Introducing B-spline basis functions in neural network approximations

Maciej Sikora[1][0009−0006−4465−2395], Kamil Doległo[1], Anna Paszyńska[2][0000−0002−0716−0619], and Maciej Paszyński[1][0000−0001−7766−6052]

[1]AGH University of Krakow, Poland
maciej.paszynski@agh.edu.pl
[2]Jagiellonian University, Krakow, Poland

Abstract. In the finite element method (FEM), the solutions of Partial Differential Equations (PDEs) are approximated using linear combinations of prescribed basis functions. The coefficients of the linear combinations are obtained by solving a system of linear equations. The FEM allows for the solution of a PDE for fixed values of the PDE parameters. It is not possible to obtain "at once" the family of solutions of parametric PDEs using FEM. We proposed to introduce B-spline basis functions into neural network approximations, where the coefficients of the basis functions used to approximate the solution are predicted by a neural network. Direct approximation of B-spline coefficients by NN has several advantages compared to standard FEM. First, it allows us to obtain a family of solutions of the parametric PDE "at once". The PDE parameters are input to the neural network, and the output involves the coefficients of the basis functions. Second, it allows obtaining the solution of a parametric PDE without the construction and solution of a system of linear equations. Third, since neural networks are universal approximators, direct approximation of B-spline coefficients by NN may find a dependence between the PDE parameters and the coefficients of the basis functions used to approximate the solution. The training of our method requires learning the dependence between the PDE parameters and the basis functions' coefficients. The approximations of B-spline coefficients by NN inherit all the features of standard FEM approximations.

Keywords: Finite Element Method, Isogeometric Analysis, Deep Neural Networks, Partial Differential Equations

1 Introduction

The classical way of solving PDEs numerically is based on the Finite Element Method (FEM). In FEM, we approximate the solution of the PDE by using a linear combination of the prescribed basis functions. The coefficients of the basis functions are obtained by solving a system of linear equations. The most accurate version of the FEM employs higher-order and continuity B-spline basis functions [9, 14, 2].

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8

The neural networks are the universal approximators [8]. They can successfully replace or support the FEM computations. Recently, there has been a growing interest in the design and training of neural networks for solving PDEs [4, 13, 3, 16]. The most popular method for training the DNN solutions of PDEs is Physics Informed Neural Networks (PINN) [19]. Since its introduction in 2019, there has been exponential growth in the number of papers and citations related to them (Web of Science search for "Physics Informed Neural Network"). It forms an attractive alternative for solving PDEs in comparison with traditional solvers such as the Finite Element Method (FEM) or Isogeometric Analysis (IGA). Physics Informed Neural Network proposed in 2019 by Prof. Karniadakis revolutionized the way in which neural networks find solutions to initial-value problems described by means of partial differential equations [19] Karniadakis has also proposed Variational Physics Informed Neural Networks VPINN [10]. VPINNs use the loss function with a weak (variational) formulation. In VPINN, we approximate the solution with a DNN (as in the PINN), but during the training process, instead of probing the loss function at points, we employ the test functions from a variational formulation to average the loss function (to average the PDE over a given domain). Karniadakis also showed that VPINNs could be extended to *hp*-VPINNs (*hp*-Variational Physics Informed Neural Networks) [11, 20], where by means of *hp*-adaptation (*h*-adaptation is breaking elements, and *p*-adaptation is raising the degrees of base polynomials) it is possible to solve problems with singularities. The incorporation of the domain decomposition methods into VPINNs is also included in the RAR-PINN method [18]. In conclusion, a family of PINN solvers based on neural networks has been developed, ranging from PINNs, VPINNs, and hp-VPINNs to RAR-PINNs.

The parametric PDEs are of great interest to the research community. The PDEs depend on several model parameters. One set of parameters instantiates the PDEs and results in one solution. Ideally, we would like to compute a family of solutions of the parametric PDEs, the functions of the PDE parameters. However, the FEM does not allow obtaining a family of solutions for the parametric PDEs "at once". A fixed set of model parameters requires the construction and solution of a single system of linear equations. Solving the parametric system of linear equations using algebraic methods is impossible. We need to formulate and solve a new system of linear equations for a new set of model parameters.

In this paper, we propose the Neural Network learning coefficients of B-splines from higher-order Finite Element Method (FEM) solver. We extend and develop further the ideas initially proposed in a short paper [6]. First, we fix the higher-order B-spline basis functions used to approximate the solutions of the parametric PDEs. Then, the neural network provides the coefficients of the linear combinations of the basis functions. The neural network learns the dependence between the parameters of the PDEs and the coefficients of the linear combinations. We employ the supervised training based on several snapshots of the FEM solutions obtained for selected values of the model parameters. Our method has several advantages with respect to higher-order FEM solvers. It allows for obtaining the family of solutions to the parametric PDEs "at once".

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8

The PDE parameters are input to the neural network, and the output involves the coefficients of the basis functions. Since neural networks are universal approximators, we may find a dependence between the PDE parameters and the coefficients of the basis functions used to approximate the solution. We only train the dependence of the coefficients of the linear combinations on the PDEs parameters. Our findings are illustrated on one-dimensional heat transfer problem and two-dimensional model L-shape domain problem.

2 One dimensional heat-transfer problem

Let us introduce the quadratic B-spline basis functions over [0,1] (see Figure 1) $B_1(x) = (1-x)^2; \quad B_2(x) = 2x(1-x); \quad B_3(x) = x^2.$

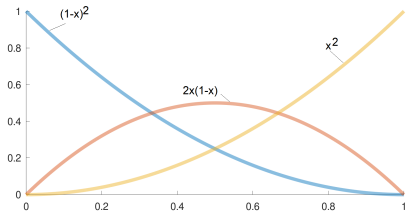

Fig. 1: Three B-splines over a single interval (element)

Let us introduce the parametric PDE with with boundary conditions (b.c.)

$$-u_n''(x) = f_n(x) \quad x \in (0,1), n \in (0,1), \quad u_n(0) = 0, u_n'(1) = g_n(x). \tag{1}$$

$g_n(x) = n\pi cos(n\pi x)$ and $f_n(x) = n^2\pi^2 sin(n\pi x)$. The family of solution is $u_n(x) = sin(n\pi x)$.

2.1 Traditional finite element method solver

The FEM can solve the parametric PDE for one selected value of n. For this purpose, we transform this problem into the weak form, multiplying by a test function v (being a distribution of the averaging of the problem) and integrating (averaging with distribution v) over the domain

$$-\int_0^1 u_n''(x)v(x)dx = \int_0^1 f_n(x)v(x)dx \quad \forall v, v(0) = 0 \tag{2}$$

We select the test functions v, so they are $v(0) = 0$ to be in agreement with the b.c. $u_n(0) = 0$. We integrate by parts

$$\underbrace{\int_0^1 u_n'(x)v'(x)dx - v(1)u_n'(1) + v(0)u_n'(0)}_{-\int_0^1 u_n''(x)v(x)dx} = \int_0^1 f_n(x)v(x)dx \quad \forall v, v(0) = 0 \tag{3}$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8

and we apply $v(0) = 0$ and $u_n'(1) = g_n(1)$

$$\int_0^1 u_n'(x)v'(x)dx = \int_0^1 f_n(x)v(x)dx + v(1)g_n(1) \quad \forall v, v(0) = 0 \tag{4}$$

In traditional FEM, we select n, and we seek the solution as a linear combination of the basis functions $u_n(x) = \sum_{i=1,2,3} u_i B_i(x)$, we also select three test functions $v \in \{B_1(x), B_2(x), B_3(x)\}$ to obtain

$$\begin{bmatrix} \int B_1'(x)B_1'(x)dx & \int B_1'(x)B_2'(x)dx & \int B_1'(x)B_3'(x)dx \\ \int B_2'(x)B_1'(x)dx & \int B_2'(x)B_2'(x)dx & \int B_2'(x)B_3'(x)dx \\ \int B_3'(x)B_1'(x)dx & \int B_3'(x)B_2'(x)dx & \int B_3'(x)B_3'(x)dx \end{bmatrix}$$

$$\begin{bmatrix} u_1(n) \\ u_2(n) \\ u_3(n) \end{bmatrix} = \begin{bmatrix} \int B_1(x)f_n(x)dx \\ \int B_2(x)f_n(x)dx \\ \int B_3(x)f_n(x)dx + n\pi cos(n\pi 1) \end{bmatrix} \tag{5}$$

(where for the sake of saving space we skept the integral limits $\int_0^1$). We compute the integrals $\int_0^1 B_i(x)B_j(x)dx$ and $\int_0^1 B_i(x)sin(n\pi x)dx$ and solve the system. The solution from traditional FEM for fixed n is $u_n(x) = u_1(n)(1-x)^2 + u_2(n)2x(1-x) + u_3(n)x^2$. To obtain the values of $u_1(n), u_2(n), u_3(n)$ for a given n, we need to generate the new right-hand-side and solve the system (5) again for new n. The cost the solver for 1D problems is $\mathcal{O}(Np)$, for 2D problems it is $\mathcal{O}(N^{1.5}p^2)$ and for 3D problems it is $\mathcal{O}(N^2p^3)$ [5] where N is the number of basis functions, and p is their polynomial order.

2.2 Neural network learning solution of parametric PDE directly from FEM solver

Let us introduce the artificial neural network $ANN(n,x) = c\sigma\left(\begin{bmatrix} a_1 a_2 \end{bmatrix}\begin{bmatrix} n \\ x \end{bmatrix} + b\right) + d$ and the activation function $\sigma(x) = \frac{1}{1+e^{-x}}$. We prepare a set of samples: input (n,x), output $y(n,x) = u_1B_1 + u_2B_2 + u_3B_3$ (obtained by calling a traditional FEM solver many times). We randomly select $n \in (0,1)$ and $x \in (0,1)$. We solve the problem (5) to obtain (u_1, u_2, u_3). We define the loss function

$$\begin{aligned} LOSS_{fem}(n,x) &= \frac{1}{2}\left(ANN(n,x) - y(n,x)\right)^2 = \\ &\frac{1}{2}\left(c\sigma\left(a_1 n + a_2 x + b\right) + d - y(n,x)\right)^2 = \\ \frac{1}{2}&\left(\left(\frac{c}{1+exp(-a_1 n - a_2 x - b)} + d\right) - y(n,x)\right)^2 \end{aligned} \tag{6}$$

We loop through data set $\{(n,x), y\}$ for $n \in (0,1), x \in (0,1)$, and we train ANN

1. Select $n \in (0,1), x \in (0,1)$
2. Compute $ANN(n,x) = c\sigma\left(\begin{bmatrix} a_1 a_2 \end{bmatrix}\begin{bmatrix} n \\ x \end{bmatrix} + b\right) + d$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8

3. Compute $LOSS_{fem}(n,x)$
4. Compute $\frac{\partial LOSS_{fem}(n,x)}{\partial k}, k \in \{a_1, a_2, b, c, d\}$
5. Correct using $\eta \in (0,1)$

$$a_1 = a_1 - \eta \frac{\partial LOSS_{fem}(n,x)}{\partial a_1}, \quad a_2 = a_2 - \eta \frac{\partial LOSS_{fem}(n,x)}{\partial a_2} \tag{7}$$

$$b = b - \eta \frac{\partial LOSS_{fem}(n,x)}{\partial b}, \quad c = c - \eta \frac{\partial LOSS_{fem}(n,x)}{\partial c} \tag{8}$$

$$d = d - \eta \frac{\partial LOSS_{fem}(n,x)}{\partial d} \tag{9}$$

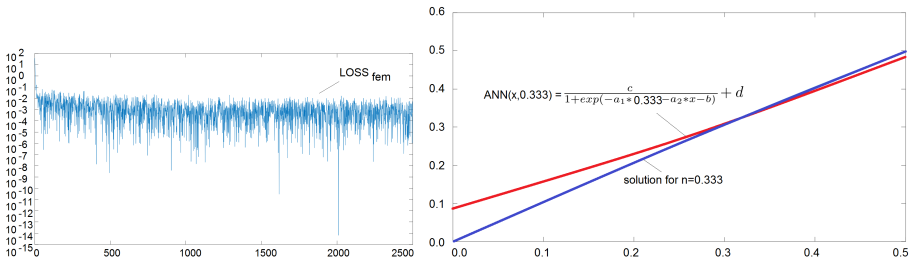

Fig. 2: Convergence of error for training of ANN. Verification of the neural network approximation of solution for $n = 0.333$ with $ANN(n,x) = \frac{c}{1+exp(-a_1 n - a_2 x - b)} + d$

In Figure 2 we present the convergence of the training. We select $n = 0.333$ and we compute $ANN(n,x) = \frac{c}{1+exp(-a_1 n - a_2 x - b)} + d$ and compare with $sin(0.333\pi x)$ in Figure 2.

2.3 Artificial neural network learning coefficients of FEM basis functions from FEM solver solutions

We introduce the artificial neural network $ANN_i(n) = u_i$, where n is the PDE parameter, $i = 1,2,3$ represents the three coefficients of basis functions $ANN_i(n) = c_i \sigma(a_i n + b_i) + d_i$.. We prepare a set of samples Input data (n), output data (u_1, u_2, u_3) (obtained by calling the FEM solver many times). We randomly select $n \in (0,1)$. We solve the FEM problem (5) to obtain $y(x) = u_1 B_1(x) + u_2 B_2(x) + u_3 B_3(x)$ (and we store u_1, u_2, u_3). Input data (n), output data (u_1, u_2, u_3). We define the loss function

$$LOSS_i(n) = \frac{1}{2}\left(ANN_i(n) - u_i(n)\right)^2 = \frac{1}{2}\left(c_i\sigma\left(a_i n + b_i\right) + d_i - u_i(n)\right)^2 = \frac{1}{2}\left(\left(\frac{c_i}{1+exp(-\left(a_i n + b_i\right)} + d_i\right) - u_i(n)\right)^2 \tag{10}$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8

We loop through the data set $\{n, (u_1(n), u_2(n), u_3(n))\}$ where $n \in (0,1)$, and we train ANN_1, ANN_2, and ANN_3

1. Select $n \in (0,1)$
2. Compute $u_i = ANN_i(n) = c_i \sigma\left(a_i n + b_i\right) + d_i$
3. Compute $LOSS_i(n)$
4. Compute $\frac{\partial LOSS_i(n)}{\partial a_i}, \frac{\partial LOSS_i(n)}{\partial b_i}, \frac{\partial LOSS_i(n)}{\partial c_i}, \frac{\partial LOSS_i(n)}{\partial d_i}$
5. Correct using $\eta \in (0,1)$

$$a_i = a_i - \eta * \frac{\partial LOSS_i(n)}{\partial a_i}, \quad b_i = b_i - \eta * \frac{\partial LOSS_i(n)}{\partial b_i} \tag{11}$$

$$c_i = c_i - \eta * \frac{\partial LOSS_i(n)}{\partial c_i}, \quad d_i = d_i - \eta * \frac{\partial LOSS_i(n)}{\partial d_i} \tag{12}$$

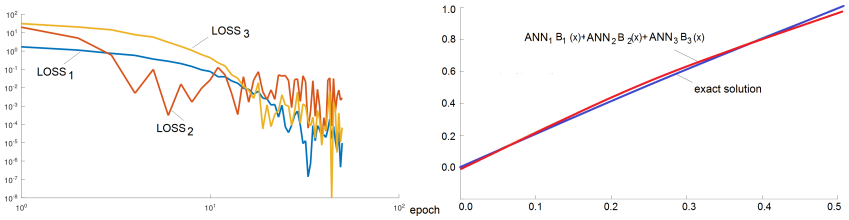

Fig. 3: Convergence of errors for training of ANN1, ANN2, ANN3. Comparison of the approximation by the neural network and exact f $n = 0.333$

In Figure 3 we present the convergence of the training. We select $n = 0.333$ and we compute

$$\begin{aligned} z(x) &= ANN_1(n)B_1(x) + ANN_2(n)B_2(x) + ANN_3(n)B_3(x) \\ &= \left(\frac{c_1}{1 + exp(-a_1 n - b_1)} + d_1\right)(1-x)^2 + \\ \left(\frac{c_2}{1 + exp(-a_2 x - b_2)} + d_2\right) 2x(1-x)^2 &+ \left(\frac{c_3}{1 + exp(-a_3 x - b_3)} + d_3\right) x^2 \end{aligned} \tag{13}$$

we compare with $sin(0.333\pi x)$ in Figure 3.

3 Two-dimensional L-shape domain model problem

We will compare the NN trained using solution obtained from higher-order FEM solver, and the solution obtained with NN learning the pointwise coefficients of B-splines from higher-order FEM solver. We focus on the so-called L-shape domain model problem. This is a historical problem introduced by prof. Ivo

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8

Babuška [1] to test the convergence of different numerical methods [7, 17]. We solve the Laplace problem

$$\Delta u = 0 \text{ in } \Omega, \qquad u = 0 \text{ on } \Gamma_D, \qquad \frac{\partial u}{\partial n} = g \text{ on } \Gamma_N. \tag{14}$$

on the computational domain presented in Figure 4. The domain consists of $\Omega = (-1,1)^2 \backslash [-1,0]^2$. On the internal part of the domain $\Gamma_D = \{(x,y) : x \in (-1,0), y = 0\} \cup \{(x,y) : x = 0, y \in (-1,0)\}$ we introduce the Dirichlet boundary condition, where $u = 0$. On the external part of the domain $\Gamma_N = \partial\Omega \backslash \Gamma_D$ we introduce the Neumann boundary condition, where we prescribe the directional derivative of the solution $\frac{\partial u}{\partial n} = \nabla \cdot u = g$.

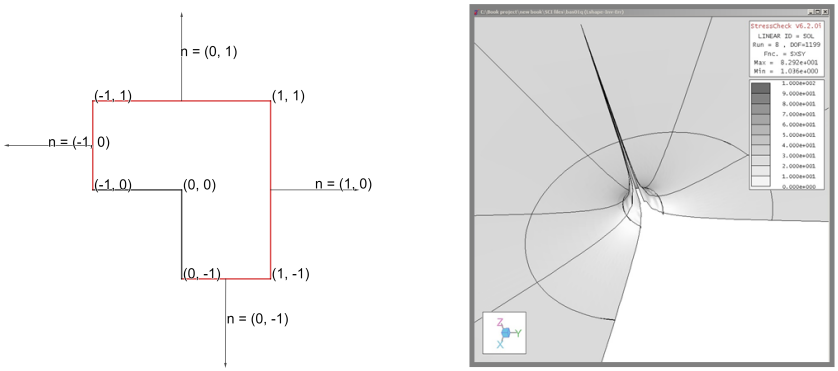

Fig. 4: The L-shape model problem. Plot of $\left(\frac{\partial u(x,y)}{\partial x}\right)^2 + \left(\frac{\partial u(x,y)}{\partial y}\right)^2$.

Usually, in the model problems designed to test the quality of the numerical methods, the Neumann boundary condition is defined in the following way. We assume the exact solution, for example, in our problem, we want to solve a family of problems parameterized by n, thus, our exact solution is

$$u_{exact}(x,y;n) = sin(2\pi n \cdot x) \cdot sin(2\pi n \cdot y), n \in (0,1). \tag{15}$$

In order to obtain this solution, we compute its normal derivatives on the boundary $g(x,y) = \frac{\partial u_{exact}}{\partial n}$. In other words, they force the exact solution by setting proper Neumann boundary conditions

$$g(x,y) = \begin{cases} \dfrac{\partial u_{exact}}{\partial x} = 2\pi \cos(2\pi n x) \sin(2\pi n y) & x = 1, y \in (-1,1), \\ \dfrac{\partial u_{exact}}{\partial x} = -2\pi \cos(2\pi n x) \sin(2\pi n y) & x = -1, y \in (0,1), \\ \dfrac{\partial u_{exact}}{\partial y} = 2\pi \sin(2\pi n x) \cos(2\pi n y) & x \in (-1,1), y = 1, \\ \dfrac{\partial u_{exact}}{\partial y} = -2\pi \sin(2\pi n x) \cos(2\pi n y) & x \in (0,1), y = -1. \end{cases} \tag{16}$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8

Why this problem is employed for testing of the quality of numerical solutions? If we plot the gradient of the solution, as illustrated in Figure 4, we can see that this gradient has a singularity at point $(0, 0)$. In other words, the value of the norm of the gradient goes to infinity at this point. The NN with higher-order FEM solver, since it employs gradients during the training process, they are indeed sensitive to gradient and this model problem is a good benchmark.

Layer	Number of neurons	Activation function
input	100 - 1000	ReLU
hidden layer 1	100 - 1000	ReLU
output 2 output 2	equal to the number of coefficients	none

Table 1: Network architecture of the NN used to approximate the coefficients of the B-spline basis functions for approximation of the model L-shape domain problem. The number of neurons in the hidden layer varies with the number of B-splines coefficients. For $13 \times 13 = 169$ output B-splines coefficients, the hidden layer has 100 neurons. For $45 \times 45 = 2025$ output B-splines coefficients, there are 1000 neurons in the hidden layer. ReLU stands for Rectified Linear Unit.

3.1 Solution of the L-shape problem with finite element method

In our method we use NN to approximate family of solution obtained from higher-order Finite Element Method (ho-FEM). It assumes that each solution is approximated with the linear combination of the basis functions. In our case, we select the B-spline basis functions, thus $u(x, y; n) \approx \sum_{i,j} NN(\theta; n)_{i,j} B_{i,p}(x) B_{j,p}(y)$ where the B-splines are defined by

$$B_{i,0}(\xi) = 1 \text{ for } \xi_i \leq \xi \leq \xi_{i+1}, \quad 0 \text{ otherwise,}$$
$$B_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} B_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} B_{i+1,p-1}(\xi) \tag{17}$$

The $NN(\theta; n)$ is the NN with parameters θ (weights of its layers), that takes model parameter n as argument, and return values of B-spline coefficients. The meaning of this formula is illustrated in Figure 5. The higher-order B-splines are created by multiplication of two lower-order B-splines by "triangles" (red lines in Figure 5).

The B-splines basis functions are defined using (17) and the knot vector $\xi = [\xi_i]_i$, e.g., $\xi =$[0, 0, 0, 0.5, 0.5, 1, 1, 1], with the assumption that the existence of the repeated knots in the denominator implies zero value of the corresponding term, e.g. $\frac{x-0}{0-0} = 0$ in (17). The tensor products of 1D B-splines create a 2D basis that approximates the solution. Exemplary 2D basis $\{B_{i,2}(x) B_{j,2}(y)\}_{i,j}$ obtained by tensor products of two sets of one-dimensional B-splines defined by knot vector [0, 0, 0, 0.5, 0.5, 1, 1, 1] are presented in Figure 5. Now, we want

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8

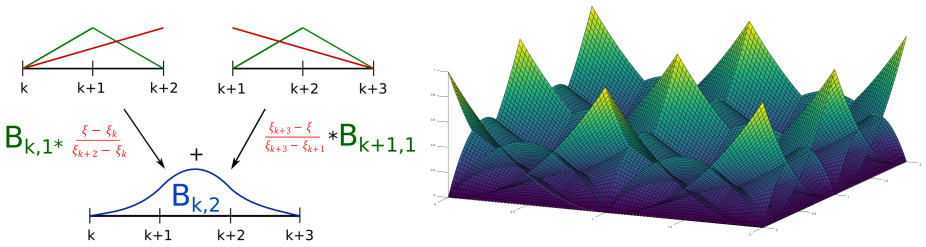

Fig. 5: Recursive definition of B-spline basis functions. 2D basis of the B-spline functions obtain from tensor product of two 1D basis of B-splines defined by the knot vector [0, 0, 0, 0.5, 0.5, 1, 1, 1]×[0, 0, 0, 0.5, 0.5, 1, 1, 1].

to know how to "stretch" the B-splines and how to make a linear combination with B-splines so they approximate a solution of PDE. We'd like the NN to learn a family of solutions of the form (15). With this approach, the number of coefficients is equal to the number of the network's output neurons; one neuron is responsible for one coefficient of one B-spline. There are two approaches possible for training NN the coefficients of B-splines; the first one is based on supervised learning, where we employ the knowledge about the values of the B-spline basis function coefficients obtained by solving the system of linear equations following the FEM formulations. The second one is based on unsupervised learning, and it will be a subject of our future work. The network architecture illustrated in

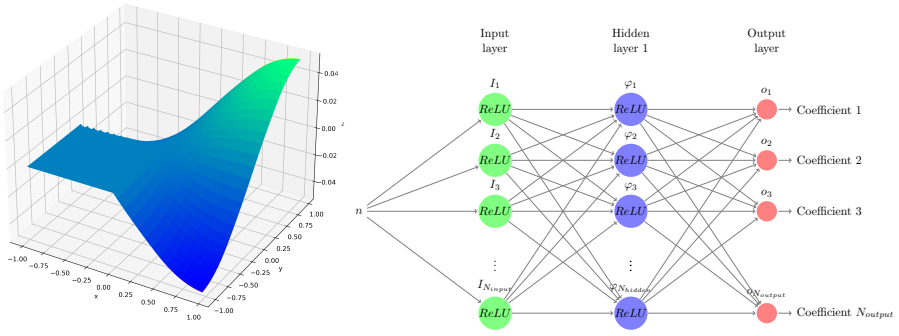

Fig. 6: **Left panel**: Visualization of the heat equation solution. There are two surfaces in this plot. The analytically calculated solution is in a green-blue color and the solution predicted by the neural network is in orange-red. Those surfaces practically overlap, so only one is visible. **Right panel**: Architecture of the neural network learning the coefficients of the B-spline basis functions approximating the solution of the L-shape model problem using the NN trained with the coefficients of B-splines. The number of neurons per layer i depends on the number of B-splines and their coefficients j.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8

Figure 6 and Table 1 varies with the number of B-splines - the output layer has to have the same number of neurons as coefficients. This is the main drawback of this approach, as the network has to be reconfigured and retrained when we change number of B-spline basis functions. In our method we apply the supervised learning, where the loss function is given by the MSE between the sets of B-spline parameters obtained by several calls to parameterised FEM solvers. Namely, $LOSS_1(\theta) = \left(\sum_{i,j} \left(NN(\theta; n)_{i,j} - u_{i,j}\right)\right)^2$, where $u_{i,j}$ are coefficients of B-splines approximating the ho-FEM solution for given n obtained by solving a system of equations resulting from ho-FEM discretization [15] using the weak formulation equivalent to (14). Here θ stands for the NN parameters learned during the training. For the FEM, the weak formulation must be derived to solve the system of equations and find values of $u_{i,j}$ coefficients. We multiply the equation (14) by test functions, and we integrate by parts to obtain the weak form: Find u such that

$$(\nabla u, \nabla v)_\Omega = (g, v)_{\Gamma_N} \quad \forall v \tag{18}$$

where $(u, v)_\Omega = \int_\Omega u(\mathbf{x})v(\mathbf{x})d\mathbf{x}$ and $(u, v)_{\Gamma_N} = \int_{\Gamma_N} u(\mathbf{x})v(\mathbf{x})dS$. The coefficients $u_{i,j}$ are obtained by solving a system of linear equations

$$\begin{bmatrix} (\nabla B_{1,1}, \nabla B_{1,1})_\Omega & \cdots & (\nabla B_{1,1}, \nabla B_{1,N})_\Omega \\ \cdots & \cdots & \cdots \\ (\nabla B_{N,N}, \nabla B_{1,1})_\Omega & \cdots & (\nabla B_{N,N}, \nabla B_{N,N})_\Omega \end{bmatrix} \begin{bmatrix} u_{1,1} \\ \cdot \\ u_{N,N} \end{bmatrix} = \begin{bmatrix} (g, B_{1,1})_{\Gamma_N} \\ \cdot \\ (g, B_{N,N})_{\Gamma_N} \end{bmatrix} \tag{19}$$

where for ho-FEM we selected linear or quadratic or cubic B-spline ($p = 1, 2, 3$ for different experiments) namely $v \in \{B_{i,j}(x, y) = B_{i,p}(x)B_{j,p}(y)\}_{i,j}$, for approximation and testing of (18). To train the neural network, we call the FEM solver 19 times, for $n = n_k \in (0, 1), n_k = \frac{1}{18}(k-1)$, $k = 1, ..., 19$ and we provide 19 sets of coefficients $\{u^k_{i,j}\}$. The samples are illustrated in Figures 8 by red dots. We select AdamW optimizer [12], with learning rate $\eta = 0.01$, and with ReduceLROnPlateau scheduler, that reduces the learning rate when a metric has stopped improving.

The NN trained with the coefficients of B-splines could learn a family of heat functions and accurately predict the coefficients for the base functions, see Figure 6. The difference between the exact solution and the solution obtained from the NN trained with the coefficients of quadratic B-splines using 20×20 mesh is shown in the Figure 7. The convergence of the training is visible in the Figure 7. Table 2 summarizes the results for several different numbers of orders of B-splines used. Coefficient approximation are satisfactory, see Figure 8.

3.2 Supervised training of point-wise solution of L-shape problem

Now, we employ the supervised training, but this time we train the point-wise solution values $\{u(x, y)\}_{x,y}$. The architecture of the NN for the supervised training of the point-wise solution is presented in Figure 9 and Table 3. The NN takes

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8

Mesh size	Number of coefficients	Spline degree	MSE value	Training time [s]
10 x 10	121	linear	1.98e-07	9
10 x 10	169	quadratic	5.29e-07	10
10 x 10	225	cubic	2.18e-07	11
20 x 20	441	linear	6.99e-07	14
20 x 20	529	quadratic	6.96e-07	18
20 x 20	625	cubic	9.14e-08	19
40 x 40	1681	linear	1.46e-06	15
40 x 40	1849	quadratic	1.37e-06	16
40 x 40	2025	cubic	7.58e-06	18

Table 2: Summary of obtained training results for the NN trained with the coefficients of B-splines.

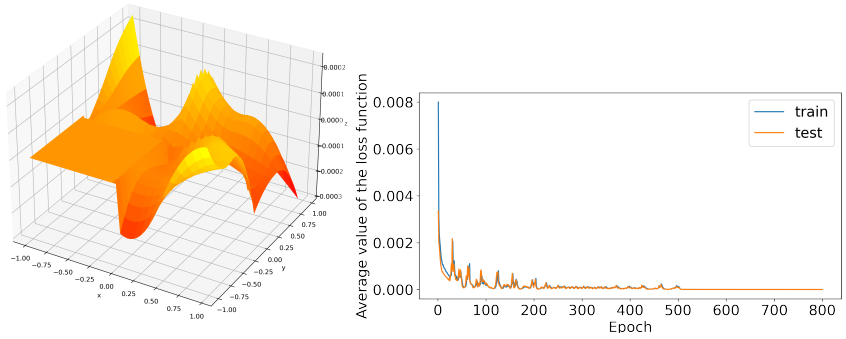

Fig. 7: **Left panel**: Difference between the NN trained with the coefficients of quadratic B-splines using 20×20 mesh and the exact solution. **Right panel**: Mean squared error loss function averaged over an epoch for the first 800 out of 2 000 epochs. Each epoch contained 19 steps (in each epoch we used 19 sets of the values of the coefficients $\{u^k_{i,j}\}_{i,j}$, $k = 1, ..., 19$ obtained from 19 calls to FEM solver for $n_k = \frac{1}{18}(k-1)$, $k = 1, ..., 19$).

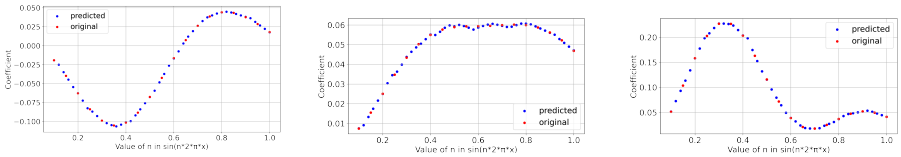

Fig. 8: **Left panel**: $B_{1,2}(x)B_{4,2}(y)$ coefficient from the NN trained the coefficients of B-splines. **Middle panel**: $B_{5,2}(x)B_{4,2}(y)$ coefficient from the NN trained the coefficients of B-splines. **Right panel**: $B_{7,2}(x)B_{7,2}(y)$ coefficient from the NN trained the coefficients of B-splines.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8

Layer	Number of neurons	Activation function
input	100	ReLU
hidden layer 1	100	ReLU
hidden layer 2	100	ReLU
output	1	none

Table 3: The architecture of point-wise solution NN.

3 arguments - the value of n, and x and y coordinates of the desired point of the solution. The output is the solution value at point (x, y) for a given n. The model was a fully-connected feed-forward neural network with 2 hidden layers, 100 neurons each:

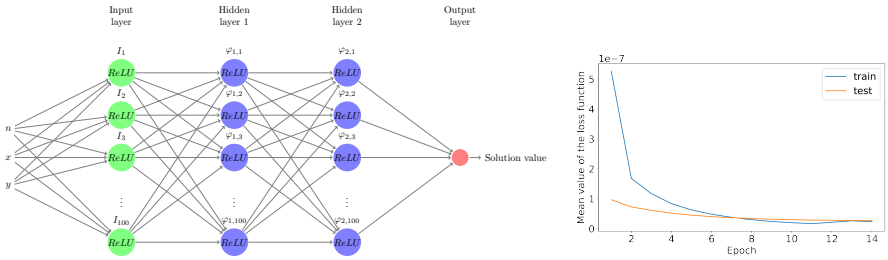

Fig. 9: **Left panel**: Architecture of the NN approximating the point-wise solution. **Right panel**: MSE loss function over 15 training epochs. Each epoch had 40 steps (in each epoch we used 40 sets of the values of the solution $\{u(x_i, y_j; n_k) = \sum_{a,b} u_{a,b} B_{a,p}(x) B_{b,p}(y)\}_{i,j}$, $k = 1, ..., 40$ obtained from 40 calls to FEM solver).

In this approach, the loss function is the MSE between the trained point-wise solution values, and the solutions obtained from FEM solver using the combination of B-splines $LOSS_2(\theta; n) = \sum_{x.y} \left(NN(\theta; x, y, n) - \sum_{i,j} u_{i,j} B_{i,p}(x) B_{j,p}(y) \right)^2$. The coefficient values $\{u_{i,j}\}_{i,j}$ are obtained by solving a system of linear equations (19) resulting from ho-FEM approximation, for 40 different values of $n = n_k \in (0, 1)$, $n_k = \frac{1}{39}(k-1)$, $k = 1, ..., 40$. We select AdamW optimizer [12], learning rate $\eta = 0.01$, and ReduceLROnPlateau scheduler. The supervised training of point-wise solution is presented in Figure 9. The trained solution and the exact solution is presented in Figure 10. The exact solution is in the blue-green palette. The trained solution is overlaid in the red-yellow colors. A difference between the exact solution and the solution obtained from the NN is shown in Figure 10. Point-wise approximation of the solution also yielded satisfactory results. The neural network could learn a family of heat functions and accurately predict the solutions, as shown in Figure 10. Table 4 summarizes the learning times for several different meshes and different B-splines and compares them

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8

to the learning times of the NN trained with the coefficients of B-splines and point-wise solution approximating NN.

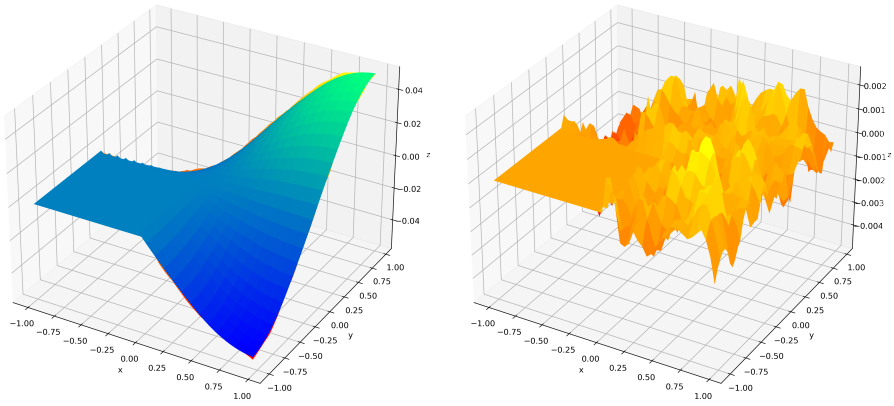

Fig. 10: **Left panel**: Visualization of the trained point-wise solution and the exact solutions. The exact solution in the blue-green palette, the trained point-wise solution is overlaid in red-yellow colors. **Right panel**: Visualization of the difference between the exact solution and the trained point-wise solution.

4 Conclusions

Our method allows us to find a family of solutions to parametric PDEs. It finds a dependence between the PDE parameters and the coefficients of the B-splines basis functions employed by higher-order FEM solver. In our method we stretch the B-spline basis to approximate the PDE solutions, and the NN tells us how to stretch the basis function when we change the PDE parameters. Our method has an advantage on traditional FEM, since it allows for solution of the parametrized PDEs at once. In our supervised training approach, we employ several calls to FEM solver to generate sets of coefficient of B-splines, and we employ them to train the general dependence between the model parameters and the solution. The future work will include the experiments with the unsupervised learning. Following the PINN idea [19] we can defined the loss function, where we train the values of the coefficients directly from the strong residual of the strong form of the PDE (14). The boundary conditions can be incorporated into the single loss by using the shift, and the resulting zero boundary conditions of the shifted formulation are enforced strongly on the NN. For this purpose, we shift the non-zero Dirichlet boundary conditions from the formulation $u(x, y) = u_0(x, y) + u_g(x, y)$ where $u_0(x, y) = 0$ on Γ_N and $u_g(x, y) = g(x, y)$ on Γ_N.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8

Mesh size	B-spline degree	training time [s] coefficients of B-splines	point-wise solution training time [s]
10 x 10	linear	9	115
10 x 10	quadratic	10	114
10 x 10	cubic	11	48
20 x 20	linear	14	43
20 x 20	quadratic	18	47
20 x 20	cubic	19	42
40 x 40	linear	15	44
40 x 40	quadratic	16	42
40 x 40	cubic	18	48

Table 4: Comparison of training times of the NN trained with the coefficients of B-splines and the point-wise solution approximating neural network. The MSE of achieved outputs had the same order of magnitude.

Thus, $\Delta u_0(x,y) = -\Delta u_g(x,y) = f(x,y)$. Now, we seek u_0 that is equal 0 on the boundary. We enforce the 0 Dirichlet boundary condition strongly. Our loss is

$$LOSS_3(\theta) = \sum_{x,y}\left(\sum_{i,j} NN(\theta;n)_{i,j}\Delta B_{i,j}(x,y) - f(x,y)\right)^2 =$$

$$\sum_{x,y}\left(\sum_{i,j} NN(\theta;n)_{i,j}\left(\frac{\partial^2 B_{i,p}(x)}{\partial x^2}B_{j,p}(y) \;\; + B_{i,p}(x)\frac{\partial^2 B_{j,p}(y)}{\partial y^2}\right) - f(x,y)\right)^2 \quad (20)$$

The applicability of this method is similar to the ho-FEM, since we employ B-spline basis functions to approximate the solution and we use NN to approximate the family of solutions obtained for different parameters.

Acknowledgement Research project partly supported by program "Excellence initiative – research university" for the AGH University of Krakow.

References

1. Babuška, I., Suri, M.: The p and h-p versions of the finite element method, basic principles and properties. SIAM Review **36**(4), 578–632 (1994), http://www.jstor.org/stable/2132721
2. Bazilevs, Y., Calo, V., Cottrell, J., Evans, J., Hughes, T., Lipton, S., Scott, M., Sederberg, T.: Isogeometric analysis using t-splines. Computer Methods in Applied Mechanics and Engineering **199**(5), 229 – 263 (2010)
3. Brevis, I., Muga, I., van der Zee, K.G.: A machine-learning minimal-residual (ml-mres) framework for goal-oriented finite element discretizations. Computers & Mathematics with Applications **95**, 186–199 (2021), recent Advances in Least-Squares and Discontinuous Petrov–Galerkin Finite Element Methods
4. Chen, R.T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.: Neural ordinary differential equations. Advances in Neural Information Processing Systems 31 (2018)

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8

5. Collier, N., Pardo, D., Dalcin, L., Paszynski, M., Calo, V.: The cost of continuity: A study of the performance of isogeometric finite elements using direct solvers. Computer Methods in Applied Mechanics and Engineering **213-216**, 353–361 (2012), https://www.sciencedirect.com/science/article/pii/S0045782511003392
6. Doległo, K., Paszyński, M., Demkowicz, L.: Deep neural networks and smooth approximation of pdes. In: Groen, D., de Mulatier, C., Paszynski, M., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M.A. (eds.) Computational Science – ICCS 2022. pp. 328–332. Springer International Publishing, Cham (2022)
7. D'Angella, D., Zander, N., Kollmannsberger, S., Frischmann, F., Rank, E., Schroder, A., Reali, A.: Multi-level hp-adaptivity and explicit error estimation. Advances in Modeling and Simulation in Enginering Sciences **3**(33) (2016)
8. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks (1989)
9. Hughes, T.J., Cottrell, J., Bazilevs, Y.: Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. Computer Methods in Applied Mechanics and Engineering (2005)
10. Kharazmi, E., Zhang, Z., Karniadakis, G.E.: Variational physics-informed neural networks for solving partial differential equations. arxiv (1912.00873v1), 1–24 (2019)
11. Kharazmi, E., Zhang, Z., Karniadakis, G.E.: hp-vpinns: Variational physics-informed neural networks with domain decomposition. Computer Methods in Applied Mechanics and Engineering (374), 113547 (2021)
12. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=Bkg6RiCqY7
13. Michoski, C., Milosavljević, M., Oliver, T., Hatch, D.R.: Solving differential equations using deep neural networks. Neurocomputing **399**, 193–212 (2020), https://www.sciencedirect.com/science/article/pii/S0925231220301909
14. Nguyen, V.P., Anitescu, C., Bordas, S.P., Rabczuk, T.: Isogeometric analysis: An overview and computer implementation aspects. Mathematics and Computers in Simulation **117**, 89 – 116 (2015)
15. Paszyński, M.: Classical and isogeometric finite element method. AGH University of Krakow (2022), https://epodreczniki.open.agh.edu.pl/handbook/1088
16. Paszyński, M., Grzeszczuk, R., Pardo, D., Demkowicz, L.: Deep learning driven self-adaptive hp finite element method. Lecture Notes in Computer Science (12742), 114–121 (2021)
17. Paszyński, M., Grzeszczuk, R., Pardo, D., Demkowicz, L.: Deep learning driven self-adaptive hp finite element method. In: Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M.A. (eds.) Computational Science – ICCS 2021. pp. 114–121. Springer International Publishing, Cham (2021)
18. Qin, S.M., Li, M., Xu, S.Q.: Rar-pinn algorithm for the data-driven vector-soliton solutions and parameter discovery of coupled nonlinear equations. arxiv (2205.10230), 1–22 (2022)
19. Raissi, M., Perdikaris, P., G.E., K.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics (378), 686–707 (2019)
20. Shin, Y., Zhang, Z., Karniadakis, G.E.: Error estimates of residual minimization using neural networks for linear pdes. arxiv (2010.08019v2), 1–22 (2020)

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_8