MOX-Report No. 69/2022

Learning Operators with Mesh-Informed Neural Networks

Franco, N.R; Manzoni, A.; Zunino, P.

MOX, Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

mox-dmat@polimi.it http://mox.polimi.it

Noname manuscript No.
(will be inserted by the editor)

Learning Operators with Mesh-Informed Neural Networks

Nicola Rares Franco · Andrea Manzoni · Paolo Zunino

Received: date / Accepted: date

Abstract Thanks to their universal approximation properties and new efficient training strategies, Deep Neural Networks are becoming a valuable tool for the approximation of mathematical operators. In the present work, we introduce Mesh-Informed Neural Networks (MINNs), a class of architectures specifically tailored to handle mesh based functional data, and thus of particular interest for reduced order modeling of parametrized Partial Differential Equations (PDEs). The driving idea behind MINNs is to embed hidden layers into discrete functional spaces of increasing complexity, obtained through a sequence of meshes defined over the underlying spatial domain. The approach leads to a natural pruning strategy which enables the design of sparse architectures that are able to learn general nonlinear operators. We assess this strategy through an extensive set of numerical experiments, ranging from nonlocal operators to nonlinear diffusion PDEs, where MINNs are compared to classical fully connected Deep Neural Networks. Our results show that MINNs can handle functional data defined on general domains of any shape, while ensuring reduced training times, lower computational costs, and better generalization capabilities, thus making MINNs very well-suited for demanding applications such as Reduced Order Modeling and Uncertainty Quantification for PDEs.

Keywords Operator Learning · Reduced Order Modeling · Nonlinear PDEs

Mathematics Subject Classification (2020) MSC 35J60 · MSC 47J05 · MSC 65N30 · MSC 68T07

Nicola Rares Franco
MOX, Department of Mathematics, Politecnico di Milano, Italy
E-mail: nicolarares.franco@polimi.it

Andrea Manzoni
MOX, Department of Mathematics, Politecnico di Milano, Italy
E-mail: andrea1.manzoni@polimi.it

Paolo Zunino
MOX, Department of Mathematics, Politecnico di Milano, Italy
E-mail: paolo.zunino@polimi.it

1 Introduction

Deep Neural Networks (DNNs) are one of the fundamental building blocks in modern Machine Learning. Originally developed to tackle classification tasks, they have become extremely popular after reporting striking achievements in fields such as computer vision [21] and language processing [30]. Not only, an in-depth investigation of their approximation properties has also been carried out in the last decade [5,8,16,18]. In particular, DNNs have been recently employed for learning (nonlinear) operators in high-dimensional spaces [9,15,20, 24], because of their unique properties, such as the ability to blend theoretical and data-driven approaches. Additionally, the interest in using DNNs to learn high-dimensional operators arises from the potential repercussions that these models would have on fields such as Reduced Order Modeling.

Consider for instance a parameter dependent PDE problem, where each parameter instance $\boldsymbol{\mu}$ leads to a solution $u_{\boldsymbol{\mu}}$. In this framework, multi-query applications such as optimal control and statistical inference are prohibitive to implement, as a numerical solver has to be run any time a new solution is to be computed. Therefore, learning the operator that maps $\boldsymbol{\mu} \to u_{\boldsymbol{\mu}}$ is a task of key interest, as it would allow one to replace the numerical solver with a much cheaper surrogate. To this end, DNNs are a valid and powerful alternative that have recently shown either comparable or superior results with respect to other state-of-the-art approaches, e.g. [3,12,13]. More in general, other works have recently exploited physics-informed machine learning for efficient reduced order modeling of parametrized PDEs [7,29]. Also, DNN models have the practical advantage of being highly versatile as, differently from other techniques such as splines and wavelets, they can easily adapt to both high-dimensional inputs, as in image recognition, and outputs, as in the so-called generative models.

However, when the dimensions into play become very high, there are some practical issues that hinder the use of as-is DNN models. In fact, classical dense architectures tend to have too many degrees of freedom, making them harder to train, computationally demanding and prone to overfitting. As a remedy, alternative architectures such as Convolutional Neural Networks (CNNs) and Graph Neural Networks (GNNs) have been employed over the years. These architectures can handle very efficiently data defined respectively over hypercubes (CNNs) or graphs (GNNs). Nevertheless, these models do not provide a complete answer, especially when the high-dimensionality arises from the discretization of a functional space such as $L^2(\Omega)$, where $\Omega \subset \mathbb{R}^d$ is some bounded domain, possibly non convex. In fact, CNNs cannot handle general geometries and they might become inappropriate as soon as Ω is not an hypercube, although some preliminary attempts to generalize CNN in this direction have recently appeared [14]. Conversely, GNNs have the benefit of considering their inputs and outputs as defined over the vertices of a graph. This appears to be a promising feature, since a classical way to discretize spatial domains is to use meshing strategies, and meshes are ultimately graphs. However, GNNs are heavily based on the graph representation itself, and their construction does

not exploit the existence of an underlying spatial domain. In particular, GNNs do not work at different fidelity levels, which would be a desirable property as it would favor mesh independence (the existence of a mesh should be an auxiliary tool and not a limitation).

Inspired by these considerations, we introduce a novel class of sparse architectures, which we refer to as Mesh-Informed Neural Networks (MINNs), to tackle the problem of learning a (nonlinear) operator

$$\mathcal{G} : V \to W,$$

where V and W are some functional spaces, e.g. $V = W \subseteq L^2(\Omega)$. The definition of $\mathcal{G}$ may involve both local and nonlocal operations, such as derivatives and integrals, and it may as well imply the solution to a Partial Differential Equation (PDE).

We cast the above problem in a setting that is more familiar to the Deep Learning literature by introducing a so-called high-fidelity discretization. This is an approach that has become widely adopted by now, and it involves the discretization of the functional spaces along the same lines of Finite Element methods, as in [3,12,22]. In short, one introduces a mesh having vertices $\{\mathbf{x}_i\}_{i=1}^{N_h} \subset \overline{\Omega}$, and defines $V_h \subset L^2(\Omega)$ as the subspace of piecewise-linear Lagrange polynomials, where $h > 0$ is the stepsize of the mesh (the idea can be easily generalized to higher order finite elements, as we show later on). Since each $v \in V_h$ is uniquely identified by its nodal values, we have $V_h \cong \mathbb{R}^{N_h}$, and the original operator to be learned can be replaced by

$$\mathcal{G}_h : V_h \cong \mathbb{R}^{N_h} \to V_h \cong \mathbb{R}^{N_h}.$$

The idea is now to approximate $\mathcal{G}_h$ by training some DNN $\Phi : \mathbb{R}^{N_h} \to \mathbb{R}^{N_h}$. As we argued previously, dense architectures are unsuited for such a purpose because of their prohibitive computational cost during training, due to the computational resources needed to optimize the network as well as to the amount of training data required to avoid overfitting.

To overcome this bottleneck, we propose Mesh-Informed Neural Networks. These are ultimately based on an *apriori* pruning strategy, whose purpose is to inform the model with the geometrical knowledge coming from Ω. As we will demonstrate later in the paper, despite their simple implementation, MINNs show reduced training times and better generalization capabilities, making them able to learn operators even in poor data regimes. Also, they allow for a novel interpretation of the so-called *hidden layers* in a way that may simplify the practical problem of designing DNN architectures.

The rest of the paper is devoted to the presentation of MINNs and it is organized as follows. In Section 2, we set some notation and formally introduce Mesh-Informed Neural Networks from a theoretical point of view. There, we also discuss their implementation and comment on the parallelism between MINNs and other emerging approaches such as DeepONets [24] and Neural

Operators [20]. We then devote Section 3 to the numerical experiments, where we test MINNs against classical fully connected models, with applications to nonlinear operators and nonlinear PDEs. After validating our approach, we take the chance in Section 4 to present an application where MINNs are employed to answer a practical problem of Uncertainty Quantification related to the microcirculation of oxygen in biological tissues. Finally, we draw our conclusions and discuss future developments in Section 5.

2 Mesh-Informed Neural Networks

In the present section we present Mesh-Informed Neural Networks, a novel class of architectures specifically built to handle discretized functional outputs, and thus of particular interest in PDE applications. Preliminary to that, we introduce some notation and recall some of the basic concepts behind classical DNNs.

2.1 Notation and preliminaries

Deep Neural Networks are a powerful class of approximators that is ultimately based on the composition of affine and nonlinear transformations. Here, we focus on DNNs having a so-called *feedforward* architecture. We report below some basic definitions.

Definition 1 Let $m, n \geq 1$ and $\rho : \mathbb{R} \to \mathbb{R}$. A layer with activation ρ is a map $L : \mathbb{R}^m \to \mathbb{R}^n$ of the form $L(\mathbf{v}) = \rho(\mathbf{W}\mathbf{v} + \mathbf{b})$, for some $\mathbf{W} \in \mathbb{R}^{n \times m}$ and $\mathbf{b} \in \mathbb{R}^n$.

In the literature, $\mathbf{W}$ and $\mathbf{b}$ are usually referred to as weight and bias of the layer, respectively. Note that the definition above contains an abuse of notation, as ρ is evaluated over an n-dimensional vector: we understand the latter operation component-wise, that is $\rho([x_1, \dots, x_n]) := [\rho(x_1), \dots, \rho(x_n)]$.

Definition 2 Let $m, n \geq 1$. A neural network of depth $l \geq 0$ is a map $\Phi : \mathbb{R}^m \to \mathbb{R}^n$ obtained via composition of $l + 1$ layers, $\Phi = L_{l+1} \circ \dots L_1$.

The layers of a neural network do not need to share the same activation function and usually the output layer, L_{l+1}, does not have one. Architectures with $l = 1$ are known as shallow networks, while the adjective deep is used when $l \geq 2$. We also allow for the degenerate case in which the network reduces to a single layer ($l = 0$). The classical pipeline for building a neural network model starts by fixing the architecture, that is the number of layers and their input-output dimensions. Then, the weights and biases of all layers are tuned accordingly to some procedure, which typically involves the optimization of a loss function computed over a given training set.

2.2 Mesh-Informed layers

We consider the following framework. We are given a bounded domain $\Omega \subset \mathbb{R}^d$, not necessarily convex, and two meshes having respectively stepsizes $h, h' > 0$ and vertices

$$\{\mathbf{x}_j\}_{j=1}^{N_h},\ \{\mathbf{x}'_i\}_{i=1}^{N_{h'}} \subset \overline{\Omega}.$$

The two meshes can be completely different and they can be either structured or unstructured. To each mesh we associate the corresponding space of piecewise-linear Lagrange polynomials, namely $V_h, V_{h'} \subset L^2(\Omega)$. Our purpose is to introduce a suitable notion of *mesh-informed layer* $L : V_h \to V_{h'}$ that exploits the apriori existence of Ω. In analogy to Definition 1, L should have N_h neurons at input and $N_{h'}$ neurons at output, since $V_h \cong \mathbb{R}^{N_h}$ and $V_{h'} \cong \mathbb{R}^{N_{h'}}$. However, thinking of the state spaces as either comprised of functions or vectors is fundamentally different: while we can describe the objects in V_h as regular, smooth or noisy, these notions have no meaning in $\mathbb{R}^{N_h}$, and similarly for $V_{h'}$ and $\mathbb{R}^{N_{h'}}$. Furthermore, in the case of PDE applications, we are typically not interested in all the elements of V_h and $V_{h'}$, rather we focus on those that present spatial correlations coherent with the underlying physics. Starting from these considerations, we build a novel layer architecture that can meet our specific needs. In order to provide a rigorous definition, and directly extend the idea to higher order finite element spaces, we first introduce some preliminary notation. For the sake of simplicity, we will restrict to simplicial finite elements.

Definition 3 Let $\Omega \subset \mathbb{R}^d$ be a bounded domain. Let $\mathcal{M} = \{K_i\}_{i \in \mathcal{I}}$ be a collection of d-simplices in Ω, that is $K \subset \overline{\Omega}$ for each $K \in \mathcal{M}$. For each *element* $K \in \mathcal{M}$, define the quantities

$$h_K := \operatorname{diam}(K), \qquad R_K := \sup\left\{\operatorname{diam}(S) \mid S \text{ is a ball contained in } K\right\}.$$

We say that $\mathcal{M}$ is an admissible mesh of stepsize $h > 0$ over Ω if the following conditions hold.

1. The elements are exhaustive, that is

$$\operatorname{dist}\left(\overline{\Omega}, \bigcup_{K \in \mathcal{M}} K\right) \leq h$$

where $\operatorname{dist}(A, B) = \sup_{x \in A} \inf_{y \in B} |x - y|$ is the distance between A and B.

2. Any two distinct elements $K, K' \in \mathcal{M}$ have disjoint interiors. Also, their intersection is either empty or results in a common face of dimension $s < d$.

4. The elements are non degenerate and their maximum diameter equals h, that is

$$\min_{K \in \mathcal{M}} R_K > 0 \quad \text{and} \quad \max_{K \in \mathcal{M}} h_K = h.$$

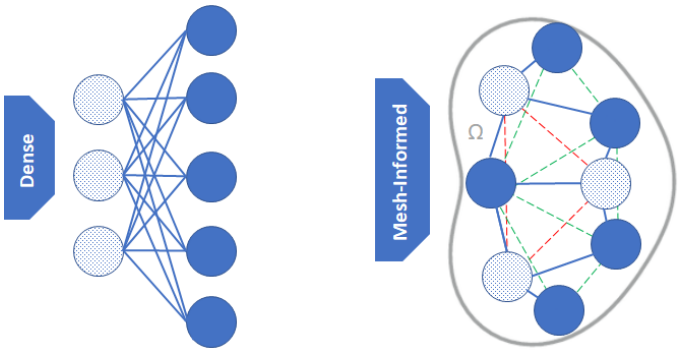

Fig. 1: Comparison of a dense layer (cf. Definition 1) and a mesh-informed layer (cf. Definition 5). The dense model features 3 neurons at input and 5 at output. All the neurons communicate: consequently the weight matrix of the layer has 15 active entries. In the mesh-informed counterpart, neurons become vertices of two meshes (resp. in green and red) defined over the same spatial domain Ω. Only nearby neurons are allowed to communicate. This results in a sparse model with only 9 active weights.

In that case, the quantity

$$\sigma = \min_{K \in \mathcal{M}} \frac{h_K}{R_K} < +\infty,$$

is said to be the aspect-ratio of the mesh.

Definition 4 Let $\Omega \subset \mathbb{R}^d$ be a bounded domain and let $\mathcal{M} = \{K_i\}_{i \in \mathcal{I}}$ be a mesh of stepsize $h > 0$ defined over Ω. For any positive integer q, we write $X_h^q(\mathcal{M})$ for the finite element space of piecewise-polynomials of degree at most q, that is

$$X_h^q(\mathcal{M}) := \{v \in \mathcal{C}(\overline{\Omega}) \text{ s.t. } v_{|K} \text{ is a polynomial of degree at most } q \ \ \forall K \in \mathcal{M}\}.$$

Let $N_h = \dim(X_h^q(\mathcal{M}))$. We say that a collection of nodes $\{\mathbf{x}_i\}_{i=1}^{N_h} \subset \Omega$ and a sequence of functions $\{\varphi_i\}_{i=1}^{N_h} \subset X_h^q(\mathcal{M})$ define a Lagrangian basis of $X_h^q(\mathcal{M})$ if

$$\varphi_j(\mathbf{x}_i) = \delta_{i,j} \qquad i, j = 1, \dots, N_h.$$

We write $\Pi_{h,q}(\mathcal{M}) : X_h^q(\mathcal{M}) \to \mathbb{R}^{N_h}$ for the function-to-nodes operator,

$$\Pi_{h,q}(\mathcal{M}) : \ \ v \to [v(\mathbf{x}_1), \dots, v(\mathbf{x}_{N_h})],$$

whose inverse is

$$\Pi_{h,q}^{-1}(\mathcal{M}) : \ \ \mathbf{c} \to \sum_{i=1}^{N_h} c_i \varphi_i.$$

We now have all we need to introduce our concept of mesh-informed layer.

Definition 5 Let $\Omega \subset \mathbb{R}^d$ be a bounded domain. Let $\mathcal{M}$ and $\mathcal{M}'$ be two meshes of stepsizes h and h' respectively. Let $V_h = X_h^q(\mathcal{M})$ and $V_{h'} = X_{h'}^{q'}(\mathcal{M})$ be the input and output spaces respectively. Denote by $\{\mathbf{x}_j\}_{j=1}^{N_h}$ and $\{\mathbf{x}'_i\}_{i=1}^{N_{h'}}$ the nodes associated to a Lagrangian basis of V_h and $V_{h'}$ respectively. A mesh-informed layer with activation function $\rho : \mathbb{R} \to \mathbb{R}$ and support $r > 0$ is a map $L : V_h \to V_{h'}$ of the form

$$L = \Pi^{-1}_{h',q'}(\mathcal{M}') \circ \tilde{L} \circ \Pi_{h,q}(\mathcal{M})$$

where $\tilde{L} : \mathbb{R}^{N_h} \to \mathbb{R}^{N'_h}$ is a layer with activation ρ whose weight matrix $\mathbf{W}$ satisfies the additional sparsity constraint below,

$$|\mathbf{x}_j - \mathbf{x}'_i| > r \implies \mathbf{W}_{i,j} = 0.$$

As a matter of fact, the projections $\Pi_{h,q}(\mathcal{M})$ and $\Pi^{-1}_{h',q'}(\mathcal{M}')$ have the sole purpose of making the definition rigorous. What actually defines the mesh-informed layer L are the sparsity patterns imposed to $\tilde{L}$. In fact, the idea is that these constraints should help the layer in producing outputs that are more coherent with the underlying spatial domain (cf. Figure 1). In light of this intrinsic duality between L and $\tilde{L}$, we will refer to the weights and biases of L as to those that are formally defined in $\tilde{L}$. Also, for better readability, from now on we will use the notation

$$L : V_h \xrightarrow{r} V_{h'},$$

to emphasize that L is a mesh-informed layer with support r. We note that dense layers can be recovered by letting $r \geq \mathrm{diam}(\Omega)$, while lighter architecture are obtained for smaller values of r. The following result provides an explicit upper bound on the number of nonzero entries in a mesh-informed layer.

Proposition 1 *Let $\Omega \subset \mathbb{R}^d$ be a bounded domain. Let $\mathcal{M}$ and $\mathcal{M}'$ be two meshes having respectively stepsizes h, h' and aspect-ratios σ, σ'. Let*

$$h_{\min} := \min_{K \in \mathcal{M}} h_K, \quad h'_{\min} := \min_{K' \in \mathcal{M}'} h_{K'}$$

be the smallest diameters within the two meshes respectively. Let $L : V_h \xrightarrow{r} V_{h'}$ be a mesh-informed layer of support $r > 0$, where $V_h := X_h^q(\mathcal{M})$ and $V_{h'} := X_{h'}^{q'}(\mathcal{M}')$. Then,

$$\|\mathbf{W}\|_0 \leq C \left(\sigma\sigma' \frac{r}{h_{\min} h'_{\min}} \right)^d$$

where $\|\mathbf{W}\|_0$ is the number of nonzero entries in the weight matrix of L, while $C = C(\Omega, d, q, q') > 0$ is a constant depending only on Ω, d, q and q'.

Proof Let $N_h := \dim(V_h)$, $N_{h'} := \dim(V_{h'})$ and let $\{\mathbf{x}_j\}_{j=1}^{N_h}, \{\mathbf{x}'_i\}_{i=1}^{N_{h'}}$ be the Lagrangian nodes in the two meshes respectively. Let $\omega := |B(\mathbf{0}, 1)|$ be the volume of the unit ball in $\mathbb{R}^d$. Since $\min_{K'} R_{K'} \geq h'_{\min}/\sigma'$, the volume of an element in the output mesh is at least

$$v_{\min}(h') := (h'_{\min}/\sigma')^d \omega.$$

It follows that, for any $\mathbf{x} \in \Omega$, the ball $B(\mathbf{x}, r)$ can contain at most

$$n_e(r, h') := \frac{\omega r^d}{v_{\min}(h')} = \left(\frac{\sigma' r}{h'_{\min}}\right)^d$$

elements of the output mesh. Therefore, the number of indices i such that $|\mathbf{x}'_i - \mathbf{x}_j| \leq r$ is at most $n_e(r, h')c(d, q')$, where $c(d, q') := (d+q')!/(q'!d!)$ bounds the number of degrees of freedom within each element. Finally,

$$\begin{aligned} \|\mathbf{W}\|_0 &\leq N_h n_e(r, h')c(d, q') \leq \\ &\leq \frac{c(d, q)|\Omega|}{v_{\min}(h)} n_e(r, h')c(d, q') = \frac{c(d, q)c(d, q')|\Omega|}{\omega} \cdot \left(\frac{\sigma\sigma' r}{h_{\min} h'_{\min}}\right)^d \end{aligned}$$

□

Starting from here, we define MINNs by composition, with a possible interchange of dense and mesh-informed layers. Consider for instance the case in which we want to define a Mesh-Informed Neural Network $\Phi : \mathbb{R}^p \to V_h \cong \mathbb{R}^{N_h}$ that maps a low-dimensional input, say $p \ll N_h$, to some functional output. Then, using our notation, one possible architecture could be

$$\Phi : \mathbb{R}^p \longrightarrow \begin{array}{c} V_{4h} \\ \cong \\ \mathbb{R}^{N_{4h}} \end{array} \xrightarrow{r = 0.5} \begin{array}{c} V_{2h} \\ \cong \\ \mathbb{R}^{N_{2h}} \end{array} \xrightarrow{r = 0.25} \begin{array}{c} V_h \\ \cong \\ \mathbb{R}^{N_h} \end{array} , \tag{1}$$

The above scheme defines a DNN of depth $l = 2$, as it is composed of 3 layers. The first layer is dense (Definition 1) and has the purpose of preprocessing the input while mapping the data onto a coarse mesh (stepsize $4h$). Then, the remaining two layers perform local transformations in order to return the desired output. Note that the three meshes need not to satisfy any hierarchy. Also, the corresponding finite element spaces need not to share the same polynomial degree. Clearly, (1) can be generalized by employing any number of layers, as well as any sequence of stepsizes $h_1, \ldots, h_n$ and supports $r_1, \ldots, r_{n-1}$. The choice of the hyperparameters remains problem specific, but a good rule of thumb is to decrease the supports as the mesh becomes finer, so that the network complexity is kept under control (cf. Proposition 1).

2.3 Implementation details

The practical implementation of mesh-informed layers is straightforward and can be done as follows. Given $\Omega \subset \mathbb{R}^d$, $h, h' > 0$, let $\mathbf{X} \in \mathbb{R}^{N_h \times d}$ and $\mathbf{X'} \in \mathbb{R}^{N_{h'} \times d}$ be the matrices containing the degrees of freedom associated to the chosen finite element spaces, that is $\mathbf{X}_{j,\cdot} := [X_{j,1}, \dots, X_{j,d}]$ are the coordinates of the jth node in the input mesh, and similarly for $\mathbf{X'}$. In order to build a mesh-informed layer of support $r > 0$, we first compute all the pairwise distances $D_{i,j} := |\mathbf{X}_{j,\cdot} - \mathbf{X}'_{i,\cdot}|^2$ among the nodes in the two meshes. This can be done efficiently using tensor algebra, e.g.

$$D = \sum_{l=1}^{d} (e_{N_{h'}} \otimes \mathbf{X}_{\cdot,l} - \mathbf{X}'_{\cdot,l} \otimes e_{N_h})^{\circ 2}$$

where $\mathbf{X}_{\cdot,l}$ is the lth column of $\mathbf{X}$, $e_k := [1, \dots, 1] \in \mathbb{R}^k$, $\otimes$ is the tensor product and $\circ 2$ is the Hadamard power. We then extract the indices $\{(i_k, j_k)\}_{k=1}^{\mathrm{dof}}$ for which $D_{i_k,j_k} \le r^2$ and initialize a weight vector $\mathbf{w} \in \mathbb{R}^{\mathrm{dof}}$. This allows us to declare the weight matrix $\mathbf{W}$ in sparse format by letting the nonzero entries be equal to $\mathbf{w}_k$ at position (i_k, j_k), so that $\|\mathbf{W}\|_0 = \mathrm{dof}$. Preliminary to the training phase, we fill the values of $\mathbf{w}$ by considering an adaptation of the so-called He initialization. More precisely, we sample independently the values $w_1, \dots, w_{\mathrm{dof}}$ from a normal distribution having mean zero and variance $1/\|\mathbf{W}\|_0$.

2.4 How MINNs relate to existing Deep Learning literature

It is worth to comment on the differences and similarities that MINNs share with other Deep Learning approaches. We discuss them below.

2.4.1 Relationship to CNNs and GNNs

Mesh-Informed architectures can work at different resolutions simultaneously, in a way that is very similar to CNNs. However, their construction comes with multiple advantages. First of all, Definition 5 adapts to any geometry, while convolutional layers typically operate on square or cubic input-output. Furthermore, convolutional layers use weight sharing, meaning that all parts of the domain are treated in the same way. This may not be an optimal choice in some applications, such as those involving PDEs, as we may want to differentiate our behavior over Ω (for instance near of far away from the boundaries).

Conversely, MINNs share with GNNs the ability to handle general geometries. As a matter of fact, we mention that these architectures have been recently applied to mesh-based data, see e.g. [10,17,28,31]. With respect to GNNs, the main advantage of MINNs lies in their capacity to work at different fidelity levels. This fact, which essentially comes from the presence of an underlying spatial domain, has at least two advantages: it favors resolution

independence and it increases the interpretability of hidden layers (now the number of neurons is not arbitrary but comes from the chosen discretization). In this sense, MINNs exploit meshing strategies as auxiliary tools and they appear to be a natural choice for learning discretized functional outputs.

2.4.2 Relationship to DeepONets and Neural Operators

Recently, some new DNN models have been proposed for operator learning. One of these are DeepONets [24], a mesh-free approach that is based on an explicit decoupling between the input and the space variable. More precisely, DeepONets consider a representation of the following form

$$(\mathcal{G}_h f)(\mathbf{x}) \approx \Psi(f) \cdot \phi(\mathbf{x}),$$

where $\cdot$ is the dot product, $\Psi : V_h \to \mathbb{R}^m$ is the so-called trunk-net, and $\phi : \Omega \to \mathbb{R}^m$ is the branch-net. DeepONets have been shown capable of learning nonlinear operators and are now being extended to include apriori physical knowledge, see e.g. [29]. We consider MINNs and DeepONets as two fundamentally different approaches that answer different questions. DeepONets were originally designed to process input data coming from sensors and, being mesh-free, they are particularly suited for those applications where the output is only partially known or observed. In contrast, MINNs are rooted on the presence of a high fidelity model $\mathcal{G}_h$ and are thus better suited for tasks such as reduced order modeling. Another difference lies in the fact that DeepONets include explicitly the dependence on the space variable $\mathbf{x}$. This can then make it harder to process general domains and include additional information such as boundary data. Conversely, MINNs can easily handle this kind of issues thanks to their global perspective.

In this sense, MINNs are much closer to the so-called Neural Operators, a novel class of DNN models recently proposed by Kovachki et al. [20]. Neural Operators are an extension of classical DNNs that was developed to operate between infinite dimensional spaces. These models are ultimately based on Hilbert-Schmidt operators, meaning that their linear part, that is ignoring activations and biases, is of the form

$$W : f \to \int_\Omega k(\cdot, \mathbf{x}) f(\mathbf{x}) d\mathbf{x} \tag{2}$$

where $k : \Omega \times \Omega \to \mathbb{R}$ is some *kernel* function that has to be identified during the training phase. Clearly, the actual implementation of Neural Operators is carried out in a discrete setting and integrals are replaced with suitable quadrature formulas.

The construction of Neural Operators is of full generality, to the point that other approaches such as DeepONets [24] can be seen as a special case. Similarly, we note that MINNs also form a special class of Neural Operators. Indeed, a rough Monte Carlo-type estimate of (2) would yield

$$(Lf)(\mathbf{x}'_i) = \int_\Omega k(\mathbf{x}'_i, \mathbf{x}) f(\mathbf{x}) d\mathbf{x} \approx \frac{|\Omega|}{N_h} \sum_j k(\mathbf{x}'_i, \mathbf{x}_j) f(\mathbf{x}_j).$$

If we let $\{\mathbf{x}_j\}_j$ and $\{\mathbf{x}'_i\}_i$ be the nodes in the two meshes, then the constraint in Definition 5 becomes equivalent to the requirement that k is supported somewhere near the diagonal, that is $\text{supp}(k) \subseteq \{(\mathbf{x}, \mathbf{x} + \varepsilon) \text{ with } |\varepsilon| \leq r\}$.

We believe that these parallels are extremely valuable, as they indicate the existence of a growing scientific community with common goals and interests.

3 Numerical experiments

We provide empirical evidence that the sparsity introduced by MINNs can be of great help in learning maps that involve functional data, such as nonlinear operators, showing a reduced computational cost and better generalization capabilities. We first detail the setting of each experiment alone, specifying the corresponding operator to be learned and the adopted MINN architecture. Then, at the end of the current Section, we discuss the numerical results.

Throughout all our experiments, we adopt a standardized approach for designing and training the networks. In general, we always employ the 0.1-leakyReLU activation for all the hidden layers, while we do not use any activation at the output. Every time a mesh-informed architecture is introduced, we also consider its dense counterpart, obtained without imposing the sparsity constraints. Both networks are then trained following the same criteria, so that a fair comparison can be made. As loss function, we always consider the mean square error computed with respect to the L^2 norm, that is

$$\mathbb{E}_{f\sim\mathbb{P}} \|\mathcal{G}_h f - \Phi(f)\|^2_{L^2(\Omega)} \tag{3}$$

where $\mathcal{G}_h$ is the (discretized) operator to be learned and Φ is the DNN model. Here, $\mathbb{P}$ is some given probability distribution over the input space Θ, which we allow to be either finite or infinite dimensional.

The optimization of the loss function is performed via the L-BFGS optimizer, with learning rate always equal to 1 and no batching. What may change from case to case are the network architecture, the number of epochs, and the size of the training set. After training, we compare mesh-informed and dense architectures by evaluating their performance on 500 unseen samples (test set), which we use to compute an unbiased estimate of the relative error below,

$$\mathbb{E}_{f\sim\mathbb{P}} \left[\frac{\|\mathcal{G}_h f - \Phi(f)\|_{L^2(\Omega)}}{\|\mathcal{G}_h f\|_{L^2(\Omega)}} \right]. \tag{4}$$

All the code was written in Python 3, mainly relying on Pytorch for the implementation of DNNs. Instead, we employed the FEniCS library for defining meshes and, when needed, solving PDEs.

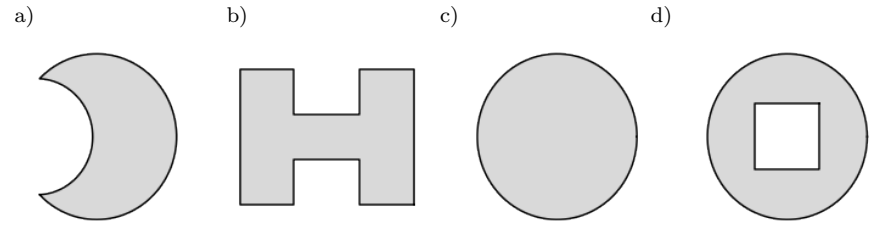

Fig. 2: Domains considered for the numerical experiments in Section 3. Panel a): upto boundaries, Ω is the difference of two circles, $B(\mathbf{0},1)$ and $B(\mathbf{x}_0,0.7)$, where $\mathbf{x}_0 = (-0.75,0)$. Panel b): A polygonal domain obtained by removing the rectangles $[-0.75,0.75]\times[0.5,1.5]$ and $[-0.75,0.75]\times[-1.5,-0.5]$ from $(-2,2)\times(-1.5,1.5)$. Panel c): the unit circle $B(\mathbf{0},1)$. Panel d): Ω is obtained by removing a square, namely $[-0.4,0.4]^2$, from the unit circle $B(\mathbf{0},1)$.

3.1 Description of the benchmark operators

Learning a parametrized family of functions

Let Ω be the domain defined as in Figure 2a. For our first experiment, we consider a variation of a classical problem concerning the calculation of the so-called *signed distance function* of Ω. This kind of functions are often encountered in areas such as computer vision [26] and real-time rendering [1]. In particular, we consider the following operator,

$$\mathcal{G} : \Theta \subset \mathbb{R}^3 \to L^2(\Omega)$$

$$\mathcal{G} : \boldsymbol{\mu} \to u_{\boldsymbol{\mu}}(\mathbf{x}) \coloneqq \min_{\substack{\mathbf{y}\in\partial\Omega,\\ y_2>\mu_1}} |\mathbf{y} - \mathbf{A}_{\mu_3}\mathbf{x}|\mathrm{e}^{x_1\mu_2}$$

where $\boldsymbol{\mu} = (\mu_1,\mu_2,\mu_3)$ is a finite dimensional vector, and $\mathbf{A}_{\mu_3} = \mathrm{diag}(1,\mu_3)$. In practice, the value of $u_{\boldsymbol{\mu}}(\mathbf{x})$ corresponds to the (weighted) distance between the dilated point $\mathbf{A}_{\mu_3}\mathbf{x}$ and the truncated boundary $\partial\Omega\cap\{\mathbf{y}:\ y_2>\mu_1\}$.

As input space we consider $\Theta \coloneqq [0,1]\times[-1,1]\times[1,2]$, endowed with the uniform probability distribution. Since the input is finite-dimensional, we can think of $\mathcal{G}$ as to the parametrization of a 3-dimensional hypersurface in $L^2(\Omega)$. We discretize Ω using P1 triangular finite elements with mesh stepsize $h = 0.02$, resulting in the high-fidelity space $V_h \coloneqq\cong \mathbb{R}^{13577}$. To learn the discretized operator $\mathcal{G}_h$, we employ the following MINN architecture

$$\mathbb{R}^3 \to \mathbb{R}^{100} \to V_{9h} \xrightarrow{r=0.4} V_{3h} \xrightarrow{0.2} V_h.$$

The corresponding dense counterpart, which servers as benchmark, is obtained by removing the sparsity constraints (equivalently, by letting the supports go to infinity). We train the networks on 50 samples and for a total of 50 epochs.

Learning a local nonlinear operator

As a second experiment, we learn a nonlinear operator that is local with respect to the input. Let Ω be as in Figure 2b. We consider the *infinitesimal area* operator $\mathcal{G} : H^1(\Omega) \to L^2(\Omega)$,

$$\mathcal{G} : u \to \sqrt{1 + |\nabla u|^2}.$$

Note in fact that, if we associate to each $u \in H^1(\Omega)$ the cartesian surface

$$S_u := \{(x_1, x_2, u(x_1, x_2)\} \subset \mathbb{R}^3,$$

then $\mathcal{G}u$ yields a measure of the area that is locally spanned by that surface, in the sense that

$$\int_{S_u} \phi(\mathbf{s}) d\mathbf{s} = \int_{\Omega} \phi(u(\mathbf{x}))\,(\mathcal{G}u)\,(\mathbf{x}) d\mathbf{x}$$

for any continuous map $\phi : S_u \to \mathbb{R}$. Over the input space $\Theta := H^1(\Omega)$ we consider the probability distribution $\mathbb{P}$ induced by a Gaussian process with mean zero and covariance kernel

$$\mathrm{Cov}(\mathbf{x}, \mathbf{y}) = \frac{1}{|\Omega|} \exp\left(-\frac{1}{2}|\mathbf{x} - \mathbf{y}|^2\right).$$

We discretize Ω using a triangular mesh of stepsize $h = 0.045$ and P1 finite elements, which results in a total of $N_h = 11266$ vertices. To sample from the Gaussian process we truncate its Karhunen-Loeve expansion at $k = 100$. Conversely, the output is computed numerically by exploiting the high-fidelity mesh as a reference.

To learn $\mathcal{G}_h$ we use the MINN architecture below,

$$V_h \xrightarrow{r=0.15} V_{3h} \xrightarrow{r=0.3} V_{3h} \xrightarrow{r=0.15} V_h,$$

which we train for 50 epochs over 500 snapshots.

Learning a nonlocal nonlinear operator

Since MINNs are based on local operations, it is of interest to assess whether they can also learn nonlocal operators. To this end, we consider the problem of learning the so-called Hardy-Littlewood Maximal Operator $\mathcal{G} : L^2(\Omega) \to L^2(\Omega)$,

$$(\mathcal{G}f)\,(\mathbf{x}) := \sup_{r>0} \fint_{|\mathbf{y}-\mathbf{x}|<r} |f(\mathbf{y})| d\mathbf{y}$$

which is known to be a continuous nonlinear operator from $L^2(\Omega)$ onto itself [25]. Here, we let $\Omega := B(\mathbf{0}, 1) \subset \mathbb{R}^2$ be the unit circle. Over the input space $\Theta := L^2(\Omega)$ we consider the probability distribution $\mathbb{P}$ induced by a Gaussian process with mean zero and covariance kernel

$$\mathrm{Cov}(\mathbf{x}, \mathbf{y}) = \exp\left(-|\mathbf{x} - \mathbf{y}|^2\right).$$

As a high-fidelity reference, we consider a discretization of Ω via P1 triangular finite elements of stepsize $h = 0.033$, resulting in a state space V_h with $N_h = 7253$ degrees of freedom. As for the previous experiment, we sample from $\mathbb{P}$ by considering a truncated Karhunen-Loeve expansion of the Gaussian process (100 modes). Conversely, the true output $u \to \mathcal{G}_h(u)$ is computed approximately by replacing the suprema over r with a maxima across 50 equispaced radii in $[h, 2]$. To learn $\mathcal{G}_h$ we use a MINN of depth 2 with a dense layer in the middle,

$$V_h \xrightarrow{r=0.25} V_{2h} \to V_{2h} \xrightarrow{r=0.25} V_h.$$

The idea is that nonlocality can still be enforced through the use of fully connected blocks, but this are only inserted at the lower fidelity levels to reduce the computational cost. We train the architectures over 500 samples and for a total of 50 epochs.

Learning the solution operator of a nonlinear PDE

For our final experiment, we consider the case of a parameter dependent PDE, which is a framework of particular interest in the literature of Reduced Order Modeling. In fact, learning the solution operator of a PDE model by means of neural networks allows one to replace the original numerical solver with a much cheaper and efficient surrogate, which enables expensive multi-query tasks such as PDE constrained optimal control, Uncertainty Quantification or Bayesian Inversion.

Here, we consider a steady version of the so-called porous media equation, defined as follows

$$-\nabla \cdot \left(|u|^2 \nabla u\right) + u = f.$$

The PDE is understood in the domain Ω defined in Figure 2d, and it is complemented with homogeneous Neumann boundary conditions. We define $\mathcal{G}$ to be the data-to-solution operator that maps $f \to u$. This time, we endow the input space with the push-forward distribution $\#\mathbb{P}$ induced by the square map $f \to f^2$, where $\mathbb{P}$ is the probability distribution associated to a Gaussian random field with mean zero and covariance kernel

$$\mathrm{Cov}(\mathbf{x}, \mathbf{y}) = \frac{1}{1 + |\mathbf{x} - \mathbf{y}|^2}.$$

To sample from the latter distribution we exploit a truncated Karhunen-Loeve expansion of the random field. We set the truncation index to $k = 20$ as that is sufficient to fully capture the volatility of the field. For the high-fidelity discretization, we consider a mesh of stepsize $h = 0.03$ and P1 finite elements, resulting in $N_h = 5987$. Finally, we employ the MINN architecture below,

$$V_h \xrightarrow{r=0.4} V_{4h} \to V_{2h} \xrightarrow{r=0.2} V_h,$$

which we train over 500 snapshots and for a total of 100 epochs. Note that, as in our third experiment, we employ a dense block at the center of the architecture.

Operator	N_h	Architecture	Test error	Gen. error
Low-dimensional manifold	13'577	Mesh-Informed Dense	4.14% 4.78%	3.08% 4.07%
Local area operator	11'266	Mesh-Informed Dense	1.49% 3.89%	1.36% 2.54%
H-L maximal operator	7'253	Mesh-Informed Dense	3.65% 6.70%	1.49% 3.09%
Nonlinear PDE solver	5'987	Mesh-Informed Dense	4.29% 18.53%	3.03% 6.56%

Table 1: Comparison of Mesh-Informed Neural Networks and Fully Connected DNNs for the test cases in Section 3. N_h = number of vertices in the (finest) mesh. Gen. error = Generalization error, defined as the gap between training and test errors. All reported errors are intended with respect to the L^2-norm, see Equation (4).

This is because the solution operator to a boundary value problem is typically nonlocal (consider for instance the so-called Green formula for the Poisson equation).

3.2 Numerical results

Table 1 reports the numerical results obtained across the four experiments. In general, MINNs perform better with respect to their dense counterpart, with relative errors that are always below 5%. As the operator to be learned becomes more and more involved, fully connected DNNs begin to struggle, eventually reaching an error of 18% in the PDE example. In contrast, MINNs are able to keep up and maintain their performance. This is also due to the fact that, having less parameters, MINNs are unlikely to overfit, instead they can generalize well even in poor data regimes (cf. last column of Table 1). Consider for instance the first experiment, which counted as little as 50 training samples. There, the dense model returns an error of 4.78%, of which 4.07% is due to the generalization gap. This means that the DNN model actually surpassed the MINN performance over the training set, as their training errors are respectively of 4.78%−4.07% = 0.71% and 4.14%−3.08% = 1.06%. However, the smaller generalization gap allows the sparse architecture to perform better on unseen inputs.

Figures 3 to 6 reports some examples of approximation on unseen input values. There, we note that dense models tend to have noisy outputs (Figures 4 and 6) and often miscalculate the range of values spanned by the output (Figures 3 and 5). Conversely, MINNs always manage to capture the main features present in the actual ground truth.

Nonetheless, Mesh-Informed Neural Networks also allow for a significant reduction in the computational cost, as reported in Table 2. In general, MINNs

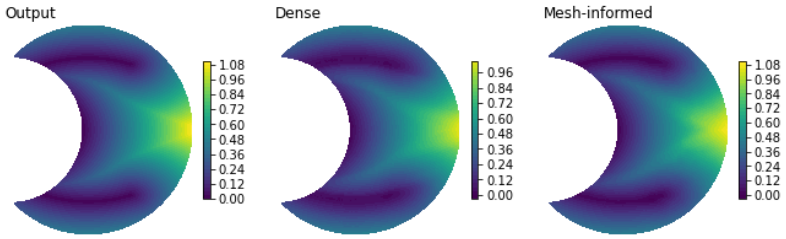

Fig. 3: Comparison of DNNs and MINNs when learning a low-dimensional manifold $\boldsymbol{\mu} \to u_{\boldsymbol{\mu}} \in L^2(\Omega)$, cf. Section 3.1. The reported results correspond to the approximations obtained on an unseen input value $\boldsymbol{\mu}^* = [0.42, 0.04, 1.45]$.

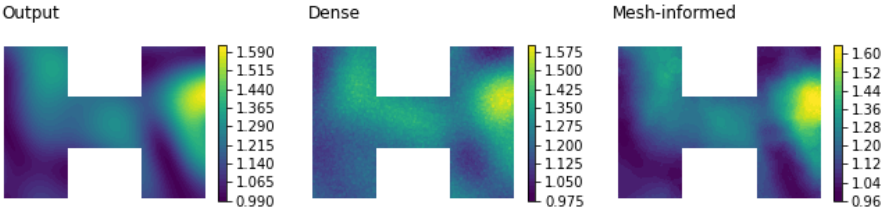

Fig. 4: Comparison of DNNs and MINNs when learning the local operator $u \to \sqrt{1 + |\nabla u|^2}$, cf. Section 3.1. The reported results correspond to the approximations obtained for an input instance outside of the training set.

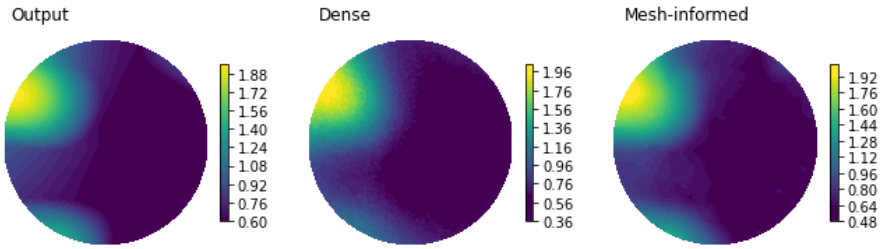

Fig. 5: Comparison of DNNs and MINNs when learning the Hardy-Littlewood Maximal Operator, cf. Section 3.1. The pictures correspond to the results obtained for an unseen input instance.

are ten to a hundred times lighter with respect to fully connected DNNs. While this is not particularly relevant once the architecture is trained (the most heavy DNN weights as little as 124 Megabytes), it makes a huge difference during the training phase. In fact, additional resources are required to optimize a DNN model, as one needs to keep track of all the operations and gradients in order to perform the so-called *backpropagation* step. This poses a significant limitation to the use of dense architectures, as the entailed computational cost can easily exceed the capacity of modern GPUs. For instance, in our experiments, fully connected DNNs required more than 2 GB of memory during training, while, depending on the operator to be learned, 10 to 250 MB were suffi-

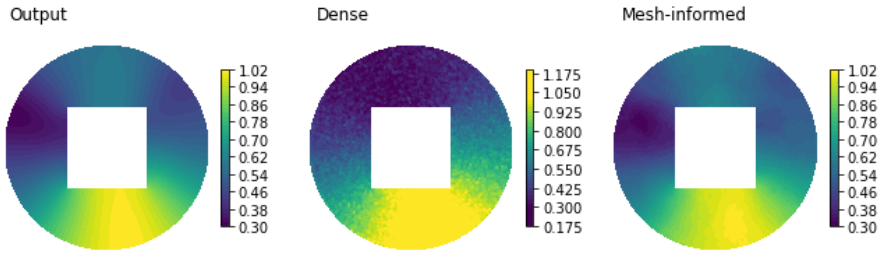

Fig. 6: Comparison of DNNs and MINNs when learning the solution operator $f \to u$ of a nonlinear PDE, cf. Section 3.1. The reported results correspond to the approximations obtained for an input instance f outside of the training set.

Architecture	dof	Training speed	Memory usage (static)	(training)
Mesh-Informed	0.3M	31.5 s/ep	1.3 Mb	0.01 Gb
Dense	21.7M	123.5 s/ep	86.9 Mb	3.49 Gb
Mesh-Informed	0.3M	60.5 s/ep	1.0 Mb	0.04 Gb
Dense	31.0M	195.3 s/ep	124.1 Mb	4.96 Gb
Mesh-Informed	1.0M	33.9 s/ep	3.9 Mb	0.16 Gb
Dense	12.8M	83.0 s/ep	51.2 Mb	2.05 Gb
Mesh-Informed	1.4M	70.0 s/ep	5.4 Mb	0.22 Gb
Dense	12.5M	130.7 s/ep	50.0 Mb	2.00 Gb

Table 2: Comparison of Mesh-Informed Neural Networks and Fully Connected DNNs in terms of their computational cost. dof = degrees of freedom, i.e. number of parameters to be optimized during the training phase. Memory usage (static) = bytes required to store the architecture. Memory usage (optimization) = bytes required to run a single epoch of the training phase. s/ep = seconds per epoch, M = millions, Mb = Megabytes, Gb = Gigabytes.

cient for MINNs. Clearly, one could also alleviate the computational burden by exploiting cheaper optimization routines, such as first order optimizers and batching strategies, however this typically prevents the network from actually reaching the global minimum of the loss function. In fact, we recall that the optimization of a DNN architecture is, in general, a non-convex and ill-posed problem. Of note, despite being 10 to 100 times lighter, MINNs are only 2 to 4 times faster during training. We believe that these results can be improved, possibily by optimizing the code used to implement MINNs.

4 An application to Uncertainty Quantification

We finally consider an application to Uncertainty Quantification (UQ) involving a partial differential equation. UQ is an essential aspect of robust modelling, which often involves expensive numerical and statistical routines. In

this Section, we provide an example on how MINNs can alleviate these costs by serving as model surrogates in the computational pipeline. In particular, starting from a suitable PDE model, we address a problem concerning oxygen transfer in biological tissues.

4.1 Model description

Oxygen is a fundamental constituent of most biological processes. In humans, oxygen is delivered by the circulatory system from the lungs to the rest of the body. At the small scales, cells receive oxygen from the vascular network of capillaries that spread all over the body. An efficient oxygen transfer is fundamental to ensure a healthy micro-environment and abnormal values in oxygen concentration are often associated to pathological scenarios. In particular, low oxygen supply, the so-called *hypoxia*, plays an important role in the development and treatment of tumors. It has been shown that hypoxic tissue opposes a resistance to chemotherapy and radiotherapy [6,27]. These issues are caused by perturbed properties of the tumor blood vessels in terms of morphology and phenotype. Here, we aim at developing a methodology to assess the role of vascular morphology on tissue hypoxia. More precisely, we wish to address the following question: how does the topology of the vascular network relate to the size of the tissue under hypoxia? We answer this question in the simplified setting that we describe below.

Within an idealized setting, we consider a portion of a vascularized tissue $\Omega := \{\mathbf{x} \in \mathbb{R}^2 : |x| < 1\}$. Let $\Lambda \subset \overline{\Omega}$ be a graph representing the vascular network of capillaries (cf. Figure 7) and let $u : \Omega \to [0,1]$ be the oxygen concentration in the tissue, normalized to the unit value. We model the oxygen transfer from the network to the tissue with the following equations,

$$\begin{cases} -\alpha \Delta u + u = (1-u)\delta_\Lambda & \text{in } \Omega \\ -\alpha \nabla u \cdot \mathbf{n} = \beta u & \text{on } \partial\Omega \end{cases}$$

where $\alpha = 0.1$ and $\beta = 0.01$ are respectively a fixed diffusion and resistance coefficient, while δ_Λ is the unique singular measure for which

$$\int_\Omega v(\mathbf{x})\delta_\Lambda(d\mathbf{x}) = \frac{1}{|\Lambda|}\int_\Lambda v(\mathbf{s})d\mathbf{s}$$

for all $v \in \mathcal{C}(\overline{\Omega})$. Here, we denote by $|\Lambda| := \int_\Lambda 1 d\mathbf{s}$ the total length of the vascular graph. The first equation in (4.1) describes the diffusion and consumption of oxygen, balanced accordingly to the amount released from the vascular network on the right hand side. Finally, the model is closed using resistance boundary conditions of Robin type.

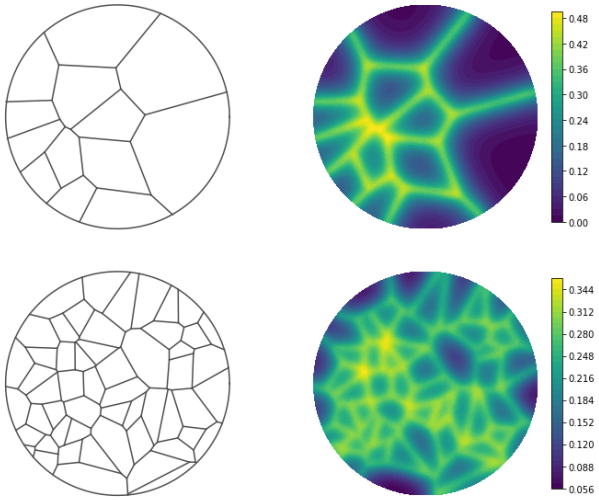

Fig. 7: Forward UQ problem (Section 4). Topology of the microvascular network Λ (left) and corresponding oxygen distribution in the tissue $u = u_\Lambda$ (right). The top and the bottom rows corresponds respectively to a poorly and a highly vascularized tissue (resp. $\lambda = 1$ and $\lambda = 3$). Globally, the two networks provide the same amount of oxygen (cf. Equation 5), but their topology significantly affects the values of u in the tissue. In the first case (top row), nearly 31% of the tissue has an oxygen level below the threshold value $u_* := 0.1$. Conversely, only 3% of the tissue reports a low oxygen concentration in the second example.

We understand (4.1) in the weak sense, meaning that define $u = u_\Lambda$ as the unique solution to the problem below

$$\int_\Omega \alpha \nabla u(\mathbf{x}) \cdot \nabla v(\mathbf{x}) d\mathbf{x} + \int_\Omega u(\mathbf{x}) v(\mathbf{x}) d\mathbf{x} + \int_{\partial\Omega} \beta u(\mathbf{s}) v(\mathbf{s}) d\mathbf{s} = \\ = \frac{1}{|\Lambda|} \int_\Lambda (1 - u(\mathbf{s})) v(\mathbf{s}) d\mathbf{s} \quad (5)$$

where the above is to be satisfied for all $v \in H^1(\Omega)$.

4.2 Uncertainty quantification setting

As we mentioned previously, we are interested in the relationship between Λ and u. To this end, we introduce the parameter space

$$\Theta := \{\Lambda \subset \overline{\Omega} : \ \Lambda \text{ is the union of finitely many segments}\}$$

which consists of all vascular networks. Note that, due to the normalizing factor $1/|\Lambda|$ in (5), all the vascular networks actually provide the same global amount of oxygen. However, as we will see later on, only those vascular graphs that

are sufficiently spread across the domain can ensure a proper oxygen supply to the whole tissue (cf. Figure 7). In other words, we explore the influence of the distribution of the network, rather than its density, on the oxygen level.

The next subsection is devoted to prescribing a suitable discretization of (5) to work with, and to introduce a class of probability measures $\{\mathbb{P}_\lambda\}_\lambda$ defined over Θ. The idea is the following. We will use a macro-scale parameter λ to describe the general perfusion of the tissue. Higher values of λ will correspond to a highly vascularized tissue. This means that the topology of the vascular network will still be uncertain, but the corresponding probability distribution $\mathbb{P}_\lambda$ will favor dense graphs. Conversely, lower values of λ will describe scenarios where capillaries are more sparse (see Figure 7, top vs bottom row). This will then bring us to consider the family of random variables

$$Q_\lambda := \frac{1}{|\Omega|}|\{u_\Lambda < 0.1\}| \text{ with } \Lambda \sim \mathbb{P}_\lambda, \tag{6}$$

that measure the portion of the tissue under the oxygen threshold 0.1, which we take as the value under which hypoxia takes place. Our interest will be to estimate the probability density function of each Q_λ and to provide a robust approximation of their expected value $\mathbb{E}[Q_\lambda]$. While these tasks can be achieved using classical Monte Carlo, the computational cost is enormous as it implies solving equation (5) repeatedly. To alleviate this burden, we will replace the original PDE solver with a suitable MINN architecture trained to learn the map $\Lambda \to u_\Lambda$.

4.3 Discretization and implementation details

For the random generation of vascular networks we exploit Voronoi diagrams [2]. Let $\mathcal{P} := \{P \subset \Omega \mid P \text{ finite}\}$ be the collection of all points tuples in Ω. To any $P \in \mathcal{P}$, we associate the vascular graph $\Lambda(P)$ defined by the edges of the Voronoi cells generated by P. In this way, we obtain a correspondence $\mathcal{P} \to \Theta$ given by $P \to \Lambda(P)$, that we can exploit to prescribe probability measures over Θ. To this end, let $\lambda > 0$ and let X_λ be a Poisson point process over Ω having a uniform intensity of 10λ. We denote by $\tilde{\mathbb{P}}_\lambda$ the probability measure induced by X_λ over $\mathcal{P}$. Then, we define $\mathbb{P}_\lambda := \#\tilde{\mathbb{P}}_\lambda$ as the push-forward measure obtained via the action $P \to \Lambda(P)$. This ensures the wished behavior: higher values of λ tend to generate more points in the domain and, consequently, denser graphs.

We now proceed to discretize the variational problem. As a first step, we note that the vascular graph Λ is not given in terms of a parametrization, which makes it harder to compute integrals of the form $\int_\Lambda v(\mathbf{s})d\mathbf{s}$. As an alternative, we consider the smoothed approximation below,

$$\int_\Lambda v(\mathbf{s})d\mathbf{s} \approx \int_\Omega v(\mathbf{x})\phi_\Lambda(\mathbf{x})d\mathbf{x}, \tag{7}$$

where

$$\phi_\Lambda := \frac{1}{\epsilon^2}\max\{\epsilon - \text{dist}(\mathbf{x}, \Lambda), 0\}.$$

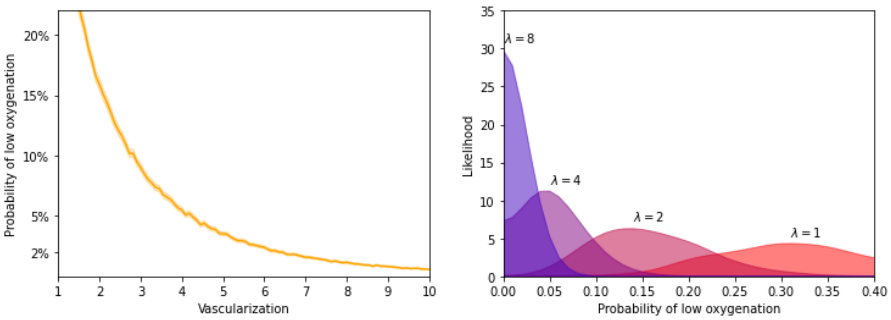

Fig. 8: Results for the UQ problem in Section 4. Left panel: expected probability of low oxygenation, $\mathbb{E}[Q_\lambda]$, as a function of the vascularization level λ. Confidence bands are computed pointwise using a 99% confidence level. Right panel: probability distribution of Q_λ for different values of λ. Colors fade from red to purple as λ grows.

Here, $\epsilon > 0$ is a smoothness parameter that we fix to $\epsilon = 0.05$. It is not hard to prove that, for each $v \in \mathcal{C}(\overline{\Omega})$ fixed, the right-hand side of equation (7) converges to the left hand-side as $\epsilon \to 0$. We refer to the appendix for a detailed proof. Then, our operator of interest becomes $\mathcal{G} : \phi_\Lambda \to u_\Lambda$, and we can proceed with our usual discretization via P1 Finite Elements. To this end, we discretize the domain using a triangular mesh of stepsize $h = 0.03$, which results in $N_h = 7253$ degrees of freedom. Then, we allow λ to vary uniformely in [1,10] and we generate a total of 4500 training snapshots accordingly to the probability distributions introduced previously. We exploit these snapshots to train the MINN model below,

$$V_h \xrightarrow{r=0.1} V_{3h} \to V_{3h} \xrightarrow{r=0.1} V_h,$$

where the architecture has been defined in analogy to the one employed for the nonlinear PDE in Section 3.1. The network is trained for a total of 50 epochs and using the same criteria presented in Section 3.

4.4 Results

Once trained, the Mesh-Informed Neural Network reported an average L^2-error of 4.99%, with errors below 10% for 488 out of 500 test instances. We considered these results satisfactory and we proceeded to sample a total of 100'000 solutions using our DNN model. More precisely, we considered 100 equally spaced values of λ in [1,10], and for each of those we sampled 1'000 independent solutions. From there, we obtained an i.i.d. sample of size 1'000 for each of the Q_{λ_i}, where $\lambda_i = \{1 + i/11\}_{i=0}^{99}$. Results are in Figure 8.

The left panel of Figure 8 shows the approximation of the map $\lambda \to \mathbb{E}[Q_\lambda]$. As the tightness of the 99% confidence bands suggests, the estimate is rather

robust. Spurious oscillations are most likely due to the numerical errors introduced by the MINN model, rather than from statistical noise. Coherently with the physical interpretation of λ, we see that the probability of low oxygenation decreases with the vascular density. Interestingly, although the total intensity of the source term is normalized to the same level in any configuration, the networks with higher gaps between neighboring edges are prone to spots of low oxygen concentration. Not only, the decay appears to be exponential. Further investigations seem to confirm this intuition, as we obtain an R^2-coefficient of 0.987 when trying to relate λ and $\log \mathbb{E}[Q_\lambda]$ via linear regression. Conversely, the right panel of Figure 8 shows how the probability distribution of Q_λ changes according to λ. The densities are more spread out when λ is near 1, while they shrink towards zero as λ increases. This is coherent with the physical intuition, and we would expect the density of Q_λ to converge to a Dirac delta as $\lambda \to +\infty$.

In real scenarios where the physical complexity of a vascularized tissue is appropriately described as in [27], this analysis would be computationally viable only with the employment of the MINN model as a surrogate for the numerical solver. In the case presented here, both the full order model and the surrogate model are computationally inexpensive. However, the former required around 2 minutes to generate 1'000 PDE solutions. Conversely, the trained DNN model was able to provide the same number of solutions in as little as 3 milliseconds, corresponding to a speed up factor of approximately 40. For multiphysics models where a simulation of a single point in the parameter space could cost hours of wall computational time, such gain could enable approaches that would be otherwise unreasonable. Even in the present simplified setting, such a boost also makes up for the computational effort required to train the network. In fact: (i) collecting the training snapshots took 575.96 seconds, (ii) training the MINN model required 125.32 seconds, (iii) generating the 100'000 new solutions took 0.3 seconds. In contrast, the numerical solver would generate at most $\approx$ 5'500 solutions within the same amount of time. These considerations support the interest in further developing model order reduction techniques based on deep neural networks that are robust for general spatial domains, such as MINNs. In fact, we are currently developing model reduction techniques applied to realistic models of the vascular microenvironment that leverage on the DL-ROM framework, previously developed in [11–13], combined with the efficiency of MINNs.

5 Conclusions

In this paper, we have introduced Mesh-Informed Neural Networks (MINNs), a novel class of sparse DNN models that can be used to learn general operators between infinite dimensional spaces. The approach is based on an apriori pruning strategy that is obtained by embedding the hidden states into discrete functional spaces of different fidelities. Despite being very easy to implement, MINNs show remarkable advantages with respect to dense architectures, such

as a massive reduction in the computational costs and an increased ability to generalize over unseen samples. This is coherent with the results available in the pruning literature [4], even though the setting differs from the one considered thereby.

We have tested MINNs over a large variety of scenarios, going from low dimensional manifolds to parameter dependent PDEs, showing that these architectures can learn nonlinear operators for general shapes of the underlying spatial domain. This opens a wide new range of research directions that we wish to investigate further in future works. For instance, one could test the use of MINNs in more sophisticated Deep Learning based Reduced Order Models for PDEs (DL-ROMs), such as those in [11–13, 23]. Finally, considering how MINNs are actually built, it may be interesting to see whether one can take advantage of multi-fidelity strategies during the training phase, as in [19].

Statements and Declarations

Funding
This project has received funding under the ERA-NET ERA PerMed / FRRB grant agreement No ERAPERMED2018-244, RADprecise - Personalized radiotherapy: incorporating cellular response to irradiation in personalized treatment planning to minimize radiation toxicity.

Data availability
Enquiries about data availability should be directed to the authors.

Appendix

Lemma 1 *Let Ω be a Lipschitz domain. Let $\Lambda \subset \overline{\Omega}$ be the union of finitely many segments, where each segment intersects $\partial\Omega$ in at most two points (the extremes). For any fixed $v \in \mathcal{C}(\overline{\Omega})$ one has*

$$\frac{1}{\epsilon^2} \int_\Omega v(\mathbf{x}) \max\{\epsilon - \mathrm{dist}(\mathbf{x}, \Lambda), 0\}\, d\mathbf{x} \to \int_\Lambda v(\mathbf{s}) d\mathbf{s} \quad \text{as } \epsilon \downarrow 0^+.$$

Proof Let φ^ϵ_Λ be the (unscaled) kernel

$$\varphi^\epsilon_\Lambda(\mathbf{x}) := \max\{\epsilon - \mathrm{dist}(\mathbf{x}, \Lambda), 0\}.$$

By definition, we note that φ^ϵ_Λ vanishes outside of the set $\Lambda + B(0,\epsilon) := \{\mathbf{x} + \epsilon\mathbf{v} \mid \mathbf{x} \in \Lambda,\ \mathbf{v} \in B(0,1)\}$. We now proceed in three steps.

Step 1. We start by proving that the lemma holds whenever Λ is composed by a single segment. Without loss of generality, we let $\Lambda = [0,1] \times \{0\}$. For the sake of simplicity, we further assume that $\Lambda \cap \partial\Omega = \varnothing$. The case in which Λ has an extreme on the boundary can be handled similarly by exploiting the Lipschitz regularity of $\partial\Omega$. Let $\epsilon < \mathrm{dist}(\Lambda, \partial\Omega)$, so

that $\Lambda + B(0,\epsilon) \subset \Omega$. By direct computation we have

$$\int_\Omega v(\mathbf{x})\varphi^\epsilon_\Lambda(\mathbf{x})d\mathbf{x} = \frac{1}{\epsilon^2}\int_{\Lambda+B(0,\epsilon)} v(\mathbf{x})\varphi^\epsilon_\Lambda(\mathbf{x})d\mathbf{x} =$$

$$= \frac{1}{\epsilon^2}\int_{A_\epsilon\cup B_\epsilon} v(\mathbf{x})\varphi^\epsilon_\Lambda(\mathbf{x})d\mathbf{x} + \frac{1}{\epsilon^2}\int_{[0,1]\times[-\epsilon,\epsilon]} v(\mathbf{x})\varphi^\epsilon_\Lambda(\mathbf{x})d\mathbf{x}$$

where A_ϵ and B_ϵ are two half circles of radius ϵ respectively centered at the extremes of the segment Λ. It is easy to see that the first contribute vanishes as $\epsilon \downarrow 0^+$. In fact, since $\|\varphi^\epsilon_\Lambda\|_\infty = \epsilon$,

$$\left|\frac{1}{\epsilon^2}\int_{A_\epsilon\cup B_\epsilon} v(\mathbf{x})\varphi^\epsilon_\Lambda(\mathbf{x})d\mathbf{x}\right| \le \frac{1}{\epsilon^2}\|v\|_\infty\cdot\epsilon|A_\epsilon\cup B_\epsilon| = \pi\epsilon\|v\|_\infty.$$

Conversely, for the second term we have

$$\int_0^1\frac{1}{\epsilon^2}\int_{-\epsilon}^{\epsilon} v(x_1,x_2)\varphi^\epsilon_\Lambda(x_1,x_2)dx_2dx_1 = \int_0^1\frac{1}{\epsilon^2}\int_{-\epsilon}^{\epsilon} v(x_1,x_2)(\epsilon-|x_2|)dx_2dx_1 =$$

$$= \int_0^1\frac{1}{\epsilon^2}\int_{-1}^{1} v(x_1,\epsilon z)(\epsilon-\epsilon|z|)\epsilon dzdx_1 = \int_0^1\int_{-1}^{1} v(x_1,\epsilon z)(1-|z|)dzdx_1.$$

By letting $\epsilon \downarrow 0^+$ we then get

$$\int_0^1\int_{-1}^{1} v(x_1,0)(1-|z|)dzdx_1 = \left(\int_\Lambda v(\mathbf{s})d\mathbf{s}\right)\left(\int_{-1}^{1}(1-|z|)dz\right) = \int_\Lambda v(\mathbf{s})d\mathbf{s}.$$

Step 2. Let $\Lambda = L_1\cup\cdots\cup L_n$ be given by the union of n segments. For any $i = 1,\dots,n$, let $\hat{L}_i := \{\mathbf{x}\in\Omega \mid \mathrm{dist}(\mathbf{x},L_i) < \mathrm{dist}(\mathbf{x},\Lambda\setminus L_i)\}$. We prove the following auxiliary result,

$$|(\Omega\setminus\hat{L}_i)\cap(L_i+B(0,\epsilon))| = o(\epsilon^2).$$

To see this, we note that, upto sets of measure zero,

$$(\Omega\setminus\hat{L}_i)\cap(L_i+B(0,\epsilon)) = (\hat{L}_1\cup\dots\hat{L}_{i-1}\cup\hat{L}_{i+1}\cup\dots\hat{L}_n)\cap(L_i+B(0,\epsilon)).$$

It is then sufficient to prove that $|\hat{L_j}\cap(L_i+B(0,\epsilon))| = o(\epsilon^2)$ for all j independently. If $L_i\cap L_j = \varnothing$, the proof is trivial. Conversely, if the two segments intersect, let θ be the angle between the two lines. It is easy to see that the intersection $\hat{L_j}\cap(L_i+B(0,\epsilon))$ is contained in a triangle of height ϵ and width $\epsilon/\tan(\theta/2)+\epsilon\tan(\theta/2)$. The conclusion follows.

Step 3. Let $\Lambda = L_1\cup\cdots\cup L_n$ and define the regions $\hat{L}_1,\dots\hat{L}_n$ as in the previous step. Fix any $v\in\mathcal{C}(\Omega)$. Then

$$\frac{1}{\epsilon^2}\int_\Omega v(\mathbf{x})\varphi^\epsilon_\Lambda(\mathbf{x})d\mathbf{x} = \frac{1}{\epsilon^2}\sum_{i=1}^n\int_{\hat{L}_i} v(\mathbf{x})\varphi^\epsilon_\Lambda(\mathbf{x})d\mathbf{x} = \sum_{i=1}^n\frac{1}{\epsilon^2}\int_{\hat{L}_i} v(\mathbf{x})\varphi^\epsilon_{L_i}(\mathbf{x})d\mathbf{x}.$$

Therefore, we can prove the original claim by showing that $\frac{1}{\epsilon^2}\int_{\hat{L}_i} v(\mathbf{x})\varphi^\epsilon_{L_i}(\mathbf{x})d\mathbf{x} \to \int_{L_i} v(\mathbf{s})d\mathbf{s}$ for each i. At this purpose, fix any $i = 1,\dots,n$. We have

$$\left|\frac{1}{\epsilon^2}\int_\Omega v(\mathbf{x})\varphi^\epsilon_{L_i}(\mathbf{x})d\mathbf{x} - \frac{1}{\epsilon^2}\int_{\hat{L}_i} v(\mathbf{x})\varphi^\epsilon_{L_i}(\mathbf{x})d\mathbf{x}\right| \le \frac{1}{\epsilon^2}\int_{\Omega\setminus\hat{L}_i}|v(\mathbf{x})||\varphi^\epsilon_{L_i}(\mathbf{x})|d\mathbf{x} \le$$

$$\le \frac{1}{\epsilon^2}\|v\|_\infty\cdot\epsilon|(\Omega\setminus\hat{L}_i)\cap(L_i+B(0,\epsilon))| = o(\epsilon)$$

and the conclusion follows from Step 1. □

References

1. Akenine-Moller, T., Haines, E., and Hoffman, N. (2019). Real-time rendering. *AK Peters/crc Press*.
2. Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3), 345-405.
3. Bhattacharya, K., Hosseini, B., Kovachki, N.B., and Stuart, A.M. (2021). Model Reduction and Neural Networks for Parametric PDEs. *SMAI Journal of Computational Mathematics*, 7, 121-157.
4. Blalock, D., Ortiz, J. J. G., Frankle, J., and Guttag, J. (2020). What is the state of neural network pruning?. *Proceedings of Machine Learning and Systems 2020 (MLSys 2020)*.
5. Berner, J., Grohs, P., Kutyniok, G., and Petersen, P. (2021). The modern mathematics of deep learning. *arXiv preprint* arXiv:2105.04026.
6. Cattaneo, L., Zunino, P. (2014). A computational model of drug delivery through microcirculation to compare different tumor treatments. *International Journal for Numerical Methods in Biomedical Engineering*, 30 (11), pp. 1347-1371.
7. Chen, W., Wang, Q., Hesthaven, J.S., and Zhang, C. (2021). Physics-informed machine learning for reduced-order modeling of nonlinear problems. *Journal of Computational Physics*, 446, 110666.
8. Daubechies, I., DeVore, R., Foucart, S., Hanin, B., and Petrova, G. (2021). Nonlinear Approximation and (Deep) ReLU Networks. *Constructive Approximation*, 1-46.
9. Elbrächter, D., Grohs, P., Jentzen, A., and Schwab, C. (2021). DNN expression rate analysis of high-dimensional PDEs: Application to option pricing. *Constructive Approximation*, 1-69.
10. Feng, Y., Feng, Y., You, H., Zhao, X., and Gao, Y. (2019, July). Meshnet: Mesh neural network for 3d shape representation. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 33, No. 01, pp. 8279-8286).
11. Franco, N. R., Manzoni, A., and Zunino, P. (2021). A Deep Learning approach to Reduced Order Modelling of Parameter Dependent Partial Differential Equations. *arXiv* preprint arXiv:2103.06183.
12. Fresca, S., Dede, L., and Manzoni, A. (2021). A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. *Journal of Scientific Computing*, 87(2), 1-36.
13. Fresca, S., and Manzoni, A. (2022). POD-DL-ROM: enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition. Computer Methods in Applied Mechanics and Engineering, 388, 114181.
14. Gao, H., Sun, L., and Wang, J.X (2021). PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain, *Journal of Computational Physics*, 428, 110079.
15. Geist, M., Petersen, P., Raslan, M., Schneider, R., and Kutyniok, G. (2021). Numerical solution of the parametric diffusion equation by deep neural networks. *Journal of Scientific Computing*, 88(1), 1-37.
16. Gribonval, R., Kutyniok, G., Nielsen, M., and Voigtlaender, F. (2022). Approximation spaces of deep neural networks. *Constructive Approximation*, 55(1), 259-367.
17. Gruber, A., Gunzburger, M., Ju, L., and Wang, Z. (2021). A Comparison of Neural Network Architectures for Data-Driven Reduced-Order Modeling. *arXiv* preprint arXiv:2110.03442.
18. Gühring, I., and Raslan, M. (2021). Approximation rates for neural networks with encodable weights in smoothness spaces. *Neural Networks*, 134, 107-130.
19. Guo, M., Manzoni, A., Amendt, M., Conti, P., and Hesthaven, J. S. (2022). Multi-fidelity regression using artificial neural networks: efficient approximation of parameter-dependent output quantities. *Computer methods in applied mechanics and engineering*, 389, 114378.
20. Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2021). Neural operator: Learning maps between function spaces. *arXiv* preprint arXiv:2108.08481.

21. Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). Imagenet classification with deep convolutional neural networks, *NIPS*.
22. Kutyniok, G., Petersen, P., Raslan, M., and Schneider, R. (2021). A theoretical analysis of deep neural networks and parametric PDEs. *Constructive Approximation*, 1-53.
23. Lee, K., and Carlberg, K. T. (2020). Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 404, 108973.
24. Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. (2021). Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3), 218-229.
25. Park, Y. J. (2018). On the Continuity of the Hardy-Littlewood Maximal Function. *Journal of the Chungcheong Mathematical Society*, 31(1), 43-46.
26. Perera, S., Barnes, N., He, X., Izadi, S., Kohli, P., and Glocker, B. (2015). Motion segmentation of truncated signed distance function based volumetric surfaces. *In 2015 IEEE Winter Conference on Applications of Computer Vision* (pp. 1046-1053). IEEE.
27. Possenti, L., Cicchetti, A., Rosati, R., Cerroni, D., Costantino, M. L., Rancati, T., and Zunino, P. (2021). A Mesoscale Computational Model for Microvascular Oxygen Transfer. *Annals of Biomedical Engineering*, 49(12), 3356-3373.
28. Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. (2020). Learning mesh-based simulation with graph networks. *arXiv* preprint arXiv:2010.03409.
29. Wang, S., Wang, H., and Perdikaris, P. (2021). Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science advances*, 7(40), eabi8605.
30. Wu, Y., Schuster, M., Chen, Z., Le, Q., Norouzi, M., Macherey, W. et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation, *arXiv preprint arXiv:1609.08144*.
31. Xu, J., Pradhan, A., and Duraisamy, K. (2021). Conditionally Parameterized, Discretization-Aware Neural Networks for Mesh-Based Modeling of Physical Systems. *arXiv* preprint arXiv:2109.09510.

MOX Technical Reports, last issues

Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

68/2022 Orlando, G.; Benacchio, T.; Bonaventura, L.
An IMEX-DG solver for atmospheric dynamics simulations with adaptive mesh refinement

67/2022 Alghamdi, M. M.; Boffi, D.; Bonizzoni, F.
A greedy MOR method for the tracking of eigensolutions to parametric elliptic PDEs

63/2022 Corti, M.; Antonietti, P.F.; Dede', L.; Quarteroni, A.
Numerical Modelling of the Brain Poromechanics by High-Order Discontinuous Galerkin Methods

64/2022 Massi, M.C., Dominoni, L., Ieva, F., Fiorito, G.
A Deep Survival EWAS approach estimating risk profile based on pre-diagnostic DNA methylation: an application to Breast Cancer time to diagnosis

66/2022 Antonietti, P.F.; Liverani, L.; Pata, V.
Lack of superstable trajectories in linear viscoelasticity: A numerical approach

65/2022 Dassi, F.; Fumagalli, A.; Mazzieri, I.; Vacca, G.
Mixed Virtual Element approximation of linear acoustic wave equation

62/2022 Ciaramella, G.; Halpern, L.; Mechelli, L.
Convergence analysis and optimization of a Robin Schwarz waveform relaxation method for periodic parabolic optimal control problems

61/2022 Gregorio, C.; Cappelletto, C.; Romani, S.; Stolfo, D.; Merlo, M.; Barbati, G.
Using marginal structural joint models to estimate the effect of a time-varying treatment on recurrent events and survival: An application on arrhythmogenic cardiomyopathy

59/2022 Boon, W. M.; Fumagalli, A.
A multipoint vorticity mixed finite element method for incompressible Stokes flow

60/2022 Cortellessa, D.; Ferro, N.; Perotto, S.; Micheletti, S.
Enhancing level set-based topology optimization with anisotropic graded meshes