algorithms

MDPI

Article

Neural-Network-Assisted Finite Difference Discretization for Numerical Solution of Partial Differential Equations

Ferenc Izsák [1,*,†] and Rudolf Izsák [2,†]

[1] Department of Applied Analysis and Computational Mathematics, Eötvös Loránd University, 1117 Budapest, Hungary
[2] Department of Algorithms and Their Applications, Eötvös Loránd University, 1117 Budapest, Hungary; irudolf@elte.hu
* Correspondence: ferenc.izsak@ttk.elte.hu
† These authors contributed equally to this work.

Abstract: A neural-network-assisted numerical method is proposed for the solution of Laplace and Poisson problems. Finite differences are applied to approximate the spatial Laplacian operator on nonuniform grids. For this, a neural network is trained to compute the corresponding coefficients for general quadrilateral meshes. Depending on the position of a given grid point $\mathbf{x}_0$ and its neighbors, we face with a nonlinear optimization problem to obtain the finite difference coefficients in $\mathbf{x}_0$. This computing step is executed with an artificial neural network. In this way, for any geometric setup of the neighboring grid points, we immediately obtain the corresponding coefficients. The construction of an appropriate training data set is also discussed, which is based on the solution of overdetermined linear systems. The method was experimentally validated on a number of numerical tests. As expected, it delivers a fast and reliable algorithm for solving Poisson problems.

Keywords: boundary value problems; Laplacian; Poisson problem; neural networks; finite difference discretization; learning data; nonuniform mesh

Citation: Izsák, F.; Izsák, R. Neural-Network-Assisted Finite Difference Discretization for Numerical Solution of Partial Differential Equations. *Algorithms* **2023**, *16*, 410. https://doi.org/10.3390/a16090410

Academic Editors: Dunhui Xiao and Shuai Li

Received: 29 July 2023
Revised: 23 August 2023
Accepted: 25 August 2023
Published: 28 August 2023

1. Introduction

In the last decade, the widespread use of artificial neural networks (ANNs) led to a significant step forward in computer science and computational mathematics. A wide range of practical problems, including segmentation and classification tasks, could be efficiently solved using this tool. Here, the related mathematical models fall into the class of discrete problems. At the same time, we have to keep in mind that the engine of the ANN-based algorithms is a gradient-based continuous optimization algorithm enriched with stochastic elements. This gives the first motivation to also apply ANN-based techniques for the numerical solution of problems arising from continuous mathematical models. The numerical solution of partial differential equations (PDEs) constitutes an important class of these, having a central role in the natural sciences and engineering.

Accordingly, a family of ANN-based methodologies was developed for solving partial differential equations (PDEs) numerically. The most common approach is the class of the so-called physics-informed neural networks (PINNs). In this framework, the solution candidates are optimized based on the most straightforward error terms. To compute these, one has to calculate consistency error in the governing equations and boundary conditions. In concrete terms, the approximations are substituted into the physical laws constituting the mathematical problem. For a general description of this method, we refer to [1,2]. A number of specific forms of this method were developed and applied for a scale of classical PDE problems arising in real-life applications; see, e.g., [3]. Beyond these, the above framework can be extended to the numerical solution of stochastic [4] and inverse problems [5,6] as well.

A possible shortcut of this procedure is to seek approximations only from a class of functions that satisfy the governing equation exactly. In this case, we only have to care about parameters (unknowns) associated with the boundary, which can largely decrease the dimension of the corresponding optimization problems. Such an algorithm is introduced in [7] and applied to the numerical solution of Dirichlet-to-Neumann problems.

To summarize these developments and further approaches in detail, a number of review articles have been published in the past years. Among them, we propose [8,9], where a really wide range of further specific references can be found. All of these approaches offer a complete procedure for the numerical solution of PDEs.

At the same time, in the last 75 years, a wide class of conventional numerical methods were developed for solving PDEs. A natural idea is to combine these with the efficient computational tools offered by the ANNs. This is also the aim of the present study. In other words, we propose here an *ANN-assisted* numerical solution of PDEs. In this framework, instead of the entire approximation procedure, only an important and technical step of a conventional numerical method is executed using an ANN.

In the course of the numerical solution of a PDE, we usually first have to discretize the underlying problem including the governing equation and the boundary data. Whenever we can choose purely function-space-based discretizations, it is mostly connected to a physical discretization of the computational domain, where the equation is posed. Generating a suitable mesh for this purpose is still a challenging problem. At this step, one can also make use of ANN-based procedures; see, e.g., [10–12].

Having this mesh at hand, choosing a specific finite element or finite difference discretization method and, finally, incorporating the given boundary data, we obtain an algebraic (or linear algebraic) system of equations. The construction of such finite-difference-based approximations will be the focus of our study.

In case of uniform rectangular meshes, we can easily derive finite difference approximations of various differential operators and convert the original problem quickly into a system of equations. Otherwise, which is the case in almost all real-life engineering problems, we need to deal with simplicial (or even hybrid) meshes and use an appropriate Galerkin-type discretization. Generating this mesh with the corresponding data structure and collecting (assembling) the system of equations may be rather time-consuming. For the details of this procedure, we refer to [13], Section 8. In the case of linear problems like the Laplacian equation, this can take even longer compared to the solution of the system of equations. Even for nonlinear problems, including iterative solvers, the assembling may take a significant portion of the computational time [14].

Here, we consider the case of the Laplacian operator. Accordingly, our aim in the present work is to develop an ANN that generates the finite difference discretization matrix of the Laplacian and meets the following requirements:

(i) It also works on nonuniform and nonrectangular grids;
(ii) It is very quick and uses only neighboring relations between the grid points;
(iii) It delivers an accurate discretization of the Laplacian.

Note that the numerical approximation of the Laplacian operator has a long history and a number of efficient approximations were elaborated. It has a central role not only in classical diffusion problems but also in several computationally intensive mathematical models, such as equations for wave propagation, Navier–Stokes equations or free-surface wave equations [15].

Summarized, the main motivation of the present work is to utilize the tool of the ANNs for the accurate discretization of the Laplacian operator. We perform this in the framework of a finite difference approximation. Our contribution here is to provide the coefficients for nonregular meshes. Instead of a pointwise optimization procedure, these coefficients will be computed by an ANN, which we train in advance.

The setup of our contribution is the following. After a short review, we discuss the principle of our algorithm. Then, the two main components, the procedure for generating learning data and the construction of the neural network, will be explained. Finally, in a

series of numerical experiments, an experimental analysis will be carried out to test the efficiency of our algorithm.

Note in advance that the methodology we develop here can be easily extended for a number of differential operators, where a computationally efficient finite difference approximation is needed on a nonuniform spatial mesh.

2. Materials and Methods

We first state the mathematical problems to solve. To ease the presentation, we perform the study in the two-dimensional case, but the principle is still valid in three space dimensions.

2.1. Problem Statement and Mathematical Background

We investigate the numerical approximation

$$\sum_{j=0}^{J} a_j u(\mathbf{x}_j) \approx \Delta u(\mathbf{x}_0) \tag{1}$$

of the Laplacian operator at a given point $\mathbf{x}_0$. Here, $\{\mathbf{x}_j\}_{j=1}^{J}$ denotes the adjacent grid points to $\mathbf{x}_0$ with given function values $\{u(\mathbf{x}_j)\}_{j=1}^{J}$ and $u(\mathbf{x}_0)$, respectively. In practice, the coefficients $\{a_j\}_{j=0}^{J}$ are to be optimized, which depend on the position of the grid points. Indeed, we apply this approximation for a spatial discretization of a domain Ω, which can be regarded as a grid. The aim of such an approximation is the numerical solution of some PDE. To demonstrate the principles, we consider the simple Poisson's problem

$$\begin{cases} \Delta u(\mathbf{x}) = f(\mathbf{x}) & \mathbf{x} \in \Omega \\ u(\mathbf{x}) = g(\mathbf{x}) & \mathbf{x} \in \partial\Omega, \end{cases} \tag{2}$$

where $\Omega \subset \mathbb{R}^2$ denotes the computational domain, $u : \Omega \to \mathbb{R}$ is the unknown function and $g : \partial\Omega \to \mathbb{R}$ is given. The well-posedness of (2) is ensured in $H^1(\Omega)$ provided that Ω is a bounded Lipschitz domain, $f \in H^{-1}(\Omega)$ and $g \in H^{\frac{1}{2}}(\partial\Omega)$ (see [16]).
Concerning the numerical solution of (2), in the most easy case, using a uniform rectangular grid, we have locally the geometric setup shown on the left in Figure 1. Here, for the approximation of $\Delta u(\mathbf{x}_0)$, we use the function values at $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ and $\mathbf{x}_4$, which are considered the neighbors of $\mathbf{x}_0$.

Then, in a given grid point $\mathbf{x}_0$, we have the classical five-point approximation

$$\Delta u(\mathbf{x}_0) \approx \frac{u(\mathbf{x}_1) - 2u(\mathbf{x}_0) + u(\mathbf{x}_3)}{h_x^2} + \frac{u(\mathbf{x}_2) - 2u(\mathbf{x}_0) + u(\mathbf{x}_4)}{h_y^2}, \tag{3}$$

so that in this case, in (1), we will have

$$a_0 = -\frac{2}{h_x^2} - \frac{2}{h_y^2}, \quad a_1 = a_3 = \frac{1}{h_x^2} \quad \text{and} \quad a_2 = a_4 = \frac{1}{h_y^2}. \tag{4}$$

This has the approximation order 2, i.e., for the difference of the two sides in (3), the choice in (4) delivers the approximation order 2 with respect to both variables:

$$\Delta u(\mathbf{x}_0) - \frac{u(\mathbf{x}_1) - 2u(\mathbf{x}_0) + u(\mathbf{x}_3)}{h_x^2} + \frac{u(\mathbf{x}_2) - 2u(\mathbf{x}_0) + u(\mathbf{x}_4)}{h_y^2} = \mathcal{O}(h_x^2) + \mathcal{O}(h_y^2), \tag{5}$$

provided that the function u is four-times continuously differentiable. Whenever it seems to be a strict assumption, for a number of PDEs (such as diffusion problems), including the Laplacian operator, we have smooth analytic solutions. In practice, the pointwise approximations in (3) are assembled into a matrix $D_{h,\mathrm{FD}}$. Incorporating also the boundary

condition and using the vector $\mathbf{f}_h$ for the pointwise values of f in the grid points, we finally have a linear algebraic problem to solve:

$$D_{h,\mathrm{FD}}\mathbf{u}_{h,\mathrm{FD}} = \mathbf{f}_h \tag{6}$$

The solution $\mathbf{u}_{h,\mathrm{FD}}$ is then called a finite difference (FD) approximation of u in (2).

In another well-known case, one can discretize any domain into (possibly curved) triangles and, based on this tessellation, a first-order finite element (FE) approximation can be applied. This situation in a special geometric case is presented on the right in Figure 1. Here, the neighbors of $\mathbf{x}_0$ are the points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_6$.

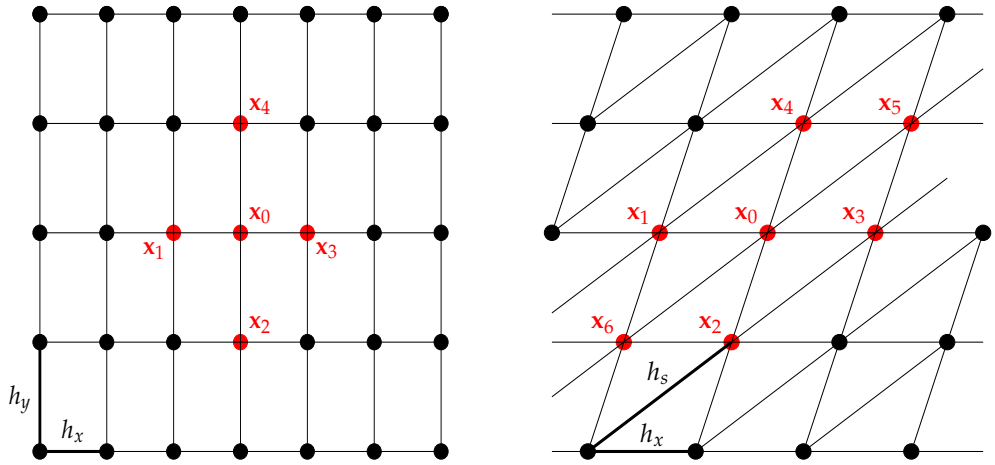

Figure 1. Two regular constellations of grid points: a fixed one $\mathbf{x}_0$ and its neighbors. They are presented on a uniform rectangular grid (**left**) and on a uniform simplicial (or triangular) grid (**right**). Neighboring points are connected. On the right-hand side, $h_s = \frac{5}{3}h_x$.

In this case, instead of the direct form of the Laplacian operator, its weak form is discretized. At the same time, in the course of the numerical solution of (2), this approach also leads to a linear system to solve:

$$D_{h,\mathrm{FE}}\mathbf{u}_{h,\mathrm{FE}} = \mathbf{f}_{h,\mathrm{FE}} \tag{7}$$

Here, the unknown vector $\mathbf{u}_{h,\mathrm{FE}} \in \mathbb{R}^N$ consists of coefficients of a finite element basis, while $\mathbf{f}_{h,\mathrm{FE}}$ represents the scalar product of f with the basis elements. Usually, a lifting also has to be applied to incorporate the boundary conditions. In any case, again, the matrix $D_h \in \mathbb{R}^{N\times N}$ can be regarded as an approximation of Δ on the entire domain Ω. Accordingly, in a specific case, shown on the right of Figure 1, we can determine the coefficients as in (4) to obtain

$$a_0 = -\frac{1}{h_x^2}\cdot\frac{44}{9}, \quad a_1 = a_2 = a_3 = a_4 = \frac{1}{h_x^2}\cdot\frac{25}{18} \quad \text{and} \quad a_5 = a_6 = -\frac{1}{h_x^2}\cdot\frac{1}{3}. \tag{8}$$

In the course of computing these coefficients, we have used the ratio $h_s = \frac{5}{3}h_x$. Note that this discretization leads to a second-order convergence with respect to the $L_2(\Omega)$-norm.

To compare the above cases, we also give the position of the nonzero elements and the total number of the nonzeros in the matrices $D_{h,\mathrm{FD}}$ and $D_{h,\mathrm{FE}}$ corresponding to the discretizations in (4) and (8), respectively. To visualize the nonzero entries of these matrices, we have depicted them in Figure 2.

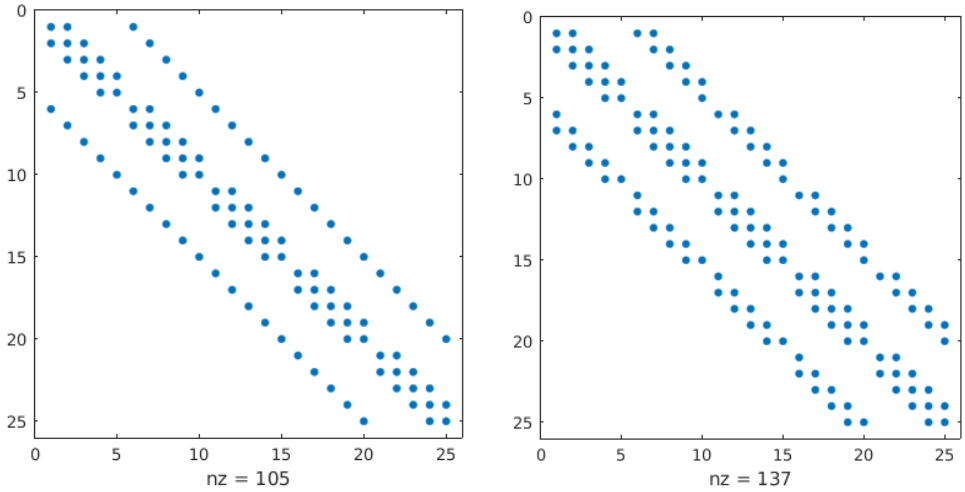

Figure 2. Location of nonzero elements in the discretization matrix $D_{h,\mathrm{FD}}$ corresponding to (4) (**left**) and $D_{h,\mathrm{FE}}$ corresponding to (8) (**right**), respectively. The total matrix size 25×25 corresponds to the number of the interior grid points. The conventional Matlab notation `nz` is for the number of nonzeros.

Whenever the finite element discretization is flexible regarding the geometry of the domain, compared to the simple finite difference approximation given by $D_{h,\mathrm{FD}}$, it has some drawbacks as well:

- We need more nonzero entries in the matrices $D_{h,\mathrm{FE}}$, as one can observe in Figure 2. It enhances both the assembling time of the matrices and the solution of the corresponding linear systems.
- We need to generate and store the entire structure of the mesh, including a list of triangles with their nodes and faces.
- In a nonuniform grid, we need to transform the gradient of basis functions on each of the triangles according to the transformation of the reference triangles computing Jacobian matrices and their inverses (see [13]).

To avoid all of these, our objective is to develop a finite difference approximation, which has the beneficial properties in (i)–(iii).

A direct way for constructing such a second-order approximation is, however, impossible even in the one-dimensional case using two neighboring grid points. This is stated, e.g., in an early contribution [17]. One should also note that, assuming a quasi-uniform one-dimensional mesh, the second-order accuracy can be preserved. Here, the difference in the neighboring grid distances should be one magnitude less compared to the grid size.

In a general case, a possible idea to obtain a second-order (or even higher-order) finite difference scheme without using a wide stencil (data from more neighboring grid points) is offered by the so-called method of compact difference schemes introduced in [18]. In this framework, the coefficients are not given explicitly; they have to be computed using a linear system. In the corresponding equalities, on the left hand side the linear combination of function values are given as in (1) or (3). At the same time, on the right hand side, again, a linear combination of the second-order derivatives arises. This idea was further developed; see, e.g., [19].

Another approach is to introduce further appropriate grid points, where the original simple finite differences deliver a higher-order approximation. Such a construction is analyzed in detail in [20], which also discusses the stability issues for the corresponding time-dependent problems.

We choose here an alternative way. Without introducing new variables or solving linear systems for optimal finite difference coefficients, an ANN will be utilized for constructing an optimal finite difference approximation.

2.2. Principles of the Present Algorithm

A natural easy choice could be to generalize (3) by preserving the second-order accuracy. As we have mentioned, we cannot just provide a simple formula for this. Instead, we could perform an optimization procedure to determine the coefficients, which approximate the Laplacian operator as accurately as possible.

Before digging into the details, we immediately reduce the setup of this problem:

- One can obviously assume the translation-invariant property of the approximation. Accordingly, we may assume that $\mathbf{x}_0 = (0,0)$.
- Also, any physically meaningful approximation should be scale-invariant. In concrete terms, if we have the optimal coefficients $\{a_j\}_{j=0}^4$ for some configuration $\{\mathbf{x}_j\}_{j=1}^4$, then we should have $\{\frac{1}{r^2}a_j\}_{j=0}^4$ for the configuration $\{r \cdot \mathbf{x}_j\}_{j=1}^4$.

However, a separate optimization procedure in all of the grid points would be extremely time-consuming. This is the point where artificial neural networks come into the picture and the following main principle is established.

An ANN should be trained to learn this optimization in the sense that it should give the coefficients $\{a_j\}_{j=0}^4$ *in* (1) *using the positions* $\{\mathbf{x}_j\}_{j=1}^4$ *as an input.*

Formally, we associate the mapping $\mathcal{N}$ to the ANN, such that

$$\mathcal{N} : (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \to (a_0, a_1, a_2, a_3, a_4)$$

First, we reduce the problem again by requiring that the Laplacian of the constant function is computed in $(0,0)$ exactly. This means that $\sum_{j=0}^{4} a_j = 0$, so that we have

$$a_0 = -\sum_{j=1}^{4} a_j.$$

With this simplification, we have to optimize only the set of parameters $\{a_1, a_2, a_3, a_4\}$.

Note that a second-order approximation can reconstruct the Laplacian for all at most second-order polynomials.

$$\begin{aligned} &p_0(x,y) = 1, \quad p_1(x,y) = x, \quad p_2(x,y) = y, \\ &p_3(x,y) = x^2, \quad p_4(x,y) = y^2, \quad p_5(x,y) = xy. \end{aligned} \tag{9}$$

2.3. Construction of Learning Data

A main cornerstone for the practical construction of a corresponding ANN is to create a suitable learning data set. To obtain this, we compute the optimal coefficients $\{a_j\}_{j=1}^4$ for a number of possible positions $\{\mathbf{x}_j\}_{j=1}^4$. In concrete terms, we start from the standard geometry $\mathbf{x}_1 = (-1,0)$, $\mathbf{x}_2 = (0,-1)$, $\mathbf{x}_3 = (1,0)$, $\mathbf{x}_4 = (0,1)$. By means of the scale-invariant property, this will be sufficient for all configurations. Adding a relatively small random number to all coordinates, we perturb them to have

$$\tilde{\mathbf{x}}_1 = (-1 + d_{11}, d_{12}),\ \tilde{\mathbf{x}}_2 = (d_{21}, -1 + d_{22}),\ \tilde{\mathbf{x}}_3 = (1 + d_{31}, d_{32}),\ \tilde{\mathbf{x}}_4 = (d_{41}, 1 + d_{42}).$$

as shown in Figure 3. This can be performed simultaneously to obtain a nonuniform tessellation of a computational domain Ω (see Figure 4).

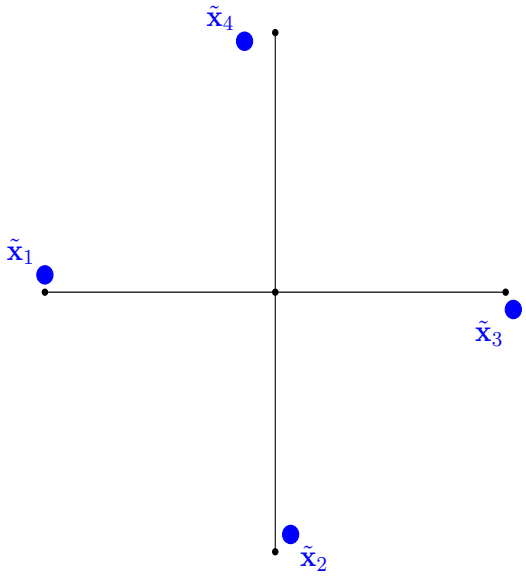

Figure 3. The standard position of neighboring discretization points (small black bullets) and their perturbations (large blue bullets) to obtain learning data. This can be considered the perturbation of the neighboring relation in the left of Figure 1.

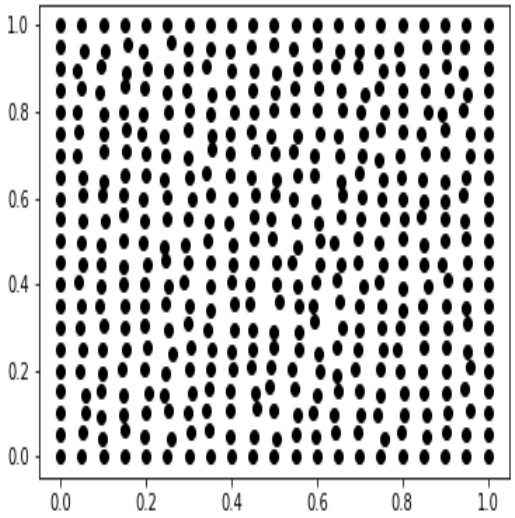

Figure 4. Grid points in a square-shaped computational domain obtained with a simultaneous perturbation of a uniform square grid. The boundary grid points are unchanged.

For this given data, we approximate the optimal coefficients $\{a_j\}_{j=1}^4$ as follows. In the perturbed grid points $\{\tilde{\mathbf{x}}_j\}_{j=1}^4$, we "solve" the system of equations

$$\begin{pmatrix} 100p_1(\tilde{\mathbf{x}}_1) & 100p_1(\tilde{\mathbf{x}}_2) & 100p_1(\tilde{\mathbf{x}}_3) & 100p_1(\tilde{\mathbf{x}}_4) \\ 100p_2(\tilde{\mathbf{x}}_1) & 100p_2(\tilde{\mathbf{x}}_2) & 100p_2(\tilde{\mathbf{x}}_3) & 100p_2(\tilde{\mathbf{x}}_4) \\ p_3(\tilde{\mathbf{x}}_1) & p_3(\tilde{\mathbf{x}}_2) & p_3(\tilde{\mathbf{x}}_3) & p_3(\tilde{\mathbf{x}}_4) \\ p_4(\tilde{\mathbf{x}}_1) & p_4(\tilde{\mathbf{x}}_2) & p_4(\tilde{\mathbf{x}}_3) & p_4(\tilde{\mathbf{x}}_4) \\ p_5(\tilde{\mathbf{x}}_1) & p_5(\tilde{\mathbf{x}}_2) & p_5(\tilde{\mathbf{x}}_3) & p_5(\tilde{\mathbf{x}}_4) \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 2 \\ 2 \\ 0 \end{pmatrix} \tag{10}$$

for the unknown coefficients (a_1, a_2, a_3, a_4) in the least-square sense. In rough terms, we are looking for the coefficients (a_1, a_2, a_3, a_4), which can reconstruct the Laplacian for the polynomials p_1, p_2, p_3, p_4 and p_5 in (9) as accurately as possible.

Here, we have applied a weight of 100 for the first-order polynomials. This is a hyperparameter: the above value could deliver a good performance regarding the final

approximation properties. The motivation for applying this is to ensure the first-order accuracy.

In the implementation, we use the efficient approximative solver `numpy.linalg.lstsq` in Python for the problem in (10). This was performed for a number of 1600 random perturbations. Note that both of the random perturbations and the least-square solution can be implemented in a vectorized form, enhancing the efficiency of the whole procedure.

2.4. Setup of the Artificial Neural Network

Having an appropriate set of learning data at hand, the final practical issue is to design a feasible neural network for computing the above coefficients $\{a_j\}_{j=1}^4$. In any case, to keep the computational costs at a low level, this should contain relatively few parameters. Also, since the mapping $\mathcal{N}$ is nonlinear, we need to incorporate nonzero activation functions. In concrete terms, we have applied the so-called SeLu activation function, which is given with

$$\mathrm{SeLu} : \mathbb{R} \to \mathbb{R},\ \mathrm{SeLu}(x) = \begin{cases} \alpha_0 x & x > 0 \\ \alpha_0 \cdot \alpha_1 e^{x-1} & x \leq 0. \end{cases}$$

Within the Keras package of Python, we have used the standard parameters $\alpha_0 \approx 1.0507$ and $\alpha_1 \approx 1.6733$, respectively.

After a long series of experiments, the following structure is proposed:

- The input layer is of size 8, obtaining the 8 coordinates of $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$.
- The first hidden layer, a dense one, is of size 4, using the SeLu activation function and a bias term.
- The second hidden layer is again a dense one of size 12 using the SeLu activation function and a bias term.
- Finally, the output layer is also dense and of size 4, and it is equipped with a linear activation function without a bias term.

The above structure is displayed in Figure 5.

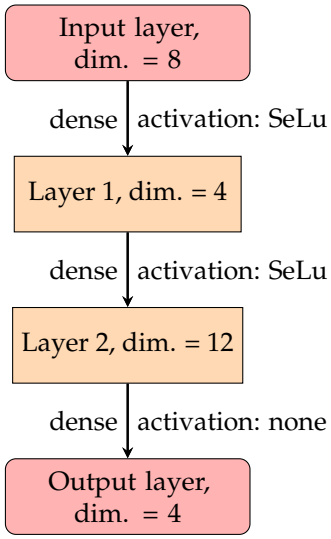

Figure 5. The structure of the ANN for computing the finite difference coefficients.

Altogether, this setup contains 144 parameters, which have to be optimized during the learning procedure. Note that in the majority of the applications of ANNs a much larger number of parameters is used. At the same time, for the efficiency of the algorithm, we need a really simple setup, which is comparable to the complexity of a single matrix-vector product. Also, if we obtain an acceptable accuracy with 144 parameters, a more sophisticated system could suffer from overfitting and would require an extra regularization procedure.

Altogether, the algorithm to build a matrix $D_{h,\mathrm{FD}}$ for the approximation of the Laplacian consists of the following steps:

(i) Generate randomly perturbed grid points and training data using the least-square approximation for (10).
(ii) Construct the ANN in Figure 5 and train it using the above data.
(iii) Construct a quadrilateral spatial grid on the computational domain Ω.
(iv) For each grid point $\mathbf{x}_0$ and its neighbors $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ and $\mathbf{x}_4$, we apply an affine linear mapping to transform the midpoint to the origin and scale one distance to be *one*.
(v) We apply the ANN to the above setup and scale back the distances to obtain the finite difference coefficients (a_1, a_2, a_3, a_4) for the neighboring points.
(vi) We compute $a_0 = -a_1 - a_2 - a_3 - a_4$ and have all the coefficients at hand.
(vii) For an arbitrary function u, we can approximate its Laplacian in all interior points by assembling the matrix $D_{h,\mathrm{NN}}$.

Remarks:

1. Note that in Python the above operation of the ANN can also be vectorized, leading to an efficient implementation.
2. We can incorporate any inhomogeneous Dirichlet boundary data, since these values are simply given on the boundary grid points.
3. We could make this procedure completely mesh-free using only grid points. At the same time, using a quadrilateral mesh, the neighbors of a certain grid point can be specified automatically. Also, these kind of meshes are practically useful (see, e.g., [21]) and these can be obtained using the frequently used mesh generators.
4. Here, $D_{h,\mathrm{NN}}$ has the same structure as $D_{h,\mathrm{FD}}$ (see the left of Figure 2), but the nonzero entries were computed using our ANN.

In practical cases, for the numerical solution of (2), we not only have to approximate the Laplacian but we also have to solve a problem given in (6).

3. A Summary: The ANN-Assisted Numerical Method

By incorporating the ANN-based approximation into the conventional FD numerical method, we summarized the proposed PDE solver for the Laplacian problem in Figure 6.

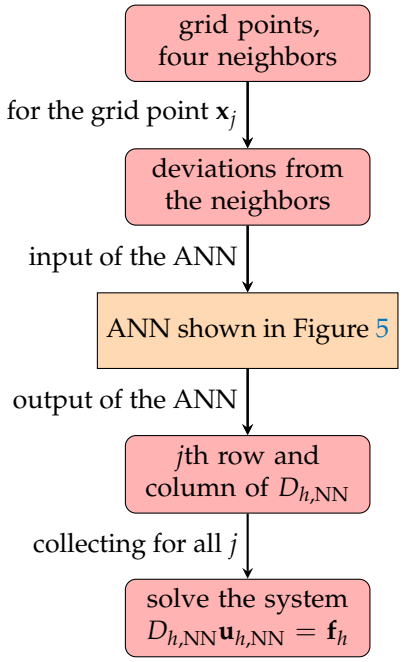

Figure 6. Block diagram of our entire algorithm for the numerical solution of (2).

4. Results: An Experimental Analysis

We first test the performance of the ANN, then the approximation property of the matrix $D_{h,\mathrm{FD}}$ and, finally, the computational error regarding $u_{h,\mathrm{FD}}$.

4.1. Performance of the ANN

In all experiments, we use the same ANN. The corresponding hyperparameters in the training procedure are chosen as follows:

- Learning rate: 0.002;
- Proportion of training and validation data: 80–20%;
- Number of epochs: 400;
- Batch size: 80;
- Loss function: mean squared error;
- Optimizer: ADAM.

The performance of the learning algorithm is shown in Figure 7.

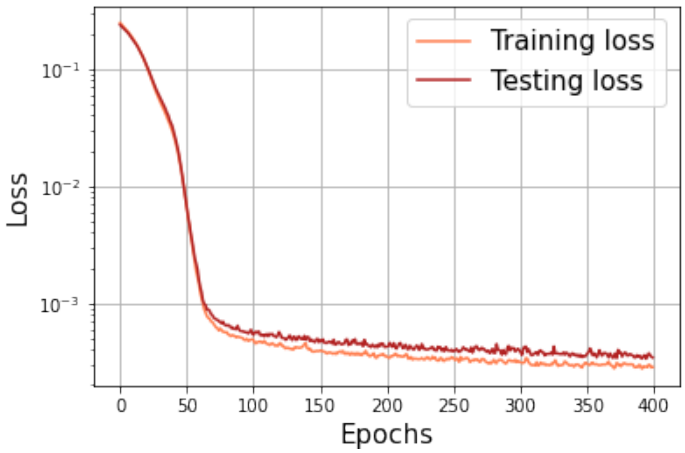

Figure 7. Learning performance of the ANN given in Section 2.4.

According to this, the ANN could learn the the coefficients of the approximation. Also, in Figure 8, one can observe a good agreement of the testing and the training loss. Note that further increasing the size of layers does not result in a further decrease in the testing loss, while we need more epochs to arrive at the same level of accuracy.

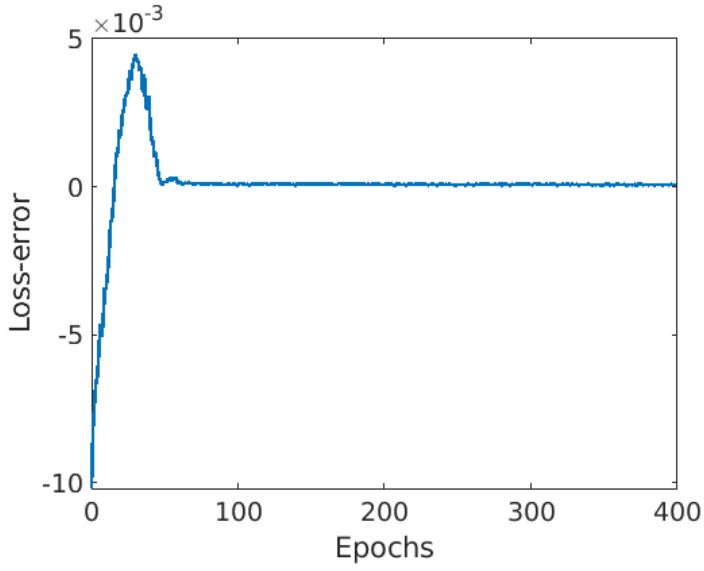

Figure 8. Error between the training and validation loss.

4.2. Performance of the ANN-Based Approximation of the Laplacian

In the first series of experiments, we investigate the accuracy of the numerical approximation of the Laplacian. In concrete terms, we compare the standard five-point approximation in (3) with the approximation provided by the trained ANN.

The computational domain is $\Omega = (0,1) \times (0,1)$, and we apply a nonuniform grid, as shown in Figure 4. Here, we have used the number of 21×21 grid points.

The approximation was tested in the case of the following model problem:

$$\begin{cases} \Delta u(x,y) = (2 - x^2y^2 - x^4) \cdot \cos xy - 4xy \sin xy \quad \text{in } \Omega \\ u(0,y) = 0,\ u(1,y) = \cos y,\ u(x,0) = x^2,\ u(x,1) = x^2 \cdot \cos x, \end{cases} \tag{11}$$

which has the analytic solution $u(x,y) = x^2 \cdot \cos xy$. Our aim here is to compare the pointwise approximation given by the ANN-based approximation with the Δ operator. To quantify this, we compute the quantity

$$\left(\sum_{\mathbf{x}_j} |\Delta u(\mathbf{x}_j) - (D_{h,\mathrm{FD}}\mathbf{u})_j|^2 \right)^{\frac{1}{2}}, \tag{12}$$

which is the ℓ_2-norm of the error of the approximation $D_{h,\mathrm{NN}} \approx \Delta$ on the grid. Here, $\mathbf{x}_j$ in the summation applies to all of the grid points.

We compare this quantity with the ℓ_2-norm

$$\left(\sum_{\mathbf{x}_j} |\Delta u(\mathbf{x}_j) - \frac{1}{0.05^2}(-4u(\mathbf{x}_j) + u(\mathbf{x}_{j+1}) + u(\mathbf{x}_{j+2}) + u(\mathbf{x}_{j+3}) + u(\mathbf{x}_{j+4}) \right)^{\frac{1}{2}}, \tag{13}$$

given by the standard five-point approximation. Here, according to the geometric setup in Figure 1, the indices $j+1, j+2, j+3$ and $j+4$ denote the indices of the neighboring grid points of $\mathbf{x}_j$.

While the first one is approximately 0.5, the second one is ≈6. This was the average of a number of experiments. Here, the random positions in Figure 4 can effect the above error terms.

This shows a real advance of our approach versus the conventional approximation. At the same time, we need to solve the problem in (11) numerically.

4.3. Performance of the ANN-Based Numerical Solution of (2)

In this series of experiments, we investigate the model problem

$$\begin{cases} \Delta u(x,y) = -16x \cos 4y \quad \text{in } \Omega \\ u(0,y) = y,\ u(\frac{\pi}{2},y) = y + \frac{\pi}{2} \cdot \cos 4y,\ u(x,0) = x \quad \text{on } \partial\Omega \\ u(x,1 + \frac{\cos x}{4}) = 1 + \frac{\cos x}{4} + x \cdot \cos 4(1 + \frac{\cos x}{4}) \quad \text{on } \partial\Omega, \end{cases} \tag{14}$$

where the computational domain is given by

$$\Omega = \{(x,y) : 0 < x < \frac{\pi}{2}, \quad 0 < y < 1 + \frac{\cos x}{4}\}.$$

Note that (14) has the analytic solution $u(x,y) = y + x \cdot \cos 4y$.

The domain Ω was discretized using a nonuniform grid. The motivation of this arises from the study of moving boundary problems for water waves [15], where the velocity potential exhibits more variation nearer to the free surface. In this way, for an equally accurate discretization, we need here more grid points compared to the bottom region. Also, the number of the grid points is frequently identical on each vertical segment. Using this convention, we can easily record the neighboring relations. A corresponding grid is shown in Figure 9.

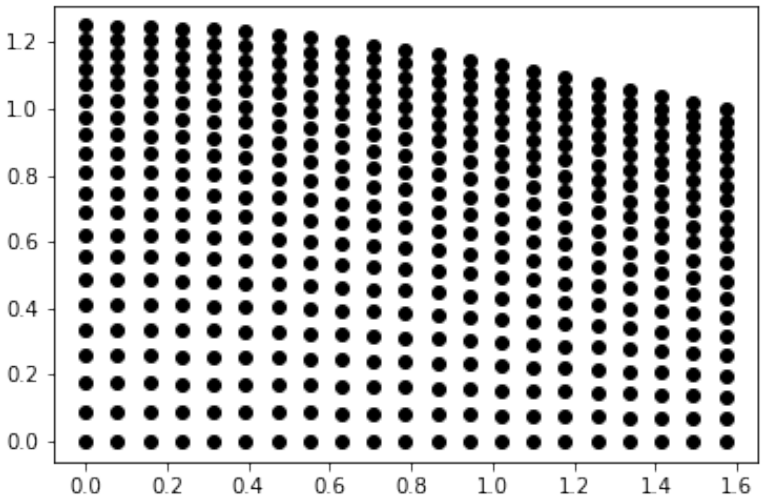

Figure 9. The computational domain Ω corresponding to the model problem in (14).

In concrete terms, we have 21 grid points along all vertical segments. Also, we have 21 segments in the discretization. Performing again a comparison between the ANN-based and conventional approximation of the Laplacian, the errors given by (12) and (13), we obtain 0.33 and 6.5, respectively. Here, we have a significant advance of our approach. Also, we have compared the final computational errors in the discrete L_2-norm to obtain

$$\left(\sum_{\mathbf{x}_j} |\mathbf{x}_j - \mathbf{x}_{j+1}| \cdot |\mathbf{x}_j - \mathbf{x}_{j+2}| \cdot |u(\mathbf{x}_j) - (\mathbf{u}_{h,\mathrm{NN}})_j|^2 \right)^{\frac{1}{2}} \approx 0.018$$

and

$$\left(\sum_{\mathbf{x}_j} |\mathbf{x}_j - \mathbf{x}_{j+1}| \cdot |\mathbf{x}_j - \mathbf{x}_{j+2}| \cdot |u(\mathbf{x}_j) - (\mathbf{u}_{h,\mathrm{FD}})_j|^2 \right)^{\frac{1}{2}} \approx 0.065.$$

Here, $\mathbf{u}_{h,\mathrm{FD}}$ denotes the numerical solution of (14) using the conventional finite difference method given with the (6) matrix based on the discretization in (13). Likewise, $\mathbf{u}_{h,\mathrm{NN}}$ denotes the solution of the equation $D_{h,\mathrm{NN}}\mathbf{u}_{h,\mathrm{NN}} = \mathbf{f}_h$, where $D_{h,\mathrm{NN}}$ is the discretization matrix obtained by our ANN. The comparison of the errors above also shows the advance of our approach.

Another way of obtaining an accurate discretization matrix is to perform the optimization procedure discussed in Section 2.3 pointwise in each $\mathbf{x}_0$. This, however, significantly increases the computational time. Regarding this, a comparison is shown in Table 1. The proposed ANN-based algorithm becomes faster only for a large number of grid points. This is typically the case in real-life situations.

Table 1. Computing time for the finite difference coefficients for different numbers of grid points.

# of Grid Points	$\mathbf{6 \cdot 10^6}$	$\mathbf{6 \cdot 10^5}$	$\mathbf{6 \cdot 10^4}$	$\mathbf{6 \cdot 10^3}$	$\mathbf{6 \cdot 10^2}$
ANN-based computation	126 s	13.6 s	1.36 s	0.286 s	0.0848 s
Pointwise optimization	331 s	33.3 s	3.3 s	0.339 s	0.0405 s

5. Discussion and Conclusions

We have presented an ANN-based finite difference approximation for Poisson problems on nonuniform grids. The ANN given in Section 2.4 could deliver an appropriate matrix $D_{h,\mathrm{FD}}$ to approximate the Laplacian. In the framework of an FD approximation (see Figure 6), this could also be used to enhance the accuracy of the numerical solution of

(2), as pointed out in Section 4.3. Also, our algorithms offer a fast computation of the FD coefficients (see Table 1).

The main benefit of the present approach is the application of this procedure for moving boundary problems. In the course of the numerical solution of these problems, we have to generate a new mesh in each time step, which requires a highly efficient procedure.

The construction of such a complex algorithm will be the continuation of this research.

Author Contributions: Conceptualization, F.I.; methodology, R.I.; software, F.I.; validation, R.I.; formal analysis, F.I.; investigation, R.I.; resources, R.I.; data curation, R.I.; writing—original draft preparation, F.I.; writing—review and editing, R.I.; visualization, R.I.; supervision, F.I.; project administration, F.I.; funding acquisition, F.I. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Research, Development and Innovation Office within the framework of the Thematic Excellence Program 2021—National Research Sub programme: "Artificial intelligence, large networks, data security: mathematical foundation and applications".

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

FD	Finite difference
FE	Finite element
PDE	Partial differential equation
PINN	Physics-informed neural network
NN	Neural network

References

1. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [CrossRef]
2. Cuomo, S.; Di Cola V.S.; Giampaolo, F.; Rozza, G.; Raissi, M.; Piccialli F. Scientific machine learning through physics-informed neural networks: Where we are and what's next. *J. Sci. Comput.* **2022**, *92*, 88.
3. Rao, C.; Sun, H.; Liu, Y. Physics-informed deep learning for incompressible laminar flows. *Theor. Appl. Mech. Lett.* **2020**, *10*, 207–212. [CrossRef]
4. Zhang, D.; Guo, L.; Karniadakis, G.E. Learning in modal space: Solving time-dependent stochastic PDEs using physics-informed neural networks. *SIAM J. Sci. Comput.* **2020**, *42*, 639–665. [CrossRef]
5. Yang, L.; Meng, X.; Karniadakis, G.E. B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *J. Comput. Phys.* **2021**, *425*, 109913. [CrossRef]
6. Jagtap, A.D.; Mao, Z.; Adams, N.; Karniadakis, G.E. Physics-informed neural networks for inverse problems in supersonic flows. *J. Comput. Phys.* **2022**, *466*, 111402. [CrossRef]
7. Izsák, F.; Djebbar, T.E. Learning Data for Neural-Network-Based Numerical Solution of PDEs: Application to Dirichlet-to-Neumann Problems. *Algorithms* **2023**, *16*, 111. [CrossRef]
8. Kovachki, N.B.; Li, Z.; Liu, B.; Azizzadenesheli, K.; Bhattacharya, K.; Stuart, A.M.; Anandkumar, A. Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. *J. Mach. Learn. Res.* **2023**, *24*, 1–97.
9. Tanyu, D.N.; Ning, J.; Freudenberg, T.; Heilenkötter, N.; Rademacher, A.; Iben, U.; Maass, P. Deep Learning Methods for Partial Differential Equations and Related Parameter Identification Problems. *arXiv* **2022**, arXiv:2212.03130.
10. Kato, H.; Ushiku, Y.; Harada, T. Neural 3D Mesh Renderer. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 3907–3916.
11. Song, T.; Wang, J.; Xu, D.; Wei, W.; Han, R.; Meng, F.; Li, Y.; Xie, P. Unsupervised Machine Learning for Improved Delaunay Triangulation. *J. Mar. Sci. Eng.* **2021**, *9*, 1398. [CrossRef]
12. Zhou, Y.; Cai, X.; Zhao, Q.; Xiao, Z.; Xu, G. Quadrilateral Mesh Generation Method Based on Convolutional Neural Network. *Information* **2023**, *14*, 273. [CrossRef]
13. Ern, A., Guermond, J-L. *Theory and Practice of Finite Elements*; Springer: New York, NY, USA, 2004; pp. 357–380.
14. Horvath, T.L.; Rhebergen, S. A conforming sliding mesh technique for an embedded-hybridized discontinuous Galerkin discretization for fluid-rigid body interaction. *Int. J. Numer. Meth. Fluids* **2022**, *94*, 1784–1809. [CrossRef]

15. Brink, F.; Izsák, F.; van der Vegt, J.J.W. Hamiltonian Finite Element Discretization for Nonlinear Free Surface Water Waves. *J. Sci. Comput.* **2017**, *73*, 366–394. [CrossRef]
16. McLean, W. *Strongly Elliptic Systems and Boundary Integral Equations*; Cambridge University Press: Cambridge, UK, 2000; pp. 128–130.
17. Kálnay de Rivas, E. On the use of nonuniform grids in finite-difference equations. *J. Comp. Phys.* **1972**, *10*, 202–210. [CrossRef]
18. Gamet, L.; Ducros, F.; Nicoud, F.; Poinsot, T. Compact finite difference schemes on non-uniform meshes. Application to direct numerical simulations of compressible flows. *Int. J. Numer. Meth. Fluids* **2022**, *29*, 159–191. [CrossRef]
19. Chen, C.; Zhang, X.; Liu, Z.; Zhang, Y. A new high-order compact finite difference scheme based on precise integration method for the numerical simulation of parabolic equations. *Adv. Differ. Equ.* **2020**, *15*, 15. [CrossRef]
20. Matus, P.P.; Hieu, L.M. Difference Schemes on Nonuniform Grids for the Two-Dimensional Convection–Diffusion Equation. *Comput. Math. Math. Phys.* **2017**, *57*, 1994–2004. [CrossRef]
21. Shan, Z.; Chen, L.; Wang, Q.; Wu, H.; Gao, B. Simplified quadrilateral grid generation of complex free-form gridshells by surface fitting. *J. Build. Eng.* **2022**, *48*, 103827. [CrossRef]