Property 2, we get

$$R_h(w^{\mathcal{NN}}) \lesssim \frac{1}{c_h}\left(\|u - w_H^{\mathcal{NN}}\|_{1,\Omega} + h^k\mathcal{N}_k(\mu, \boldsymbol{\beta}, \sigma)\|w_H^{\mathcal{NN}}\|_{k,\mathcal{T}_H} + h^k\mathcal{N}_k(f, \psi)\right)\,. \tag{4.24}$$

In order to bound the terms containing $w_H^{\mathcal{NN}}$, we introduce the quantity

$$e^{\mathcal{NN}} = u - Bw^{\mathcal{NN}}\,, \tag{4.25}$$

which, recalling the definitions (2.4) and (2.5), can be written as

$$e^{\mathcal{NN}} = \Phi(\tilde{u} - w^{\mathcal{NN}})\,, \tag{4.26}$$

and we formulate a final assumption.

Assumption 5 *(smoothness of the solution and the neural network manifold)* The solution u can be represented as in (2.4) with

$$\tilde{u} \in \mathrm{H}^{k+1}(\Omega) \qquad \text{and} \qquad \Phi \in \mathrm{W}^{k+1,\infty}(\Omega) \tag{4.27}$$

for k satisfying (4.7). Furthermore, the manifold formed by the neural network functions satisfies the smoothness condition

$$U^{\mathcal{NN}} \subset \mathrm{H}^2(\Omega)\,. \tag{4.28}$$

Note that (4.27) implies in particular $u \in \mathrm{H}^{k+1}(\Omega)$ with the bound $\|u\|_{k+1,\Omega} \lesssim \|\tilde{u}\|_{k+1,\Omega}\,\|\Phi\|_{k+1,\infty,\Omega}$; on the other hand, (4.28) implies $e^{\mathcal{NN}} \in \mathrm{H}^2(\Omega)$. (We refer to Remark 4 for another set of assumptions on the neural network.)

Recalling (4.22) and using the identity

$$u - w_H^{\mathcal{NN}} = (u - \mathcal{I}_H u) + \mathcal{I}_H e^{\mathcal{NN}} = (u - \mathcal{I}_H u) - e^{\mathcal{NN}} + (I - \mathcal{I}_H)e^{\mathcal{NN}}\,, \tag{4.29}$$

we can write

$$\|u - w_H^{\mathcal{NN}}\|_{1,\Omega} \lesssim \|u - \mathcal{I}_H u\|_{1,\Omega} + \|e^{\mathcal{NN}}\|_{1,\Omega} + H|e^{\mathcal{NN}}|_{2,\Omega} \tag{4.30}$$

and, using a standard inverse inequality in $\mathbb{P}_k(G)$ for any $G \in \mathcal{T}_H$,

$$\begin{aligned}\|w_H^{\mathcal{NN}}\|_{k,\mathcal{T}_H} &\lesssim \|\mathcal{I}_H u\|_{k,\mathcal{T}_H} + H^{1-k}\|\mathcal{I}_H e^{\mathcal{NN}}\|_{1,\Omega} \\ &\lesssim \|\mathcal{I}_H u\|_{k,\mathcal{T}_H} + H^{1-k}\left(\|e^{\mathcal{NN}}\|_{1,\Omega} + H|e^{\mathcal{NN}}|_{2,\Omega}\right)\,.\end{aligned} \tag{4.31}$$

Keeping into account (4.19) and (4.20), in order to conclude we need to identify a function $\tilde{w}^{\mathcal{NN}} \in U^{\mathcal{NN}}$ for which a bound of the type

$$|e^{\mathcal{NN}}|_{m,\Omega} \lesssim |\tilde{u} - \tilde{w}^{\mathcal{NN}}|_{m,\Omega} \lesssim H^{k+1-m}|\tilde{u}|_{k+1,\Omega} \tag{4.32}$$

holds true for $m = 1, 2$. The existence of such a function is guaranteed by one of the available results on the approximation of functions in Sobolev spaces by neural networks (see [20, Theorem 5.1, Remark 5.2]; see also [9]), provided the number of layers L and the widths of the layers in the chosen $\mathcal{NN}$ satisfy suitable conditions depending on the target accuracy (hence, in our case depending on H^k). Indeed, suppose one is interested in using meshes with meshsize as small as $H_{\min}$ in the domain Ω (here assumed to satisfy $\Omega \subset [0,1]^n$ for the sake of simplicity), and let $N \in \mathbb{N}$ be such that

$$3^n(1+\delta)(2(m+1))^{3m}\max\{R^m, \ln^m(\beta N^{k+n+3})\}\frac{C(n,m,k,\tilde{u})}{N^{k+1-m}} \leq H_{\min}^{k+1-m}|\tilde{u}|_{k+1,\Omega}, \quad (4.33)$$

where δ, R, β and C are constants not depending on N defined in [20]. Then, a function $\tilde{w}^{\mathcal{NN}}$ exists which fulfils (4.32) and is represented as a neural network with the hyperbolic tangent as activation function and two hidden layers with N_1 and N_2 neurons respectively, satisfying

$$N_1 \leq 3\left\lceil \frac{k+1}{2} \right\rceil \left|P_{k,n+1}\right| + n(N-1), \quad N_2 \leq 3\left\lceil \frac{n+2}{2} \right\rceil \left|P_{n+1,n+1}\right| N^n, \quad (4.34)$$

where

$$\left|P_{a,b}\right| = \binom{a+b-1}{a}, \qquad \forall a,b \in \mathbb{N},\ b \geq 2.$$

Substituting (4.32) into (4.30) and (4.31), and using inequalities (4.21) and (4.24), we arrive at the following bound on the loss $R_h(u^{\mathcal{NN}})$.

Lemma 4.2 *Under the previous assumptions, it holds*

$$R_h(u^{\mathcal{NN}}) \lesssim \frac{1}{c_h}\left(H^k|u|_{k+1,\Omega} + H^k|\tilde{u}|_{k+1,\Omega} + h^k\mathcal{N}_k(\mu,\boldsymbol{\beta},\sigma)\|\tilde{u}\|_{k+1,\Omega} + h^k\mathcal{N}_k(f,\psi)\right).$$

We remark that such a bound, when the involved neural network is comprised of at least two hidden layers and is such that there exists N satisfying both (4.33) and (4.34), does not depend on the network hyperparameters.

Concatenating Lemmas 4.1 and 4.2, and using once more $H \lesssim h$, we obtain the following a priori error estimate for the solution of Problem (3.12).

Theorem 6 (a priori error estimate) *Let* $u_H^{\mathcal{NN}} \in U_H$ *be defined by* (4.1). *Under Assumptions 1, 3 and 5, for h sufficiently small it holds*

$$\|u - u_H^{\mathcal{NN}}\|_{1,\Omega} \lesssim \left(1 + \frac{C_h}{c_h}\right)h^k\left[(1+\mathcal{N}_k(\mu,\boldsymbol{\beta},\sigma))\|\tilde{u}\|_{k+1,\Omega} + \mathcal{N}_k(f,\psi)\right]. \quad (4.35)$$

Remark 3 *(on the equivalence constants c_h, C_h)* If a classical Lagrange basis is used in (3.10), and the triangulation $\mathcal{T}_h$ is quasi-uniform, then for constants weights $\gamma_i = 1$ one has $c_h \simeq h^{1-d/2}$ and $C_h \simeq h^{-d/2}$, whence $\frac{C_h}{c_h} \simeq h^{-1}$. On the other hand, if a hierarchical basis is used instead, then $c_h \simeq C_h \simeq 1$, hence, $\frac{C_h}{c_h} \simeq 1$ in dimension $d = 1$, whereas $c_h \simeq |\log h|^{-1}$, $C_h \simeq 1$, hence, $\frac{C_h}{c_h} \simeq |\log h|$ in dimension $d = 2$.

Thus, the presence of the ratio $\frac{C_h}{c_h}$ in (4.35), which originates from the control of the loss function, makes this estimate sub-optimal. However, our numerical experiments in Sect. 6.1 indicate that this adverse effect is not seen in practice. The reason may be related to the decay of the loss function $R_h(u^{\mathcal{NN}})$, which is significantly faster than the decay of the approximation error when h is reduced, thereby compensating for the growth of ratio. See Remark 5.

Remark 4 *(low-regularity $\mathcal{NN}$)* When the condition $U^{\mathcal{NN}} \subset \mathrm{H}^2(\Omega)$ fails to be satisfied, as for the ReLU activation function, we may provide a different set of assumptions which still lead to an $O(h^k)$-error estimate as in Theorem 6. Precisely, we may assume that $\tilde{u} \in \mathrm{W}^{k+1,\infty}(\Omega)$ and $U^{\mathcal{NN}} \subset \mathrm{W}^{1,\infty}(\Omega)$. Then, referring to the first equality in (4.29), one has

$$\begin{aligned} \|u - w_H^{\mathcal{NN}}\|_{1,\Omega} &\leq \|(u - \mathcal{I}_H u)\|_{1,\Omega} + \|\mathcal{I}_H e^{\mathcal{NN}}\|_{1,\Omega} \\ &\lesssim \|(u - \mathcal{I}_H u)\|_{1,\Omega} + H^{-1}\|\mathcal{I}_H e^{\mathcal{NN}}\|_{0,\Omega}\,, \end{aligned}$$

with

$$\begin{aligned} \|\mathcal{I}_H e^{\mathcal{NN}}\|_{0,\Omega}^2 = \sum_{G\in\mathcal{T}_H} \|\mathcal{I}_H e^{\mathcal{NN}}\|_{0,G}^2 &\leq \sum_{G\in\mathcal{T}_H} \|\mathcal{I}_H e^{\mathcal{NN}}\|_{\mathrm{L}^\infty(G)}^2 |G| \\ &\lesssim \sum_{G\in\mathcal{T}_H} \|e^{\mathcal{NN}}\|_{\mathrm{L}^\infty(G)}^2 |G| \lesssim H^d \|e^{\mathcal{NN}}\|_{\mathrm{L}^\infty(\Omega)}^2 . \end{aligned}$$

The conclusion easily follows if $\tilde{w}^{\mathcal{NN}}$ is chosen to satisfy the error bound $\|\tilde{u}-\tilde{w}^{\mathcal{NN}}\|_{\mathrm{L}^\infty(\Omega)} \lesssim H^{k+1}|\tilde{u}|_{\mathrm{W}^{k+1,\infty}(\Omega)}$, which is possible according to the results in [6, 7].

5 Implementation Issues

As specified in Sect. 3.1, we use a fully-connected feed-forward neural network architecture, which is fixed and depends neither on the PDE nor on its discretization. For each simulation, we initialize the neural network with a completely new set of weights, which is important to show that our results are not initialization dependent. The activation function is the hyperbolic tangent. It has been proven in [20] that such neural networks with two hidden layers enjoy exponential converge properties with respect to the number of weights. Nevertheless, in order to simplify the training and enrich the space in which we seek the numerical solution, we consider five layers (namely, $L = 5$ with the notation of Sect. 3.1) with 50 neurons each ($N_\ell = 50$, $\ell = 1, ..., 5$). We also highlight that it is always possible to approximate the identity function with a neural network with a single layer with just one neuron. The best approximation obtainable with a neural network with more than two layers is thus more accurate than the one computable with a neural network with just two layers, because, in the worst possible case, the L-layers neural network can be obtained by combining an $(L-2)$-layers identity neural network with a suitable 2-layers neural network. Numerical tests have been performed to investigate the influence of the activation function on the model accuracy; we observed that all the commonly used activation functions led to equivalent results, thus we omit such a comparison from the present work. We remark that the hyperparameters $L = 5$ and $N_\ell = 50$ have been chosen to obtain a neural network sufficiently large to satisfy condition (4.32) on all grids used in our experiments. Numerical evidence that the neural network best approximation error is negligible when compared with other sources of error is presented in Sect. 6.2.

In order to compute the results shown in Sect. 6, we compared various state of the art optimizers to find the most efficient way to minimize the loss function. We observe that most of the momentum-based first-order methods have similar performances (the presented results are computed with the ADAM optimizer [21]), but it is convenient to use a learning rate scheduler in order to reduce the learning rate during the process. We tested both cyclical learning rate schedulers and exponential learning rate schedulers [22], the differences were very subtle and we thus choose to adopt the most common exponential learning rate scheduler and decided not to report images about such a comparison. To further reduce the loss function we also use the BFGS method and its limited-memory version: the L-BFGS method [23].

The Dirichlet boundary conditions are imposed via the mapping B defined in (2.5). The construction of the function Φ is particularly simple when Ω is a convex polygon, since in this case Φ can be defined as the product of the linear polynomials which vanish on each Dirichlet edge; this is precisely how we define Φ in the numerical examples discussed in the next session. In other geometries, one can build Φ either as described in [17], or by using a level-set method, or even by training an auxiliary neural network to (approximately) vanish on Γ_D. Similarly, in order to obtain an analytical expression of the extension $\overline{u}$ of the Dirichlet data g, one can train another neural network to (approximately) match the values of g on Γ_D or use a data transfinite interpolation [24].

5.1 VPINN Efficiency and the Inf-sup Condition

In Sect. 3 we introduced a discretization method in which the loss function is built by a piecewise polynomial interpolation of the neural network; on the other hand, we also mentioned in Remark 2 the possibility of building the loss function directly from the (non-interpolated) neural network.

From the theoretical point of view, only the former approach can be considered mathematically reliable, since the error control is based on the validity of an inf-sup condition, as detailed in Sect. 4. On the contrary, if the neural network is used without interpolation, one usually gets an under-determined system, for which the error control may be problematic. In fact, for instance, the discrete solution with zero data may not be identically zero, as documented in Sect. 6.3, which rules out uniqueness. Nonetheless, there is empirical evidence (see, e.g., [25–28]) that non-interpolated neural networks do succeed in computing accurate solutions even in complex scenarios. Actually, in the next section we will provide numerical evidence that the two approaches are always (in the considered cases) comparable in terms of rate of convergence and, when the solution is regular, smaller errors are obtained minimizing the same loss function without interpolation.

From the computational point of view, the two approaches have comparable advantages and disadvantages. Let us first consider non-interpolated VPINNs. The corresponding loss functions can be more easily implemented thanks to the existing deep-learning frameworks, which allow the direct computation of neural network derivatives via automatic differentiation [29]. One only needs to generate a mesh and the corresponding test functions, associate a quadrature rule with each element and assemble all the tensors required to efficiently compute the loss function. The main difference with the interpolated neural network approach is that, in the latter, the interpolation matrices have to be assembled too (see “Appendix A.1” for a detailed description of the construction of the interpolation operators), while automatic differentiation is not required. Depending on the problem at hand, this may be an advantage or not. Indeed, the interpolation matrices assembly may be tricky but, using them, all derivatives can be efficiently computed by matrix-vector multiplications that are much cheaper than the entire automatic differentiation procedure, especially when higher order derivatives are required. Therefore, for fast-converging optimization processes, a non-interpolated neural network approach may be efficient and can be more easily implemented; otherwise, each optimization step may be much more expensive than the analogous operation performed with an interpolated neural network. Furthermore, we observed that the training phase is faster when the neural network is interpolated because the procedure converges in fewer steps. This is probably related to the fact that the solution is sought in a significantly smaller space, that can be more easily explored during the training phase.

6 Numerical Results

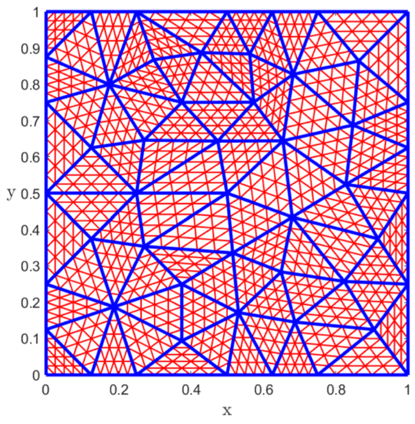
In this section we present several numerical results concerning the VPINN discretization of Problem (2.1) in the square $\Omega = (0, 1)^2$. We will vary the coefficients of the operator, the boundary conditions and the smoothness of the exact solution. For each test case, we vary the degree k_{test} of the test functions, the order q of the quadrature rule and, correspondingly, we choose the polynomial degree k of the interpolating functions as $k = k_{\text{int}} = q + 2 - k_{\text{test}}$, according to (4.7). We only report results obtained with Gaussian rules, as Newton-Cotes formulas of the same order give comparable errors (see [30] for a larger set of numerical experiments about this and other comparisons).

The theoretical results in Sect. 4 suggest that it is convenient to maintain k_{test} as low as possible; consequently, we only use piecewise linear ($k_{\text{test}} = 1$) or piecewise quadratic ($k_{\text{test}} = 2$) test functions. Recalling condition (3.3), we thus choose $q = 3$ or $q = 5$ if $k_{\text{test}} = 1$, and $q = 5$ if $k_{\text{test}} = 2$.

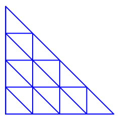
The triangulations we use are generic Delaunay triangular meshes. In order to satisfy the discrete inf-sup condition, we choose $\mathcal{T}_H$ and $\mathcal{T}_h$ as nested meshes whose meshsizes satisfy $H = k_{\text{int}}h$. A pair $(\mathcal{T}_H, \mathcal{T}_h)$ of used meshes is represented in Fig. 1, together with the elemental refinement corresponding to $k_{\text{int}} = 4$, $k_{\text{int}} = 5$ and $k_{\text{int}} = 6$.

6.1 Error Decays

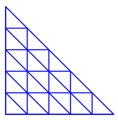
Hereafter, we empirically confirm, with numerical experiments, the a priori error estimate established in Sect. 4. We also compare the behavior of the proposed NN with that of other NNs defined by different strategies. In the following, we denote the interpolated VPINN as



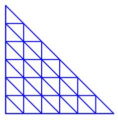
a Delaunay meshes $\mathcal{T}_H$ (blue lines) and $\mathcal{T}_h$ (red lines). Refinement corresponding to $k_{int} = 5$.



b Elemental refinement for $k_{int} = 4$.



c Elemental refinement for $k_{int} = 5$.



d Elemental refinement for $k_{int} = 6$.

Fig. 1 One of the meshes used in Sect. 6.1 and elemental refinements chosen to obtain $\mathcal{T}_h$ from $\mathcal{T}_H$

IVPINN to distinguish it from the non-interpolated VPINN [14], simply denoted as VPINN, and the standard PINN [1].

In the subsequent plots, we report by blue dots the error $\|u - u_H^{\mathcal{NN}}\|_{1,\Omega}$, where $u_H^{\mathcal{NN}}$ is the interpolated VPINN defined on the mesh $\mathcal{T}_H$ as in (4.1), versus the size H of the mesh $\mathcal{T}_H$. We also show a blue solid line and a blue dashed one: the former is the regression line fitting the blue dots (possibly ignoring the first ones); its slope in the log-log plane yields the empirical convergence rate. The latter is used as a reference, since its slope corresponds to an error decay proportional to $h^{k_{\text{int}}}$, which is the expected convergence rate of the H^1 error as indicated by Theorem 6, assuming that the ratio $\frac{C_h}{c_h}$ may be neglected (see Remark 5). The dashed line represents the best convergence rate we can expect from the proposed discretization scheme.

For comparison, we also report by green dots the error $\|u - \hat{u}^{\mathcal{NN}}\|_{1,\Omega}$, where $\hat{u}^{\mathcal{NN}}$ is the non-interpolated VPINN defined in Remark 2, and by red dots the error $\|u - \widetilde{u}^{\mathcal{NN}}\|_{1,\Omega}$, where $\widetilde{u}^{\mathcal{NN}}$ is the standard PINN proposed in [1], with the same architecture of the used VPINNs and the loss function computed as described in [31]. To obtain a fair comparison, the regularization coefficient and the ratio between the control points inside the domain and the ones on the boundary are chosen as described in [31]. Since we are interested in convergence rates with respect to mesh refinement, but the PINN does not require any mesh, the corresponding errors are computed by training the network with the same numbers of inputs used during the training of $u_H^{\mathcal{NN}}$; to be precise, the PINN is trained using the same number of collocation points as the number of interpolation nodes used by the interpolated VPINN.

Furthermore, in order to better analyze the trade-off between the model accuracy and the training efficiency and complexity, we plot the same errors versus the dataset size. Whenever these dots, possibly after a pre-asymptotic phase, sit close to their regression line, we draw it as well in green or red, respectively.

Convergence test #1: $u \in C^\infty(\bar{\Omega})$

Consider problem (2.1) with $\Gamma_D = \{(x, y) \in \partial\Omega : x = 0 \text{ or } x = 1\}$ and $\Gamma_N = \partial\Omega \backslash \Gamma_D$. Let us choose the following operator coefficients

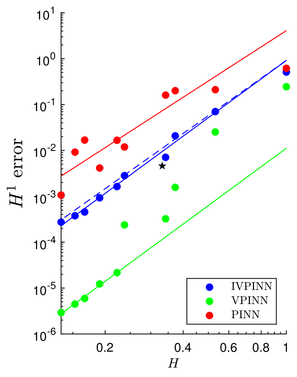
$$\mu(x, y) = 2 + \sin(x + 2y), \qquad \beta(x, y) = \begin{bmatrix} \sqrt{x - y^2 + 5} \\ \sqrt{y - x^2 + 5} \end{bmatrix}, \qquad \sigma(x, y) = e^{\frac{x}{2} - \frac{y}{3}} + 2\,,$$

and the data f, g, and ψ such that the exact solution is

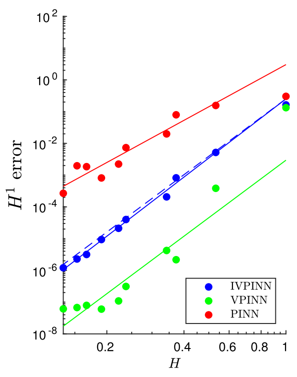
$$u(x, y) = \sin(3.2x(x - y))\cos(4.3y + x) + \sin(4.6(x + 2y))\cos(2.6(y - 2x)).$$

The corresponding error decays with respect to the meshsize H are shown in Fig. 2.

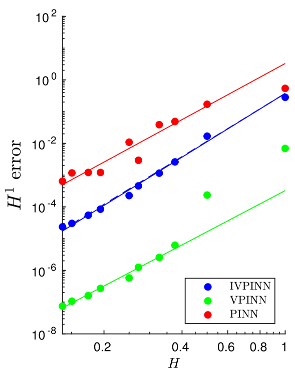
In Fig. 2a, where the IVPINN (blue dots) and the VPINN (green dots) are trained with $q = 3$ and $k_{\text{test}} = 1$, we observe that the points are distributed, possibly after an initial preasymptotic phase, along straight lines with slopes very close to $k_{\text{int}} = 4$. We highlight that the PINN convergence is significantly noisier and that the corresponding H^1 error is, on average, about 7 times the IVPINN one. A similar phenomenon can be seen in Fig. 2b, although the finite precision of the used Tensorflow software prevents convergence to display at full for small values of H. In this test we use $q = 5$ and $k_{\text{test}} = 1$ and the regression lines for the IVPINN and the VPINN have slopes close to $k_{\text{int}} = 6$, while the PINN accuracy is again much lower. Finally, the data in Fig. 2c are obtained with $q = 5$ and $k_{\text{test}} = 2$ and the blue regression line slope is 5.05, almost coinciding with $k_{\text{int}} = 5$.



a $q = 3$, $k_{test} = 1$, $k_{\text{int}} = 4$.
Expected decay rate: 4.
Obtained decay rate: 4.15.

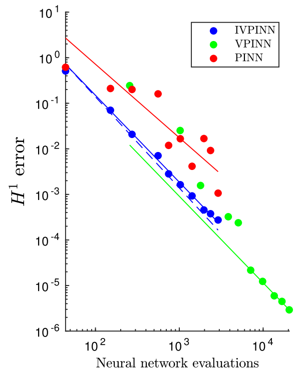


b $q = 5$, $k_{test} = 1$, $k_{\text{int}} = 6$.
Expected decay rate: 6.
Obtained decay rate: 6.19.

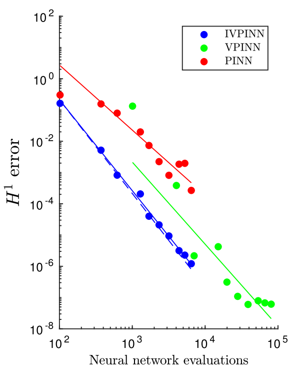


c $q = 5$, $k_{test} = 2$, $k_{\text{int}} = 5$.
Expected decay rate: 5.
Obtained decay rate: 5.05.

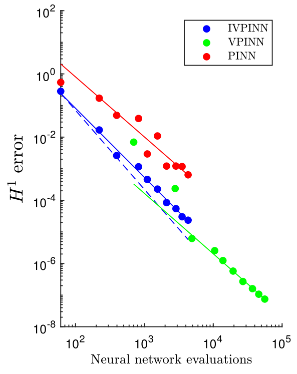
Fig. 2 Error decays versus H for *Convergence test #1: $u \in C^\infty(\bar{\Omega})$*



a $q = 3$, $k_{test} = 1$, $k_{\text{int}} = 4$.
Expected decay rate: -2.
Obtained decay rate: -1.91.



b $q = 5$, $k_{test} = 1$, $k_{\text{int}} = 6$.
Expected decay rate: -3.
Obtained decay rate: -2.91.



c $q = 5$, $k_{test} = 2$, $k_{\text{int}} = 5$.
Expected decay rate: -2.5.
Obtained decay rate: -2.19.

Fig. 3 Error decays versus dataset size for *Convergence test #1: $u \in C^\infty(\bar{\Omega})$*

Such results highlight that, although the VPINN implementation is more complex than the PINN one, the former produces more accurate solutions than the latter, when the exact solution is regular.

In Fig. 3, the same error decays are expressed in terms of the number of training points, i.e., the number of neural network forward evaluations required to construct the loss function in a single epoch. Such an alternative visualization highlights that the performances of the IVPINN and the VPINN are very similar when trained with similar training sets. This is due to the fact that, since we stabilize the VPINN by projecting it on a space of continuous piecewise polynomials, we need fewer interpolation points (input data of the IVPINN) than quadrature points (input data of the VPINN) to evaluate the loss function.

Remark 5 (on the quotient $\frac{C_h}{c_h}$) Theorem 6 indicates that the best possible convergence rate, when the solution is regular enough, is k_{int}. However, as discussed in Remark 3, the quotient $\frac{C_h}{c_h}$ is of order $O(h^{-1})$ when test functions are picked from the Lagrange basis associated with a quasi-uniform triangulation and the weights γ_i are equal to 1. In this case, the term $\left(1 + \frac{C_h}{c_h}\right)$ in (4.35) reduces the predicted convergence by exactly one order.

On the other hand, in Fig. 2, we have shown cases where the order of convergence is optimal. Such a behavior is related to the fact that the loss $R_h(u^{\mathcal{NN}})$ decays much faster than expected, in the considered cases, namely at least as $O(h^8)$. Therefore, when h is small enough, the term $C_h R_h(u^{\mathcal{NN}})$ in Lemma 4.1 can be neglected, and the predicted convergence rate is not affected by the presence of the quotient $\frac{C_h}{c_h}$.

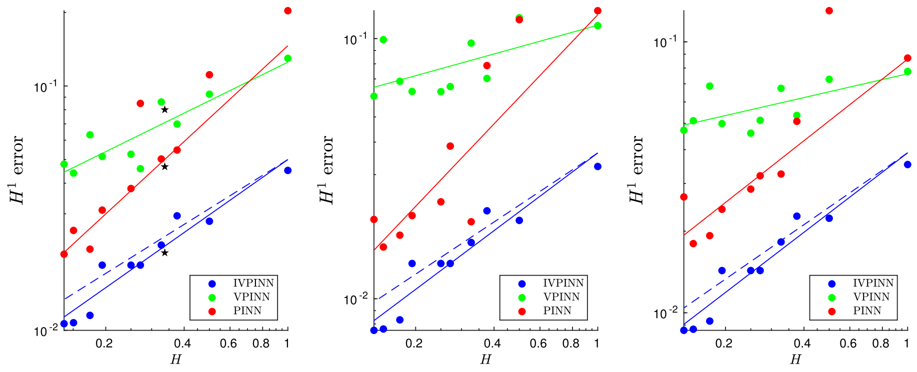
Convergence test #2: $u \in H^{5/3-\varepsilon}(\Omega)$

Let us now focus on a less smooth solution, whose regularity is commonly found in domains with reentrant corners. The problem is characterized by $\Gamma_D = \partial\Omega$, $\mu = 1$, $\beta = [2, 3]^T$, $\sigma = 4$, whereas the forcing term and boundary conditions are such that the exact solution is, in polar coordinates,

$$u(r, \theta) = r^{\frac{2}{3}} \sin\left(\frac{2}{3}\left(\theta + \frac{\pi}{2}\right)\right).$$

Since $u \in H^{5/3-\varepsilon}(\Omega)$ for any $\varepsilon > 0$, we expect a convergence rate close to 2/3; indeed, $k_{\text{int}} \geq 4$ and the rate of convergence is always limited by the solution regularity as expected. The error decays are shown in Fig. 4. Notice that the IVPINN is even more stable and accurate than the VPINN trained on the same mesh (i.e., with more input data). The PINN behaves better in this test case than in the previous one, but still the accuracy is worse than the one provided by our IVPINN.

It is also interesting to analyze the behavior of the loss function and of the error during training as documented in Fig. 5, where the first 3000 epochs are performed with the ADAM optimizer, while the remaining ones with the BFGS optimizer. Such plots correspond to the loss function and the H^1 error associated with the dots marked by the black stars in Fig. 4a.



a $q = 3$, $k_{test} = 1$, $k_{\text{int}} = 4$. Expected decay rate: 2/3. Obtained decay rate: 0.75.

b $q = 5$, $k_{test} = 1$, $k_{\text{int}} = 6$. Expected decay rate: 2/3. Obtained decay rate: 0.75.

c $q = 5$, $k_{test} = 2$, $k_{\text{int}} = 5$. Expected decay rate: 2/3. Obtained decay rate: 0.74.

Fig. 4 Error decays versus H for *Convergence test #2: $u \in H^{5/3-\varepsilon}(\Omega)$*

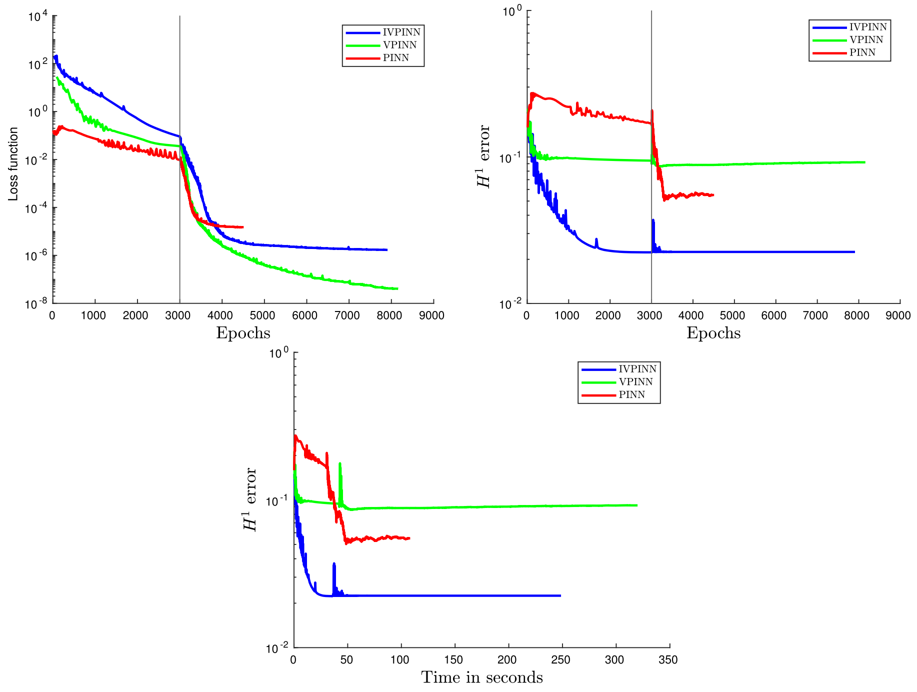
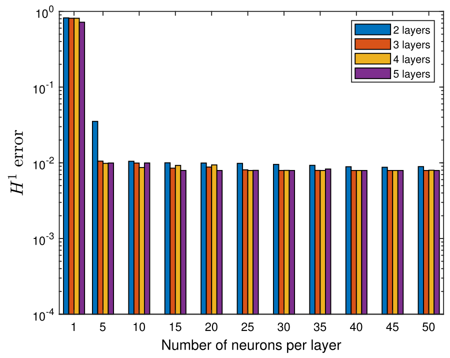


Fig. 5 Top row: loss function (left) and H^1 error (right) evaluations as functions of the number of executed epochs. The first 3000 epochs are performed with the ADAM optimizer, the subsequent ones with the BFGS optimizer. Bottom row: H^1 error as a function of the elapsed time

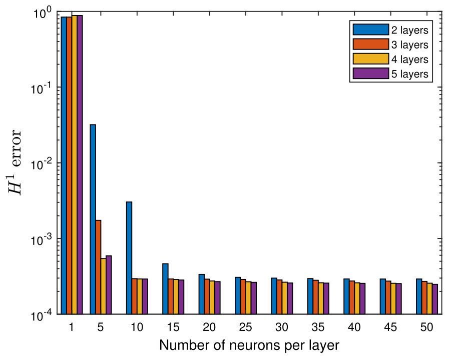
It can be noted that the IVPINN and the VPINN initially converge very fast with the ADAM optimizer; eventually, after the initial phase in which both the loss and the error decrease, the error reaches a constant value despite the loss function keeps diminishing. This implies that there exist other sources of error that prevail when the loss function decays. On the other hand, using a standard PINN, one observes that the convergence of the loss and the error is much slower than for the VPINNs, and the second-order optimizer is needed to converge to an accurate solution. The average epoch execution time is approximately 0.0599 seconds for the PINN, 0.0587 seconds for the VPINN, and 0.0479 seconds for the IVPINN. Such a gain is due to the fact that the model derivatives are computed via automatic differentiation in the non-interpolated models, while the gradient of the IVPINN can be computed by a simple matrix-vector multiplication. Note that the gain increases when higher derivatives are involved in the PDE.

6.2 How the VPINN Dimension Affects Accuracy

We now focus on the dependence of the error on the neural network dimension. For the sake of simplicity, we fix the problem discretization and vary only the number of layers and the number of neurons in each layer, assuming that each layer contains exactly the same number of neurons. The considered domain, parameters, forcing term and boundary conditions are the ones described in *Convergence test #1*. The VPINN is trained with piecewise linear test



a VPINN trained on the mesh used to compute the blue point close to the black star in Fig. 2.a.



b VPINN trained on the finest mesh used in Fig. 2.a.

Fig. 6 H^1 error varying the number of layers and the number of neurons in each layer of the neural network

functions ($k_{\text{test}} = 1$) and quadrature rules of order $q = 3$ on the finest mesh used to produce Fig. 2a and on the mesh associated with the blue dot close to the black star in the same figure.

We can observe, in Fig. 6, that the error is very high for small networks, but then it rapidly decreases while increasing the number of neurons in each layer, until a plateau is reached depending on the chosen problem discretization. Essentially, on both meshes, 3 layers with 10 neurons each suffice to achieve the lowest possible discretization error for the given loss function.

This analysis confirms that the error decays reported in Sect. 6.1 are all insensitive to the neural network hyper-parameters, as they have been obtained by a large neural network (5 layers with 50 neurons each). Such results validate the assumption made in Sect. 4 about the neural network, namely that its dimension – provided it is sufficiently large – does not influence the predicted convergence rate.

6.3 On the Importance of the Inf-sup Condition

In this section we show that the inf-sup condition, assumed in Proposition 4 to derive the a priori error estimate, is crucial in order to avoid spurious modes in the numerical solution. To prove such a claim, let us consider the simplest one-dimensional Poisson's problem with zero forcing term and zero Dirichlet boundary conditions:

$$\begin{cases} -u'' = 0 & \text{in } \Omega\,, \\ u = 0 & \text{on } \Gamma_D\,, \end{cases} \tag{6.1}$$

where $\Omega = (0, 1)$ and $\Gamma_D = \{0, 1\}$. For the sake of simplicity, we use piecewise linear test functions and quadrature rules of order $q = 3$. Note that since both $U^{\mathcal{NN}}$ and U_H contain the exact solution $u \equiv 0$, it is always possible to obtain a numerical solution that is identical to the exact one (up to numerical precision).

Let us denote by u_δ any discrete solution defined in Sect. 3.2, namely, either a solution $u_H^{\mathcal{NN}}$ obtained by interpolated VPINNs, or a solution $\hat{u}^{\mathcal{NN}}$ obtained by non-interpolated VPINNs. These discrete solutions are represented in Fig. 7 in logarithmic scale to allow a

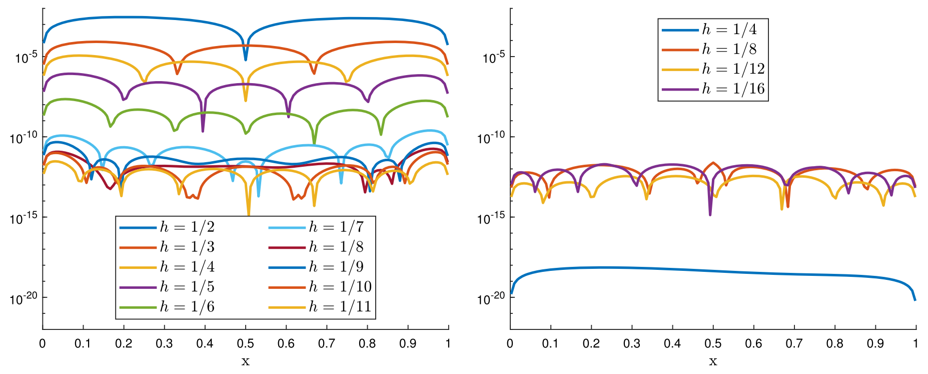


Fig. 7 Numerical solutions of problem (6.1) computed with different meshes, solutions obtained with non-interpolated neural networks (left) and with interpolated neural networks (right). Quadrature rule order $q = 3$. Test functions order $k_{test} = 1$

direct comparison. In order to avoid numerical issues due to the logarithmic scale of the plot when u_δ gets close to 0, a truncation procedure is applied.

The functions u_δ produced by non-interpolated networks are represented in the left plot of Fig. 7. Each one is obtained by minimizing the loss function up to machine accuracy; despite this, when the mesh is fairly coarse the discrete solution is significantly different from the null solution. Indeed, the initial weights in the training process are non-zero, and the minimization process is under-determined, thereby allowing the existence of non-zero global minima. Refining the mesh, the approximation improves up to a maximum precision imposed by the chosen network architecture and the Tensorflow deep learning framework.

Conversely, the plots in the subfigure on the right-hand side of Fig. 7, produced by interpolated networks, clearly indicate that the obtained discrete solutions are numerically zero, irrespective of the meshsize. Note that the case $h = 1/4$ differs from the others since here the interpolation mesh $\mathcal{T}_H$ is formed by just one element. The corresponding function $u_H^{\mathcal{NN}}$ is thus differentiable everywhere and the used gradient-based optimizers are able to minimize the loss function more effectively. We highlight that, since $q = 3$ and $k_{\text{test}} = 1$, we have to choose $H = 4h$ to satisfy the inf-sup condition.

To illustrate the mechanism that may lead to the onset of spurious modes, in "Appendix A.2" we provide an analytical example of a neural network which significantly differs from a PDE solution, yet it is a global minimizer of the corresponding loss function.

These results, although obtained in overly simple functional settings, show the potential existence of uncontrolled components in the discrete solutions obtained by non-interpolated neural networks. In more complex scenarios, the presence of spurious modes may be even more pronounced. In practice, as observed in Fig. 2, when the PDE solution is smooth enough non-interpolated solutions appear to be more accurate than the corresponding solutions obtained by interpolated neural networks using the same test functions; however, a rigorous analysis of NN-based discretization schemes should also cope with the presence of spurious components, which we have avoided by resorting to an inf-sup condition. We believe that these observations shed new light on the use of deep learning in numerical PDEs.

7 Application to Nonlinear Parametric Problems

In the previous sections, we investigated the features of the proposed VPINN discretization for the linear boundary-value problem (2.1). Hereafter, we provide an application where the nonlinear nature of neural networks can be exploited at best. It is well known that solving nonlinear PDEs by PINNs or VPINNs comes at little extra cost with respect to linear PDEs, since nonlinearities just impact the computation of the loss function. Similarly, parametric problems can be easily and efficiently handled by neural networks, even when the dependence of the solution upon the parameters is nonlinear. Indeed, it is enough to add as many inputs as the number of parameters in the definition of the network, and train it on a proper subset of the parameter space.

To illustrate the behavior of our VPINN in these situations, let us consider the following nonlinear parametric equation:

$$\begin{cases} -\nabla \cdot (\mu \nabla u) + \boldsymbol{\beta} \cdot \nabla u + \sigma \, \mathrm{e}^{-pu^2} = f & \text{in } \Omega \,, \\ u = g & \text{on } \partial\Omega \,, \end{cases} \tag{7.1}$$

where μ, β, σ, f and g are suitably smooth functions, and $p \in \Omega_p \subset \mathbb{R}$ is an additional parameter. Our goal is to train a neural network to compute the numerical solution for any given value of p in a prescribed parametric domain Ω_p.

We fix $\Omega = (0, 1)^2$, $\mu = 1$, $\beta = [2, 3]^T$, $\sigma = 4$ and choose $f = f(\cdot, p)$ and $g = g(\cdot, p)$ such that the exact solution is

$$u(x, y; p) = \frac{\cos\left(5\left(px + \frac{y}{2}\right)\right)}{1 + p} + \left(x + \frac{y}{2}\right)^2 .$$

To approximate such a solution in the parametric domain $\Omega_p = [0.5, 2]$, we consider a neural network with three inputs (x, y, p) and we train it using the loss function:

$$R_h^2(w) = \sum_{p \in \Omega_p^{\#}} \sum_{i \in I_h} r_{h,i;p}^2(w)\, \gamma_i^{-1} \,, \tag{7.2}$$

where $\Omega_p^{\#} = \{p_1, ..., p_{n_p}\} \subset \Omega_p$ is a finite set of parameter values, and the residuals $r_{h,i;p}$ are defined as in (3.10), considering the new equation. In this numerical test $\Omega_p^{\#}$ contains $n_p = 13$ equally spaced values $0.5 = p_1 < p_2 < ... < p_{n_p} = 2$.

After the training phase, the neural network can be evaluated at new parameter values to analyze its accuracy. The error diagram is presented in Fig. 8: the blue line is computed as the H^1 error between the exact solution $u = u(\cdot, p)$ and the corresponding numerical solution, while the red dots represent the error associated with parameters in $\Omega_p^{\#}$. Despite the small number of parameter values used during training, the model provides accurate solutions for the whole range of parameter values. Note that the error increases for larger values of p because the solution is more and more oscillating as p increases.

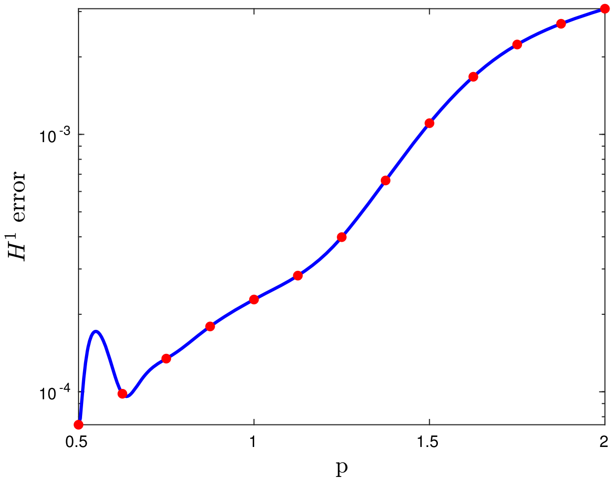


Fig. 8 H^1 error for different values of the parameter p. The VPINNs are trained on $n_p = 13$ parameters. The red dots are associated with parameter values used in the training phase

8 Conclusions

We have investigated VPINN methods for elliptic boundary-value problems, in what concerns the choice of test functions and quadrature rules. The aim was the derivation of rigorous a priori error estimates for some projection of the neural network solution. The neural network is trained using as test functions finite-element nodal Lagrange basis functions of degree k_{test} on a mesh $\mathcal{T}_h$, where Gaussian quadrature rules of order q are applied in each element of $\mathcal{T}_h$. For a fixed neural network architecture with tanh activation function, we studied how the error in the energy norm depends upon the mesh parameter h, for different values of k_{test} and q.

Error control was obtained for the finite-element interpolant of degree $k_{\text{int}} = q + 2 - k_{\text{test}}$ of the neural network on an auxiliary mesh $\mathcal{T}_H$; such an interpolation enters also in the definition of the residuals which are minimized through the loss function. A key ingredient in the error control is the validity of an inf-sup condition between the spaces of test functions and interpolating functions. Indeed, the neural network solution might be affected by spurious modes due to the under-determined nature of the minimization problem, as we documented for a problem with zero data; instead, the onset of such modes is prevented by the adopted interpolation procedure.

Our analysis reveals that the convergence rate in the energy norm is at least of order $q + 1 - k_{\text{test}}$ for sufficiently smooth functions, and it increases to $q + 2 - k_{\text{test}}$ when the value of the loss function obtained by minimization is sufficiently small. The main message stemming from the analysis is that it is convenient to choose test functions of the lowest degree $k_{\text{test}} = 1$ in order to get the highest convergence rate for a fixed quadrature rule. Furthermore, for smooth solutions the convergence rate may be arbitrarily increased by increasing the precision of the quadrature rule, although the realization of this theoretical statement is hampered in practice by the finite precision of machine arithmetics.

We also investigated the influence of the neural network hyperparameters on the overall accuracy of the discretization, and we found that a small network with few layers and neurons suffices to reach accuracies of practical interest. To stay on the safe side, we used a larger network in our experiments, thereby obtaining results that are essentially independent of the network hyperparameters.

For the sake of comparison, we also implemented a standard VPINN without projection upon piecewise polynomials, as well as a standard PINN trained with the same number of inputs as those used in training our VPINN. Interestingly, we experimentally observed that in general the error decay rate for the non-interpolated neural network solution replicates the one theoretically predicted for the interpolated network. The PINN solutions appear to be less accurate and noisier than the interpolated VPINN's.

We have shown that interpolated VPINNs are able to efficiently solve nonlinear parametric problems without the need for additional nonlinear solvers or globalization methods, due to their intrinsic nonlinear nature. The VPINN can be trained in an off-line phase on a subset of the parameter domain and then efficiently evaluated on-line on any other parameter value. This is a key difference between the proposed method and standard numerical techniques such as FEM, even if the solution is sought in the same finite dimensional space. Indeed, the latter would require some iterative technique to handle nonlinearities, as well as some form of interpolation/extrapolation to get the solution for the whole range of parameters. All this is provided for free by the NN machinery.

Possible extensions of this work are related to the investigation of more advanced neural networks architectures to improve the method accuracy and efficiency [32, 33] or to more complex problems. Indeed, neural networks are known to be able to manage very high-dimensional problems, overcoming the so-called curse of dimensionality, therefore we expect them to be able to efficiently solve parametric PDEs with multiple parameters [33] or high-dimensional PDEs [4, 34]. Other possible applications are related, for instance, to inverse problems [35] or integration between PDEs and data [36].

Funding Open access funding provided by Politecnico di Torino within the CRUI-CARE Agreement. The authors performed this research in the framework of the Italian MIUR Award "Dipartimenti di Eccellenza 2018-2022" granted to the Department of Mathematical Sciences, Politecnico di Torino (CUP: E11G18000350001). The research leading to this paper has also been partially supported by the SmartData@PoliTO center for Big Data and Machine Learning technologies. SB was supported by the Italian MIUR PRIN Project 201744KLJL-004, CC was supported by the Italian MIUR PRIN Project 201752HKH8-003. The authors are members of the Italian INdAM-GNCS research group.

Data Availibility The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Appendix

A.1 Construction of the Interpolation Operator

In this section we provide details on the practical construction of the operator $\mathcal{I}_H : \mathrm{C}^0(\bar{\Omega}) \to U_H$ introduced in Sect. 3.2.

Since U_H is the linear subspace of U containing all the piecewise polynomial of degree k_{int} defined over $\mathcal{T}_H$, there exists a Lagrange basis $\{\hat{\varphi}_i\}_{i=1}^{n_I}$ such that $U_H = \text{span}\{\hat{\varphi}_i : i = 1, ..., n_I\}$, which is associated with a corresponding set of points $\{\mathbf{x}_i\}_{i=1}^{n_I} \subset \Omega$. The basis functions satisfy the relations $\hat{\varphi}_i(\mathbf{x}_j) = \delta_{i,j}, \forall i, j = 1, ..., n_I$. Therefore, the operator $\mathcal{I}_H$ maps the generic function $v \in \mathrm{C}^0(\bar{\Omega})$ to the function $\mathcal{I}_H v = \sum_{i=1}^{n_I} v_i \hat{\varphi}_i \in U_H$, uniquely identified by the vector $\mathbf{v} = \{v_i\}_{i=1}^{n_I}$, where $v_i = v(\mathbf{x}_i)$.

In order to evaluate the function $\mathcal{I}_H B u^{\mathcal{NN}}$ at the required quadrature points $\{\mathbf{x}_j^q, j = 1, ..., n_q\}$ during the loss function computation or the final evaluation, one just needs to compute the quantities

$$\mathcal{I}_H B u^{\mathcal{NN}}(\mathbf{x}_j^q) = \sum_{i=1}^{n_I} \left(B u^{\mathcal{NN}}\right)(\mathbf{x}_i) \hat{\varphi}_i(\mathbf{x}_j^q).$$

In practice, it is more convenient to introduce the sparse matrix $M \in \mathbb{R}^{n_q \times n_I}$ such that $M_{i,j} = \hat{\varphi}_j(\mathbf{x}_i^q)$. This allows us to evaluate the interpolated function at each quadrature point with a matrix-vector multiplication as follows:

$$\begin{bmatrix} \mathcal{I}_H B u^{\mathcal{NN}}\left(\mathbf{x}_1^q\right) \\ \mathcal{I}_H B u^{\mathcal{NN}}\left(\mathbf{x}_2^q\right) \\ \vdots \\ \mathcal{I}_H B u^{\mathcal{NN}}\left(\mathbf{x}_{n_q}^q\right) \end{bmatrix} = M \begin{bmatrix} \left(B u^{\mathcal{NN}}\right)(\mathbf{x}_1) \\ \left(B u^{\mathcal{NN}}\right)(\mathbf{x}_2) \\ \vdots \\ \left(B u^{\mathcal{NN}}\right)(\mathbf{x}_{n_I}) \end{bmatrix}. \tag{A.1.1}$$

In the same way, the derivatives of $\mathcal{I}_H B u^{\mathcal{NN}}$ can be computed, at the same points, as

$$\frac{\partial^{|\alpha|} \mathcal{I}_H B u^{\mathcal{NN}}}{\partial x^\alpha}(\mathbf{x}_j^q) = \sum_{i=1}^{n_I} \left(B u^{\mathcal{NN}}\right)(\mathbf{x}_i) \frac{\partial^{|\alpha|} \hat{\varphi}_i(\mathbf{x}_j^q)}{\partial x^\alpha}, \tag{A.1.2}$$

where $\alpha = (\alpha_1, ..., \alpha_n) \in \mathbb{Z}_+^n$ and $|\alpha| = \sum_{i=1}^n \alpha_i$. Defining the matrix $M^\alpha \in \mathbb{R}^{n_q \times n_I}$ such that $M_{i,j}^\alpha = \frac{\partial^{|\alpha|} \hat{\varphi}_j(\mathbf{x}_i^q)}{\partial x^\alpha}$, it is possible to compute all the required derivatives simply by replacing M by M^α on the right hand side of (A.1.1). In this way, the VPINN derivatives can be computed without relying on automatic differentiation, further improving the method efficiency during training.

A.2 An Example of 'Spurious' Neural Network

Consider again the boundary-value problem (6.1), which admits the null solution. We are interested in solving this problem by a plain PINN (VPINN) solver. To train the network, we choose a set of $n_{\mathcal{S}}$ control (quadrature) points,

$$\mathcal{S} = \{x_i : i = 1, ..., n_{\mathcal{S}}\}$$

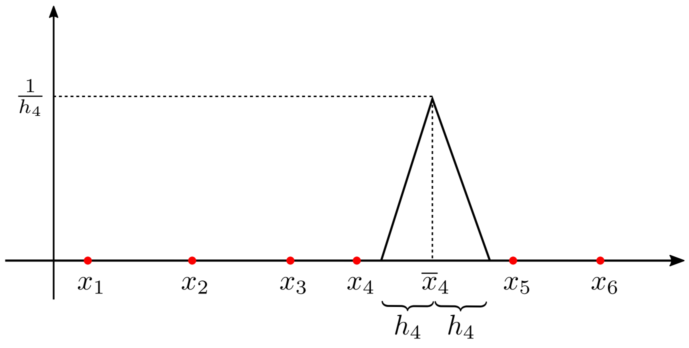


Fig. 9 Graphical representation of a function $u_4^{\mathcal{NN}}$ defined in (B.2.1)

satisfying $0 \le x_1 < x_2 < ... < x_{n_\mathcal{S}-1} < x_{n_\mathcal{S}} \le 1$. Using the architecture defined in (3.1), it is possible to construct a ReLU neural network w with just a single hidden layer and 3 neurons with the following weights:

$$\mathbf{A}_1^j = \begin{bmatrix} 1/h_j \\ 1/h_j \\ 1/h_j \end{bmatrix}, \qquad \mathbf{b}_1^j = \begin{bmatrix} (-\overline{x}_j + h_j)/h_j \\ -\overline{x}_j/h_j \\ (-\overline{x}_j - h_j)/h_j \end{bmatrix}, \qquad \mathbf{A}_2^j = \begin{bmatrix} \frac{1}{h_j} & -\frac{2}{h_j} & \frac{1}{h_j} \end{bmatrix}, \qquad \mathbf{b}_2^j = 0,$$

where, for any fixed index $j \in \{1, ..., n_\mathcal{S} - 1\}$, we denote by $\overline{x}_j = \frac{x_j + x_{j+1}}{2}$ the mean of two consecutive nodes, and by h_j the difference between $\overline{x}_j$ and $x_j + \varepsilon_j$, for some $\varepsilon_j \in (0, \overline{x}_j - x_j)$. The function represented by this set of weights is:

$$\begin{aligned} w_j(x) =& \frac{1}{h_j} \max\left(0, \frac{x}{h_j} + \frac{-\overline{x}_j + h_j}{h_j}\right) - \frac{2}{h_j} \max\left(0, \frac{x}{h_j} + \frac{-\overline{x}_j}{h_j}\right) + \\ &+ \frac{1}{h_j} \max\left(0, \frac{x}{h_j} + \frac{-\overline{x}_j - h_j}{h_j}\right) \\ =& \begin{cases} -\dfrac{\overline{x}_j - h_j}{h_j^2} + \dfrac{x}{h_j^2} & \text{if } x \in (\overline{x}_j - h_j, \overline{x}_j], \\ \dfrac{\overline{x}_j + h_j}{h_j^2} - \dfrac{x}{h_j^2} & \text{if } x \in (\overline{x}_j, \overline{x}_j + h_j), \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{B.2.1}$$

An example of such a function is shown in Fig. 9.

It is easily seen that $w_j(x_i) = w_j'(x_i) = w_j''(x_i) = 0$ for any $i = 1, ..., n_\mathcal{S}$, therefore the PINN (VPINN) loss function is exactly equal to 0. However, this does not ensure the accuracy of the approximation, in fact $\|w_j\|_{L^1(\Omega)} = \|w_j - u\|_{L^1(\Omega)} = 1$.

Note that it is possible to define larger networks with analogous properties. Moreover, the same phenomenon can be observed with any sigmoid activation function, exploiting the fact that the accuracy in the evaluation of both the activation function and the loss function is bounded by machine precision. We also highlight that the phenomenon can be partially alleviated by introducing a regularization term in the loss function; however, it adds noise to the optimization process, possibly resulting in losses of accuracy when the PDE solution is characterized by large gradients.

This proves that it is not possible to guarantee the inf-sup stability for standard PINNs or VPINNs, and that spurious modes cannot be controlled simply minimizing the loss function. On the contrary, the interpolation operator proposed in this paper acts as a VPINN stabilizer, preventing the onset of spurious components in the solution.

References

1. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. **378**, 686–707 (2019). https://doi.org/10.1016/j.jcp.2018.10.045
2. Tartakovsky, A.M., Marrero, C.O., Perdikaris, P., Tartakovsky, G.D., Barajas-Solano, D.: Learning Parameters and Constitutive Relationships with Physics Informed Deep Neural Networks. arXiv:1808.03398 (2018)
3. Yang, Y., Perdikaris, P.: Adversarial uncertainty quantification in physics-informed neural networks. J. Comput. Phys. **394**, 136–152 (2019)
4. Lanthaler, S., Mishra, S., Karniadakis, G.E.: Error estimates for DeepONets: a deep learning framework in infinite dimensions. Trans. Math. Appl. **6**(1), tnac001 (2022). https://doi.org/10.1093/imatrm/tnac001
5. Elbrächter, D., Perekrestenko, D., Grohs, P., Bölcskei, H.: Deep neural network approximation theory. IEEE Trans. Inf. Theory **67**(5), 2581–2623 (2021)
6. Gühring, I., Kutyniok, G., Petersen, P.: Error bounds for approximations with deep ReLU neural networks in $W^{s,p}$ norms. Anal. Appl. **18**(05), 803–859 (2020)
7. Opschoor, J.A., Petersen, P.C., Schwab, C.: Deep ReLU networks and high-order finite element methods. Anal. Appl. **18**(05), 715–770 (2020)
8. Kutyniok, G., Petersen, P., Raslan, M., Schneider, R.: A theoretical analysis of deep neural networks and parametric PDEs. Constr. Approx., 1–53 (2021)
9. Opschoor, J.A., Schwab, C., Zech, J.: Exponential ReLU DNN expression of holomorphic maps in high dimension. Constr. Approx., 1–46 (2021)
10. Gonon, L., Schwab, C.: Deep ReLU Neural Networks Overcome the Curse of Dimensionality for Partial Integrodifferential Equations. arXiv:2102.11707 (2021)
11. Kharazmi, E., Zhang, Z., Karniadakis, G.E.: VPINNs: Variational Physics-Informed Neural Networks For Solving Partial Differential Equations. arXiv:1912.00873 (2019)
12. Zang, Y., Bao, G., Ye, X., Zhou, H.: Weak adversarial networks for high-dimensional partial differential equations. J. Comput. Phys. **411**, 109409 (2020)
13. Khodayi-Mehr, R., Zavlanos, M.: VarNet: Variational neural networks for the solution of partial differential equations. In: Learning for Dynamics and Control, pp. 298–307, PMLR (2020)
14. Kharazmi, E., Zhang, Z., Karniadakis, G.E.: *hp*-VPINNs: Variational physics-informed neural networks with domain decomposition. Comput. Methods Appl. Mech. Eng. **374**, 113547 (2021)
15. Mishra, S., Molinaro, R.: Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs. IMA J. Numer. Anal. (2021). https://doi.org/10.1093/imanum/drab032
16. Berrone, S., Canuto, C., Pintore, M.: Solving Pdes by Variational Physics-informed Neural Networks: an a Posteriori Error Analysis. arXiv:2205.00786 (2022)
17. Sukumar, N., Srivastava, A.: Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. Comput. Methods Appl. Mech. Eng. **389**, 114333–50 (2022). https://doi.org/10.1016/j.cma.2021.114333
18. Nitsche, J.A.: Uber ein Variationsprinzip zur Losung Dirichlet-Problemen bei Verwendung von Teilraumen, die keinen Randbedingungen unteworfen sind. Abh. Math. Sem. Univ., Hamburg 36, 9–15 (1971)
19. Ciarlet, Ph.G.: The Finite Element Method for Elliptic Problems. Classics in Applied Mathematics, vol. 40. Society for Industrial and Applied Mathematics (SIAM), Philadephia (2002). https://doi.org/10.1137/1.9780898719208
20. De Ryck, T., Lanthaler, S., Mishra, S.: On the approximation of functions by tanh neural networks. Neural Netw. (2021). https://doi.org/10.1016/j.neunet.2021.08.015
21. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. arXiv:1412.6980, (2014)
22. Smith, L.N.: Cyclical learning rates for training neural networks. In: 2017 IEEE Winter Conference on Applications of Computer Vision (WACV), pp. 464–472, IEEE (2017)
23. Wright, S., Nocedal, J., *et al.*: Numerical optimization. Springer Science 35(67-68), 7 (1999)

24. Rvachev, V.L., Sheiko, T.I., Shapiro, V., Tsukanov, I.: Transfinite interpolation over implicitly defined sets. Comput. Aided Geom. Design **18**(3), 195–220 (2001). https://doi.org/10.1016/S0167-8396(01)00015-2
25. Zhang, E., Yin, M., Karniadakis, G.E.: Physics-informed neural networks for nonhomogeneous material identification in elasticity imaging. arXiv:2009.04525 (2020)
26. Sahli Costabal, F., Yang, Y., Perdikaris, P., Hurtado, D.E., Kuhl, E.: Physics-informed neural networks for cardiac activation mapping. Front. Phys. **8**, 42 (2020)
27. Ji, W., Qiu, W., Shi, Z., Pan, S., Deng, S.: Stiff-pinn: Physics-informed neural network for stiff chemical kinetics. J. Phys. Chem. A **125**(36), 8098–8106 (2021). https://doi.org/10.1021/acs.jpca.1c05102
28. Wight, C.L., Zhao, J.: Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks. Commun. Comput. Phys. **29**(3), 930–954 (2021)
29. Baydin, A.G., Pearlmutter, B.A., Radul, A.A., Siskind, J.M.: Automatic differentiation in machine learning: a survey. J. Mach. Learn. Res. **18**, 1–43 (2018)
30. Berrone, S., Canuto, C., Pintore, M.: Variational Physics Informed Neural Networks: The Role of Quadratures and Test Functions. arXiv:2109.02095v1 (2021)
31. Mishra, S., Molinaro, R.: Estimates on the generalization error of physics-informed neural networks for approximating PDEs. IMA J. Numer. Anal. (2022). https://doi.org/10.1093/imanum/drab093
32. Rodriguez-Torrado, R., Ruiz, P., Cueto-Felgueroso, L., Green, M.C., Friesen, T., Matringe, S., Togelius, J.: Physics-informed attention-based neural network for hyperbolic partial differential equations: application to the buckley-leverett problem. Sci. Rep. **12**(1), 1–12 (2022)
33. Gao, H., Sun, L., Wang, J.-X.: Phygeonet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. J. Comput. Phys. **428**, 110079 (2021)
34. Han, J., Jentzen, A., Weinan, E.: Solving high-dimensional partial differential equations using deep learning. Proc. Nat. Acad. Sci. **115**(34), 8505–8510 (2018)
35. Chen, Y., Lu, L., Karniadakis, G.E., Negro, L.D.: Physics-informed neural networks for inverse problems in nano-optics and metamaterials. Opt. Express **28**(8), 11618–11633 (2020). https://doi.org/10.1364/OE.384875
36. Chen, Z., Liu, Y., Sun, H.: Physics-informed learning of governing equations from scarce data. Nat. Commun. **12**(1), 1–13 (2021)