POLITECNICO DI TORINO
Repository ISTITUZIONALE

The Lowest-Order Neural Approximated Virtual Element Method

Original
The Lowest-Order Neural Approximated Virtual Element Method / Berrone, S., Oberto, D., Pintore, M., Teora, G.. - 153 - 1:(2025). (ENUMATH: European Conference on Numerical Mathematics and Advanced Applications Lisbona (Portogallo) Settembre 4-8 2025) [10.1007/978-3-031-86173-4_13].

Availability:
This version is available at: 11583/2999718 since: 2025-05-28T12:56:25Z

Publisher:
Springer

Published
DOI:10.1007/978-3-031-86173-4_13

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
Springer postprint/Author's Accepted Manuscript

This version of the article has been accepted for publication, after peer review (when applicable) and is subject to Springer Nature's AM terms of use, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: http://dx.doi.org/10.1007/978-3-031-86173-4_13

(Article begins on next page)

18 November 2026

The Lowest-Order Neural Approximated Virtual Element Method

Stefano Berrone[0000−0001−8642−4258] and
Davide Oberto[0000−0001−8735−3148] and
Moreno Pintore[0000−0002−2837−2792] and
Gioana Teora[0000−0002−8540−3639]

Abstract We introduce the lowest-order Neural Approximated Virtual Element Method, a novel polygonal method that relies on neural networks to eliminate the need for projection and stabilization operators in the Virtual Element Method. In this paper, we discuss its formulation and detail the strategy for training the underlying neural network. The viability of the new method is tested through numerical experiments on elliptic problems.

1 Introduction

In the last years, the Virtual Element Method (VEM), introduced in [1], has gained considerable interest for its flexibility in handling arbitrary geometries and for the simplicity to build high-order methods. This versatility stems from the introduction of some non-polynomial functions in the local space, which are not known in a closed-form, and from the careful choice of the local degrees of freedom that allow to exactly compute suitable polynomial projections of these functions. However, these virtual functions also represent the main limitation of the VEM: the requirement to compute the polynomial projection of the virtual functions to access their pointwise evaluation leads not only to many limitations in a post-processing phase [2], but also to a lack of coercivity of the method. Indeed, their introduction entails the need to

Stefano Berrone, Davide Oberto, Gioana Teora (✉)
Dipartimento di Scienze Matematiche "G. L. Lagrange", Politecnico di Torino, Corso Duca degli Abruzzi, 24, 10129 Turin, Italy. The authors are members of the Italian INdAM-GNCS research group.
e-mail: stefano.berrone@polito.it,davide.oberto@polito.it,gioana.teora@polito.it

Moreno Pintore
Inria, Laboratoire Jacques-Louis Lions, Sorbonne Université, 4 place Jussieu, 75005 Paris, France.
e-mail: moreno.pintore@inria.fr

define a proper stabilization term to enforce the coercivity of the discrete bilinear form, which is highly problem-dependent [3] and whose isotropic nature can limit the accuracy of the method in case of strongly anisotropic problems [4, 5]. Furthermore, the presence of the stabilization term and the heavy usage of projections may induce issues also in complex non-linear problems [6, 7].

Very recently, these drawbacks has been addressed in [2] and [8], which use a reduced basis method and the *Lightning Laplace solver* developed in [9], respectively, to approximate the VEM basis functions. However, they further increase the additional computational cost already required by VEM with respect to standard procedures like the Finite Element Method (FEM).

Inspired by the recent success of Scientific Machine Learning [10], in this paper we introduce the lowest-order *Neural Approximated Virtual Element Method* (NAVEM), a novel method in which neural networks are used to approximate the unknown VEM basis functions. Through such approximations, the need for computing the local projection matrices and defining a stability operator is circumvented, thereby aligning NAVEM with a FEM on polygonal meshes.

2 The Virtual Element Method

Let $\Omega \subset \mathbb{R}^2$ be an open convex bounded polygonal domain and let $\Gamma = \partial\Omega$ be its boundary. Let $\mathcal{T}_h$ be a decomposition of Ω into polygons E and let $\mathcal{E}_{h,E}$ be the set of edges of the element $E \in \mathcal{T}_h$. Furthermore, we denote by N_E^v the number of vertices (and of edges) of the element E and by h_E its diameter. Here, as usual, h denotes the maximum diameter of the polygons in $\mathcal{T}_h$. We assume that the following mesh assumptions hold true [11].

Assumption 1 (Mesh assumptions)**.** *There exists a positive constant ρ, independent of E and h, such that*

- *each polygon $E \in \mathcal{T}_h$ is star-shaped with respect to a ball of radius $\geq \rho h_E$;*
- *for each edge $e \in \mathcal{E}_{h,E}$, it holds: $|e| \geq \rho h_E$.*

Given a polygon E, for each integer $k \geq 0$, we denote by $\mathbb{P}_k(E)$ the set of two-dimensional polynomials of degree up to k defined on E, with dimension $n_k = \dim \mathbb{P}_k(E) = \frac{(k+1)(k+2)}{2}$. Furthermore, we introduce the set

$$\mathbb{B}_1(\partial E) = \left\{ v \in C^0(\partial E) : v_{|e} \in \mathbb{P}_1(e) \, \forall e \in \mathcal{E}_{h,E} \right\},$$

whose dimension is $\dim \mathbb{B}_1(\partial E) = N_E^v$. For all $E \in \mathcal{T}_h$, we define the lowest-order local virtual element space [1] as the set

$$V_{h,1}(E) = \left\{ v \in H^1(E) : \quad (i)\ \Delta v = 0, \quad (ii)\ v_{|\partial E} \in \mathbb{B}_1(\partial E) \right\}, \tag{1}$$

with dimension $N_E^{\text{dof}} = \dim V_{h,1}(E) = N_E^v$, and we consider the value of $v_h \in V_{h,1}(E)$ at the vertices of E as local degrees of freedom.

2.1 The Model Problem and its Virtual Element Discretization

Given $f \in L^2(E)$, we consider the Poisson problem with homogeneous Dirichlet boundary conditions:

$$\begin{cases} -\Delta u = f & \text{in } \Omega, \\ u = 0 & \text{on } \Gamma. \end{cases} \tag{2}$$

The variational formulation of problem (2) reads: *Find* $u \in V = H_0^1(\Omega)$ *such that*

$$a(u,v) = \sum_{E\in\mathcal{T}_h} a^E(u,v) = \sum_{E\in\mathcal{T}_h} \int_E \nabla u \cdot \nabla v = (f,v)_\Omega, \quad \forall v \in V. \tag{3}$$

Now, we can define the lowest-order global virtual element space as

$$V_{h,1} = \{v \in H_0^1(\Omega) \cap C^0(\overline{\Omega}) : v_{|E} \in V_{h,1}(E) \ \forall E \in \mathcal{T}_h\}. \tag{4}$$

We introduce the *computable* local polynomial projectors

- $\Pi_1^\nabla : V \to \mathbb{P}_1(\mathcal{T}_h)$ such that, for each $E \in \mathcal{T}_h$,

$$(\nabla v - \nabla \Pi_1^\nabla v, \nabla p)_E = 0, \quad \forall p \in \mathbb{P}_1(E) \text{ and } \int_{\partial E} \Pi_1^\nabla v = \int_{\partial E} v;$$

- $\Pi_0^0 : V \to \mathbb{P}_0(\mathcal{T}_h)$ which is locally defined as the $L^2(E)$-projector onto element-wise constants.

Finally, the virtual element discretization of problem (3) reads: *Find* $u_h \in V_{h,1}$ *such that*

$$\sum_{E\in\mathcal{T}_h} a_h^E(u_h, v_h) = \sum_{E\in\mathcal{T}_h} (f, \Pi_0^0 v_h)_E \quad \forall v_h \in V_{h,1}, \tag{5}$$

where the discrete bilinear form $a_h^E(\cdot,\cdot)$ is the sum of a consistency term related to the accuracy of the method and of a stabilization term $S^E(\cdot,\cdot)$ enforcing the coercivity [1], that is

$$a_h^E(u_h, v_h) = \int_E \nabla \Pi_1^\nabla u_h \cdot \nabla \Pi_1^\nabla v_h + S^E((I - \Pi_1^\nabla)u_h, (I - \Pi_1^\nabla)v_h). \tag{6}$$

In particular, the stabilization term $S^E(\cdot,\cdot)$ can be chosen as any symmetric positive definite bilinear form which scales like $a^E(\cdot,\cdot)$.

3 The Neural Approximated Virtual Element Method

Let us introduce the set of the VEM Lagrangian basis functions $\{\varphi_i\}_{i=1}^{N^{\text{dof}}}$ corresponding to the aforementioned degrees of freedom, each of them associated with a differ-

ent internal vertex v_i of the tessellation $\mathcal{T}_h$. We denote by $\mathbb{S}_i = \text{supp}\,(\varphi_i) = \cup_{j=1}^{N_{v_i}} E_j$ the support of φ_i, i.e. the union of the N_{v_i} elements $E_j \in \mathcal{T}_h$ adjacent to the vertex v_i. Furthermore, given an element $E \in \mathcal{T}_h$, for the sake of brevity, we indicate by $\{\varphi_{j,E}\}_{j=1}^{N_E^{\text{dof}}}$ the set of the restrictions to E of the Lagrangian basis functions related to the vertices of E. Clearly, the local and the global virtual element spaces can be written as

$$V_{h,1}(E) = \text{span}\{\varphi_{j,E} : \ j = 1, \dots, N_E^{\text{dof}}\}$$

and

$$V_{h,1} = \text{span}\{\varphi_i : \ i = 1, \dots, N^{\text{dof}}\}.$$

3.1 The Approximated Basis Functions

Let us introduce the space $\mathbb{H}_{\ell_E}(E)$ of the harmonic polynomials of degree up to $\ell_E \geq 1$ with dimension $\dim \mathbb{H}_{\ell_E}(E) = 2\ell_E + 1$, where we introduce the subscript E to underline that ℓ_E depends on the number of vertices N_E^v of E. In the following, we assume ℓ_E to be an hyperparameter of the neural network and we define it such that $\ell_{E_1} = \ell_{E_2}$ if $N_{E_1}^v = N_{E_2}^v$ with $E_1, E_2 \in \mathcal{T}_h$. We point out that the parameter ℓ_E must be large enough to ensure the coercivity of the discrete problem [4]. In this regard, a necessary condition is $\ell_E \geq \left\lceil \frac{N_E^v - 1}{2} \right\rceil$.

Now, we introduce a neural network which aims to learn the following highly non-linear map:

$$(v_j, E) \mapsto \varphi_{j,E}^{\mathcal{NN}} \in \mathbb{H}_{\ell_E}(E), \text{ for each vertex } v_j \text{ of } E \text{ and } \forall E \in \mathcal{T}_h, \tag{7}$$

where the functions $\varphi_{j,E}^{\mathcal{NN}}$ aim to locally approximate the Lagrangian VEM basis functions. To achieve this goal, firstly we must require that the functions $\varphi_{j,E}^{\mathcal{NN}}$ belong to the VEM space $V_{h,1}(E)$, and, in particular, that they satisfy Properties (i) and (ii) defined in (1).

In this regard, we note that Property (i) is trivially satisfied by the functions $\varphi_{j,E}^{\mathcal{NN}}$ by construction. Instead, Property (ii) is, in general, not satisfied by functions belonging to $\mathbb{H}_{\ell_E}(E)$. Nevertheless, we overcome this issue by training the neural network to learn functions $\varphi_{j,E}^{\mathcal{NN}}$ in such a way that they mimic the Lagrangian basis functions $\varphi_{j,E}$ at the boundary of the element E, where all the virtual functions are known in a closed form. By proceeding in this way, the NAVEM functions $\varphi_{j,E}^{\mathcal{NN}}$ approximate the related VEM Lagrangian basis functions, i.e. $\varphi_{j,E}^{\mathcal{NN}} \approx \varphi_{j,E}$ on $E, \forall j = 1, \dots, N_E^{\text{dof}}, \ \forall E \in \mathcal{T}_h$.

Now, for each internal vertex v_i of the tessellation $\mathcal{T}_h$, we can define the piecewise functions $\varphi_i^{\mathcal{NN}}$ as

$$\varphi_i^{\mathcal{NN}} = \begin{cases} \varphi_{j,E}^{\mathcal{NN}} & \text{if } v_i \text{ is the } j\text{-the vertex of } E \text{ and } E \in \mathbb{S}_i, \\ 0 & \text{otherwise.} \end{cases}$$

and the local and global lowest-order NAVEM spaces as the sets

$$V_{h,1}^{\mathcal{NN}}(E) = \text{span}\{\varphi_{j,E}^{\mathcal{NN}},\ j = 1, \ldots, N_E^{\text{dof}}\} \subset \mathbb{H}_{\ell_E}(E),$$

$$V_{h,1}^{\mathcal{NN}} = \text{span}\{\varphi_i^{\mathcal{NN}},\ i = 1, \ldots, N^{\text{dof}}\},$$

which approximate the corresponding VEM spaces by construction.

Finally, the NAVEM discretization of problem (3) reads: *Find* $u_h^{\mathcal{NN}} \in V_{h,1}^{\mathcal{NN}}$ *such that*

$$\sum_{E \in \mathcal{T}_h} a^E(u_h^{\mathcal{NN}}, v_h^{\mathcal{NN}}) = \sum_{E \in \mathcal{T}_h} (f, v_h^{\mathcal{NN}})_E \quad \forall v_h^{\mathcal{NN}} \in V_{h,1}^{\mathcal{NN}}. \tag{8}$$

where we highlight that the bilinear form $a^E(\cdot,\cdot)$ used therein is the continuous bilinear form defined in (3) (up to errors introduced by numerical integration).

3.2 The Neural Network

In this section, we show how to train the neural network underlying the NAVEM method (offline phase) and the procedure to predict the NAVEM basis functions (online phase).

3.2.1 The Offline Phase

Let us consider the reference square $\tilde{S}$ centred at the origin with diameter $h_{\tilde{S}}$. Now, we introduce an orthonormal polynomial basis $\{\tilde{p}_\beta\}_{\beta=1}^{2\ell_E+1}$ for $\mathbb{H}_{\ell_E}(\tilde{S})$, which is built by orthogonalizing the scaled polynomial basis for $\mathbb{H}_{\ell_E}(\tilde{S})$ defined by the recursive strategy used in [4]. More precisely, we orthogonalize this basis by applying the modified Gram-Schmidt algorithm twice to the Vandermonde matrix whose columns contain the evaluations of the scaled polynomials at points defined as nodes of a lattice built over $\tilde{S}$.

Now, for each $N^v \geq 3$, we perform the following steps:

1. We consider a set of randomly generated polygons $\mathcal{S}_{N^v}$ with N^v vertices and edges. This process is subject to the restrictions imposed by the mesh assumptions 1.
2. On each element $E \in \mathcal{S}_{N^v}$ we perform the mapping $F_E : \widehat{E} \to E$ proposed in [12] to reduce the variability of elements seen by the neural network. This mapping is an affine transformation whose definition is based on the inertia tensor of the element E. We recall that this mapping generates polygons $\widehat{E}$ with a (scaled) identity inertia tensor, unit diameter and centred at the axes origin.
3. For each vertex v_j of $\widehat{E} = F_E^{-1}(E) : E \in \mathcal{S}_{N^v}$, we encode the information representing the pair $(v_j, \widehat{E})$ into a vector $\boldsymbol{z}_E \in \mathbb{R}^{N_0}$ of a suitable dimension N_0. More precisely, we consider an affine map $G_{j,\widehat{E}} : \tilde{E}_j \to \widehat{E}$ that rotates and rescales $\widehat{E}$ to place its j-vertex into $(1, 0)$. Then, we set $\boldsymbol{z}_E = [\tilde{x}_1^2, \tilde{x}_2^2, \tilde{x}_1^3, \tilde{x}_2^3, \ldots, \tilde{x}_1^{N^v}, \tilde{x}_2^{N^v}]$,

where $(\tilde{x}_1^k, \tilde{x}_2^k)$ are the coordinates of the k-th vertices of $\tilde{E}_j = G_{j,\widehat{E}}^{-1}(\widehat{E})$ and the vertices of $\tilde{E}_j$ are ordered counter-clockwise starting from $(\tilde{x}_1^1, \tilde{x}_2^1) = (1, 0)$. Thus, the dimension of the input of the neural network is $N_0 = 2(N^v - 1)$. This process is done to reduce the dimension of the input of the neural network and it is possible since the prediction of a basis function is not affected by the prediction of basis functions related to the remaining vertices of the element.

4. Given an activation function σ, we define a feed-forward fully-connected neural network with L layers and N_n, $n = 1, \dots, L$, neurons per layer, i.e. a function $c^{\mathcal{NN}} : \mathbb{R}^{N_0} \to \mathbb{R}^{2\ell_E+1}$ defined by the recursive expression:

$$\begin{aligned} z_n &= \sigma(\mathbf{A}_n z_{n-1} + \mathbf{b}_n), \qquad && n = 1, ..., L-1, \\ \boldsymbol{c}^{\mathcal{NN}} &= c^{\mathcal{NN}}(z_E) = \mathbf{A}_L z_{L-1} + \mathbf{b}_L \end{aligned} \tag{9}$$

where $z_0 = z_E$ is the input, $\mathbf{A}_n \in \mathbb{R}^{N_n \times N_{n-1}}$ and $\mathbf{b}_n \in \mathbb{R}^{N_n}$, $n = 1, ..., L$ are the matrices and vectors containing the network weights, while the output $\boldsymbol{c}^{\mathcal{NN}} \in \mathbb{R}^{2\ell_E+1}$ represents the vector of coefficients of functions $\varphi_{j,\tilde{E}_j}^{\mathcal{NN}} := \varphi_{j,E}^{\mathcal{NN}} \circ F_E \circ G_{j,\widehat{E}}$ with respect to the basis $\{\tilde{p}_\beta\}_{\beta=1}^{2\ell_E+1}$, i.e. $\varphi_{j,\tilde{E}_j}^{\mathcal{NN}} = \sum_{\beta=1}^{2\ell_E+1} \boldsymbol{c}_\beta^{\mathcal{NN}}(z_E)\tilde{p}_\beta$.

5. We train the neural network by minimizing the loss function:

$$\mathcal{L}_{N^v} = \sum_{E \in \mathcal{S}_{N^v}} \sum_{j=1}^{N^v} \sum_{\tilde{\boldsymbol{x}} \in X_{\partial\tilde{E}_j}} \left[\left(\varphi_{j,\tilde{E}_j}^{\mathcal{NN}}(\tilde{\boldsymbol{x}}) - \varphi_{j,\tilde{E}_j}(\tilde{\boldsymbol{x}}) \right)^2 + \left(\frac{\partial \varphi_{j,\tilde{E}_j}^{\mathcal{NN}}}{\partial \tilde{\boldsymbol{t}}}(\tilde{\boldsymbol{x}}) - \frac{\partial \varphi_{j,\tilde{E}_j}}{\partial \tilde{\boldsymbol{t}}}(\tilde{\boldsymbol{x}}) \right)^4 \right], \tag{10}$$

where $X_{\partial\tilde{E}_j}$ denotes the set of control points distributed on $\partial\tilde{E}_j$ and $\dfrac{\partial}{\partial \tilde{\boldsymbol{t}}}$ represents the tangential derivative. We emphasize that the different exponents used in (10) are set in order to obtain two contributions with comparable orders of magnitude.

It is worth mentioning that the diameter of the reference square $\tilde{S}$ is set in such a way all the elements $\tilde{E}_j$ are contained in $\tilde{S}$. Consequently, the polynomial basis $\{\tilde{p}_\beta\}_{\beta=1}^{2\ell_E+1}$ is well-scaled for each element $\tilde{E}_j$. Finally, we remark that ℓ_E depends on E only through N_E^v.

3.2.2 The Online Phase

To assemble the discrete NAVEM system, for each element $E \in \mathcal{T}_h$, firstly we perform the inertial mapping $\widehat{E} = F_E^{-1}(E)$. Subsequently, for each vertex v_j of $\widehat{E}$:

1. We encode the information $(v_j, \widehat{E})$ to generate the input z_E of the neural network.
2. We predict the coefficients of the corresponding NAVEM basis function $\varphi_{j,\tilde{E}_j}^{\mathcal{NN}}$ with respect to the polynomial basis $\{\tilde{p}_\beta\}_{\beta=1}^{2\ell_E+1}$ using the neural network related to the number of vertices of E.
3. We map the function $\varphi_{j,\tilde{E}_j}^{\mathcal{NN}} = \sum_{\beta=1}^{2\ell_E+1} \boldsymbol{c}_\beta^{\mathcal{NN}}(z_E)\tilde{p}_\beta$ and its gradient $\tilde{\nabla}\varphi_{j,\tilde{E}_j}^{\mathcal{NN}} = \sum_{\beta=1}^{2\ell_E+1} \boldsymbol{c}_\beta^{\mathcal{NN}}(z_E)\tilde{\nabla}\tilde{p}_\beta$ back to the original element E.

Finally, we can assemble and solve the linear system associated with Problem (8) as in standard FEM solvers.

4 Numerical Results

We consider the advection-diffusion-reaction problem

$$\begin{cases} \nabla \cdot (-\boldsymbol{D}(\boldsymbol{x})\nabla u) + \boldsymbol{\beta}(\boldsymbol{x}) \cdot \nabla u + \gamma(\boldsymbol{x})u = f & \text{in } \Omega, \\ u = g_D & \text{on } \Gamma, \end{cases} \tag{11}$$

where

$$\boldsymbol{D}(\boldsymbol{x}) = \begin{bmatrix} 1 + x_2^2 & -x_1 x_2 \\ -x_1 x_2 & 1 + x_1^2 \end{bmatrix}, \quad \boldsymbol{\beta}(\boldsymbol{x}) = \begin{bmatrix} x_1 \\ -x_2 \end{bmatrix}, \quad \gamma(\boldsymbol{x}) = x_1 x_2.$$

Furthermore, the forcing term f and the Dirichlet boundary condition g_D are set in such a way the exact solution is

$$\begin{aligned} u(\boldsymbol{x}) =& 3\left((x_1 - 0.2) + \frac{x_2 - 0.3}{2}\right)^2 + 2\left(\frac{x_1 - 0.7}{2} + (x_2 - 0.8)\right)^3 \\ &+ \sin(2\pi x_1)\sin(3\pi x_2). \end{aligned} \tag{12}$$

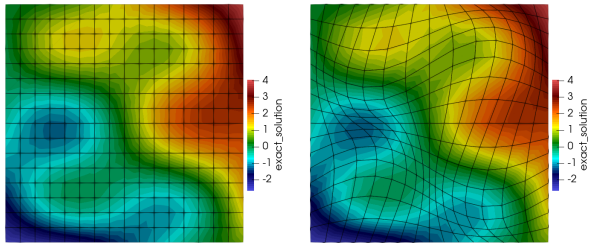
The contour line plot of the exact solution is shown in Figure 1.

4.1 The Neural Network Architecture

Our neural network is made up of 3 hidden layers of 30 neurons each. We employ the hyperbolic tangent as nonlinear activation function, and initialize the weights using a Glorot normal distribution. We use a first-order Adam optimizer for the first 3000 epochs with an exponentially decaying learning rate and successively we switch to a second-order BFGS optimizer until convergence of the loss is reached.

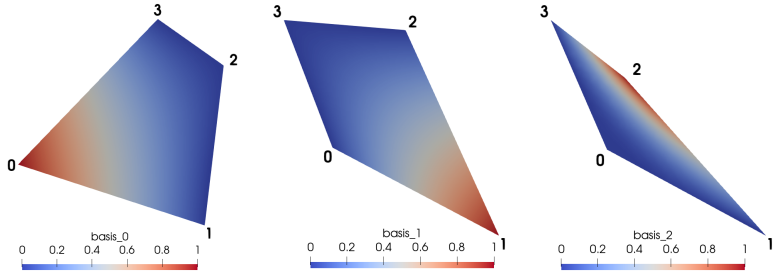
Furthermore, in our experiments, we numerically solve Problem (11) on two different families of meshes made up of quadrilateral elements. The first family consists of 4 cartesian meshes made up of 16, 64, 256 and 1024 identical squares, respectively. The meshes in the second family are derived from a sine distortion of the related cartesian counterparts. The third refinement of each family is shown in Figure 1.

Since we consider only quadrilateral meshes, we only need to train a single neural network which predicts functions belonging to $\mathbb{H}_{\ell_E}(E)$, with $\ell_E = 5$ for all $E \in \mathcal{T}_h$. Moreover, the training dataset is made up of 100 randomly generated quadrilaterals. Examples of such generated quadrilaterals coloured by one of the corresponding NAVEM basis functions are shown in Figure 2.



(a) (b)

Fig. 1: The third mesh of the family coloured by the interpolated exact solution (12). Left: a cartesian mesh. Right: a sine distorted mesh.



(a) (b) (c)

Fig. 2: Three different training polygons coloured by the predicted NAVEM basis functions associated with vertices 0, 1 and 2 respectively.

4.2 The Convergence Curves

To assess the accuracy of our procedure, we solve Problem (11) with the NAVEM and analyze the behaviour of the following errors

$$\mathrm{err}_2^2 = \sum_{E\in\mathcal{T}_h} \|u - u_h^{\mathcal{NN}}\|_{0,E}^2, \quad \mathrm{err}_1^2 = \sum_{E\in\mathcal{T}_h} \|\nabla u - \nabla u_h^{\mathcal{NN}}\|_{0,E}^2, \tag{13}$$

when h varies. Furthermore, to make a comparison with standard VEM, we also solve Problem (11) with a VEM approach by employing the *dofi-dofi* stabilization [1]. Since we are dealing with variable coefficients, we opt for the more suitable VEM formulation, detailed in [11], which is based on the definition of the local enhanced virtual element space and on the introduction of the L^2-projector of derivatives. Moreover, since the VEM functions are pure virtual, we are not able to compute the errors as expressed in (13). Thus, in the case of VEM, we define

$$\mathrm{err}_2^2 = \sum_{E\in\mathcal{T}_h} \|u - \Pi_1^0 u_h\|_{0,E}^2, \quad \mathrm{err}_1^2 = \sum_{E\in\mathcal{T}_h} \|\nabla u - \Pi_0^0 \nabla u_h\|_{0,E}^2. \tag{14}$$

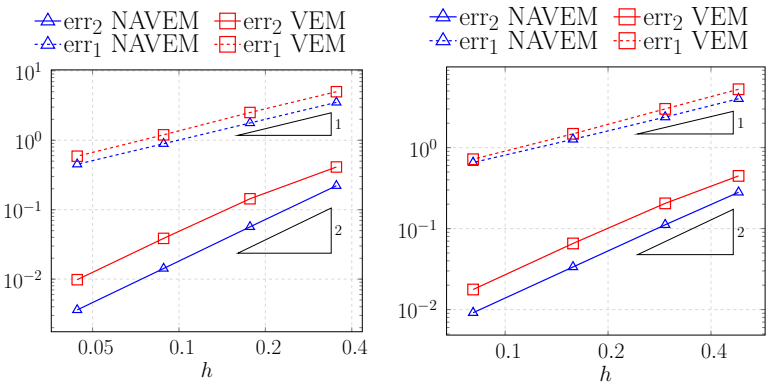


Fig. 3: The NAVEM errors (13) and the VEM errors (14) versus h. Left: cartesian meshes. Right: sine distorted meshes.

The behaviour of the errors (13) and (14) are shown in Figure 3. We observe that the NAVEM errors (13) decay with the same rate of convergence of the related VEM errors, that is $O(h^2)$ for the error in the L^2-norm and $O(h^1)$ for the error in the H^1-seminorm.

Furthermore, we note that the error constants related to NAVEM are smaller than the VEM error constants, that is the curves of convergences of NAVEM are downward shifted with respect to the VEM ones. Since we are approximating the same space, we suppose that a good training of the neural networks could ensure NAVEM achieves in any case the same theoretical convergence results that hold for the VEM, as supported by the numerical experiments reported here. We highlight that at the moment there exist only few examples of a priori error estimates for neural networks-based solvers (see for instance [13]). However, in the future, we aim to present a more theoretical discussion.

5 Conclusions

In this paper, we introduce the lowest-order Neural Approximated Virtual Element Method. It is a polytopal method that relies on a neural network to approximate the VEM basis functions avoiding the introduction of any stabilization term or polynomial projectors. We numerically observe for NAVEM the same empirical convergence rates and smaller error constants with respect to VEM on quadrilateral elements.

Acknowledgements The author S.B. kindly acknowledges partial financial support provided by PRIN project “Advanced polyhedral discretisations of heterogeneous PDEs for multiphysics prob-

lems" (No. 20204LN5N5_003) and by PNRR M4C2 project of CN00000013 National Centre for HPC, Big Data and Quantum Computing (HPC) (CUP: E13C22000990001). The author D. O. kindly acknowledges the financial support provided by INdAM - GNCS Project CUP_E53C23001670001. The author M.P. kindly acknowledges the financial support provided by the Politecnico di Torino where the research has been carried out. The author G.T. kindly acknowledges financial support provided by the MIUR programme "Programma Operativo Nazionale Ricerca e Innovazione 2014 - 2020" (CUP: E11B21006490005) and by INdAM - GNCS Project CUP_E53C23001670001.

Competing Interests The authors have no conflicts of interest to declare that are relevant to the content of this chapter.

References

1. Beirão Da Veiga L., Brezzi F., Cangiani A., Manzini G., Marini L.D., and Russo A.: Basic principles of virtual element methods. Math. Models Methods Appl. Sci. **23**, 199–214, (2013)
2. Credali F., Bertoluzza S., and Prada D.: Reduced basis stabilization and post-processing for the virtual element method. Computer Methods in Applied Mechanics and Engineering. **420**, (2023)
3. Russo A. and Sukumar N.: Quantitative study of the stabilization parameter in the virtual element method. (2023)
4. Berrone S., Borio A., Marcon F., and Teora G.: A first-order stabilization-free virtual element method. Appl. Math. Lett., **142**, 108641, (2023)
5. Borio A., Lovadina C., Marcon F., and Visinoni M.: A lowest order stabilization-free mixed virtual element method. Computers & Mathematics with Applications. **160** 161–170, (2024).
6. Adak D, Natarajan E. and Kumar S.: Convergence analysis of virtual element methods for semilinear parabolic problems on polygonal meshes. Numer. Methods Partial Differential Eq., **35** 222–245, (2019)
7. Frittelli M., Madzvamuse A. and Sgura I.: Bulk-surface virtual element method for systems of PDEs in two-space dimensions. Numerische Mathematik, **147** 305–348, (2021)
8. Trezzi M.L. and Zerbinati U.: When rational functions meet virtual elements: The lightning virtual element method. (2023)
9. Gopal A. and Trefethen L.N.: Solving Laplace problems with corner singularities via rational functions. SIAM J. Numer. Anal. **57**, 2074–2094 (2019)
10. Cuomo S., Di Cola V. S. , Giampaolo F., Rozza G., Raissi M., and Piccialli F.: Scientific machine learning through physics-informed neural networks: Where we are and what's next. J. Sci. Comput. **92**, (2022)
11. Beirão Da Veiga L., Brezzi F., Marini L.D., and Russo A.: Virtual element methods for general second order elliptic problems on polygonal meshes. Math. Models Methods Appl. Sci. **26**, 729–750, (2015)
12. Berrone S., Teora G., and Vicini F.: Improving high-order vem stability on badly-shaped elements. Math. Comput. Simul., **216**, 367–385, (2024)
13. Berrone S., Canuto C., and Pintore M.: Variational physics informed neural networks: the role of quadratures and test functions. J. Sci. Comput. **92**, 1–27, (2022)