CSMA 2024
16ème Colloque National en Calcul des Structures
13-17 Mai 2024, Presqu'île de Giens (Var)

Error Reduction in Neural Network Approximations of Boundary-Value Problems by a Multi-level Approach

Z. Aldirany[1], R. Cottereau[2], M. Laforest[1], S. Prudhomme[1]

[1] *Département de mathématiques et de génie industriel, Polytechnique Montréal, {ziad.aldirany,marc.laforest,serge.prudhomme}@polytml.ca*
[2] *Aix-Marseille Université, CNRS, Centrale Marseille, LMA UMR 7031, cottereau@lma.cnrs-mrs.fr*

Résumé — A new methodology to control the error in approximations of solutions to boundary-value problems obtained with deep learning methods is presented here. The main idea consists in computing an initial approximation to the problem using a simple neural network and in estimating, in an iterative manner, a correction by solving the problem for the residual error with a new network of increasing complexity. This sequential reduction of the residual of the partial differential equation allows one to decrease the solution error, which, in some cases, can be reduced to machine precision.
Mots clés — Neural networks, Partial differential equations, Physics-informed neural networks, Numerical error, Convergence.

1 Introduction

In recent years, the solution of partial differential equations using deep learning [1, 2, 3] has gained popularity and is emerging as an alternative to classical discretization methods, such as the finite element or the finite volume methods. Deep learning techniques can be used to either solve a single initial boundary-value problem [4, 5, 6] or approximate the operator associated with a partial differential equation [7, 8, 9, 10]. The primary advantages of deep learning approaches lie in their ability to provide meshless methods, and hence address the curse of dimensionality, and in the universality of their implementation for various initial and boundary-value problems. However, one of the main obstacles remains their inability to consistently reduce the relative error in the computed solution. Although the universal approximation theorem [11, 12] guarantees that a single hidden layer network with a sufficient width should be able to approximate smooth functions to a specified precision, one often observes in practice that the convergence with respect to the number of iterations reaches a plateau, even if the size of the network is increased. This is primarily due to the use of gradient-based optimization methods, e.g. Adam [13], for which the solution may get trapped in local minima. These optimization methods applied to classical neural network architectures, e.g. feedforward neural networks [14], do indeed experience difficulties in controlling the large range of scales inherent to a solution, even with some fine-tuning of the hyper-parameters, such as the learning rate or the size of the network. In contrast, this is one of the main advantages of classical methods over deep learning methods, in the sense that they feature well-defined techniques to consistently reduce the error, using for instance mesh refinement [15, 16] or multigrid structures [17].

We introduce in this work a novel approach based on the notion of multi-level neural networks, which are designed to consistently reduce the residual associated with a partial differential equation, and hence, the errors in the numerical solution. The approach is versatile and can be applied to various neural network methods that have been developed for the solution of boundary-value problems [5, 6], but we have chosen, for the sake of simplicity, to describe the method on the particular case of physics-informed neural networks (PINNs) [4]. Once an approximate solution to a linear boundary-value problem has been computed with the classical PINNs, the method then consists in finding a correction, namely, estimating the solution error, by minimizing the residual using a new network of increasing complexity. The process can subsequently be repeated using additional networks to minimize the resulting residuals, hence allowing one to reduce the error to a desired precision.

The development of the proposed method is based on two key observations. First, each level of the correction process introduces higher frequencies in the solution error, as already discussed in [18] and highlighted again in the numerical examples. This is the reason why the sequence of neural networks should be of increasing complexity. Moreover, a key ingredient will be to use the Fourier feature mapping approach [19] to accurately approximate the functions featuring high frequencies. Second, the size of the error, equivalently of the residual, becomes at each level increasingly smaller. Unfortunately, feedforward neural networks employing standard parameter initialization, e.g. Xavier initialization [20] in our case, are tailored to approximate functions whose magnitudes are close to unity. We thus introduce a normalization of the solution error at each level based on the Extreme Learning Method [21], which also contributes to the success of the multi-level neural networks.

2 Preliminaries : Neural Networks and PINNs

Neural networks have been extensively studied in recent years for solving partial differential equations [1, 4]. A neural network can be viewed as a mapping between an input and an output by means of a composition of linear and nonlinear functions with adjustable weights and biases. Training a neural network consists in optimizing the weights and biases by minimizing some measure of the error between the output of the network and corresponding target values obtained from a given training dataset. As a predictive model, the trained network is then expected to provide accurate approximations of the output when considering a wider set of inputs. Several neural network architectures, e.g. convolutional neural networks (CNNs) [22] or feedforward neural networks (FNNs) [14], are adapted to specific classes of problems.

We shall consider here FNNs featuring n hidden layers, each layer having a width N_i, $i = 1,\dots,n$, an input layer of width N_0, and an output layer of width N_{n+1}. Denoting the activation function by σ, the neural network with input $\boldsymbol{z}_0 \in \mathbb{R}^{N_0}$ and output $\boldsymbol{z}_{n+1} \in \mathbb{R}^{N_{n+1}}$ is defined as

$$\begin{aligned} &\text{Input layer :} && \boldsymbol{z}_0, \\ &\text{Hidden layers :} && \boldsymbol{z}_i = \sigma(W_i \boldsymbol{z}_{i-1} + \boldsymbol{b}_i), \quad i = 1,\cdots,n, \\ &\text{Output layer :} && \boldsymbol{z}_{n+1} = W_{n+1}\boldsymbol{z}_n + \boldsymbol{b}_{n+1}, \end{aligned} \tag{1}$$

where W_i is the *weights* matrix of size $N_i \times N_{i-1}$ and $\boldsymbol{b}_i$ is the *biases* vector of size N_i. To simplify the notation, we combine the weights and biases of the neural network into a single parameter denoted by θ. The neural network (1) generates a finite-dimensional space of dimension $N_\theta = \sum_{i=1}^{n+1} N_i(N_{i-1}+1)$. To keep things simple, throughout this work we shall use the tanh activation function and the associated Xavier initialization scheme [20] to initialize the weights and biases.

We briefly review the PINNs approach to solving partial differential equations, as described in [4]. Let Ω be an open bounded domain in $\mathbb{R}^d$, $d = 1, 2$, or 3, with boundary $\partial\Omega$. For two Banach spaces U and V of functions over Ω, we assume a linear differential operator $A : U \to V$. Our goal is to find the solution $u \in U$ that satisfies, for a given $f \in V$, the partial differential equation cast here in its residual form :

$$R\big(\boldsymbol{x}, u(\boldsymbol{x})\big) := f(\boldsymbol{x}) - Au(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in \Omega, \tag{2}$$

and the following boundary conditions :

$$B\big(\boldsymbol{x}, u(\boldsymbol{x})\big) = 0, \quad \forall \boldsymbol{x} \in \partial\Omega. \tag{3}$$

For the sake of simplicity in the presentation, but without loss of generality, we consider here only the case of homogeneous Dirichlet boundary conditions, such that the residual B is given by

$$B\big(\boldsymbol{x}, u(\boldsymbol{x})\big) := u(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \partial\Omega. \tag{4}$$

The primary objective in PINNs is to use a neural network with parameters θ to find an approximation $\tilde{u}_\theta(\boldsymbol{x})$ of the solution $u(\boldsymbol{x})$ to problem (2)-(3). For the sake of simplicity in the notation, we shall omit in the rest of the paper the subscript θ when referring to the approximate solutions $\tilde{u}_\theta$, and thus simply write $\tilde{u}(\boldsymbol{x})$. The training, i.e. the identification of the parameters θ of the neural network, is performed

by minimizing a loss function, defined here as a combination of the residual associated with the partial differential equation and that associated with the boundary condition in terms of the L^2 norm :

$$\mathcal{L}(\theta) := w_r \int_\Omega R\big(\boldsymbol{x}, \tilde{u}(\boldsymbol{x})\big)^2 dx + w_{bc} \int_{\partial\Omega} B\big(\boldsymbol{x}, \tilde{u}(\boldsymbol{x})\big)^2 dx, \tag{5}$$

where w_r and w_{bc} are penalty parameters. In other words, by minimizing the loss function (5) one obtains a weak solution $\tilde{u}$ that weakly satisfies the boundary condition.

Alternatively, the homogeneous Dirichlet boundary condition could be strongly imposed, as done in [23], by multiplying the output of the neural network by a function $g(\boldsymbol{x})$ that vanishes on the boundary. For instance, if $\Omega = (0,\ell) \in \mathbb{R}$, one could choose $g(x) = x(\ell - x)$. The trial functions $\tilde{u}$ would then be constructed, using the feedforward neural network (1), as follows :

$$\begin{aligned}
&\text{Input layer :} && \boldsymbol{z}_0 = \boldsymbol{x}, \\
&\text{Hidden layers :} && \boldsymbol{z}_i = \sigma(W_i \boldsymbol{z}_{i-1} + \boldsymbol{b}_i), \quad i = 1, \dots, n, \\
&\text{Output layer :} && z_{n+1} = W_{n+1} \boldsymbol{z}_n + \boldsymbol{b}_{n+1}, \\
&\text{Trial function :} && \tilde{u} = g(\boldsymbol{x}) z_{n+1}.
\end{aligned} \tag{6}$$

where the input and output layers have a width $N_0 = d$ and $N_{n+1} = 1$, respectively. The dimension of the finite-dimensional space of functions generated by the neural network (6) is now given by $N_\theta = \sum_{i=1}^{n+1} N_i(N_{i-1}+1) = N_1(d+1) + \sum_{i=2}^{n} N_i(N_{i-1}+1) + (N_n+1)$.

For the rest of this work, the boundary conditions will be strongly imposed, so that the loss function will henceforth be

$$\mathcal{L}(\theta) = \int_\Omega R\big(\boldsymbol{x}, \tilde{u}(\boldsymbol{x})\big)^2 dx. \tag{7}$$

The problem that one solves by PINNs can thus be formulated as :

$$\min_{\theta \in \mathbb{R}^{N_\theta}} \mathcal{L}(\theta) = \min_{\theta \in \mathbb{R}^{N_\theta}} \int_\Omega R\big(\boldsymbol{x}, \tilde{u}(\boldsymbol{x})\big)^2 dx. \tag{8}$$

One major issue that one faces when using PINNs is that it is very difficult, even impossible, to effectively reduce the L^2 or H^1 error in the solutions to machine precision. The main reason, from our own experience, is that the solution process may get trapped in some local minima, without being able to converge to the global minimum, when using non-convex optimization algorithms. We study the performance of Adam solver and L-BFGS solver, when applied to a simple one-dimensional Poisson problem. This numerical example will also serve later as a model problem for further verifications of the underlying principles in our approach.

Example 1 *Given a function $f(x)$, the problem consists in finding $u = u(x)$, for all $x \in [0,1]$, that satisfies*

$$\begin{aligned}
-\partial_{xx} u(x) &= f(x), \qquad \forall x \in (0,1), \\
u(0) &= 0, \\
u(1) &= 0.
\end{aligned} \tag{9}$$

For the purpose of the study, the source term f is chosen such that the exact solution to the problem is given as

$$u(x) = e^{\sin(k\pi x)} + x^3 - x - 1, \tag{10}$$

where k is a given integer. We take $k = 2$ in this example.

We consider here a network made of only one hidden layer of a width of 20, i.e. $n = 1$ and $N_1 = 20$. Moreover, $N_0 = N_2 = 1$. The learning rates for the Adam optimizer and L-BFGS are set to 10^{-2} and unity, respectively. In the first experiment, the network is trained for 10,000 iterations using Adam. In the second experiment, it is trained with Adam for 4,000 iterations followed by 100 iterations of L-BFGS. Figure 1 compares the evolution of the loss function with respect to the number of iterations for these two scenarios. In the first case, we observe that the loss function laboriously reaches a value around 10^{-2} after 10,000 iterations. The loss function further decreases in the second case but still plateaus around 5×10^{-5} after about 30 iterations of L-BFGS. Note that the scale along the x-axis in the figure on the right has been adjusted in order to account for the large discrepancy in the number of iterations used with Adam and L-BFGS.

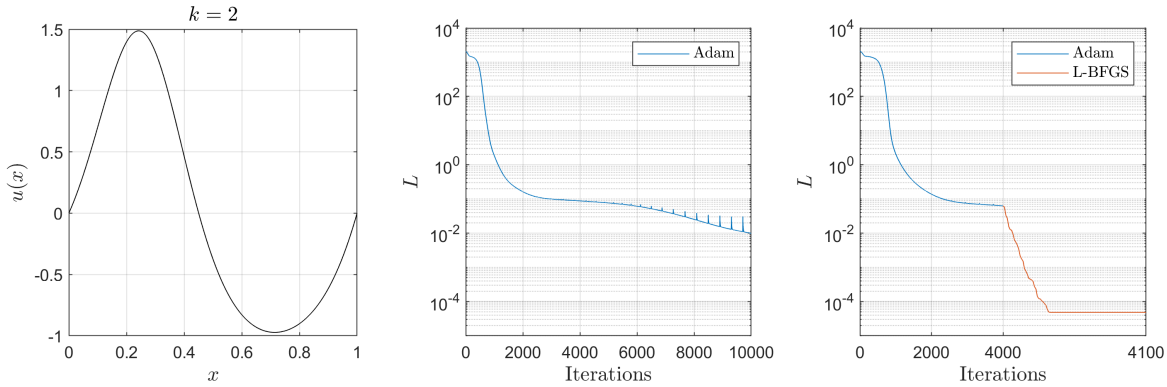

FIGURE 1 – Results from Example 1. (Left) Exact solution with $k = 2$. (Middle) Evolution of the loss function using the Adam optimizer only. (Right) Evolution of the loss function using the Adam optimizer and L-BFGS.

3 Multi-level neural networks

In this section, we describe the multi-level neural networks, whose main objective is to improve the accuracy of the solutions obtained by PINNs. Supposing that an approximation $\tilde{u}$ of the solution u to Problem (2)-(3) has been computed, the error in $\tilde{u}$ is defined as $e(\boldsymbol{x}) = u(\boldsymbol{x}) - \tilde{u}(\boldsymbol{x})$ and satisfies :

$$\begin{aligned} R(\boldsymbol{x}, u(\boldsymbol{x})) = f(\boldsymbol{x}) - Au(\boldsymbol{x}) = f(\boldsymbol{x}) - A\tilde{u}(\boldsymbol{x}) - Ae(\boldsymbol{x}) = R(\boldsymbol{x}, \tilde{u}(\boldsymbol{x})) - Ae(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in \Omega, \\ B(\boldsymbol{x}, u(\boldsymbol{x})) = B(\boldsymbol{x}, \tilde{u}(\boldsymbol{x})) + B(\boldsymbol{x}, e(\boldsymbol{x})) = B(\boldsymbol{x}, e(\boldsymbol{x})) = 0, \quad \forall \boldsymbol{x} \in \partial\Omega, \end{aligned}$$

where we have used the fact that A and B are linear operators and $\tilde{u}$ strongly verifies the boundary condition. In other words, the error function $e(x)$ satisfies the new problem in the residual form :

$$\tilde{R}(\boldsymbol{x}, e(\boldsymbol{x})) = R(\boldsymbol{x}, \tilde{u}(\boldsymbol{x})) - Ae(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in \Omega, \tag{11}$$

$$B(\boldsymbol{x}, e(\boldsymbol{x})) = 0, \quad \forall \boldsymbol{x} \in \partial\Omega. \tag{12}$$

We notice that the above problem for the error has exactly the same structure as the original problem, with maybe two exceptions : 1) the source term $R(\boldsymbol{x}, \tilde{u}(\boldsymbol{x}))$ in the error equation may be small, 2) the error $e(\boldsymbol{x})$ may be prone to higher frequency components than in $\tilde{u}$. Our earlier observations suggest we find an approximation $\tilde{e}$ of the error using the PINNs approach after normalizing the source term by a scaling parameter μ, in a way that scales the error to a unit maximum amplitude. The new problem becomes :

$$\tilde{R}(\boldsymbol{x}, e(\boldsymbol{x})) = \mu R(\boldsymbol{x}, \tilde{u}(\boldsymbol{x})) - Ae(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in \Omega, \tag{13}$$

$$B(\boldsymbol{x}, e(\boldsymbol{x})) = 0, \quad \forall \boldsymbol{x} \in \partial\Omega. \tag{14}$$

The dimension of the new neural network to approximate e should be larger than that used to find $\tilde{u}$, due to the presence of higher frequency modes in e. In particular, the number of wave numbers M in the Fourier feature mapping should be increased. The idea is to some extent akin to a posteriori error estimation techniques developed for Finite Element methods. Finally, one should expect that the optimization algorithm should once again reach a plateau after a certain number of iterations and that the process should be repeated to estimate a new correction to the error e.

We thus propose an iterative procedure, referred here to as the “multi-level neural network method", in order to improve the accuracy of the solutions when using PINNs (or any other neural network procedure based on residual reduction). We start by modifying the notation due to the iterative nature of the process. As mentioned in the previous section, the source term f may need to be normalized by a scaling parameter μ_0, so that we reconsider the initial solution u_0 satisfying a problem in the form :

$$R_0(\boldsymbol{x}, u_0(\boldsymbol{x})) = \mu_0 f(\boldsymbol{x}) - Au_0(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in \Omega, \tag{15}$$

$$B(\boldsymbol{x}, u_0(\boldsymbol{x})) = 0, \quad \forall \boldsymbol{x} \in \partial\Omega. \tag{16}$$

The above problem can then be approximated using a neural network to obtain an approximation $\tilde{u}_0$ of u_0. Hence, the first approximation $\tilde{u}$ to u reads after scaling $\tilde{u}_0$ with μ_0 :

$$\tilde{u}(\boldsymbol{x}) = \frac{1}{\mu_0}\tilde{u}_0(\boldsymbol{x}). \tag{17}$$

We would like now to estimate the error in $\tilde{u}$. However, we find it easier to work in terms of $\tilde{u}_0$. Therefore, we look for a new correction u_1 that solves the problem :

$$R_1(\boldsymbol{x}, u_1(\boldsymbol{x})) = \mu_1 R_0(\boldsymbol{x}, \tilde{u}_0(\boldsymbol{x})) - Au_1(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in \Omega, \tag{18}$$

$$B(\boldsymbol{x}, u_1(\boldsymbol{x})) = 0, \quad \forall \boldsymbol{x} \in \partial\Omega. \tag{19}$$

Once again, one can compute an approximation $\tilde{u}_1$ of u_1 using PINNs. Since $\tilde{u}_1$ can be viewed as the normalized correction to the error in $\tilde{u}_0(\boldsymbol{x})$, the new approximation to u is now given by :

$$\tilde{u}(\boldsymbol{x}) = \frac{1}{\mu_0}\tilde{u}_0(\boldsymbol{x}) + \frac{1}{\mu_0\mu_1}\tilde{u}_1(\boldsymbol{x}). \tag{20}$$

The process can be repeated L times to find corrections u_i at each level $i = 1, \ldots, L$ given the prior approximations $\tilde{u}_0$, $\tilde{u}_1$, ..., $\tilde{u}_{i-1}$. Each new correction u_i then satisfies the boundary-value problem :

$$R_i(\boldsymbol{x}, u_i(\boldsymbol{x})) = \mu_i R_{i-1}(\boldsymbol{x}, \tilde{u}_{i-1}(\boldsymbol{x})) - Au_i(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in \Omega, \tag{21}$$

$$B(\boldsymbol{x}, u_i(\boldsymbol{x})) = 0, \quad \forall \boldsymbol{x} \in \partial\Omega. \tag{22}$$

After finding an approximation $\tilde{u}_i$ to each of those problems up to level i, one can obtain a new approximation $\tilde{u}$ of u such that :

$$\tilde{u}(\boldsymbol{x}) = \frac{1}{\mu_0}\tilde{u}_0(\boldsymbol{x}) + \frac{1}{\mu_0\mu_1}\tilde{u}_1(\boldsymbol{x}) + \ldots + \frac{1}{\mu_0\mu_1\ldots\mu_i}\tilde{u}_i(\boldsymbol{x}). \tag{23}$$

Once the approximations $\tilde{u}_i$ have been found at all levels $i = 0, \ldots, L$, the final approximation at the end of the process would then be given by :

$$\tilde{u}(\boldsymbol{x}) = \sum_{i=0}^{L} \frac{1}{\Pi_{j=0}^{i}\mu_j}\tilde{u}_i(\boldsymbol{x}). \tag{24}$$

Using PINNs, the neural network approximation $\tilde{u}_i$ (which implicitly depends on the network parameters θ) for each error correction will be obtained by solving the following minimization problem :

$$\min_{\theta \in \mathbb{R}^{N_{\theta,i}}} \mathcal{L}_i(\theta) = \min_{\theta \in \mathbb{R}^{N_{\theta,i}}} \int_\Omega R_i\big(\boldsymbol{x}, \tilde{u}_i(\boldsymbol{x})\big)^2 dx, \tag{25}$$

where $N_{\theta,i}$ denotes the dimension of the function space generated by the neural network used at level i. We recall that the boundary conditions are strongly imposed and, hence, do not appear in the loss functions $\mathcal{L}_i(\theta)$. Since each correction $\tilde{u}_i$ is expected to have higher frequency contents, the size $N_{\theta,i}$ of the networks should be increased at each level. Moreover, the number of iterations used in the optimization algorithms Adam and L-BFGS will be increased as well, since more iterations are usually needed to approximate higher frequency functions.

4 Numerical example

To illustrate the proposed approach, we consider the one-dimensional numerical example presented in Example 1. We solve Problem (9) with $k = 2$ whose exact solution is given by (10). We consider three levels of the multi-level neural networks, i.e. $L = 3$, in addition to the initial solve, so that the approximation $\tilde{u}$ will be obtained using four sequential neural networks. We choose networks with a single hidden layer of width N_1 given by $\{10, 20, 40, 20\}$. The networks are first trained with 4,000 iterations of Adam followed by $\{200, 400, 600, 0\}$ iterations of L-BFGS. The mappings of the input and boundary conditions are chosen with $M = \{1, 3, 5, 1\}$ wave numbers. In this example, the scaling

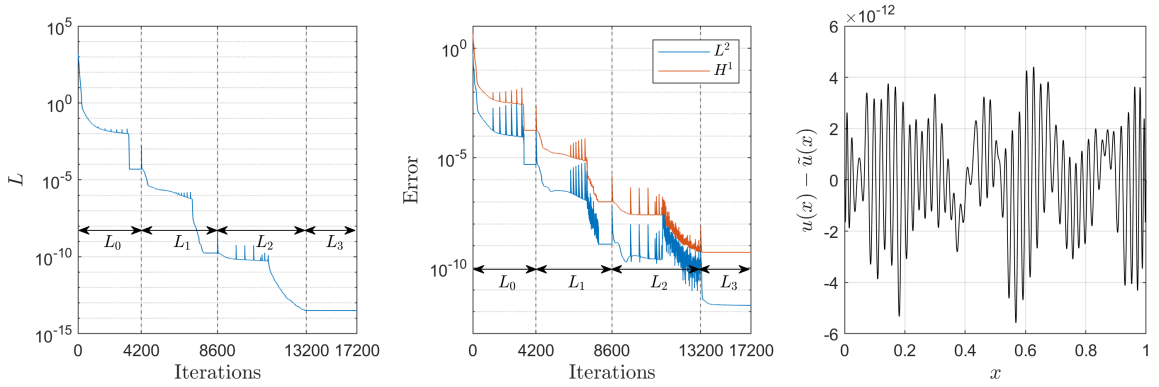

FIGURE 2 – (Left) Evolution of the loss function. (Middle) Evolution of the error $e(x) = u(x) - \tilde{u}(x)$ measured in the L^2 and H^1 norms. (Right) Pointwise error after three error corrections. The regions shown by L_i, $i = 0, \ldots, 3$, in the first two graphs indicate the region in which the neural network at level i is trained.

parameter μ_i, $i = 0, \ldots, 3$, for $\tilde{u}_0$ and the three corrections $\tilde{u}_i$, are chosen here as $\mu_i = \{1, 10^3, 10^3, 10^2\}$. In the next section, we will present a simple approach to evaluate these normalization factors. We note that the last network has been designed to approximate functions with a low-frequency content. This choice will be motivated below.

We present in Figure 2 the evolution of the loss function and of the errors in the L^2 and H^1 norms with respect to the number of optimization iterations, along with the pointwise error at the end of the training. We first observe that each error correction allows one to converge closer to the exact solution. More precisely, we gain almost seven orders of magnitude in the L^2 error thanks to the introduced corrections. Indeed, after three corrections, the maximum pointwise error is around 6×10^{-12}, which is much smaller than the error we obtained with $\tilde{u}_0$ alone. To better explain our choice of the number M of wave numbers at each level, we show in Figure 3 the computed corrections $\tilde{u}_i$. We observe that each correction approximates higher frequency functions than the previous one, except $\tilde{u}_3$. In fact, once we start approximating the high-frequency errors, it becomes harder to capture the low-frequencies with larger amplitudes. Here, we see that the loss function eventually decreases during the training of $\tilde{u}_2$, but that the L^2 error has the tendency to oscillate while slightly decreasing. It turns out that this behavior can be attributed to the choice of the loss function $\mathcal{L}_2$, in which the higher frequencies are penalized more than the lower ones. In other words, we have specifically designed the last network to approximate only low-frequency functions and be trained using Adam only. Thanks to this architecture, the L^2 error significantly decreases during the training of $\tilde{u}_3$, without a noticeable decrease in the loss function. As a remark, longer training for $\tilde{u}_2$ would correct the lower frequencies while correcting high frequencies with smaller amplitudes, but it is in our opinion more efficient, with respect to the number of iterations, to simply introduce a new network targeting the low frequencies.

5 Conclusions

We have presented in this paper a novel approach to control and reduce errors in deep learning approximations of solutions to linear boundary-value problems. The method has been referred to as the multi-level neural network method in the sense that, at each level of the process, one uses a new neural network, possibly of different sizes, to compute a correction corresponding to the error in the previous approximation. Each successive correction aims at reducing the global error in the resulting approximation of the solution. Although the conceptual idea seems straightforward, the efficiency of the approach relies on two key ingredients : normalization of the residual before computing a new correction based on the Extreme Learning Method and use of Fourier feature mapping to the input data and the functions used to strongly impose the Dirichlet boundary conditions. We believe that the multi-level neural network method is a versatile approach and could be applied to other deep learning techniques designed for solving boundary-value problems. The numerical results presented here illustrate the fact that the method can provide highly accurate approximations to the solutions and, in some cases, allows

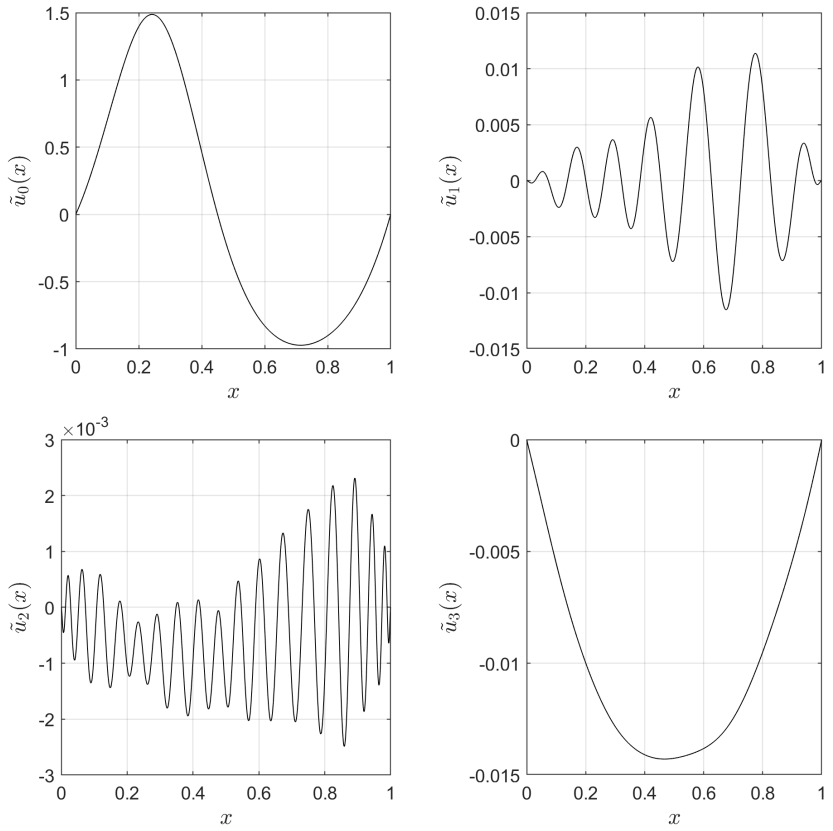

FIGURE 3 – Approximation $\tilde{u}_0(x)$ and corrections $\tilde{u}_i(x)$, $i = 1,2,3$.

one to reduce the numerical errors in the L^2 and H^1 norms down to the machine precision.

Références

[1] Justin Sirignano and Konstantinos Spiliopoulos. DGM : A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375 :1339–1364, 2018.

[2] Alex Bihlo and Roman O Popovych. Physics-informed neural networks for the shallow-water equations on the sphere. *Journal of Computational Physics*, 456 :111024, 2022.

[3] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. NSFnets (Navier-Stokes flow nets) : Physics-informed neural networks for the incompressible navier-stokes equations. *Journal of Computational Physics*, 426 :109951, 2021.

[4] Maziar Raissi, Paris Perdikaris, and George 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.

[5] E Weinan and Bing Yu. The deep Ritz method : a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1) :1–12, 2018.

[6] Yaohua Zang, Gang Bao, Xiaojing Ye, and Haomin Zhou. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics*, 411 :109409, 2020.

[7] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3) :218–229, 2021.

[8] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv :2010.08895*, October 2020.

[9] Ziad Aldirany, Régis Cottereau, Marc Laforest, and Serge Prudhomme. Operator approximation of the wave equation based on deep learning of Green's function. *arXiv :2307.13902*, July 2023.

[10] Dhruv Patel, Deep Ray, Michael RA Abdelmalik, Thomas JR Hughes, and Assad A Oberai. Variationally mimetic operator networks. *arXiv :2209.12871*, September 2022.

[11] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4) :303–314, 1989.

[12] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2) :251–257, 1991.

[13] Diederik P. Kingma and Jimmy Ba. ADAM : A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553) :436–444, 2015.

[15] W. Bangerth and R. Rannacher. *Adaptive finite element methods for differential equations*. Birkhäuser, Basel, 2003.

[16] R. Sevilla, S. Perotto, and K. Morgan. *Mesh Generation and Adaptation : Cutting-Edge Techniques*. SEMA SIMAI Springer Series. Springer International Publishing, 2022.

[17] Wolfgang Hackbusch. *Multi-Grid Methods and Applications*. Springer, Berlin, 1985.

[18] Mark Ainsworth and Justin Dong. Galerkin neural networks : A framework for approximating variational equations with error control. *SIAM Journal on Scientific Computing*, 43(4) :A2474–A2501, 2021.

[19] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33 :7537–7547, 2020.

[20] 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*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[21] Guang-Bin Huang, Dian Hui Wang, and Yuan Lan. Extreme learning machines : a survey. *International journal of machine learning and cybernetics*, 2 :107–122, 2011.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6) :84–90, 2017.

[23] Kevin Stanley McFall and James Robert Mahan. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. *IEEE Transactions on Neural Networks*, 20(8) :1221–1233, 2009.