École Polytechnique Fédérale de Lausanne
Chair of Computational Mathematics and Simulation Science

Variational Physics-Informed Neural Networks For The Helmholtz Impedance Problem

Master Semester Project
June 9, 2022

Author: Sepehr Mousavi

Supervisors: Jan S. Hesthaven
Fernando Henriquez

Contents

Abstract

In this work, after presenting the methodology and the results of a finite element solver for the Helmholtz impedance problem and showing its limits in high wave numbers, a framework for solving this problem with variational physics-informed neural networks is presented and implemented. Using this framework, different aspects have been investigated in order to understand the behavior of this kind of solvers for the Helmholtz impedance problem. The behavior of the network with different architectures and different number of test functions is examined and it has been concluded that the best approach for reinforcing the network is to increase the number of hidden layers and to scale the number of test functions with the square root of the number of the parameters of the network. It has been shown that a coercive variational formulation with suitable test functions could be beneficial for high wave numbers, and it has been observed that a least-squares based initialization method could effectively facilitate the training. Some ideas for future works are suggested at the end of the report.

1 Introduction

The Helmholtz equation has the general form

$$-\nabla^2 u(\underline{x}) - k^2 u(\underline{x}) = f(\underline{x}), \qquad \underline{x} \in \Omega, \tag{1}$$

where $k \in \mathbb{R}_+$ is often called the wave number of the equation since larger values of this parameter result in more oscillating solutions. This equation represents a time-harmonic form of the wave equation after applying the Fourier transform and often arises in various physical problems such as seismology [1] and acoustics [2]. As will be shown in the following sections, there are some difficulties regarding solving this equation, especially for high wave number k. Graham et al. [3] have studied the performance of multiple iterative methods such as Krylov methods [4] and multi-grid methods [5] for the Helmholtz equation. They suggested that for solving this equation, hand-tailored techniques might be beneficial to improving the performance.

In this work, we will consider the one-dimensional Helmholtz equation with impedance boundary conditions in the domain $\Omega = (a, b)$

$$\begin{aligned} -u''(x) - k^2 u(x) &= f(x) \\ -u'(a) - \imath k u(a) &= g_a \in \mathbb{C} \\ +u'(b) - \imath k u(b) &= g_b \in \mathbb{C} \end{aligned} \tag{2}$$

We will mainly consider (2) in the domain $\Omega = (-1, +1)$. The Helmholtz equation with impedance boundary conditions is well-posed, and has a unique complex solution.

With the recent improvements in knowledge and practical aspects of machine learning techniques, and more specifically, deep learning and neural networks, these methods are being successfully implemented in domains and applications such as computer vision [6], recommender systems [7], and generative models [8], where they can benefit from the abundance of data and learn the features that best represent the problem. However, in the context of analyzing complex physical, biological, or engineering systems, these methods are facing the challenge of high cost of data acquisition, which forces us to find a way to make decisions in a semi-supervised fashion while

retaining the robustness and convergence of the methods. Recent studies in the literature have introduced deep learning frameworks for solving non-linear partial differential equations (PDEs) that have achieved this goal.

Raissi *et. al.* [9] have introduced the so-called physics-informed neural networks (PINNs) scheme, which uses the prior knowledge that usually stems from the physics of the system in a structured way as a regularization method to narrow the range of the admissible solutions. They consider the general form of a time-dependant non-linear PDE as

$$u_t + N[u; \lambda] = 0, x \in \Omega, t \in [0, T] \tag{3}$$

where $u(t, x)$ denotes the unknown solution, $N[.; \lambda]$ is a non-linear operator parameterized by λ, and Ω is a subspace which defines the domain of the system. With $f(t, x)$ as the left-hand side of this equation, the method consists of defining two neural networks for $u(t, x)$ and $f(t, x)$ with shared parameters, and optimizing these parameters based on a suitable loss function that takes into account the initial condition and the boundary conditions, and enforces the equation strictly in quadrature points inside the domain. They have shown that their method performs well even with small data for several non-linear PDEs by comparing their results with the exact solution of those equations.

Kharazmi *et. al.* [10] have taken the next step by developing a Petrov-Galerkin version of the PINNs scheme by selecting the trial space to be the space of the neural networks and the test space to be the space of trigonometric functions or Legendre polynomials. They introduce the variational physics-informed neural networks (VPINNs) by incorporating the variational residual of the PDE in the loss function of the network. They show that by integrating by parts the integrand part of the variational form, they can reduce the training time of the VPINNs and increase their accuracy compared to PINNs. They obtain the explicit form of the variational residual for shallow neural networks with one hidden layer, and suggest that numerical integration techniques should be employed for deep neural networks.

Outline of the Report

The goal of this study is to implement the VPINNs scheme introduced in [10] for solving (2) and to explore the behavior of these networks with different architecture, activation functions, initialization techniques, etc. for different wave numbers k. For this purpose, we will present the variational formulation of (2) and an alternative coercive variational formulation in Section 2.1. Then, we will proceed with presenting our framework for finite element method and for VPINNs scheme in Section 2.2 and Section 2.3, respectively. In Section 3, we will present and discuss the results of the implemented frameworks with various parameters and conditions. Section 4 summarizes our conclusions and provides some suggestions for future works.

2 Frameworks And Methods

2.1 Variational Formulation

We consider the one-dimensional version of the Helmholtz equation as expressed in (2), and test this equation by an arbitrary test function $v \in H^1(\Omega)$ to get

$$\int_\Omega (-u'' - k^2 u)v dx = \int_\Omega f v dx, \quad f \in L^2(\Omega) \qquad \forall v \in H^1(\Omega). \tag{4}$$

Integration by parts gives

$$\int_\Omega u'' v dx = [u'v]_{-1}^{+1} - \int_\Omega u'v' dx, \tag{5}$$

which could be replaced into (4) to eliminate the second derivative from the equation. Doing this, and by imposing the boundary conditions as

$$\begin{aligned} u'(-1)v(-1) &= (-g_a - \imath k u(-1))v(-1) \\ u'(+1)v(+1) &= (g_b + \imath k u(+1))v(+1), \end{aligned} \tag{6}$$

we get the variational formulation of the Helmholtz impedance problem, which is to find $u \in H^1(\Omega)$ such that $\mathsf{a}(u,v) = l(v)$ for all $v \in H^1(\Omega)$, where $\mathsf{a} : H^1(\Omega) \times H^1(\Omega) \to \mathbb{C}$ and $l : H^1(\Omega) \to \mathbb{C}$ are defined as

$$\mathsf{a}(u,v) = \int_{-1}^{+1} u'v' dx - k^2 \int_{-1}^{+1} uv dx - \imath k(u(-1)v(-1) + u(+1)v(+1)) \tag{7}$$

$$l(v) = \int_{-1}^{+1} f v dx + g_a v(-1) + g_b v(+1) \tag{8}$$

2.1.1 Coercive Variational Formulation

Another version of the variational formulation of the Helmholtz impedance problem is presented and analyzed in [11]. This formulation is proposed as a coercive formulation to overcome one of the challenges of the original variational formulation of the Helmholtz impedance problem. The accuracy and the convergence rate of the FEM with this new formulation is analyzed in [12]. In this section, we present a modification of this formulation for the one-dimensional case in the domain $\Omega = (-1, +1)$. Let us consider the Hilbert space $V := H^2(\Omega)$ with the inner product $(.,.)_V : V \times V \to \mathbb{C}$ defined as

$$\begin{aligned} (u,v)_V :=& k^2 (u,v)_{L^2(\Omega)} + (u',v')_{L^2(\Omega)} + k^{-2}(u'',v'')_{L^2(\Omega)} \\ &+ 2k^2 \left(u(-1)\overline{v(-1)} + u(+1)\overline{v(+1)} \right) \\ &+ k^2 \left(u'(-1)\overline{v'(-1)} + u'(+1)\overline{v'(+1)} \right), \quad u, v \in V \end{aligned} \tag{9}$$

and the induced norm $\|.\|_V := \sqrt{(.,.)_V}$, where $(.,.)_{L^2(\Omega)}$ corresponds to the $L^2(\Omega)$-inner product in Ω. For $k, \beta \in \mathbb{R}_+$, we define the sesquilinear form $\mathsf{a} : V \times V \to \mathbb{C}$ as

$$\begin{aligned} \mathsf{a}(u,v) :=& \int_\Omega u'\overline{v'} dx + k^2 \int_\Omega u\overline{v} dx + \int_\Omega \left(\mathcal{M}u + \frac{A}{k^2}\mathcal{L}u \right) \overline{\mathcal{L}v} dx \\ &- \imath k(\overline{\mathcal{M}v(b)}u(b) + \overline{\mathcal{M}v(a)}u(a)) - \overline{v'}\mathcal{M}u|_{\partial\Omega} - xk^2 u\overline{v}|_{\partial\Omega} + xu'\overline{v'}|_{\partial\Omega} \end{aligned} \tag{10}$$

and the linear form $l : V \to \mathbb{C}$ as

$$l(v) := \int_\Omega \left(\overline{\mathcal{M}v} - \frac{A}{k^2}\overline{\mathcal{L}v}\right) f dx + \overline{\mathcal{M}v(b)}g_b + \overline{\mathcal{M}v(a)}g_a, \tag{11}$$

where $A \in \mathbb{R}_+$ and the operators $\mathcal{M}$ and $\mathcal{L}$ are defined as

$$\begin{aligned} \mathcal{M}v &:= xv' - \imath k\beta v, \\ \mathcal{L}v &:= v'' + k^2 v. \end{aligned} \tag{12}$$

The problem would be defined as finding $u \in V$ such that $\mathsf{a}(u, v) = l(v)$ for all $v \in V$. It could be proved that setting $\beta = 1$ and $A = 1/3$ makes the sesquilinear form $\mathsf{a} : V \times V \to \mathbb{C}$ V-elliptic according to

$$\Re\{\mathsf{a}(u, v)\} \geq \gamma \|v\|_V^2 \quad \forall v \in V, \tag{13}$$

where $\gamma = \dfrac{1}{6}$.

2.2 The Finite Element Method

We define $N + 1$ basis functions as

$$\varphi_j(x) = \begin{cases} 1 - \frac{N}{2}|x - x_j|, & x_{j-1} < x < x_{j+1} \\ 0, & otherwise, \end{cases} \tag{14}$$

where $x_j = -1 + 2j/N$ for $j = 0, 1, \cdots, N$. Just in sake of completeness of the formulation we define the ghost points $x_{-1} = a$, and $x_{N+1} = b$. The resulting basis functions with $N = 9$ are illustrated in Figure 1. Let $V_N = span(\varphi_j)_{j=0,\cdots,N}$ be a finite-dimensional subspace of $H^1(\Omega)$. We search for solutions in this subspace, and modify the variational version of the Helmholtz impedance problem as finding $u_N \in V_N$ such that $\mathsf{a}(u_N, v_N) = l(v_N)$ for all $v_N \in V_N$, where $\mathsf{a}(u, v)$ and $l(v)$ are defined in Equations (7) and (8), respectively. It can easily be shown that it is sufficient to ensure that the aforementioned equation is satisfied for all the basis functions φ_i, $i = 0, \cdots, N$:

$$\mathsf{a}(u_N, \varphi_i) = b(\varphi_i). \tag{15}$$

Let $u_N \in V_N$ be a linear combination of the basis of V_N: $u_N(x) = \sum_{j=0}^{N} c_j \varphi_j$. By plugging u_N into (15), we can verify that the stated problem could be expressed as a linear system of equations $A\underline{c} = \underline{f}$, where

$$A = \begin{bmatrix} a(\varphi_0, \varphi_0) & a(\varphi_1, \varphi_0) & \cdots & a(\varphi_N, \varphi_0) \\ a(\varphi_0, \varphi_1) & a(\varphi_1, \varphi_1) & \cdots & a(\varphi_N, \varphi_1) \\ \vdots & \vdots & \ddots & \vdots \\ a(\varphi_N, \varphi_N) & a(\varphi_0, \varphi_N) & \cdots & a(\varphi_N, \varphi_N) \end{bmatrix}, \quad \underline{c} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_N \end{bmatrix}, \quad \underline{f} = \begin{bmatrix} l(\varphi_0) \\ l(\varphi_1) \\ \vdots \\ l(\varphi_N) \end{bmatrix}. \tag{16}$$

The solution of this system of equations could be calculated using methods such as Gauss-elimination. We don't need any quadrature rule to calculate the integrals in $a(\varphi_j, \varphi_i)$ as these result in determined values depending on i and j.

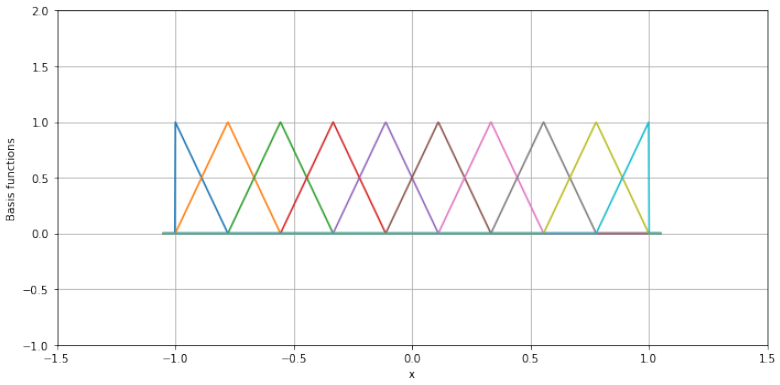


Figure 1: The finite element basis functions over the domain $\Omega = (-1, +1)$ for $N = 9$.

2.3 Framework of the VPINNs Scheme

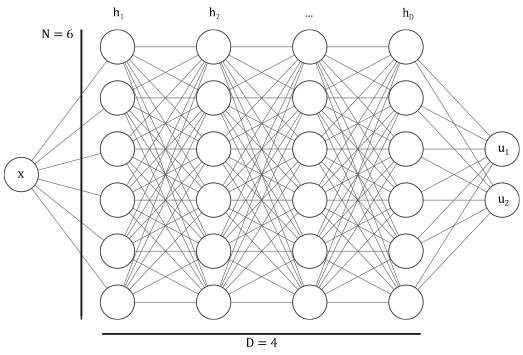
2.3.1 Multilayer Perceptron

Let $\mathcal{N}$ be the space of a fully-connected multilayer perceptron (MLP) with 1 node at the input layer, N nodes on each of D hidden layers, and 2 nodes at the output layer as depicted in Figure 2a. The 2 dimensional output represents the real and the imaginary parts of the solution: $u_N(x) = u_{N,1}(x) + \imath u_{N,2}(x)$. In the case of shallow networks, it could be easily shown that this will be the same as using complex model parameters. For this generic architecture of the MLP, each element of the the two-dimensional solution could be expressed as

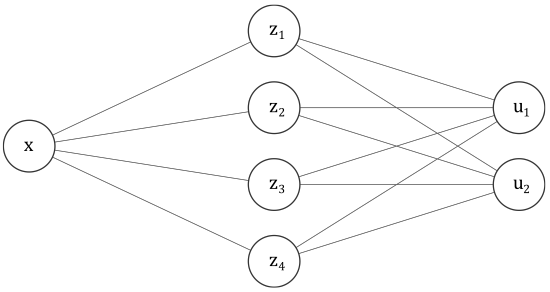
$$u_{N,\,i} = c_0^i + \sum_{j=1}^{N} c_j^i \sigma(h^D \circ h^{D-1} \circ \cdots \circ h^1(x)), \quad i \in \{1, 2\}, \tag{17}$$

where $\sigma(x)$ is a non-linear activation function and the output of the hidden layers $(h^d)_{d=1}^D$ are defined as

$$\begin{aligned} h_i^1 &= \sigma(w_i^1 x + b_i^1) \\ h_i^d &= \sigma(\sum_{n=1}^{N} w_{i,n}^d h_n^{d-1} + b_i^d), \quad d = 2, \cdots, D. \end{aligned} \tag{18}$$



(a) Generic multilayer perceptron with 4 hidden layer and $N = 6$.



(b) Shallow multilayer perceptron with $N = 4$.

With this architecture, the network has $2N$ parameters for the first hidden layer, $(D-1)(N^2+N)$ parameters for the other hidden layers, and $2N+2$ parameters for the last year, which sums up as

$$N_p = (D-1)N^2 + (D+3)N + 2, \tag{19}$$

where N_p is the total number of parameters of the network.

2.3.2 Variational Formulation for VPINNs

Let $V_K = span(v_1, v_2, \cdots, v_K)$ be the space of the test functions. The Helmholtz impedance problem in the context of VPINNs is defined as finding $u_{\mathcal{N}} \in \mathcal{N}$ such that the loss function

$$\mathcal{L}(u_{\mathcal{N}}) = \frac{1}{K}\sum_{k=1}^{K} |a(u_{\mathcal{N}}, v_k) - l(v_k)|^2 \tag{20}$$

is minimized, where a and l are the bilinear and the linear forms defined in either of the variational formulations presented in Section 2.1.

2.3.3 Least-squares Initialization

An important factor for training a neural network is the initialization of the parameters. Because of the unsupervised nature of VPINNs, the initialization could be more important and more helpful than for the networks with supervised learning purposes. Some methods such as new reptile initialization based physics-informed neural network (NRVPINN) [13] have been proved to be substantially helpful to the training. In this work, we investigate the effect of a specialized least-squares based initialization for the case of shallow networks which is described in this section. Considering N nodes in the hidden layer, we initialize the weight of this layer as $w_i^1 = k$, where k is the wave number of the equation. The bias of this layer initialized as $b_i^1 = -w_i^1 x_i^*$, where x_i^* for $i = 1, 2, ..., N$ is uniformly distributed on the interval Ω. With this initialization, the output of the shallow network could be expressed As

$$u_{N,\,i} = c_0^i + \sum_{j=1}^{N} c_j^i h_j^1 = c_0^i + \sum_{j=1}^{N} c_j^i \sigma(k(x - x_j^*)). \tag{21}$$

Initializing the biases of the last layers with zeros, i.e $c_0^i = 0$, the weights of the last layer are initialized as the real and imaginary part of the least-square solution of the system of equations $A\underline{c} = \underline{f}$, where A is a complex $K \times N$ matrix with $A_{i,j} = \mathsf{a}(\phi_i, h_j^1)$ and $f_i = l(\phi_i)$ for $i = 1, 2, ..., K$. An example of such an initialization is given in Figure 3 for $N = K = 20$ and the wave number $k = 16$. We can see that by such an initialization we can take advantage of the information we have about the frequency of oscillations of the solution to get a very good first guess. However, one needs to put more thought into it for implementing this initialization for deep networks, or other types of source functions. Furthermore, the method could be improved by choosing x^*'s adaptive to the wave number k, since these are the x-intercepts of the bases of the initial solution.

2.3.4 Quadrature Rules for Integrations

The integrals appearing in the loss function of the VPINNs scheme defined in (20) are evaluated using the Legendre-Gauss-Lobato quadrature rule, which for an arbitrary one-dimensional

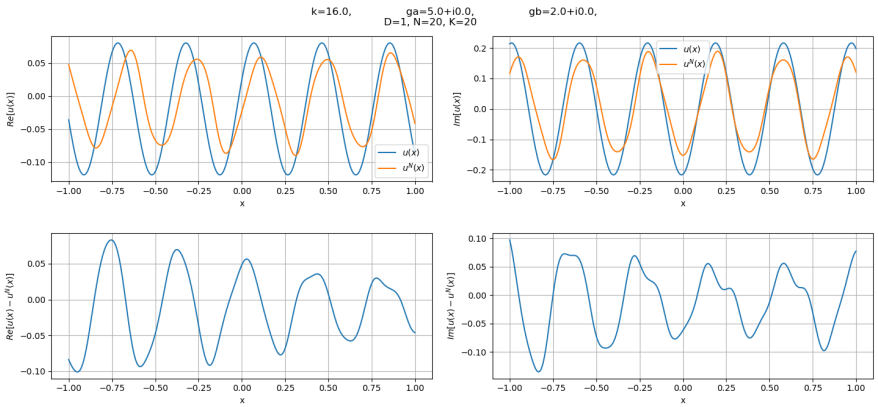


Figure 3: Example of the least-squares initialization for a shallow neural network.

function $h(x)$ is defined as

$$\int_a^b h(x)dx \simeq \sum_{i=1}^{N_q} w_i h(x_i), \tag{22}$$

where $(w_i)_{i=1}^{N_q}$ are the quadrature weights, and $(x_i)_{i=1}^{N_q} \subset \overline{\Omega}$ are the quadrature points. When using test functions with local support (such as finite element hat functions defined in (14)), the quadrature points should be chosen carefully. Since these test functions are only defined in a subset of the whole domain, choosing global quadrature points results in a poor approximation of the integral. Therefore, we use local quadrature points to evaluate the integrals that involve test functions with local support, and global quadrature points when the function being integrated is globally defined.

3 Results and Discussion

The frameworks described in the previous section are implemented and their results are presented and discussed in this section. We define the error of the solution as the $H^1(\Omega)$ norm of difference between the analytical solution $u(x)$, and the solution of the numerical scheme $u_N(x)$:

$$||u - u_N||_{H^1(\Omega)} = ||u - u_N||_{L^2(\Omega)} + ||u' - u'_N||_{L^2(\Omega)}, \tag{23}$$

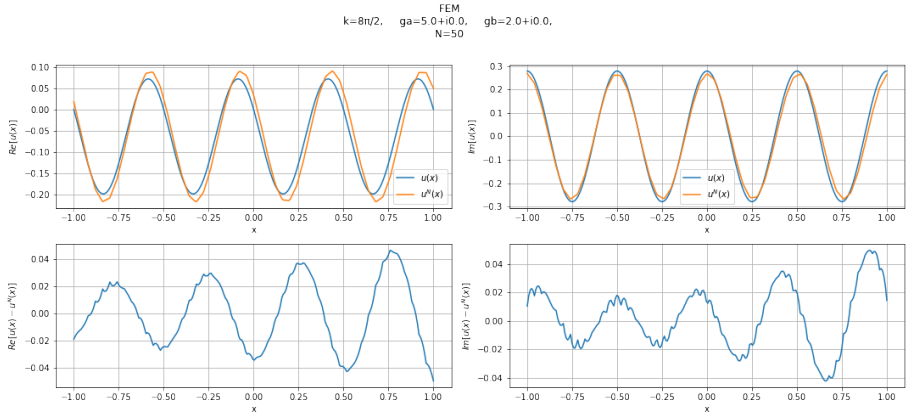
where

$$||f||_{L^2(\Omega)} = \left(\int_\Omega |f(x)|^2 dx \right)^{1/2}, \tag{24}$$

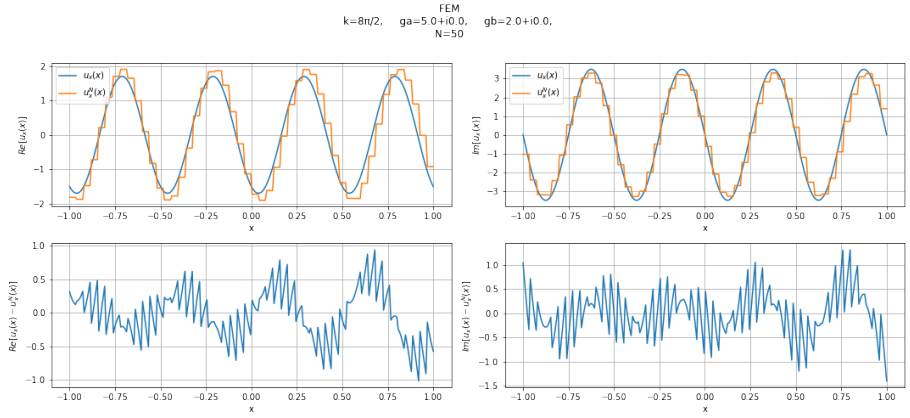
and use this error as a measure to evaluate the solution from each numerical scheme.

3.1 Finite Element Method

Implementing the framework described in Section 2.2, we investigate the results of the finite element method for the Helmholtz impedance problem with different wave numbers. The results for a relatively moderate k validated against the exact solution are presented in Figures 4 and 5. These plots show how using a finer mesh on the domain, i.e. corresponding to a bigger N, improves the quality of the solution. The same trend has been observed for other k's.



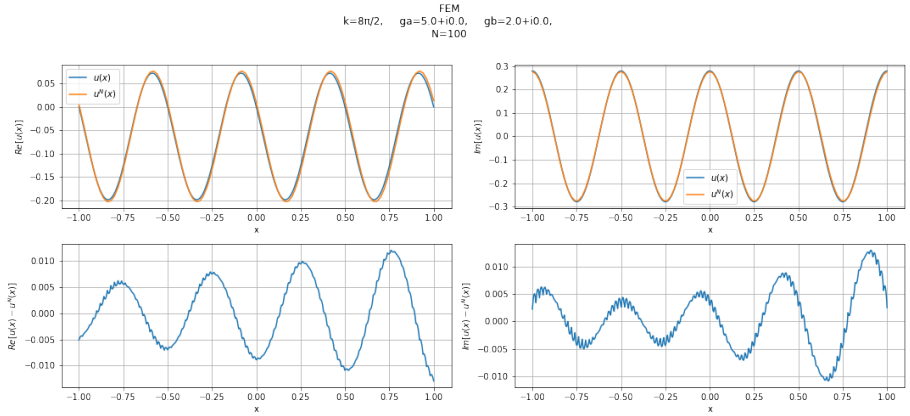
(a) $u_N(x)$.



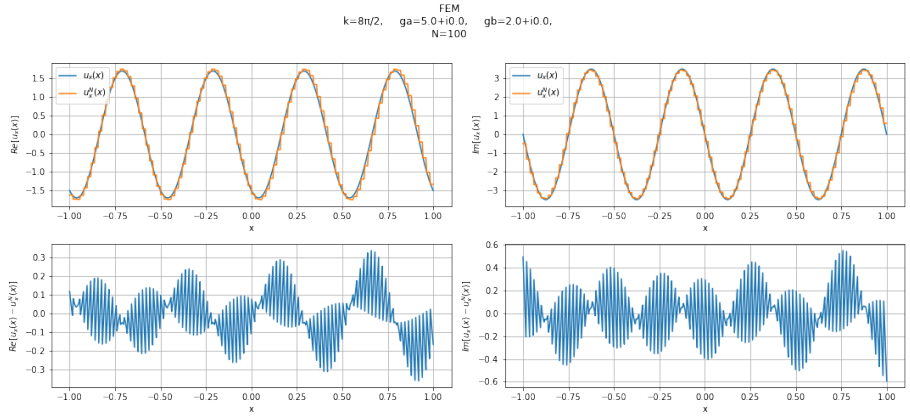
(b) $u'_N(x)$.

Figure 4: $u_N(x)$ and $u'_N(x)$ plotted against the exact solution for a relatively moderate k and $N = 50$. On each subfigure, the bottom row represents the difference between the numerical and exact solutions. The source function has a constant value of $f(x) = 10$, and the other parameters are indicated on the plots.

However, for larger values of k, there is an issue with this method. From the nature of the equation we know that for larger k's, we have more oscillations in the solution. Since the finite element method estimates the solution with piecewise linear basis functions in a uniform mesh, the resulting numerical solution will be piecewise linear. This will require a minimum number of grid points on each oscillation for the estimation to have a decent accuracy. Consider the solution in Figure 5 where we used 100 grid points. If we wanted to estimate this function with 8 grid points, for instance, it was not possible to even capture the general shape of the function. This phenomenon is the major observation in Figure 6, where the error of the numerical solution is plotted against mesh refinement parameter, N, for different values of k. For each constant k, there is no improvement in the error as we refine the mesh until a certain N, which we call N_c. However, for $N > N_c$, we can see the first-order accuracy of the method as expected. Another important observation is that the value of N_c increases with increasing k, which means that for high values of k, the improvement in the accuracy only exists for extremely fine meshes.



(a) $u_N(x)$



(b) $u'_N(x)$

Figure 5: $u_N(x)$ (a) and $u'_N(x)$ (b) plotted against the exact solution for a relatively moderate k and $N = 100$. On each subfigure, the bottom row represents the difference between the numerical and exact solutions. The source function has a constant value of $f(x) = 10$, and other parameters are indicated on the figures.

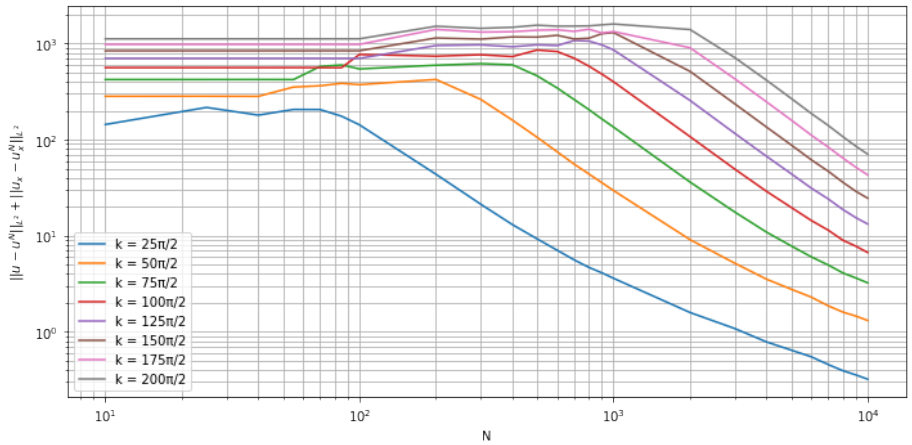


Figure 6: Order of accuracy of the FEM scheme for different values of k.

3.2 VPINNs Scheme

Implementing the framework explained in Section 2.3, we investigate the performance of this scheme for different network architecture, different activation functions, different initialization techniques, different formulations, and for different wave numbers k. The Adam optimizer [14] is used for all the trainings. Except for the results presented in Section 3.2.3, the weights of each layer are initialized using the normal Xavier initialization [15], and the biases are initialized using a uniform distribution in the domain Ω. This initialization, when compared to the least-squares initialization presented in Section 2.3.3, will be called random initialization. In all the trainings except the ones with the coercive variational formulation presented in Section 2.1.1 where the first K Legendre polynomials are used, we use the local hat functions defined by (14) for the test functions. Each training is done multiple times to ensure independence from randomness and the best one is selected for comparisons. The biggest learning rate lr that ensures convergence and accuracy for most of the networks being compared turned out to be 0.05 (unless stated otherwise), which is kept fix in all the comparisons.

3.2.1 Shallow Networks

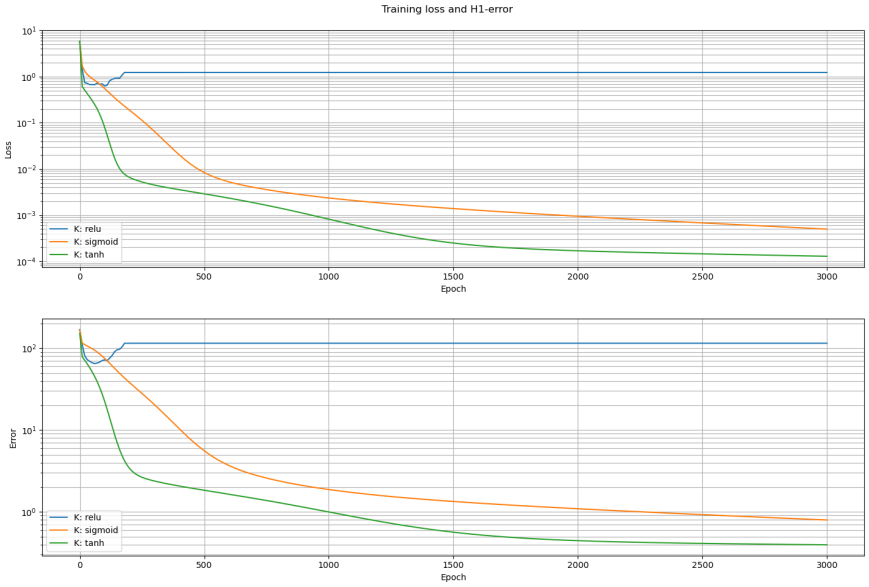
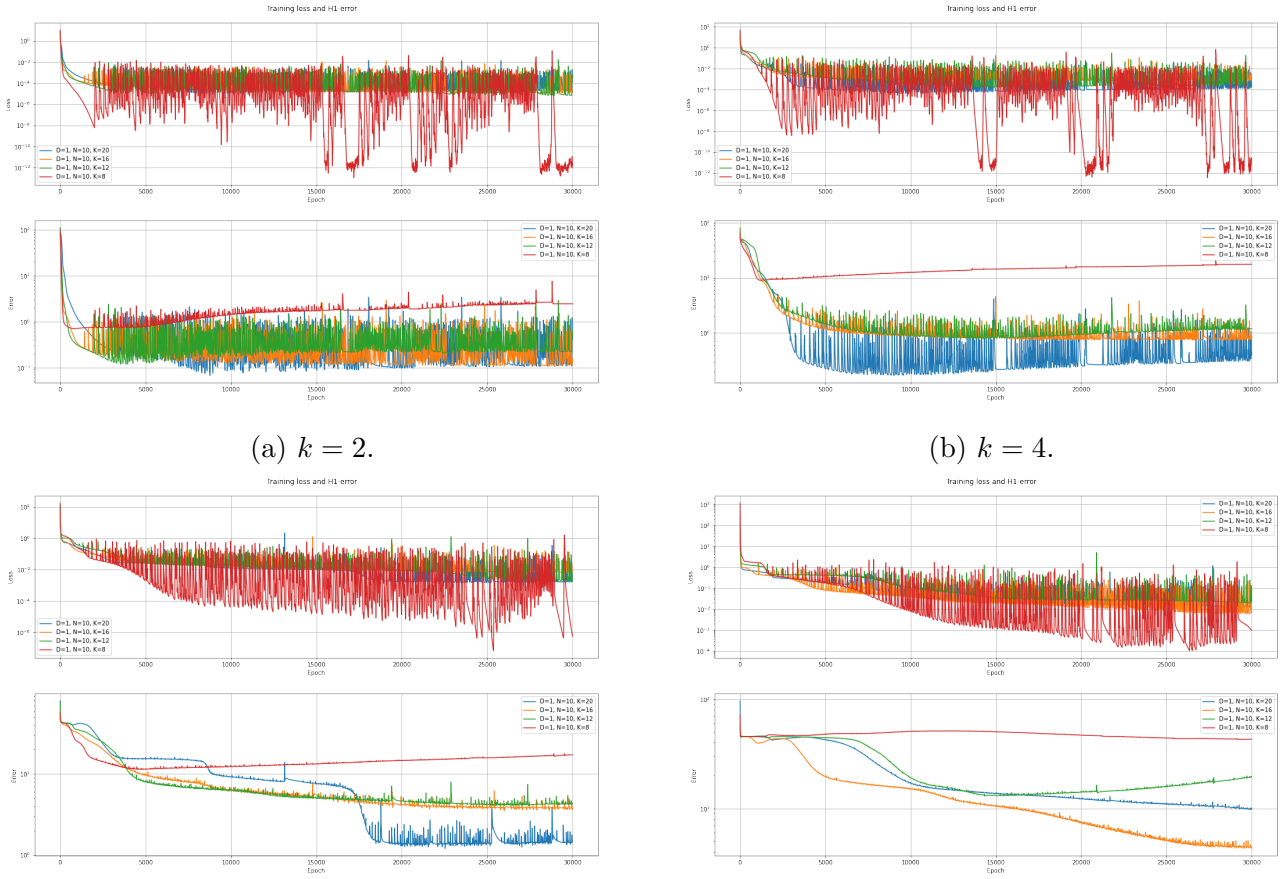


Figure 7: Training loss and solution error of a shallow network with $N = 10$ and $K = 10$ with different activation functions for $k = 1.0$.

We start with shallow neural networks with only one hidden layer, i.e. $D = 1$. We use the local hat functions presented in (14) as test functions, and train a network with $N = 10$ and 10 test functions ($K = 10$) with different types of activation functions for a low wave number. The loss functions and the errors are compared in Figure 7. The learning rate is 0.1 for all the activation functions. We can see that the hyperbolic tangent activation function outperforms the sigmoid activation function for our setup both in terms of accuracy and the convergence rate. Regarding the ReLU activation function, there are some issues inherited with this set-up of networks. If we train the weights and biases of the hidden layer, these parameters drive the threshold of each ReLU outside of the support domain of the test functions, and the gradient of the loss with

respect to these parameters goes to zero, so technically, the hidden neurons will be lost one by one. As done in [16], it could be easily shown that in case of the ReLU activation function, the weights of the hidden layer are redundant for this kind of networks and it is suggested to set the weights to 1. To address the issue with ReLU activation function, the network is trained with fixed biases being uniformly distributed in the domain $\Omega = (a, b)$ and weights fixed to 1, and the training could be completed without this issue. However, by fixing the parameters of the hidden layer and the ReLU activation function, this framework has no advantage over the original FEM scheme in case of capability and adaptability. In the rest of the experiments, the hyperbolic tangent activation function is used for the trainings.



(a) $k = 2$.

(b) $k = 4$.

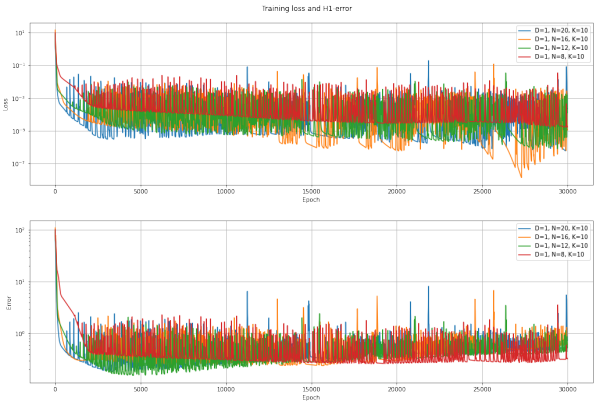
(c) $k = 6$.

(d) $k = 8$.

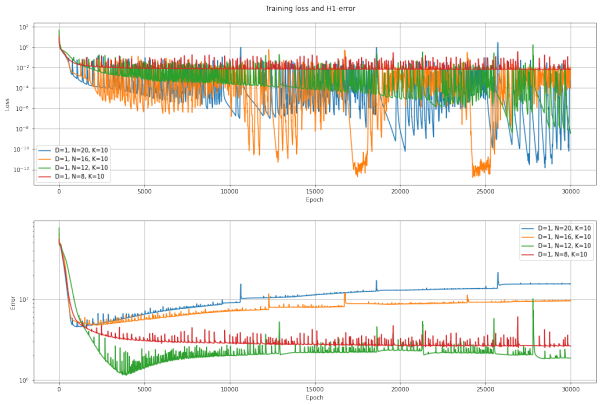
Figure 8: Training loss and solution error with fixed network architecture and increasing number of test functions for different wave numbers.

In an effort to investigate the effect of increasing the number of test functions on the accuracy and the training of the network, with a fixed width $N = 10$, the training is done for different values of K. The results are presented in Figure 8. For the lowest wave number $k = 2$ in Figure 8a, we can see that with 8 test functions, the training loss becomes unstable and the solution error takes an increasing trend after around 2000 epochs. With 12 test functions, this behavior is not observed and the solution error converges. Increasing the number of test functions to 16, we can see a slight improvement, which does not repeat itself with further increasing the number of test functions to 20. For a higher wave number $k = 4$ in Figure 8b, we can see the same behavior

for 8 test functions but with the difference that the final solution error keeps being improved with increasing the number of test functions to 20. For the highest wave number $k = 8$, we can observe the unstable behavior extends to 12 test functions, and the improvement with increasing the number of test functions becomes more evident. We can conclude that as the wave number increases, more number of test functions will be needed for training the network.



(a) $k = 2$.



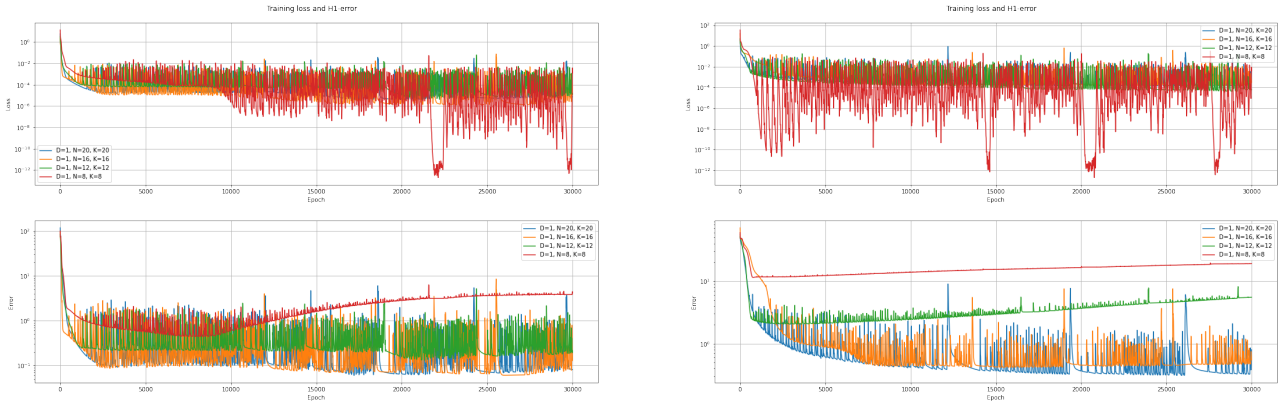
(b) $k = 4$.

(c) $k = 6$.

(d) $k = 8$.

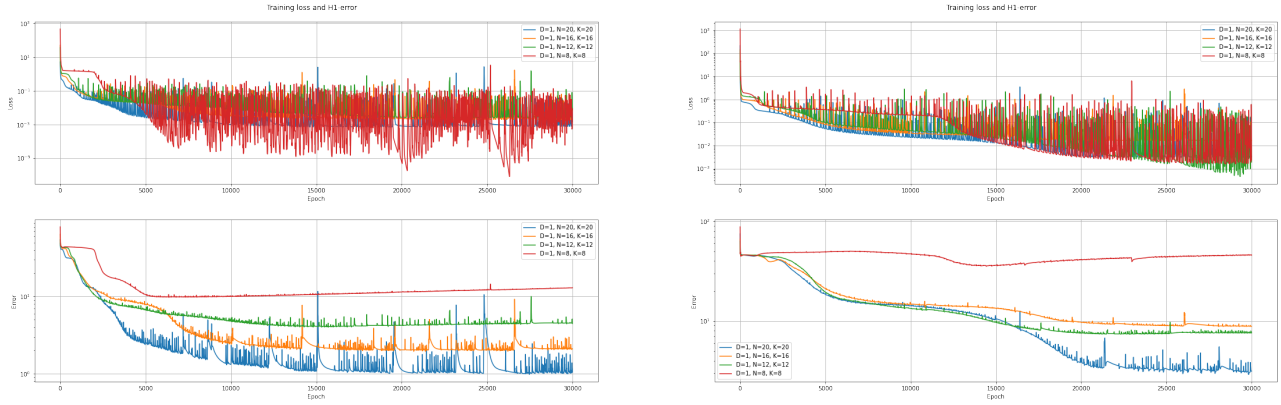
Figure 9: Training loss and solution error with fixed number of test function and increasing width of the network.

To investigate the effect of the width of the network, a similar thing is done with fixing the number of test functions to $K = 10$, and training the network for different values of N. The results are presented in Figure 9. For the lowest wave number $k = 2$ in Figure 9a, we can see that increasing the width of the network does not bring much to the table. However, the network reaches its minimum solution error faster. For $k = 4$ in Figure 9b, we can see that when the width of the network is increased to 16 or 20, the unstable behavior emerges even for sufficient number of test functions. For higher wave numbers, the network is not able to converge to the solution with any of these widths, and we can see that increasing the width even makes things worse in Figure 9d. These observations allow us to conclude that increasing the width of the network cannot heal the need for higher number of test functions, and even requires more test functions in order to converge to the solution.



(a) $k = 2$.

(b) $k = 4$.



(c) $k = 6$.

(d) $k = 8$.

Figure 10: Training loss and solution error with fixed number of test function and increasing width of the network.

In another set of experiments, the width of the network and the number of test functions are kept equal and are increased together. The idea was to treat these two hyper-parameters as one parameter since they are bounded together in the FEM scheme. The results are presented in Figure 10. We can see that the final solution error is consistently improved with increasing both N and K together for all of the wave numbers, but the improvement becomes more evident as the wave number gets higher. These experiments provide evidence that the best approach is to keep N and K proportional, and increase them together in order to study the convergence rate of this method.

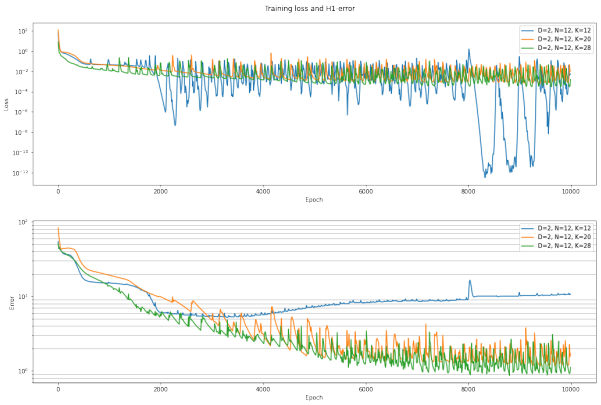
Figure 11 shows how a quite strong network with $N = K = 20$ performs for different wave numbers. Although the network is doing a good job for all of the wave numbers, we can see that as the wave number increases, the accuracy of the final solution decreases, which again implies the need to enrich the network for higher wave numbers.

3.2.2 Deep Networks

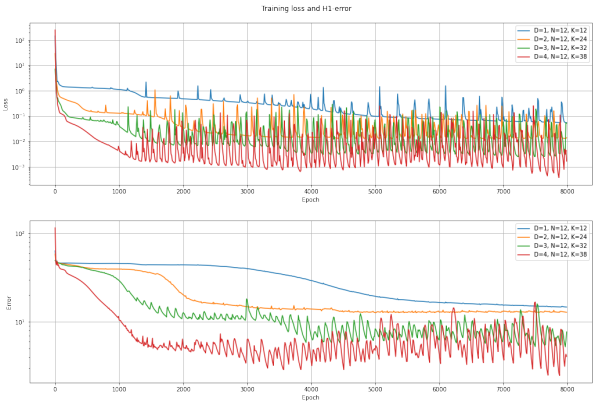
From (19), we can see that for deeper neural networks ($D > 1$), the number of parameters has a quadratic dependance on the width of the network, N. This gave us the notion that just as

Figure 11: Training loss and solution error of a shallow network with $N = 20$ and $K = 20$ for different wave numbers.

increasing N would require more test functions, increasing D might also require the training to be done on more test functions. To address this, we compared the training process for a network with two hidden layers and $N = 12$ with different number of test functions. The results are presented in Figure 12a. We can see that with the same number of test functions as for shallow networks, the model is overfitted to the test functions. While with 20 test functions we can get a better solution error, further increasing the number of test functions to 28 is not advantageous. We consider 24 test functions adequate for this network architecture with 206 parameters. Comparing this number to the number of parameters (50) of the shallow network with the same width, we can assume that the number of adequate test functions scales with the square root of the number of parameters: $K_{adequate} \sim \sqrt{N_p}$.



(a) Training loss and solution error of a network with 2 hidden layers and $N = 12$ for wave number $k = 8.0$, with different number of test functions.



(b) Training loss and solution error of a network with $N = 20$ for wave number $k = 8.0$, with different depths and number of test functions.

In Figure 12b, we keep the width of the network to $N = 12$, increase the depth of the network, and increase the number of test functions as $K \simeq 12\sqrt{N_p/50}$ as concluded earlier. It shows that with this approach, both the convergence speed and the final accuracy could be improved very well. Comparing this approach with the best approach in shallow neural networks (increasing the width and the number of test functions equally), we saw that on our CPUs, the deep network

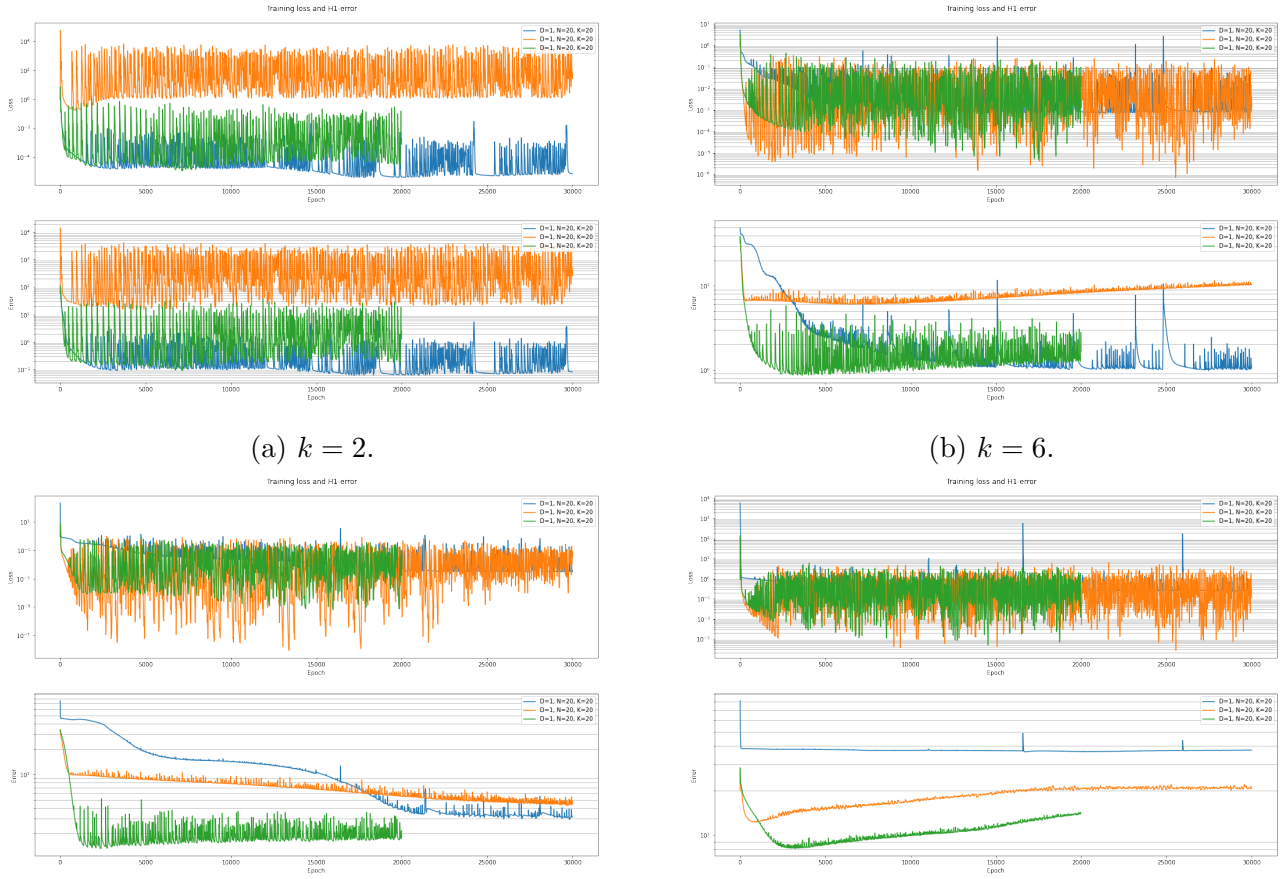
with $D = 4$, $N = 12$ and 38 test functions reaches a decent solution error in 15 minutes, while a shallow neural network with 20 nodes and 20 test functions requires 65 minutes for reaching the same accuracy.

3.2.3 Least-squares Initialization

Instead of random initialization, we initialized the weights and the biases of the shallow network by the method described in Section 2.3.3 to see how it will affect the training. We trained a shallow network with 20 nodes and 20 test functions for $k = 2$, $k = 6$, $k = 8$, and $k = 16$, and compared the training loss and the solution error to the ones of the same architecture with random initialization (as described in the previous sections). The learning rate is fixed to 0.05 in all the trainings, and the same optimizer configurations is used. The results are compared in Figure 13. First of all, we can observe that the initialization is mostly effective for high wave numbers. For $k = 2$, although the least-squares initialization gives a good first guess, the random initialization outperforms it both in the final accuracy and in the convergence speed. For other wave numbers, we can see a huge improvement in terms of the convergence speed. For $k = 6$, the solution error hits its minimum in less than 500 epochs, which is reached by random initialized network in around 4000 training epochs. A similar trend could be observed for $k = 8$ where the least-squares initialized network reaches and solution error of 10 in 500 training epochs, while with random initialization this error is only reached after 15000 epochs. For $k = 16$, we can see that the random initialized network fails to converge to the solution but the least-squares initialized performs quite well.

However, after a very sharp decrease in the solution error in the initial epochs, the least-squares initialized networks were not successful in reaching the potential of the network completely, or with a good convergence rate. We can see that for all the wave numbers below $k = 16$, the final solution error is much better with random initialization. This might lie on the fact that although the least-squares initialization gives a solid first guess of the final solution, it might lack the randomness usually demanded by the neural networks to adapt to doing their task perfectly. This could be originated in the initialization of the parameters of the hidden layer, where we set all the weights equally. This issue could be easily solved by adding a noise to these weights. In a slightly different version of this initialization, instead of setting all the weights of the hidden layer to the wave number k, we sampled them from a normal distribution with the mean $k^{0.75}$ and the standard deviation $k^{0.2}$. The green curves in Figure 13 correspond to this set of experiments. We can see that this technique improves the training with this initialization impressively. For $k = 6.0$, the network with improved initialization hits a minimum solution error in less than 3000 epochs, compared to 20000 epochs with random initialization. For $k = 8$, the minimum error is reached in 2000 epochs, which the randomly initialized network does not reach even in 30000 epochs, and for $k = 16$, the best accuracy reached is clearly improved.

Another important aspect regarding the proposed initialization method is that the performance of the first guess is not necessarily improved with increasing the width of the network, N. This is simply because of the uniform distribution of x^*'s, as indicated in Section 2.3.3. In fact, for $k = 6$, for instance, the least-squares initialized network with 12 nodes resembles the exact solution much better than with 20 nodes. This behavior of the initialization could also be decisive in the instable behavior of the network in Figures 13b and 13d, and explains the better performance for $k = 8$ in Figure 13c. Having a more suitable distribution of the x^*'s based



(a) $k = 2$.

(b) $k = 6$.

(c) $k = 8$.

(d) $k = 16$.

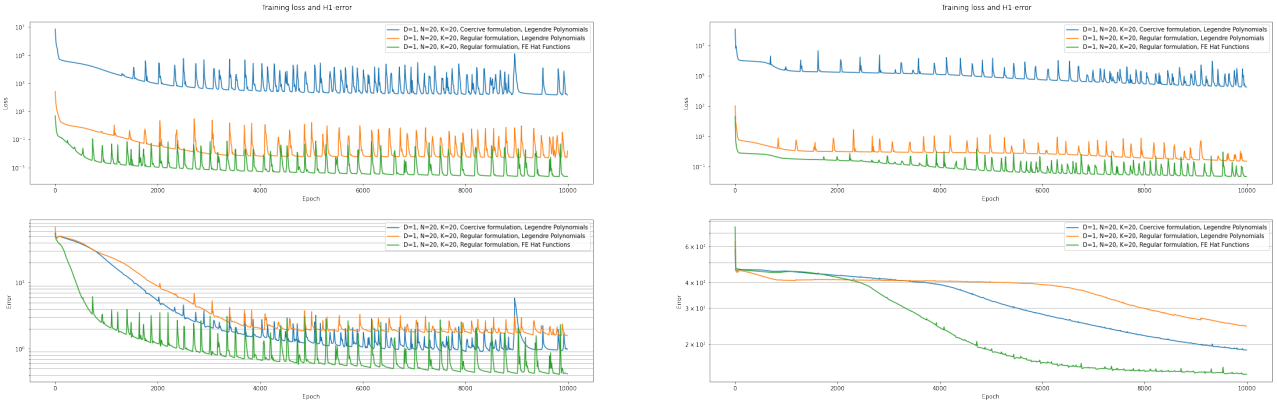
Figure 13: Training loss and solution error of the same network architecture with random initialization (blue), least-squares initialization (orange), and improved least-squares initialization (green) for different wave numbers.

on the wave number k and the type of source function could solve this issue and improve the initialization method very well.

3.2.4 Coercive Variational Formulation

In order to investigate the behavior of the network using the coercive variational formulation presented in Section 2.1.1, we trained a shallow neural network with 20 nodes on the hidden layer and 20 test functions with this formulation, and compared the training process to the original variational formulation. As the coercive formulation requires test functions in $H^2(\Omega)$, the finite element hat functions could not be used for this formulation. Therefore, the first 20 Legendre polynomials, which have global support, are used as test functions. As illustrated in Figure 14, the coercive formulation with Legendre polynomials as test functions converges more slowly than the original formulation with hat test functions, but slightly faster than the original formulation with Legendre polynomials. Comparing the time it took for each of these experiments to perform 10000 epochs (on the same device), we can conclude that using global test functions is computationally more expensive than using local test functions, and the coercive formulation is

computationally more expensive than the original variational formulation. In the case of $k = 8$, the coercive formulation with Legendre test functions took 260 minutes to evolve 10000 epochs, which took only 24 minutes for the original formulation with hat test functions.



(a) $k = 4$.

(b) $k = 8$.

Figure 14: Training loss and solution error for the same architecture with different formulations and test functions.

However, the advantages of the coercive formulation reveal themselves when being compared to the regular formulation with Legendre polynomials. Comparing the difference of these two methods for $k = 4$ and $k = 8$, we can see that as the wave number increases, the advantages of the coercive formulation become more significant. This brings up the idea that using this formulation in conjunction with suitable test functions in $H^2(\Omega)$ with local support could lead to better convergence rates for high wave numbers.

4 Conclusion

In Section 3.1, we showed that although the Helmholtz impedance problem could be solved by the finite element method accurately and efficiently, the first-order convergence (in the case of using the hat test functions) only starts after a certain level of refinement, and as the wave number k increases, this critical level of refinement goes higher. This motivated us to implement a VPINNs scheme for this problem and investigate the effect of different configurations of the network on its performance. These observations can eventually be put together to craft some specialized techniques to get a better performance in high wave numbers.

In Section 3.2.1, we observed that the hyperbolic tangent is the most suitable activation function for this setup, and kept using this activation function in the rest of the experiments. By doing several experiments with different network widths (N) and different number of test functions (K), we also observed that the best refinement approach for shallow networks is to increase these two hyper-parameters correlated to each other. In Section 3.2.2, we investigated the existence of a similar correlation with the depth of the network and concluded that the correlation should be defined on the number of parameters of the network, and proposed to scale the number of test functions with the square root of the number of parameters of the network. We also showed that

with adequate width, increasing the depth of the network is a better approach than increasing its width, in terms of convergence time.

In Section 3.2.3, we investigated the initialization method described in Section 2.3.3 and showed that a suitable initialization technique could effectively improve the convergence speed of the network. We showed that for $k = 8$ with sufficient width and sufficient test functions, the network with suitable initialization could achieve its capacity in 2000 epochs, while with random initialization, even after 30000 epochs it does not reach its capacity. We also pointed out some suggestions for improving this initialization technique such as adapting the biases to the nature of the solution, instead of distributing them uniformly in the domain. In Section 3.2.4, we compared the convergence rate of the formulation presented in Section 2.1.1 with the original variational formulation and concluded that the coercive variational formulation could be beneficial in high wave numbers, but it is computationally more expensive in low wave numbers. We also pointed out that using a set of suitable test functions in $H^2(\Omega)$ with local support could speed up the training with this formulation.

In future works, some of the ideas developed in the previous sections could be investigated to see if they help the training or making the network more powerful:

- The use of ReLU and similar activation function could be mathematically analyzed and the reasons of their failure should be investigated. Once done, the architecture of the network could be changed to take advantage of features of this activation function. As a suggestion, a local version of the ReLU activation function (e.g., ReLU with support in $[0, 1]$) could be applied only on the first hidden layer in order to make the network behave differently in different parts of the domain. This way, depending on where the input of the network is, only some of the nodes in the first layer will be activated. Such a network is implemented in the `VPINN_HelmholtzImpedanceRF` class but fails to converge for the same reasons mentioned in Section 3.2.1. The results and the detailed implementation are provided in Appendix A.
- Similar to the proposed initialization method in Section 2.3.3, the weights of the last layer could be calculated by least-squares on every iteration. Doing this, we will only optimize the parameters of the hidden layers, thus we will solve for the best basis functions that suite the solution. Such methods have been proposed in [17] and [16].
- Inspired by data augmentation techniques in supervised machine learning, we can interpret the test functions as our data, and try out some similar techniques on the way we use them. It was understood that increasing the number of test functions has the most impact on the computational cost. Therefore, using the test functions more smartly could improve the training in terms of speed and accuracy. Translating the stochastic gradient decent method, we can take the gradient decent steps on a batch of the test functions, instead of summing up the gradients for all of them before taking the step. Doing this, we can also shuffle the order of the test functions on each epoch. Another preposition which could massively increase the capacity of the network without adding significant computational costs, is to randomly choose K test functions among a set of $K' > K$ number of test functions. That is to say, we can, instead of defining K local hat functions and training on all of them, define K' local hat functions, which discretized the domain better, but still keep training on only K of them, randomly chosen on each epoch. Stochastically, the network will be

optimized for all the test functions.

- Investigating the evolution of the solution throughout the training, we could see that with some configurations, the network manages to capture either the real or the imaginary part of the solution very well, but lacks accuracy in the other part of the solution. The evolution of the solution with random initialization and with least-squares initialization are available in Appendix B. Regarding this observation, we suggest that training two networks for each part (imaginary, real) of the solution could be beneficial.

References

[1] Tariq Alkhalifah, Chao Song, Umair bin Waheed, and Qi Hao. Wavefield solutions from machine learned functions constrained by the Helmholtz equation. *Artificial Intelligence in Geosciences*, 2:11–19, 2021.

[2] Qiang Sun, Evert Klaseboer, Boo-Cheong Khoo, and Derek Y. C. Chan. Boundary regularized integral equation formulation of the Helmholtz equation in acoustics. *Royal Society Open Science*, 2(1):140520, Jan 2015.

[3] Ivan G Graham, Thomas Y Hou, Omar Lakkis, and Robert Scheichl. *Numerical analysis of multiscale problems*, volume 83. Springer Science & Business Media, 2012.

[4] Shitao Fan. An introduction to Krylov subspace methods, 2018.

[5] Jinchao Xu and Ludmil T Zikatanov. Algebraic multigrid methods, 2016.

[6] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[7] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.

[8] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[9] M. Raissi, P. Perdikaris, and G.E. Karniadakis. 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.

[10] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. Variational physics-informed neural networks for solving partial differential equations, 2019.

[11] Andrea Moiola and Euan A Spence. Is the Helmholtz equation really sign-indefinite? *Siam Review*, 56(2):274–312, 2014.

[12] Ganesh C. Diwan, Andrea Moiola, and Euan A. Spence. Can coercive formulations lead to fast and accurate solution of the Helmholtz equation?, 2018.

[13] Xu Liu, Xiaoya Zhang, Wei Peng, Weien Zhou, and Wen Yao. A novel meta-learning initialization method for physics-informed neural networks, 2021.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[15] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[16] Min Liu, Zhiqiang Cai, and Jingshuang Chen. Adaptive two-layer relu neural network: I. best least-squares approximation, 2021.

[17] Eric C. Cyr, Mamikon A. Gulian, Ravi G. Patel, Mauro Perego, and Nathaniel A. Trask. Robust training and initialization of deep neural networks: An adaptive basis viewpoint, 2019.

A Appendix: The `VPINN_HelmholtzImpedanceRF` Network

Listing 1 includes the code implementation of the proposed network on top of the parent class. In this architecture, we use a local version of ReLU as the activation function of the first hidden layer, and a sinusoidal (hyperbolic tangent) activation function for other hidden layers. The idea is to impose different behaviors for different regions of the domain. Figure 15 shows that without any other consideration, training such a network fails. The solution of the failed training is shown in Figure 16. Some other similar architectures have been tried out, which also failed. Among those were putting the ReLU activation function only on the last layer, and non-local ReLU activation functions on the first two layers.

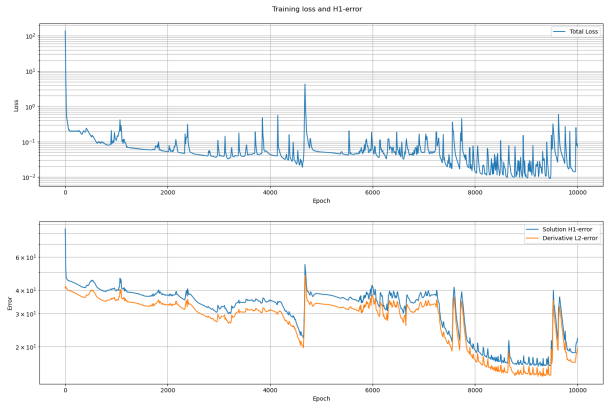


Figure 15: Training loss and solution error of the `VPINN_HelmholtzImpedanceRF` network with 4 hidden layers for wave number $k = 8.0$.

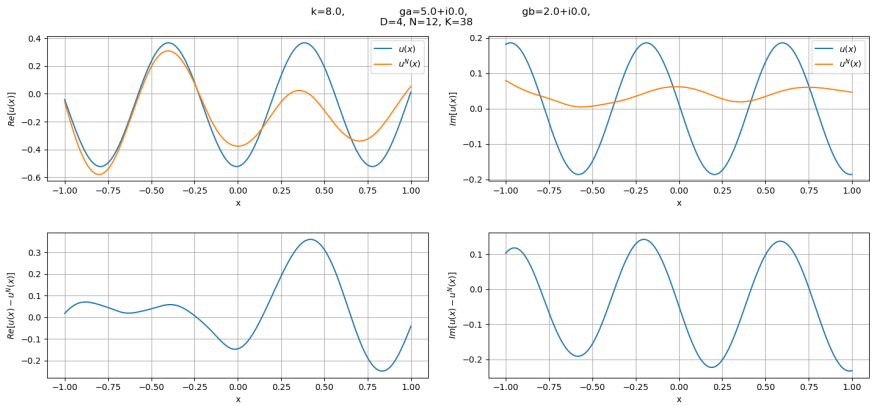


Figure 16: The solution of the `VPINN_HelmholtzImpedanceRF` network with 4 hidden layers for wave number $k = 8.0$.

```
class VPINN_HelmholtzImpedanceRF(VPINN_HelmholtzImpedance):
    def __init__(self, f: Union[Callable, float], k: float, a: float, b: float
   , ga: complex, gb: complex, *, layers=[1, 10, 10, 10, 2], activation=torch
   .tanh, penalty=None, quad_N=80, seed=None, cuda=False):

        # Initialize
        assert len(layers) >= 5, "this network only works for D > 2."
```

```
        super().__init__(f, k, a, b, ga, gb,
                layers=layers, activation=activation,
                penalty=penalty, quad_N=quad_N, seed=seed, cuda=cuda)
        self.stepact = lambda x: torch.heaviside(-(x-1.), torch.zeros(1)) *
    torch.relu(x)

    def forward(self, x):
        for i, f in zip(range(self.length), self.lins):
            if i == len(self.lins) - 1:
            # Last layer
                x = f(x)
            elif i == 0:
            # First hidden layer
                x = self.stepact(f(x))
            else:
            # other hidden layers
                x = self.activation(f(x))
        return x
```

Listing 1: Code snippet of the `VPINN_HelmholtzImpedanceRF` class which inherits from the main `VPINN_HelmholtzImpedance` class.

B Appendix: Evolution of the Solution

Figure 17 illustrates the evolution of the solution of a network architecture that converges to the solution in 30000 epochs. We can see that from the initial random initialization, the network is able to capture the main features of the solution in 2000 epochs, and converges to the final solution in the following 28000 epochs. An interesting observation is that there is a bias from left to right which could also be captured in the initial result of the network (epoch 0). We can see that the solution converges faster in the ranges closer to the left boundary.

The same bias to the left boundary could also be observed in Figure 18 which illustrates the evolution of solution of the same network with least-squares initialization in the initial 2000 epochs. In Figure 13c, we can see that the solution error of this network does not improve much with least-square initialization after around epoch 2000. Another observation is that the most of the error comes from the imaginary part of the solution, which is not the case with random initialization.

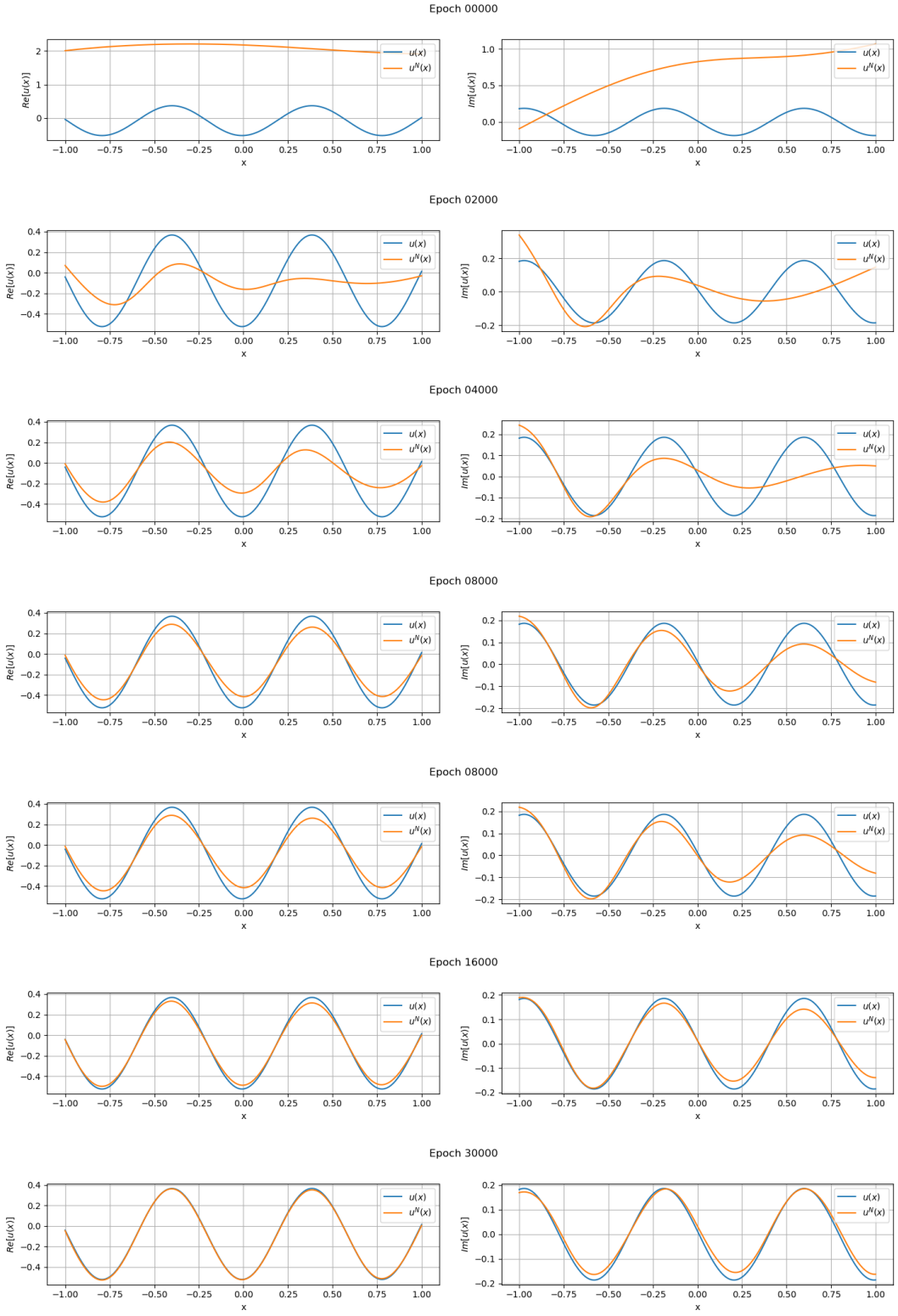


Figure 17: The evolution of the solution during the training of a shallow network with 20 nodes and 20 test functions for $k = 8.0$.

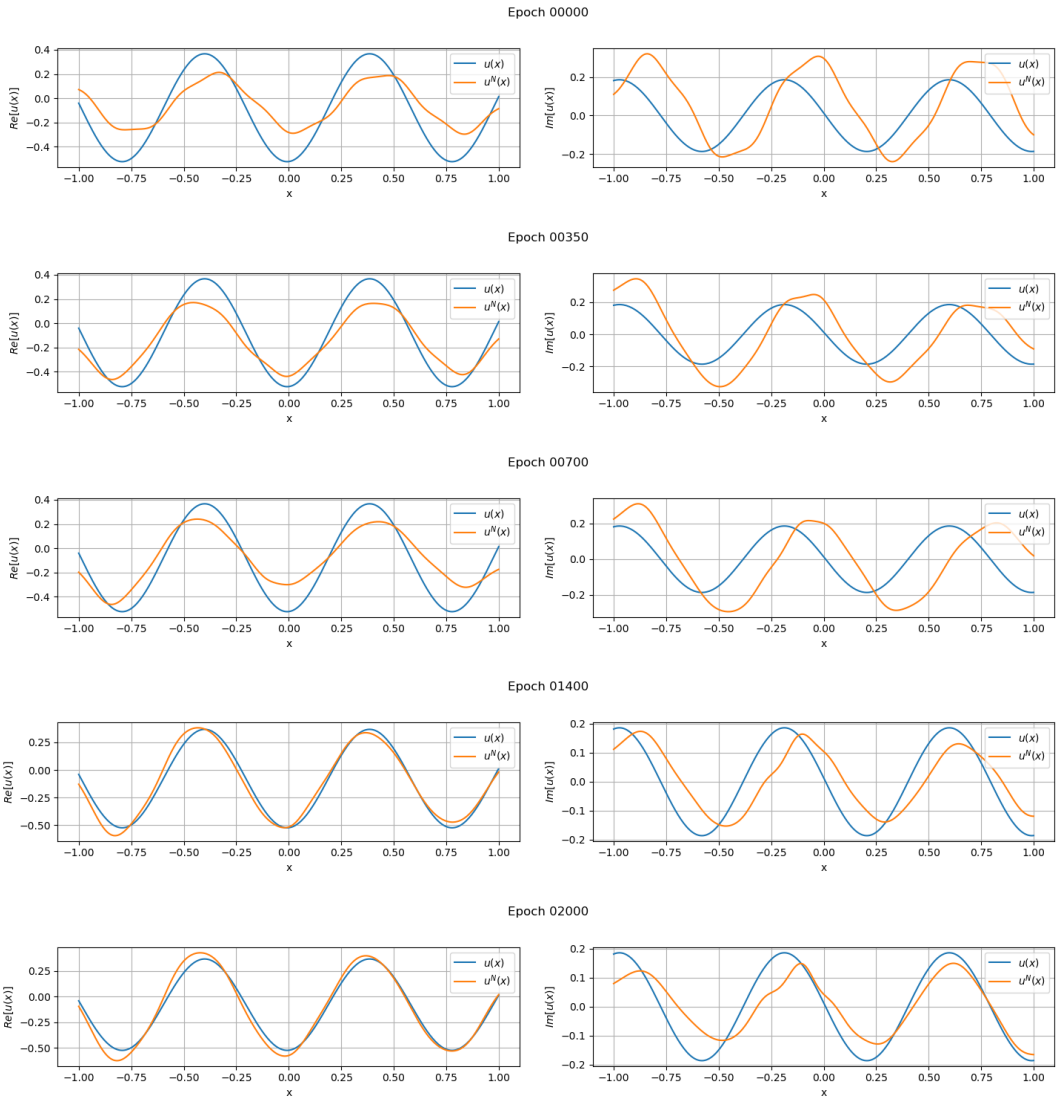


Figure 18: The evolution of the solution during the training of a shallow network with 20 nodes and 20 test functions with least-squares initialization (constant weights) for $k = 8.0$.