Journal homepage http://revistas.unitru.edu.pe/index.php/SSMM

SELECCIONES MATEMÁTICAS
Universidad Nacional de Trujillo
ISSN: 2411-1783 (Online)
2023; Vol. 10(1): 1-15.

Special issue: Peruvian Conference on Scientific Computing 2022, Cusco - Perú

Domain Decomposition with Neural Network Interface Approximations for time-harmonic Maxwell's equations with different wave numbers

Tobias Knoke, Sebastian Kinnewig, Sven Beuchler, Ayhan Demircan, Uwe Morgner and Thomas Wick

Received, Jan. 15, 2023 *Accepted, Mar. 03, 2023*

How to cite this article:
Knoke T. et al. *Domain Decomposition with Neural Network Interface Approximations for time-harmonic Maxwell's equations with different wave numbers*. Selecciones Matemáticas. 2023;10(1):1–15. http://dx.doi.org/10.17268/sel.mat.2023.01.01

Abstract

In this work, we consider the time-harmonic Maxwell's equations and their numerical solution with a domain decomposition method. As an innovative feature, we propose a feedforward neural network-enhanced approximation of the interface conditions between the subdomains. The advantage is that the interface condition can be updated without recomputing the Maxwell system at each step. The main part consists of a detailed description of the construction of the neural network for domain decomposition and the training process. To substantiate this proof of concept, we investigate a few subdomains in some numerical experiments with low frequencies. Therein the new approach is compared to a classical domain decomposition method. Moreover, we highlight current challenges of training and testing with different wave numbers and we provide information on the behaviour of the neural-network, such as convergence of the loss function, and different activation functions.

Keywords . Time-Harmonic Maxwell's Equations, Machine Learning, Feedforward Neural Network, Domain Decomposition Method.

1. Introduction. The Maxwell's equations for describing electro-magnetic phenomena are of great interest in current research fields, such as optics. One present example of employing Maxwell's equations can be found in the Cluster of Excellence PhoenixD (Photonics Optics Engineering Innovation Across Disciplines)[1] at the Leibniz University Hannover, in which modern methods for optics simulations are being developed. Therein, one focus is on the efficient and accurate calculation of light distribution in an optical material to design optical devices on the micro- and nanoscale [1, 2]. In comparison to other partial differential equations, such as in solid mechanics or fluid flow, the Maxwell's equations have some peculiarities such as the curl operator, which has in two-dimensional problems, a one-dimensional image, but in three-dimensional problems, it has a three-dimensional image. Moreover, the requirements for the discretization and definiteness of the final linear system are specific. In more detail, in numerical mathematics, Maxwell's equations are of interest because of their specific mathematical structures [3, 4, 5, 6], requirements for finite elements [3, 5, 7, 8, 9, 10, 11], their numerical solution [12, 13, 14, 15] as well as postprocessing such as

*Institute of Applied Mathematics at Leibniz University Hannover, Germany (tobias.knoke@stud.uni-hannover.de).
†Institute of Applied Mathematics at Leibniz University Hannover, Germany (kinnewig@ifam.uni-hannover.de).
‡Institute of Applied Mathematics at Leibniz University Hannover, Germany (beuchler@ifam.uni-hannover.de).
§Institute of Quantum Optics at Leibniz University Hannover, Germany (demircan@iqo.uni-hannover.de).
¶Institute of Quantum Optics at Leibniz University Hannover, Germany (morgner@iqo.uni-hannover.de).
‖Institute of Applied Mathematics at Leibniz University Hannover, Germany (thomas.wick@ifam.uni-hannover.de).
[1] https://www.phoenixd.uni-hannover.de/en/

a posteriori error control and adaptivity [16, 17]. As their numerical solution is challenging due to their ill-posed nature, e.g., [18], one must apply suitable techniques. The most prominent approach in the literature is based on domain decomposition (DD) techniques [19, 15]. The geometric multigrid solver developed by Hiptmair [12] can only be applied to the problem in the time domain (i.e., the well-posed problem).

In this work, we concentrate on the numerical solution using a domain decomposition method. Specifically, our starting point is the method developed in [18], based on ideas from [20], and which was realized in the modern open-source finite element library deal.II [21, 22]. The domain decomposition method's crucial point is the interface operator derivation [20]. Our main objective in the current work is to design a proof of concept to approximate the interface operator with the help of a feedforward neural network (NN) [23, 24, 25]. We carefully derive the governing algorithms and focus on a two-domain problem to study our new approach's mechanism and performance. Implementation-wise, the previously mentioned deal.II library (in C++) is coupled to the PyTorch (in python) [26] library, which is one of the standard packages for neural network computations. Our main aim is to showcase that our approach is feasible and can be a point of departure for further future extensions. We notice that the current work is an extension of the conference proceedings paper [27] with more mathematical and algorithmic details, and different numerical tests, specifically the studies on different wave numbers and comparison of two NN activation functions.

The outline of this work is as follows: In Section 2, we introduce the time-harmonic Maxwell's equations and our notation. Next, in Section 3, domain decomposition and neural network approximations are introduced. Afterwards, we address in detail the training process in Section 4. In Section 5, some numerical tests demonstrate our proof of concept. Our work is summarized in Section 6.

2. Equations.

For the sake of simplicity, we only consider the two-dimensional time-harmonic Maxwell's equations. In the following, we will introduce these equations in detail.

2.1. Fundamental operators.

To comprehensively describe the problem, we introduce the basic operators needed to describe two-dimensional electro-magnetic problems. Therefore, let us assume a scalar function $\phi : \mathbb{R} \to \mathbb{R}$ and $v \in \mathbb{R}^2$ to be a two-dimensional vector. Then the gradient of ϕ is given by $\nabla\phi = \left(\frac{\partial\phi}{\partial x_1}, \frac{\partial\phi}{\partial x_2}\right)$, and the divergence of v is given by $\operatorname{div}(v) := \nabla \cdot v := \sum_{i=1}^{2} \frac{\partial v_i}{\partial x_i}$. Next, $a \cdot b = (a_1, a_2)^T \cdot (b_1, b_2)^T = a_1 b_1 + a_2 b_2$ denotes the scalar product. We can furthermore write down the description of the two-dimensional curl operator

$$\operatorname{curl}(v) = \frac{\partial v_2}{\partial x_1} - \frac{\partial v_1}{\partial x_2}, \tag{2.1}$$

and the curl operator applied to a scalar function

$$\underline{\operatorname{curl}}(\phi) = \begin{pmatrix} \frac{\partial\phi}{\partial x_2} \\ -\frac{\partial\phi}{\partial x_1} \end{pmatrix}. \tag{2.2}$$

2.2. Time-harmonic Maxwell's equations.

Let $\Omega \subset \mathbb{R}^2$ be a bounded domain with sufficiently smooth boundary Γ. The latter is partitioned into $\Gamma = \Gamma^{\infty} \cup \Gamma^{\text{inc}}$. The main governing function space is defined as

$$H(\operatorname{curl}, \Omega) := \{v \in \mathcal{L}^2(\Omega) \mid \operatorname{curl}(v) \in \mathcal{L}^2(\Omega)\},$$

where $\mathcal{L}^2(\Omega)$ is the well-known space of square-integrable functions in the Lebesgue sense. In order to define boundary conditions, we introduce the traces

$$\gamma^t : H(\operatorname{curl}, \Omega) \to H_{\times}^{-1/2}(\operatorname{div}, \Gamma),$$
$$\gamma^T : H(\operatorname{curl}, \Omega) \to H_{\times}^{-1/2}(\operatorname{curl}, \Gamma),$$

which are defined by

$$\gamma^t(\phi) = \begin{pmatrix} \phi\, n_2 \\ -\phi\, n_1 \end{pmatrix} \quad \text{and} \quad \gamma^T(v) = v - (n \cdot v) \cdot n,$$

where $n \in \mathbb{R}^2$ is the normal vector of Ω, $H_{\times}^{-1/2}(\operatorname{div}, \Gamma) := \{v \in H^{-1/2}(\Gamma) \mid v \cdot n = 0,\ \operatorname{div}_{\Gamma} v \in H^{-1/2}(\Gamma)\}$ is the space of well-defined surface divergence fields and $H(\operatorname{curl}, \Gamma) := \{v \in H^{-1/2}(\Gamma) \mid v \cdot n = 0,\ \operatorname{curl}_{\Gamma}(v) \in H^{-1/2}(\Gamma)\}$ is the space of well-defined surface curls, see [3, Chapter 3.4]. In the

following, we first state the strong form of the system. The time-harmonic Maxwell's equations are then defined as follows: Find the electric field $E : \Omega \to \mathbb{C}^2$ such that

$$\begin{cases} \underline{\operatorname{curl}}\left(\mu^{-1}\operatorname{curl}(E)\right) - \varepsilon\omega^2 E & = 0 \quad \text{in } \Omega \\ \mu^{-1}\gamma^t\left(\operatorname{curl}(E)\right) - i\kappa\omega\gamma^T(E) & = 0 \quad \text{on } \Gamma^{\infty} \\ \gamma^T(E) & = E^{\text{inc}} \quad \text{on } \Gamma^{\text{inc}}, \end{cases} \tag{2.3}$$

where $E^{\text{inc}} : \mathbb{R}^2 \to \mathbb{C}^2$ is some given incident electric field, $\mu \in \mathbb{R}^+$ is the relative magnetic permeability, $\kappa = \sqrt{\varepsilon}$, $\varepsilon \in \mathbb{C}$ relative permittivity, $\omega = \frac{2\pi}{\lambda}$ is the wave number and $\lambda \in \mathbb{R}^+$ is the wave length and i denotes the imaginary number. System (2.3), as well as its weak form, is called time-harmonic, because the time dependence can be expressed by $e^{i\omega\tau}$, where $\tau \geq 0$ denotes the time.

2.3. Weak formulation. In this subsection, we derive the weak form. This is the starting point for a finite element method (FEM) discretization. For the derivation, we first begin by rewriting the curl product with the help of integration by parts:

$$\int_{\Omega} \underline{\operatorname{curl}}(\phi) \cdot u \, \mathrm{d}x = \int_{\Omega} \phi \operatorname{curl}(u) \, \mathrm{d}x + \int_{\partial\Omega} \gamma^t(\phi) \cdot u \, \mathrm{d}s, \tag{2.4}$$

see for instance [28, 3]. We want to derive the weak formulation from the strong formulation (2.3) in the following:

$$\begin{aligned} &\int_{\Omega} \underline{\operatorname{curl}}\left(\mu^{-1}\operatorname{curl}(E)\right) \cdot \varphi \, \mathrm{d}x - \varepsilon\omega^2 \int_{\Omega} E \cdot \varphi \, \mathrm{d}x = 0, \\ \overset{(2.4)}{\Rightarrow} &\int_{\Omega} \mu^{-1}\operatorname{curl}(E)\operatorname{curl}(\varphi) \, \mathrm{d}x - \varepsilon\omega^2 \int_{\Omega} E \cdot \varphi \, \mathrm{d}x + \int_{\partial\Omega} \mu^{-1}\gamma^t\left(\operatorname{curl}(E)\right) \cdot \varphi \, \mathrm{d}s = 0. \end{aligned} \tag{2.5}$$

By applying the definition of the boundaries Γ^{∞} and Γ^{inc} from equation (2.3) to equation (2.5), we obtain the weak formulation of the time-harmonic Maxwell's equations. Find $E \in H(curl, \Omega)$ such that for all $\varphi \in H(curl, \Omega)$

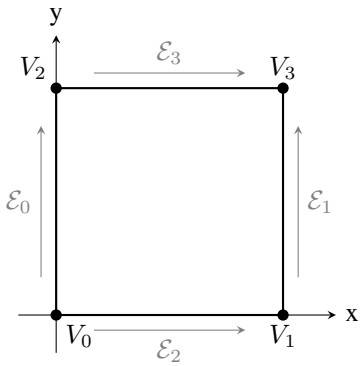
$$\begin{aligned} \int_{\Omega} \left(\mu^{-1}\operatorname{curl}(E)\operatorname{curl}(\varphi) - \varepsilon\omega^2 E \cdot \varphi\right) \mathrm{d}x + i\kappa\omega \int_{\Gamma^{\infty}} \gamma^T(E) \cdot \gamma^T(\varphi) \, \mathrm{d}s \\ = \int_{\Gamma^{\text{inc}}} \gamma^T\left(E^{\text{inc}}\right) \cdot \gamma^T(\varphi) \, \mathrm{d}s. \end{aligned} \tag{2.6}$$

2.4. Two-dimensional Nédélec elements. For the implementation with the help of a Galerkin finite element method (FEM), we need the discrete weak form. Based on the De-Rham cohomology, we must choose our basis functions out of the Nédélec space V_h. Therefore, we want to introduce the definition of the space V_h in the following, based on the formalism introduced by Zaglmayr [29, Chapter 5.2].

As a suitable polynomial basis, we introduce the integrated Legendre polynomials. Let $x \in [-1, 1]$. The following recursive formula defines the integrated Legendre polynomials:

$$\begin{aligned} L_1(x) &= x, \\ L_2(x) &= \tfrac{1}{2}\left(x^2 - 1\right), \\ (n+1)L_{n+1}(x) &= (2n-1)xL_n(x) - (n-2)L_{n-1}(x), \quad \text{for } n \geq 2. \end{aligned} \tag{2.7}$$

Let us choose the quadrilateral reference element as $Q = [0, 1] \times [0, 1]$.



$$\lambda_0 = (1-x)(1-y), \quad \sigma_0 = (1-x)+(1-y),$$
$$\lambda_1 = x(1-y), \quad \sigma_1 = x+(1-y),$$
$$\lambda_2 = (1-x)y, \quad \sigma_2 = (1-x)+y,$$
$$\lambda_3 = xy, \quad \sigma_3 = x+y$$

Figure 2.1: Left: Vertex and edge ordering on the reference cell, right: parametrisation of the reference cell.

We continue by defining the set of all edges $\mathfrak{E} = \{\mathcal{E}_m\}_{0\le m<4}$ with local edge-ordering $\mathcal{E}_m = \{V_i, V_j\}$, where $(i,j) \in \{(0,2),(1,3),(0,1),(2,3)\}$, see Figure 2.1. We denote the cell itself with local vertex-ordering $C = \{V_0, V_1, V_2, V_3\}$. The polynomial order is given by $p = \left(\{p_\mathcal{E}\}_{\mathcal{E}\in\mathfrak{E}}, p_C\right)$.

$H(curl)$ **conforming basis function**	
Vertex-based shape functions	
There are no DoFs on the vertices.	
Edge-based shape functions for $0 \le i < p_\mathcal{E}$, $\mathcal{E} \in \mathfrak{E}$, where λ_α and $\sigma_\alpha,\ \alpha \in \{0,1,2,3\}$ are defined in Figure 2.1	
Lowest order	$\varphi^{\mathcal{N}_0}_{\mathcal{E}_m} = \frac{1}{2}\nabla\left(\sigma_{e_2} - \sigma_{e_1}\right)\left(\lambda_{e_1} + \lambda_{e_2}\right)$
Higher-order	$\varphi^{\mathcal{E}_m}_i = \nabla\left(L_{i+2}\left(\sigma_{e_2} - \sigma_{e_1}\right)\left(\lambda_{e_1} + \lambda_{e_2}\right)\right)$
Cell-based functions $0 \le i,j < p_C$	
Type 1:	$\varphi^{C,1}_{(i,j)} = \nabla(L_{i+2}(\xi_F)L_{j+2}(\eta_F))$
Type 2:	$\varphi^{C,2}_{(i,j)} = \widetilde{\nabla}(L_{i+2}(\xi_F)L_{j+2}(\eta_F))$ where $\widetilde{\nabla}(a\ b) := (a'\ b - a\ b')$
Type 3:	$\varphi^{C,3}_{(0,j)} = L_{i+2}(2y-1)e_x$ $\varphi^{C,3}_{(i,0)} = L_{i+2}(2x-1)e_y$

Figure 2.2: The definition of the $H(\mathrm{curl})$ basis-functions on the reference element with barycentric coordinates λ_α and $\sigma_\alpha,\ \alpha \in \{0,1,2,3\}$.

With the help of these basis functions, we define the two-dimensional Nédélec space

$$V_h := V_h^{\mathcal{N}_0}(\mathcal{T}_h) \bigoplus_{\mathcal{E}\in\mathfrak{E}} V_h^{\mathcal{E}}(\mathcal{T}_h) \bigoplus_{C\in\mathcal{C}} V_h^{C}(\mathcal{T}_h), \tag{2.8}$$

where $V_h^{\mathcal{N}_0}$ is the space of the *lowest-order Nédélec* function, $V_h^{\mathcal{E}}$ is the space of the *edge-bubbles* and V_h^{C} is the space of the *cell-bubbles*. All basis functions on one element with baryzentric coordinates are displayed in Figure 2.2. Visualizations of some basis functions are displayed in Figure 2.3. The description of $V_h(\Omega)$ is still not complete, so far we only described $V_h(Q)$, with Q as previously defined. It remains to introduce the Piola transformation, which is used to transform the reference element to any given physical element, see Monk [3] (Lemma 3.57, Corollary 3.58).

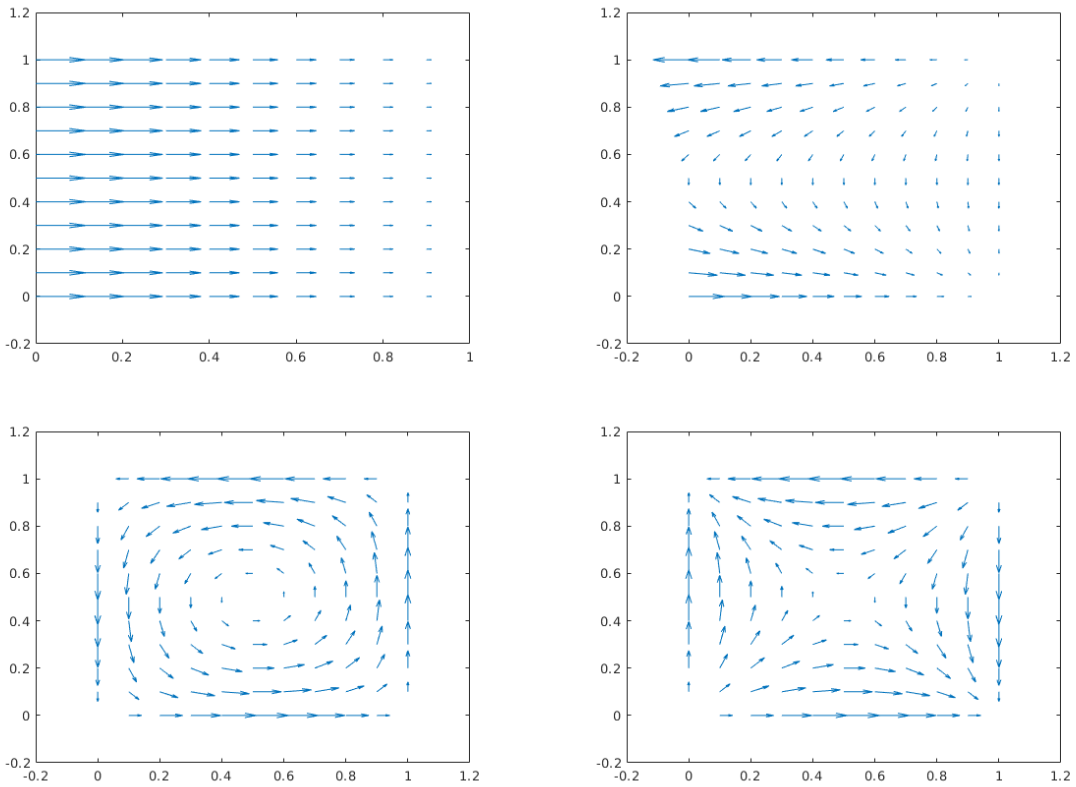


Figure 2.3: Plots of basis functions on $(0,1)^2$: low order edge function (left above), high order edge based basis function for $p=2$ (right above) to the edge $\mathcal{E}_0$, high order cell based basis functions for $p=2$ of type 1 and 2 (below).

2.5. Discrete weak formulation. We have gathered everything to write down the discrete weak formulation of the time-harmonic Maxwell's equations. We obtain the discrete weak formulation by applying the Galerkin method to the equation (2.6). Find $E_h \in V_h(\Omega)$ such that

$$\int_\Omega \left(\mu^{-1}\operatorname{curl}(E_h)\operatorname{curl}(\varphi_h) - \varepsilon\omega^2 E_h\cdot\varphi_h\right)\,\mathrm{d}x + i\kappa\omega\int_{\Gamma^\infty}\gamma^T(E_h)\cdot\gamma^T(\varphi_h)\,\mathrm{d}s = \int_{\Gamma^{\text{inc}}}\gamma^T\left(E^{\text{inc}}\right)\cdot\gamma^T(\varphi_h)\,\mathrm{d}s \quad \forall\varphi_h\in V_h(\Omega). \tag{2.9}$$

3. Numerical approach. In this section, we first describe domain decomposition and afterwards the neural network approximation. In the latter, we also outline how to replace the interface operator by the neural network.

3.1. Domain decomposition. Since the solution of Maxwell's equation system (2.3) is challenging, as already outlined in the introduction, we apply a non-overlapping domain composition method (DDM)[19] in which the domain is divided into subdomains as follows

$$\overline{\Omega} = \bigcup_{i=0}^{n_{\text{dom}}}\overline{\Omega}_i \quad \text{with}$$
$$\Omega_i\cap\Omega_j = \varnothing \quad \forall i\neq j,$$

where $n_{\text{dom}}+1$ is the number of subdomains. In such a way, every subdomain Ω_i becomes small enough so that we can handle it with a direct solver. The global solution of the electric field E is computed via an iterative method, where we solve the time-harmonic Maxwell's equations on each subdomain with suitable interface conditions between the different subdomains. Thus, we obtain a solution E_i^k for every subdomain Ω_i, where k denotes the k-th iteration step. The initial interface condition is given by

$$g_{ji}^{k=0} := -\mu^{-1}\gamma_i^t\left(\operatorname{curl}\left(E_i^{k=0}\right)\right) - i\kappa S\left(\gamma_i^T\left(E_i^{k=0}\right)\right) = 0, \tag{3.1}$$

where S describes the interface operator, i is the index of the current domain, and j is the index of the neighbouring domain [20]. Afterwards, the electric-field E_i^{k+1} is computed at each step by solving the following system

$$\begin{cases} \underline{\text{curl}}\left(\mu^{-1}\,\text{curl}\left(E_i^{k+1}\right)\right) - \omega^2\varepsilon E_i^{k+1} & = 0 & \text{in } \Omega_i, \\ \mu^{-1}\gamma_i^t\left(\text{curl}\left(E_i^{k+1}\right)\right) - i\omega\kappa\gamma_i^T\left(E_i^{k+1}\right) & = 0 & \text{on } \Gamma_i^{\infty}, \\ \gamma_i^T\left(E_i^{k+1}\right) & = \gamma_i^T\left(E_i^{\text{inc}}\right) & \text{on } \Gamma_i^{\text{inc}}, \\ \mu^{-1}S\left(\gamma_i^t\left(\text{curl}\left(E_i^{k+1}\right)\right)\right) - i\omega\kappa\gamma_i^T\left(E_i^{k+1}\right) & = g_{ji}^k & \text{on } \Sigma_{ij}, \end{cases} \tag{3.2}$$

where $\Sigma_{ij} = \Sigma_{ji} := \partial\Omega_i \cap \partial\Omega_j$ denotes the interface of two neighbouring elements and the interface condition is updated by

$$g_{ji}^{k+1} = -\mu^{-1}\gamma_i^t\left(\text{curl}\left(E_i^{k+1}\right)\right) - i\kappa S\left(\gamma_i^T\left(E_i^{k+1}\right)\right) = -g_{ij}^k - 2i\kappa S\left(\gamma_i^T\left(E_i^{k+1}\right)\right). \tag{3.3}$$

In case of success we obtain $\lim_{k\to\infty} E_i^k = E|_{\Omega_i}$, but this convergence depends strongly on the chosen interface operator S (see [15, 20]). The implementation of this approach into deal.II was done in [18].

3.2. Our new approach: Neural network approximation of S. Since the computation of a good approximation of S is challenging, we examine a new approach in which we attempt to approximate this operator with the help of a neural network (NN). For a first proof of concept, we choose a prototype example and explore whether an NN can approximate the interface values. As it is not feasible to compute the exact interface operator S, we aim to compute g_{ij}^{k+l}, $l > 0$ with an NN, using g_{ij}^k and E_i^{k+1} as input. Another benefit of this approach is that we can quickly generate a training data set from a classical domain decomposition method, as described in Section 4.4. We choose $S = \mathbb{1}$ for simplicity inside our classical domain decomposition method. Hence, the advantage of this approach is that one can update the interface condition without recomputing the system (3.2) at each step, raising the hope of reducing the computational cost.

4. Neural network training. The first step in neural network approximations is the training process, which is described in this section. Besides the mathematical realization, we also need to choose the software libraries. We utilize deal.II [22] to discretize the time-harmonic Maxwell's equations with the finite element method. The neural network is trained with PyTorch [26]. The exchange of information between the results of the deal.II code and the PyTorch code take place via the hard disk.

4.1. Basic definitions. First of all, we give a short definition of the neural network type employed in this work, and we introduce the basic parameters. Further information can be found in [23, 30, 31, 32, 33]. The following notation and descriptions of this subsection are mainly based on [34].

Definition 4.1 (Artificial neuron). *An (artificial) neuron (also known as unit [35], [23][Section 5.1]) u is a tuple of the form (x, w, σ). The components have the following meanings:*

- *$x = (x_0, \ldots, x_n) \in \mathbb{R}^{n+1}$ is the input vector. It contains the information, that the neuron receives.*
- *$w = (w_0, \ldots, w_n) \in \mathbb{R}^{n+1}$ is the weight vector, which determines the influence of the individual input information on the output of the neuron. Later, w denotes the weight vector of all neurons.*
- *$\sigma : \mathbb{R} \to \mathbb{R}$, with $\sigma = \sum_{i=0}^{n} x_i w_i \mapsto a$ is the activation function. It determines the so-called activation level a from the input and the weights, which represent the output of the neuron.*

Definition 4.2 (Neural network). *An (artificial) neural (feedforward-) network is a set of neurons U with a disjoint decomposition $U = U_0 \dot\cup \ldots \dot\cup U_l$. The partition sets $U_\mathsf{k}, \mathsf{k} = 0, \ldots, l$ are called layers. Here, U_0 is the input layer. It contains the neurons that receive information from outside. Moreover, U_l is the output layer with the neurons that return the output. Finally, $U_1, \ldots, U_{l-1}$ are the so-called hidden layers.* Starting from any neuron $u \in U_\mathsf{k}$, there is a connection to each neuron $\hat u \in U_{\mathsf{k}+1}$ for $\mathsf{k} = 0, \ldots, l-1$. Such a connection illustrates that the output a of the neuron u is passed on to the neuron $\hat u$. This property is the reason for the name feedforward network.

Each $U_0, \ldots, U_{l-1}$ contains a so-called bias neuron of the form $(0, 0, 1)$. It has no input, weights and a constant output value 1 and only transfers a constant bias in the form of the weight to each neuron of the subsequent layer.

Remark 4.1. *In the following examples all neurons of the layer U_k will have the same activation function, given by $\sigma^{(\mathsf{k})}$ for $\mathsf{k} = 0, \ldots, l$. Here, $D_\mathsf{k} := |U_\mathsf{k}| - 1$ for $k = 0, \ldots, l-1$ denotes the number of neurons of the k-th layer (without bias neuron) and $D_l := |U_l|$ is the number of neurons of the output layer.*

4.2. Decomposing the domain. Before constructing the NN, we choose the domain, the decomposition and the grid on which the system (3.2) is solved to obtain the training values because they will influence the network size. The domain in our chosen example, given by

$$\Omega = (0,1) \times (0,1),$$

is divided into two subdomains

$$\Omega_0 = (0,1) \times (0,0.5) \quad \text{and} \quad \Omega_1 = (0,1) \times (0.5,1) \quad \text{(see Figure 4.1)},$$

and the grid on which the FEM is applied is a mesh of 32×32 elements with quadratic Nédélec elements. Hence 32 elements with each 4 degrees of freedom (dofs) are located on the interface in both subdomains.

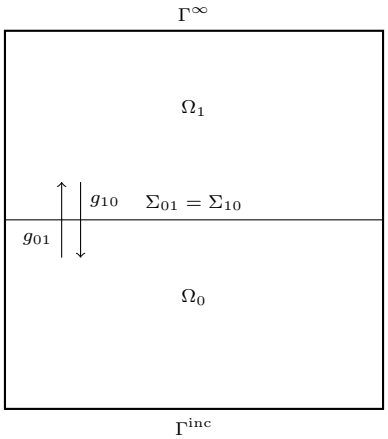


Figure 4.1: Visualization of the domain Ω with the chosen decomposition.

We evaluate the interface condition and the solution on each dof and use the values as the NN's input and target. Therefore the input contains $4 \cdot \dim(g_{ij}) + 4 \cdot \dim(E_i) = 16$ values, and the output consists of $4 \cdot \dim(g_{ji}) = 8$ values, and we obtain 32 input-target pairs with one computation.

4.3. Neural network construction. Regarding the considerations above, we need an input layer with 16 neurons (without bias) and an output layer with 8 neurons. Furthermore, we use one hidden layer with 500 neurons (without bias). Hence, for the governing network, we have

$$U = U_0 \dot{\cup} U_1 \dot{\cup} U_2$$

with $D_0 = 16$, $D_1 = 500$ and $D_2 = 8$. Our tests, presented in Section 5, revealed that this is a sufficient size for our purpose. The activation function per layer is chosen as follows:

$$\begin{aligned}
\sigma^{(0)} &= id \quad \text{(input layer)}, \\
\sigma^{(1)} &= \frac{1}{1+e^{-x}} \quad \text{(hidden layer)}, \\
\sigma^{(2)} &= id \quad \text{(output layer)},
\end{aligned}$$

where $\sigma^{(1)}$ is known as the sigmoid function, which turned out to be the most effective since the error could be reduced more and more quickly than with other functions we tested e.g.

$$\begin{aligned}
\sigma^{(1)} &= \tanh(x), \\
\sigma^{(1)} &= \log\left(\frac{1}{1+e^{-x}}\right) \quad \text{(LogSigmoid)}, \\
\sigma^{(1)} &= \max(0,x) + \min(0, e^x - 1) \quad \text{(CELU)}, \\
\sigma^{(1)} &= a\left(\max(0,x) + \min\left(0, b\left(e^x - 1\right)\right)\right) \quad \text{(SELU)}, \\
&\quad \text{with } a \approx 1.0507, \text{ and } b \approx 1.6733.
\end{aligned}$$

An exception represents the ReLU function, which we will discuss later in Section 5.4. Moreover, we apply separate networks U^{01} and U^{10} of the same shape for both interface conditions g_{01} and g_{10} since it

turned out that they are approximated differently, fast and accurately. The resulting programming code is displayed in Figure 4.2.

```
import torch.nn as nn
import torch.nn.functional as F
class Maxwell(nn.Module):
def __init__(self):
super(Maxwell, self).__init__()
        self.lin1 = nn.Linear(16, 500)
        self.lin2 = nn.Linear(500, 8)

    def forward(self, x):
        x = torch.sigmoid(self.lin1(x))
        #x = F.relu(self.lin1(x))
        x = self.lin2(x)
        return x

net = Maxwell()
print(net)
```

Figure 4.2: PyTorch code of the implementation of the network construction.

4.4. Training. To obtain enough training data, we vary the boundary condition E^{inc} and create training and test values to control the network during the training and avoid overfitting. The training and test sets are generated by the boundary values listed in Table 4.1.

E^{inc} for the training set		E^{inc} for the test set
$\begin{pmatrix} e^{\frac{-(x-0.7)^2}{0.008}} \\ 0 \end{pmatrix}$	$\begin{pmatrix} \cos(\pi^2 y) + \sin(\pi^2 x)i \\ \sin(\pi^2 y) + 0.5\cos(\pi^2 x)i \end{pmatrix}$	$\begin{pmatrix} e^{\frac{-(x-0.5)^2}{0.003}} \\ 0 \end{pmatrix}$
$\begin{pmatrix} e^{\frac{-(x-0.2)^2}{0.002}} \\ 1 \end{pmatrix}$	$\begin{pmatrix} \sin(\pi^2 x) + \sin(\pi^2 x)i \\ \sin(\pi^2 y) + 0.5\cos(\pi^2 x)i \end{pmatrix}$	$\begin{pmatrix} \cos(\pi^2 y) + \sin(\pi^2 x)i \\ \cos(\pi^2 y) + 0.5\cos(\pi^2 x)i \end{pmatrix}$
$\begin{pmatrix} e^{\frac{-(x-0.7)^2}{0.003}} \\ 1 \end{pmatrix}$	$\begin{pmatrix} \sin(\pi^2 x) + \sin(\pi^2 x)i \\ \sin(\pi^2 x) + 0.5\cos(\pi^2 x)i \end{pmatrix}$	
$\begin{pmatrix} e^{\frac{-(x-0.8)^2}{0.003}} \\ \sin(\pi^2 x) \end{pmatrix}$	$\begin{pmatrix} \cos(\pi^2 y) + \sin(\pi^2 x)i \\ \cos(\pi^2 x) + 0.5\cos(\pi^2 x)i \end{pmatrix}$	
$\begin{pmatrix} e^{\frac{-(x-0.5)^2}{0.003}} \\ \cos(\pi^2 x) \end{pmatrix}$	$\begin{pmatrix} \cos(\pi^2 x) + \sin(\pi^2 x)i \\ \cos(\pi^2 y) + 0.5\cos(\pi^2 x)i \end{pmatrix}$	

Table 4.1: Boundary values for generating the training set and the test set

Since we choose 10 different boundary values for the training set and 2 for the test set and each of them generates a set of 32 training/test values (one per element on the interface), we obtain all in all a set of $32 \cdot 10 = 320$ training values and a set of $32 \cdot 2 = 64$ test values for both networks. To keep the computation simple in a first set of tests, we choose a small wave number $\omega = \frac{2\pi}{3}$, and compute the sets with the iterative DDM in 4 steps. Afterwards we use the results $\left(g_{ij}^1, E_i^2\right)$ as the input and g_{ji}^3 as the targets to train our NNs with the application of the mean squared error as the loss function, given by

$$\text{Loss}(\mathtt{w}) = \frac{1}{2}\sum_{i=1}^{N} \|t^{(i)} - y(\mathtt{x}^{(i)}, \mathtt{w})\|^2,$$

where N denotes the number of input-target pairs (in our case $N = 320$ for the training set and $N = 64$ for the test set), $t^{(i)}$ is the target vector, y is the function generated by the network and hence $y(\mathtt{x}^{(i)}, \mathtt{w})$ denotes

the output of the NN. We refer the reader to Section 5.1 for the specific realization. As the optimizer, we use the Adam algorithm [36], which is a line search method based on the following iteration rule

$$x^{\rho+1} = x^{\rho} + \alpha^{\rho} p^{\rho},$$

where p^{ρ} is called the search direction and α^{ρ} is the step size (or learning rate in case of NN) for the iteration step ρ. The search direction of the Adam algorithm depends on four parameters β_1, β_2, m_1 and m_2, where β_1 and β_2 are fixed values in the interval $[0, 1)$, and m_1 and m_2 are updated in each step via

$$m_1^0 = m_2^0 = 0, \quad m_1^{\rho+1} = \beta_1 m_1^{\rho} + (1-\beta_1) \cdot \nabla \mathrm{Loss}(\mathtt{w})^{\rho}$$
$$\text{and} \quad m_2^{\rho+1} = \beta_2 m_2^{\rho} + (1-\beta_2) \cdot \|\nabla \mathrm{Loss}(\mathtt{w})^{\rho}\|^2.$$

The search direction is then given by

$$p^{\rho-1} = -\widehat{m_1^{\rho}} \Big/ \sqrt{\widehat{m_2^{\rho}} + \varepsilon}$$

with $\widehat{m_1^{\rho}} = m_1^{\rho}/(1-(\beta_1)^{\rho})$, $\widehat{m_2^{\rho}} = m_2^{\rho}/(1-(\beta_2)^{\rho})$ and $0 < \varepsilon \ll 1$.

The implementation of this training process in PyTorch is displayed in Figure 4.3.

```
import torch.optim as optim
import time
start_time = time.time()
tol = 3e-3
max_iter = 20000
iterations = 0
loss_test = tol + 1
optimizer = optim.Adam(net.parameters(), lr=1e-5)
criterion = nn.MSELoss()
while(iterations < max_iter and loss_test > tol):

    out = net(inp_training)
    optimizer.zero_grad()
    loss = criterion(out, target_training)
    loss_test = criterion(net(inp_test), target_test)
    print("Loss: %.5f" % loss, "Test-Loss: %.5f" % loss_test))
    loss.backward()
    optimizer.step()
    iterations += 1
print("Final Loss: %.5f" % loss_test)
print("Number of iterations: %.0f" % iterations)
time_taken = time.time() - start_time
print("Run-Time: %.4f s" % time_taken)
```

Figure 4.3: PyTorch code of the implementation of the network training.

The network U^{01} is trained with the learning rate 10^{-5}. The initial training error of 3.12 and the test error of 5.87 are reduced to $1.7 \cdot 10^{-4}$ and $3 \cdot 10^{-3}$ after 29 843 training steps. At U^{10}, the initial training error of 0.72 and the test error of 1.28 are reduced to $3 \cdot 10^{-4}$ and $4 \cdot 10^{-3}$ after 20 326 steps with learning rate of 10^{-5} and after further training with a learning rate of 10^{-6} in 3706 steps, we finally achieve the training error $2.9 \cdot 10^{-4}$ and the test error $3 \cdot 10^{-3}$.

5. Numerical tests. In this section, we investigate several numerical experiments to demonstrate the current capacities of our approach. In addition, we highlight and analyze shortcomings and challenges.

5.1. Comparison of new approach and classical DDM. In this first numerical example, we apply the implemented and trained NNs for the following boundary condition

$$E^{\mathrm{inc}}(x,y) = \begin{pmatrix} \cos\left(\pi^2 (y-0.5)\right) + \sin\left(\pi^2 x\right) i \\ \cos\left(\pi^2 y\right) + 0.5 \sin\left(\pi^2 x\right) i \end{pmatrix},$$

and compute the first interface conditions g_{10}^1 and g_{01}^1 and the solutions E_1^2 and E_0^2 by solving (3.2) and (3.3) with the use of the parameters given in Table 5.1. Afterwards, these values are passed on to the networks U^{01} and U^{10}. The output they return is then handled as our new interface condition, which we use to solve system (3.2) one more time. With that, we obtain the final solution. Moreover, we compute the same example with the DDM in 4 steps. The results that are displayed in Figure 5.1 show excellent agreement.

Parameter	Definition	Value
μ	relative magnetic permeability	1.00
ε	relative electric permittivity	1.49^2
κ		$\sqrt{\varepsilon} = 1.49$
λ	wave length	3.00
ω	wave number	$\frac{2\pi}{\lambda} = \frac{2\pi}{3.00}$
	grid size	$\frac{1}{32}$

Table 5.1: Parameters for the DDM

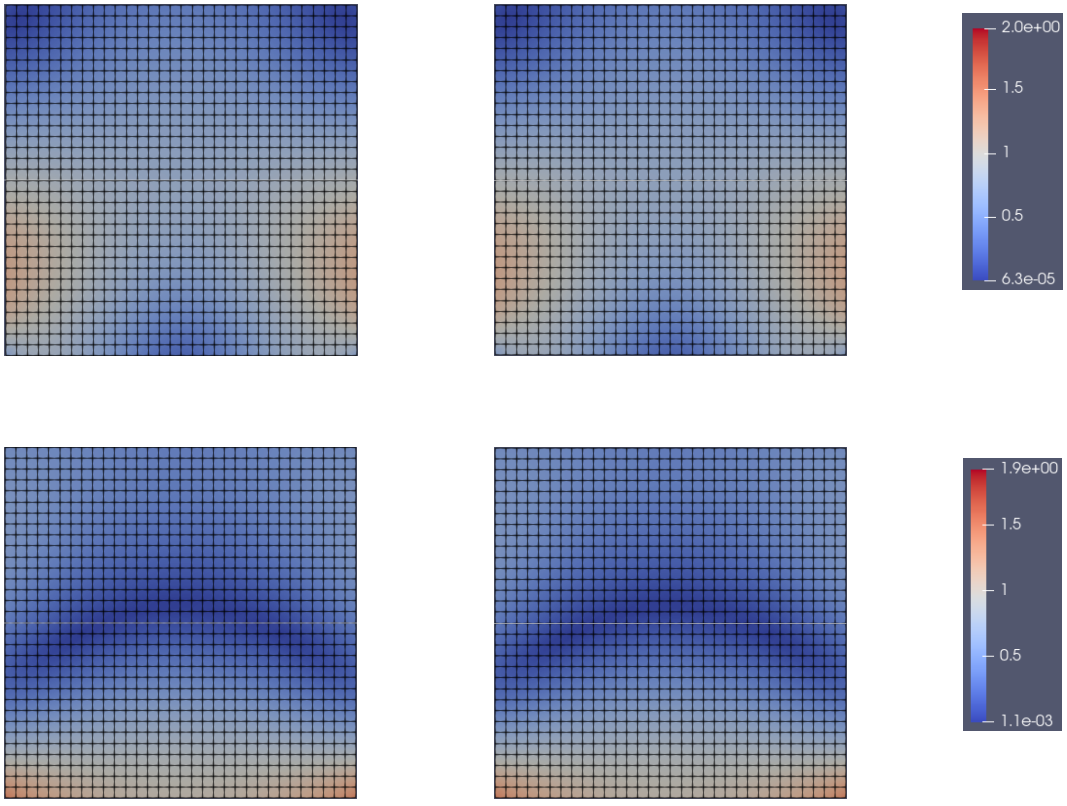


Figure 5.1: First example: Real part (above) and imaginary part (below) of the NN solution (left) and the DDM solution (right).

5.2. Higher wave numbers. As a second example, we increase the wave number, which leads to a more complicated problem. Therefore we repeat the same computation with $\omega = \pi$ and leave the other parameters (especially the parameters and hyperparameters of the neural networks) unchanged. In contrast to the previous example, the results that are displayed in Figure 5.2 show differences. While the imaginary

part is still well approximated, the real part of the NN solution differs significantly from the DDM solution and shows a discontinuity on the interface.

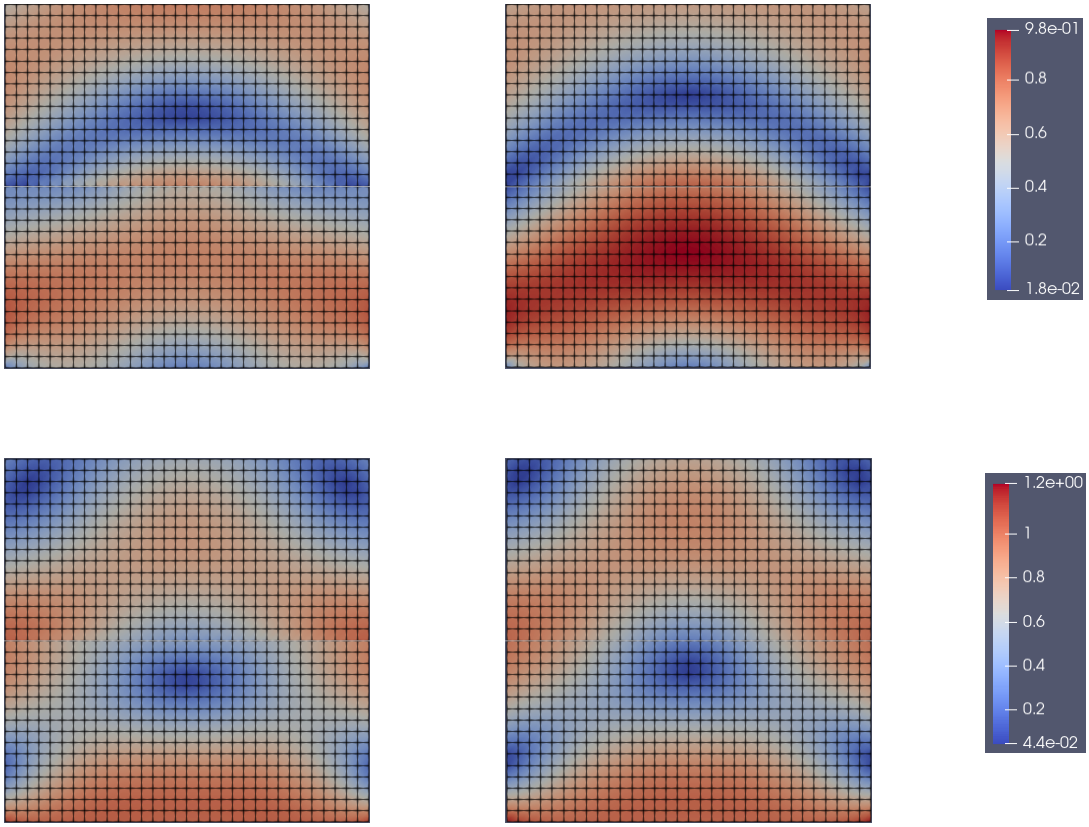


Figure 5.2: Second example: Real part (above) and imaginary part (below) of the NN solution (left) and the DDM solution (right).

5.3. Refined computational analysis for intermediate wave numbers. A possible reason for the mismatching results in Section 5.2 is the "problem of big wave numbers", which is very well-studied for Helmholtz-type problems [37]. The same problem also applies to Maxwell's equations [18]. To verify this conjecture and because of the very distinctive results in Section 5.1 and Section 5.2, we attempt two more computations with other wave numbers, namely $\omega = \frac{2\pi}{2.9}$ and $\omega = \frac{2\pi}{3.1}$. The results, that are displayed in Figures 5.3 and 5.4, in which we neglect the representation of the meshes to make the differences more visible, show that the approximation becomes inaccurate if the wave number differs slightly from the one we used for the training, regardless of whether it is larger or smaller. Therefore the bad approximation is not due to the big size of the wave number. Instead of this, it can be assumed that the NNs are specialized for the specific wave number they are trained with and "learn along" this value during the training process.

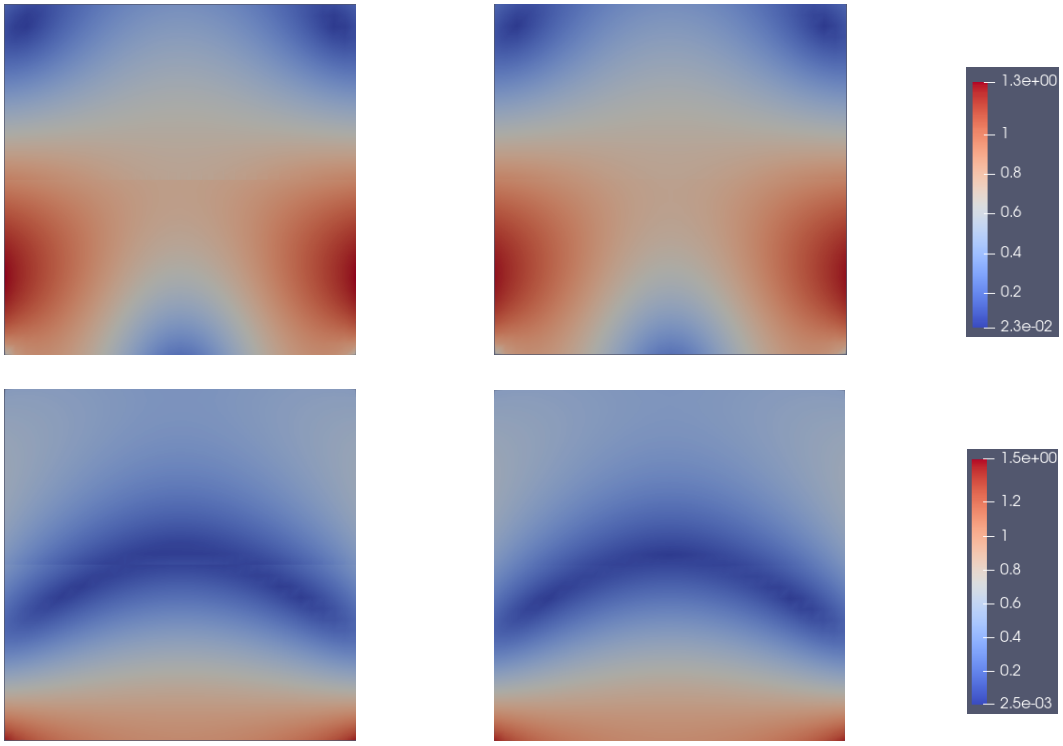


Figure 5.3: Third example: Real part (above) and imaginary part (below) of the NN solution (left) and the DDM solution (right) with $\omega = \frac{2\pi}{2.9}$

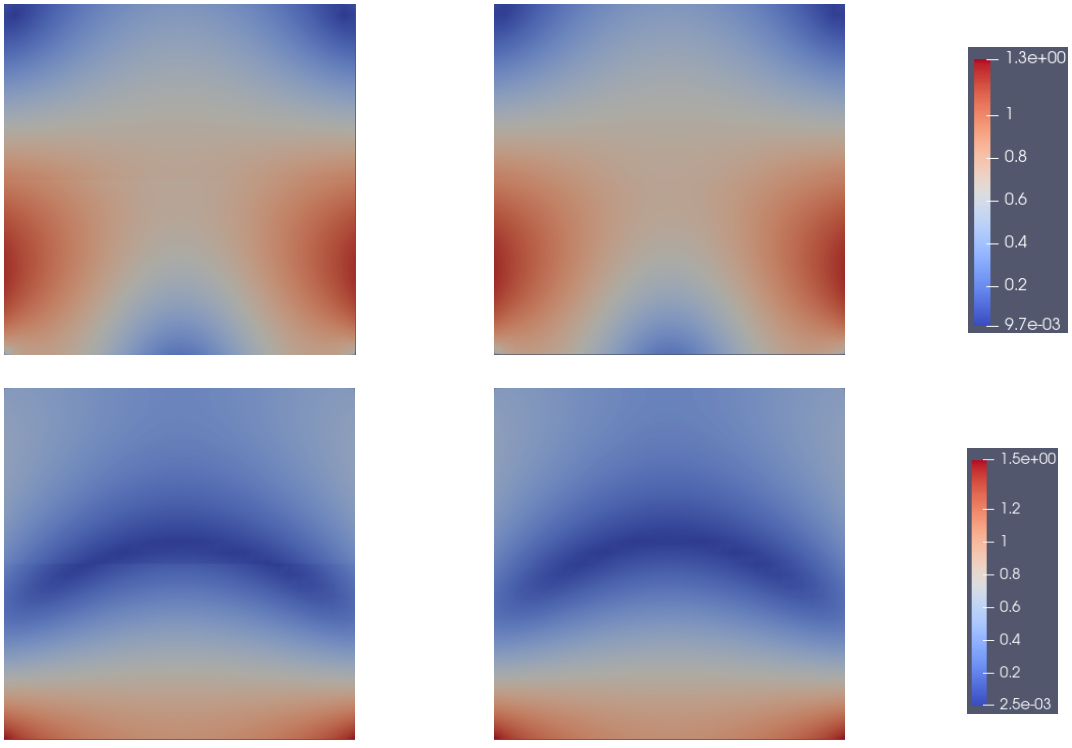


Figure 5.4: Fourth example: Real part (above) and imaginary part (below) of the NN solution (left) and the DDM solution (right) with $\omega = \frac{2\pi}{3.1}$.

5.4. Comparison of different neural network activation functions: Sigmoid vs. ReLU. As mentioned in Section 4.3, we tested different activation functions to train the NNs before using sigmoid. One

of these is the ReLU function given by

$$f(x) = \max(0, x),$$

which is implemented in the PyTorch class torch.nn.functional. This function allows a greater and faster error reduction than the others we tested, including sigmoid. In most cases, the test error of the network U^{01} can be reduced after approx. 16000 steps with a learning rate of 10^{-5} and ca. 6500 steps with a learning rate of 10^{-6} to $8 \cdot 10^{-4}$, which is almost a quarter compared to the final error in the training of the same NN with sigmoid as the activation function (see Section 4.4). Also, the test error of U^{10} can be reduced more quickly, namely to $2 \cdot 10^{-3}$ after ca. 3500 steps with a learning rate of 10^{-5}. However, we also observed that the test error grows after a short reduction phase in other cases. But in contrast, the training error continues to shrink, revealing that the training of our ReLU-networks is more susceptible to overfitting. This suspicion is strengthened when we apply the successfully trained ReLU-networks to the first example with the same procedure described in Section 5.1. The results displayed in Figure 5.5 show a discontinuity in the interface. This suggests that even in the lucky cases in which the test error is reduced very well, we are dealing with overfitting, and the resulting NNs cannot accurately capture the actual problem. Because of the unreliable training of the ReLU-NNs, it is reasonable to use sigmoid as the activation function instead.

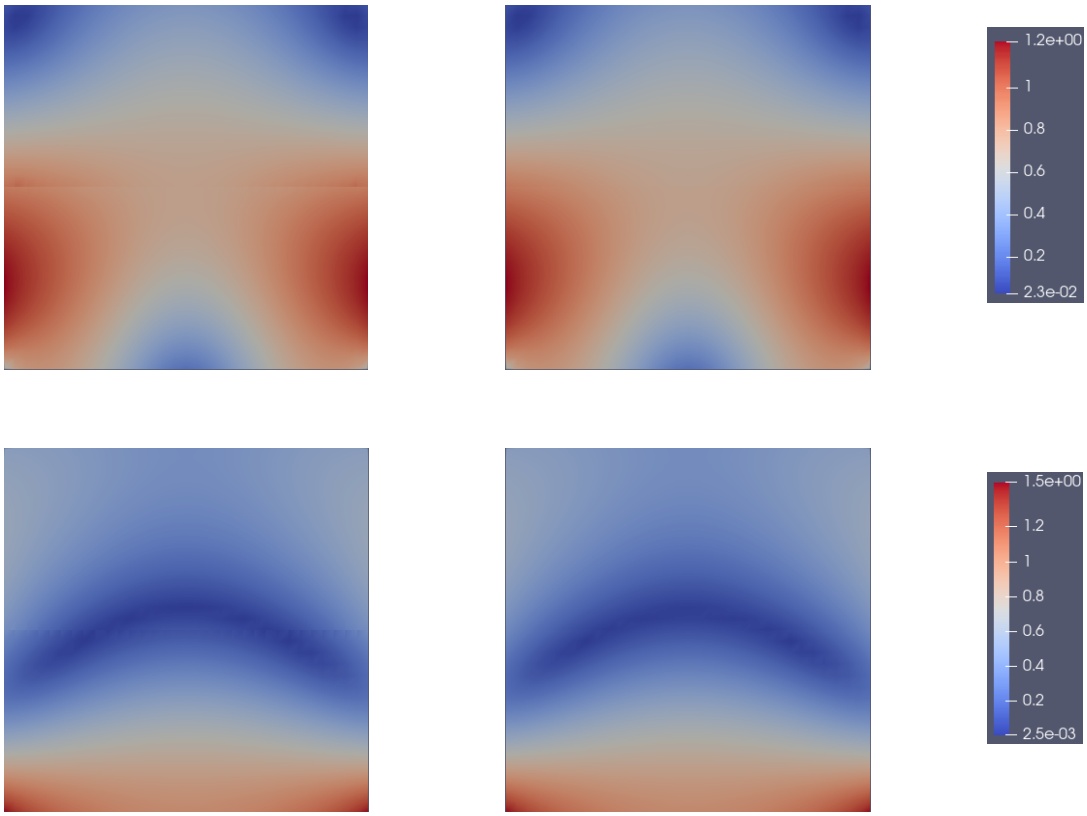


Figure 5.5: Real part (above) and imaginary part (below) of the NN solution with the use of ReLU (left) and Sigmoid (right) as activation function.

6. Conclusion. In this contribution, we provided a proof of concept and feasibility study for approximating the interface operator in domain decomposition with a feedforward neural network. These concepts are applied to the time-harmonic Maxwell's equations. We carefully described the numerical framework from the algorithmic and implementation point of view. In the realization, we coupled deal.II (C++) for solving the Maxwell's equations with PyTorch for the neural network solution. Afterwards, we conducted various numerical tests that included comparing our new approach with classical domain decomposition. Then, we studied higher wave numbers in more detail. Therein, we detected difficulties, which we further investigated, revealing that the training and testing of the neural network is highly sensitive to the specific wave number. Finally, a comparison of two different neural network activation functions was undertaken. As an outlook, we plan to increase the number of subdomains to study other wave numbers further and apply the method to three-dimensional Maxwell's equations.

Acknowledgment. This work is funded by the Deutsche Forschungsgemeinschaft (DFG) under Germany's Excellence Strategy within the Cluster of Excellence PhoenixD (EXC 2122, Project ID 390833453).

ORCID and License
Tobias Knoke https://orcid.org/0000-0003-2987-5110
Sebastian Kinnewig https://orcid.org/0000-0002-0923-7413
Sven Beuchler https://orcid.org/0000-0001-9411-8701
Ayhan Demircan https://orcid.org/0000-0002-0015-2077
Uwe Morgner https://orcid.org/0000-0001-5103-9632
Thomas Wick https://orcid.org/0000-0002-1102-6332

References

[1] Shi L, Babushkin I, Husakou A, Melchert O, Frank B, Yi J, et al. Femtosecond field-driven on-chip unidirectional electronic currents in nonadiabatic tunneling regime. Laser & Photonics Reviews. 2021;(15).
[2] Melchert O, Kinnewig S, Dencker F, Perevoznik D, Willms S, Babushkin I, et al. Soliton compression and supercontinuum spectra in nonlinear diamond photonics. Diamond and Related Materials. 2023;136:109939. Available from: https://www.sciencedirect.com/science/article/pii/S0925963523002649.
[3] Monk P. Finite element methods for Maxwell's equations. Oxford Science Publications; 2003.
[4] Demkowicz L. Computing with hp-adaptive finite elements. Volume 1 One and Two Dimensional Elliptic and Maxwell Problems. Chapman and Hall/CRC; 2006.
[5] Langer U, Pauly D, Repin S, editors. Maxwell's Equations: Analysis and Numerics. De Gruyter; 2019. Available from: https://www.degruyter.com/document/doi/10.1515/9783110543612/html.
[6] Rodriguez AA, Bertolazzi E, Valli A. In: Langer U, Pauly D, Repin S, editors. 1. The curl–div system: theory and finite element approximation. Berlin, Boston: De Gruyter; 2019. p. 1–44. Available from: https://doi.org/10.1515/9783110543612-001 [cited 2023-01-18].
[7] Nédélec JC. Mixed finite elements in $\mathbb{R}3$. Numerische Mathematik. 1980 Sep;35(3):315–341. Available from: https://doi.org/10.1007/BF01396415.
[8] Nédélec JC. A new family of mixed finite elements in $\mathbb{R}3$. Numerische Mathematik. 1986 Jan;50(1):57–81. Available from: https://doi.org/10.1007/BF01389668.
[9] Kinnewig S, Wick T, Beuchler S. Algorithmic realization of the solution to the sign conflict problem for hanging nodes on hp-hexahedral Nédélec elements; 2023. In preparation.
[10] Carstensen C, Demkowicz L, Gopalakrishnan J. Breaking spaces and forms for the DPG method and applications including Maxwell equations. Computers & Mathematics with Applications. 2016;72(3):494–522. Available from: https://www.sciencedirect.com/science/article/pii/S0898122116302620.
[11] Nicaise S, Tomezyk J. In: Langer U, Pauly D, Repin S, editors. 9. The time-harmonic Maxwell equations with impedance boundary conditions in polyhedral domains. Berlin, Boston: De Gruyter; 2019. p. 285–340. Available from: https://doi.org/10.1515/9783110543612-009 [cited 2023-01-18].
[12] Hiptmair R. Multigrid method for Maxwell's equations. SIAM Journal on Numerical Analysis. 1998;36(1):204–225.
[13] Henneking S, Demkowicz L. A numerical study of the pollution error and DPG adaptivity for long waveguide simulations. Computers & Mathematics with Applications. 2021;95:85–100.
[14] Faustmann M, Melenk JM, Parvizi M. $\mathcal{H}$-matrix approximability of inverses of FEM matrices for the time-harmonic Maxwell equations. Advances in Computational Mathematics. 2022;48(5).
[15] Dolean V, Gander Mj, Gerardo-Giorda L. Optimized Schwarz Methods for Maxwell's Equations. SIAM Journal on Scientific Computing. 2009;31(3):2193–2213. Available from: https://epubs.siam.org/doi/abs/10.1137/080728536.
[16] Schöberl J. A Posteriori Error Estimates for Maxwell Equations. Mathematics of Computation. 2008;77(262):633–649. Available from: https://www.jstor.org/stable/40234527.
[17] Bürg M. A residual-based a posteriori error estimator for the hp-finite element method for Maxwell's equations. Applied Numerical Mathematics. 2012 08;62:922–940.
[18] Beuchler S, Kinnewig S, Wick T. In: Brenner SC, Chung ETS, Klawonn A, Kwok F, Xu J, Zou J, editors. Parallel domain decomposition solvers for the time harmonic Maxwell equations. vol. 145 of Lecture Notes in Computational Science and Engineering. Springer; 2023. p. 615–622.
[19] Toselli A, Widlund O. Domain decomposition methods - algorithms and theory. Volume 34 of Springer Series in Computational Mathematics. Berlin, Heidelberg: Springer; 2005.
[20] El Bouajaji M, Thierry B, Antoine X, Geuzaine C. A quasi-optimal domain decomposition algorithm for the time-harmonic Maxwell's equations. Journal of Computational Physics. 2015;294:28–57. Available from: https://www.sciencedirect.com/science/article/pii/S0021999115001965.
[21] Arndt D, Bangerth W, Davydov D, Heister T, Heltai L, Kronbichler M, et al. The deal.II finite element library: Design, features, and insights. Computers & Mathematics with Applications. 2020;Available from: http://www.sciencedirect.com/science/article/pii/S0898122120300894.
[22] Arndt D, Bangerth W, Feder M, Fehling M, Gassmöller R, Heister T, et al. The deal.II library, Version 9.4. Journal of Numerical Mathematics. 2022;30(3):231–246. Available from: https://dealii.org/deal94-preprint.pdf.
[23] Bishop CM. Pattern recognition and machine learning. Information Science and Statistics. Springer New York, NY; 2006. Available from: https://link.springer.com/book/9780387310732.

[24] Higham CF, Higham DJ. Deep Learning: An introduction for applied mathematicians. SIAM review. 2019;61(4):860–891.
[25] Kinnewig S, Kolditz L, Roth J, Wick T. Numerical methods for algorithmic systems and neural networks. Hannover : Institutionelles Repositorium der Leibniz Universität Hannover, Lecture Notes. Institut für Angewandte Mathematik, Leibniz Universität Hannover; 2022. Available from: https://doi.org/10.15488/11897.
[26] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. PyTorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32. Curran Associates, Inc.; 2019. p. 8024–8035. Available from: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.
[27] Knoke T, Kinnewig S, Wick T, Beuchler S. Neural network interface condition approximation in a domain decomposition method applied to Maxwell's equations; 2023, accepted for publication. In: Domain Decomposition Methods in Science and Engineering XXVII, Springer.
[28] Girault V, Raviart PA. Finite element methods for Navier-Stokes equations. vol. 5 of Springer Series in Computational Mathematics. Springer-Verlag, Berlin; 1986. Theory and algorithms. Available from: https://doi.org/10.1007/978-3-642-61623-5.
[29] Zaglmayr S. High order finite element methods for electromagnetic field computation. Johannes Kepler University Linz; 2006.
[30] Copony S. Dynamisches Verhalten in neuronalen Netzen. Universität Hamburg; 2007. Available from: https://www.math.uni-hamburg.de/home/gunesch/Vorlesung/SoSe2007/Sem_DS_ODE/Vortrag/Copony_Vortrag.pdf.
[31] Kriesel D. A Brief Introduction to Neural Networks. http://www.dkriesel.com/en/science/neural_networks; 2005.
[32] Ben-David S, Shalev-Shwartz S. Understanding machine learning : from theory to algorithms. Cambridge: Cambridge University Press; 2014. Available from: http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning.
[33] Nielson A. Neural networks and Deep Learning. Determination Press; 2015.
[34] Knoke T, Wick T. Solving differential equations via artificial neural networks: Findings and failures in a model problem. Examples and Counterexamples. 2021;1:100035. Available from: https://www.sciencedirect.com/science/article/pii/S2666657X21000197.
[35] Ellacott SW. Aspects of the numerical analysis of neural networks. Acta Numerica. 1994;3:145–202.
[36] Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. In: Bengio Y, LeCun Y, editors. 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings;. Available from: http://arxiv.org/abs/1412.6980.
[37] Ernst OG, Gander MJ. In: Why it is Difficult to Solve Helmholtz Problems with Classical Iterative Methods. Lecture Notes in Computational Science and Engineering. Springer; 2012. p. 325–363. Available from: https://doi.org/10.1007/978-3-642-22061-6_10.