Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 387 (2021) 114188
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Extreme learning machine collocation for the numerical solution of elliptic PDEs with sharp gradients

Francesco Calabrò[a,b,*], Gianluca Fabiani[b], Constantinos Siettos[a,b]

[a] *Dipartimento di Matematica e Applicazioni "Renato Caccioppoli", Università degli Studi di Napoli Federico II, Italy*
[b] *Scuola Superiore Meridionale di Napoli, Università degli Studi di Napoli Federico II, Italy*

Received 25 January 2021; received in revised form 7 September 2021; accepted 11 September 2021
Available online 5 October 2021

Abstract

We address a new numerical method based on machine learning and in particular based on the concept of the so-called Extreme Learning Machines, to approximate the solution of linear elliptic partial differential equations with collocation. We show that a feedforward neural network with a single hidden layer and sigmoidal transfer functions and fixed, random, internal weights and biases can be used to compute accurately enough a collocated solution for such problems. We discuss how one can set the range of values for both the weights between the input and hidden layer and the biases of the hidden layer in order to obtain a good underlying approximating subspace, and we explore the required number of collocation points. We demonstrate the efficiency of the proposed method with several one-dimensional diffusion–advection–reaction benchmark problems that exhibit steep behaviors, such as boundary layers. We point out that there is no need of iterative training of the network, as the proposed numerical approach results to a linear problem that can be easily solved using least-squares and regularization. Numerical results show that the proposed machine learning method achieves a good numerical accuracy, outperforming central Finite Differences, thus bypassing the time-consuming training phase of other machine learning approaches.
© 2021 Elsevier B.V. All rights reserved.

Keywords: Partial differential equations; Collocation methods; Artificial Neural Networks; Extreme Learning Machine; Boundary layer; Sigmoidal transfer functions

1. Introduction

The use of machine learning for the numerical solution of PDEs goes back to the end of 90s [1,2]. Today, due to both theoretical and technological advances there is a growing interest in developing and implementing new numerical approaches based on machine learning for the solution of Partial Differential Equations (PDEs) (see e.g. [3–19]). The main focus of such techniques is on the numerical solution of some very difficult problems, such as the solution of high-dimensional PDEs, and nonlinear PDEs with complex geometries. Machine learning approaches have proven to be enough accurate and extremely flexible. However, one of their major drawbacks, when compared with conventional numerical methods (such as Finite Differences and Finite Elements) is the computational cost of

* Corresponding author at: Dipartimento di Matematica e Applicazioni "Renato Caccioppoli", Università degli Studi di Napoli Federico II, Italy.
E-mail addresses: calabro@unina.it (F. Calabrò), gianluca.fabiani@unina.it (G. Fabiani), constantinos.siettos@unina.it (C. Siettos).

https://doi.org/10.1016/j.cma.2021.114188
0045-7825/© 2021 Elsevier B.V. All rights reserved.

the training phase which in general is very high. While a good numerical accuracy obtained with Artificial Neural Networks (ANNs) can be theoretically justified by the celebrated theorem of the universal approximation of ANNs, their broad use is still limited in the field of numerical analysis/scientific computing, due to the lack of theoretical justification on their convergence. For example, while it has been proven that the approximation error, with ANNs, vanishes asymptotically, results about explicit error bounds for the approximation errors or about the polynomial reproduction are still limited (see e.g. [20–23]). Among all types of machine learning and in particular ANNs, recently, great attention has been given to the so-called Extreme Learning Machines (ELMs) [24–26]. As discussed in [27], ELMs are built on Rosenblatt's perceptron concept [28], spectral methods, numerical analysis and linear algebra matrix methods but with "essential extensions". It is worthwhile to mention that ELMs are inspired by earlier works addressing ANN with Random Weights (NNRW) [29] and Random Vector Functional Link networks (RVFL) [30]. A thorough discussion about the (subtle) differences between these approaches can be found in [27]. Unlike other machine learning algorithms, ELMs learn in one step, thus bypassing the need to tune in an iterative manner all the weights and biases. ELMs have been used mainly for classification purposes, yet their efficiency and applicability for the numerical solution of differential equations is still underrepresented. Indeed, to the best of our knowledge, [7] is the only study on the subject, where the authors however report a failure of ELMs to deal with sharp gradient problems.

Here, we show how ELMs can be exploited to deal with boundary-value problems, in particular 1D second-order linear elliptic problems that exhibit steep behaviors, such as boundary layers. It is well known, that the numerical solution of such problems using classical methods such as Finite Differences may lead to several numerical instabilities such as spurious oscillations (see e.g. [31,32]) even if the solution is regular. Here, for our illustrations, we focus on linear advection–reaction boundary-value problems with constant coefficients, where the solution is known analytically and it is regular, but the advection-dominated or the reaction-dominated cases may result to boundary layers. For such problems, several approaches (for example artificial diffusion, upwind schemes or mesh adaptivity schemes [33,34]) have been addressed to deal with the emergent steep gradients. Compared to the above techniques, our approach is free from problem-dependent modifications and it is shown to be able to efficiently approximate the steep gradients that arise. Our results reveal that ELMs can serve as a robust numerical approach to obtain accurate solutions of boundary value problems without the need of stabilization techniques. In particular, we pinpoint that the training process of ELMs with one hidden layer reduces to the solution of a under-determined linear system of algebraic equations.

Our findings reveal the ability of ELMs to approximate the solutions efficiently, thus extending and giving a new insight on the application of such types of machine learning methods and in particular of ANNs. What we propose here, is that one does not need to iteratively train an ANN in order to obtain a sufficiently accurate solution. By appropriately fixing the values of both internal weights and biases, one can in principle obtain accurate enough solutions by solving for linear problems once a least-squares/regularization problem to adjust the weights of the connections between the hidden and the output layer of the network. Obviously, such solutions can be taken as initial choices for a iterative process if the problem is non-linear and/or there is a demand for a higher numerical accuracy.

The paper is organized as follows. In Section 2, we introduce the approximation space on the basis of the proposed ELM scheme, and in Section 3, we present the proposed collocation method. The approximation efficiency and convergence property of the scheme is analyzed in Section 4. In Section 5, we present the numerical results obtained with the proposed approach for the solution of several benchmark 1D boundary-layer elliptic problems and compare them with both the exact-analytical and the Finite Difference numerical solutions.

2. The proposed machine learning scheme: the ELM network

In this section, we briefly introduce the proposed ELM scheme, and the main results related to the properties of ELMs as universal approximators. The structure of the proposed scheme is a simple feedforward ANN with a single hidden layer with n nodes (neurons). The first layer is the *input layer*, where the inputs (here, the domain of the PDE, i.e. the collocation points, where the solution is sought) are fed and the last one is the so-called *output layer*, where one gets the computed collocated solution. Every neuron in one layer is connected to every neuron in the next layer, i.e. the structure is fully connected and at each connection is assigned a weight. Finally, to each neuron in the hidden layer is also assigned a *bias* and an *activation (transfer) function* that takes as input the linear weighted sum of its (input) connections (see e.g. [23]).

Here, we have chosen the standard logistic sigmoid function as activation (transfer) function:

$$\sigma(x) = \frac{1}{1+\exp(-x)} \ .$$

The output function is the linear activation function, so that the output of the network turns out to be a linear weighted sum of the outputs of the n sigmoid functions. In the network, one has to assign values to the biases of the neurons of the hidden layer, denoted by the vector $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_n) \in \mathbb{R}^n$, the (internal) weights between the input layer and the hidden layer, denoted by the vector $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{R}^n$, and the (external) weights of the connections between the hidden and the output layer, denoted by the vector $\boldsymbol{w} = (w_1, w_2, \ldots, w_n) \in \mathbb{R}^n$. Therefore, this type of ANN describes a map $G : \mathbb{R} \to \mathbb{R}$ that can be written as:

$$G(x; \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{w}) = \sum_{i=1}^{n} w_i \sigma(\alpha_i x + \beta_i). \tag{2.1}$$

As it is well-known, this kind of ANN is a universal approximator for any L^1 function as stated, for example in [35–37]. For the completeness of the presentation, we state this fundamental theorem below.

Theorem 2.1. *Let G be a ANN function as in (2.1). For any function $f \in L^1([a,b])$ and for all $\varepsilon > 0$ there exist a choice of $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{w}$ such that*

$$\|G - f\|_{L^1} < \varepsilon.$$

where $\|\cdot\|_{L^1}$ is the usual L^1 norm, i.e. $\|\psi\|_{L^1} = \int_a^b |\psi(x)| dx$.

Similar results apply for continuous functions and for the derivatives of differentiable functions [23,36,38].

These results open the way to the use of ANN for the numerical solution of differential equations. The most common approach when using an ANN is the one that minimizes a cost function with respect to the network's parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{w}$ in an iterative manner (e.g. using back-propagation). This optimization procedure is called "training". This approach can be also exploited for the solution of differential problems. In this case, the optimization process can be facilitated by the analytical derivation of the derivatives as defined by the ANN input–output map (see e.g. [1]). Two main issues arise when training an ANN via such an iterative optimization approach: the initial choice of the degrees of freedom (weights and biases) and the overall computational cost. When dealing with time-dependent problems, usually the above optimization problem is solved by tessellating the time interval where the solution is sought in smaller subintervals, thus solving the optimization problem in each one of the subintervals sequentially. Hence, upon convergence of the optimization problem in a subinterval, one can use as initial guesses of the unknown parameters of the network for the numerical solution in the next subinterval the ones obtained from the previous time subinterval (see e.g. [17,39]); in other problems, where the main difficulties are related to the geometry of the boundary, the initial choice is efficiently chosen as a boundary lift (see e.g. [1,5]). Conversely, in the problems that we consider here, i.e. elliptic PDEs that may lead to an internal layer or a boundary layer problem, there is no such a good indication by the problem itself that may provide "good" initial guesses for the network parameters. Here, we present a procedure that approximates a solution of a linear PDE in the framework of ELMs bypassing the traditional optimization approach. Toward to the above aim, we first show how one can set the internal weights and biases of the hidden layer to get "good" approximating functions.

We start by considering the function G in (2.1) that depends non-linearly on the internal weights and biases. Let us now take the function:

$$\sigma_i(x) = \sigma(\alpha_i x + \beta_i) = \frac{1}{1+\exp(-\alpha_i x - \beta_i)} \ . \tag{2.2}$$

The first and second derivatives of (2.2) with respect to the independent variable x read:

$$\begin{aligned} \sigma_i'(x) =& \alpha_i \frac{\exp(z)}{(1+\exp(z))^2} \\ \sigma_i''(x) =& \alpha_i^2 \frac{\exp(z)(1-\exp(z))}{(1+\exp(z))^3}, \end{aligned} \tag{2.3}$$

where $z = \alpha_i x + \beta_i$. Note that if one takes two functions σ_i and σ_j, where at least one of the parameters is different, i.e. $\alpha_i \neq \alpha_j$ or $\beta_i \neq \beta_j$, then these are linearly independent (see [40]). Finally, the ratio between α_i and β_i gives the location of the inflection point, while the internal weights α_i govern the variation of the amplitude of the *S-shape*:

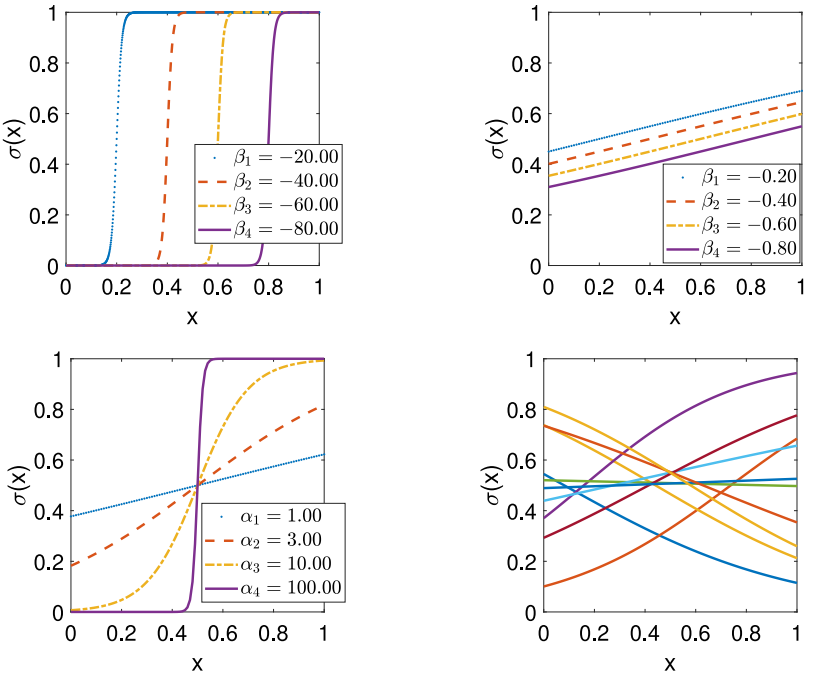


Fig. 1. The functions σ_i of (2.2) with varying parameters. On the top left panel, we have set $\alpha_i = 100$, on the top right, $\alpha_i = 1$. On the bottom left panel we show functions with a fixed center $C_i = 1/2$. On the bottom right panel, a set of 10 functions obtained by varying the coefficients as stated in Section 3 are depicted.

- σ_i has an inflection point at $x = -\dfrac{\beta_i}{\alpha_i}$, that we call the *center* C_i of the sigmoid function;
- σ_i is monotone, $\lim_{x\to-\infty} = 0$; $\lim_{x\to+\infty} = 1$ if α_i is positive, the other way if negative. Moreover, the range where the values are between 0.05 and 0.95 is $\left[C_i - \frac{2.945}{\alpha_i}, C_i + \frac{2.945}{\alpha_i}\right]$. We denote this interval as the *S-shape amplitude* I_S of the function.

The values of α_i, β_i are set randomly, and kept constant throughout the computations. This is the case of the so-called Extreme Learning Machine (ELM) networks [26,41,42]. The only free parameters that one needs to adjust are the output weights, i.e. the weights of the connections between the hidden layer and the output layer. The analysis presented in [25] gives the conditions that are required to ensure the interpolation property, thus the non-singularity of the collocation matrices; we report this result in Section 4.

Outside the domain I_S, the function σ_i is (almost) constant; then, we focus on the values of the parameters α_i, β_i such that the intersection of the S-shape amplitude I_S and the domain of interest is non-null, thus leading to functions σ_i with C_i inside the interval. The choice of α_i is related to the behavior of the derivative of the function. When taking all internal weights $\alpha_i \equiv K$, one obtains Heaviside-like functions if K is big (as used in [43]), or almost-linear functions, if K is small (as discussed in [22,44]). In our case, the use of both kind of functions can help to approximate both steep gradients and global behaviors. In Fig. 1, we plot different cases of such functions: on the top panels, α_i is a constant, on the bottom left panel, C_i is set equal to the mid-point of the interval and finally on the bottom right panel, we plot functions with random values of α_i, β_i. Note, that some functions can have similar shapes: interpolation properties are not changed but the resolution of the interpolation problem may become ill-conditioned.

3. The proposed numerical algorithm

We address a simple algorithm for the numerical solution of linear PDEs with steep gradients with the aid of ELMs with a single hidden layer with n neurons. As discussed above, the main idea is to fix in an appropriate way the values of the internal weights and the biases of the neurons in the hidden layer, in order to construct a

"good" basis of n sigmoid functions σ_i. Then, the values of the external weights are computed by solving a linear least-squares problem with regularization.

To present the approach, we have chosen a steady-state one-dimensional PDE with constant coefficients.

$$\begin{cases} -\mu u''(x) + \gamma u'(x) + \lambda u(x) = f(x), & x \in I := [0,1] \subseteq \mathbb{R} \\ \nu_0 u'(0) + \rho_0 u(0) = g_0\,, \;\; \nu_1 u'(1) + \rho_1 u(1) = g_1, \end{cases} \tag{3.1}$$

where μ, γ and λ are the diffusion, advection and reaction terms, respectively. The domain is fixed to [0, 1] for our convenience. The boundary conditions (BC) are in general of the Robin form; Dirichlet and Neumann boundary conditions are derived by the Robin BC by setting $\nu_i = 0$ or $\rho_i = 0$. The approximate solution $\tilde{u}(x, \boldsymbol{w})$ obtained by the proposed scheme can be written as:

$$\tilde{u}(x; \boldsymbol{w}) = \sum_{i=1}^{n} w_i \sigma_i(x). \tag{3.2}$$

We choose $\boldsymbol{\alpha}$ to be random uniformly distributed in a range that depends on the number of neurons n; if we fix the domain of the differential problem to have unitary length, we propose to set:

$$\boldsymbol{\alpha} := \text{rand}\left(\left[-\frac{n-10}{10} - 4, \frac{n-10}{10} + 4\right]\right). \tag{3.3}$$

In this way, if $n = 10$ we have a maximum weight e.g. 4 and as n grows by 10, then the maximum weight grows by 1. Note that the choice (3.3) is empirically derived and based on the general consideration described in Section 2. The biases $\boldsymbol{\beta}$ are fixed so that the centers $C_i = -\beta_i/\alpha_i$ of the sigmoid functions are located on random uniformly distributed points inside the interval.[1]

Then, the ELM network is collocated in (3.1) on $M-2$ equidistant collocation points $x_j \in (0, 1)$ getting:

$$-\mu \tilde{u}''(x_j) + \gamma \tilde{u}'(x_j) + \lambda \tilde{u}(x_j) = f(x_j) \qquad \text{for } j = 2, \ldots, M-1. \tag{3.4}$$

Analytical derivatives of the sigmoid function σ_i can be computed by (2.3). Then:

$$-\mu \sum_{i=1}^{n} w_i \sigma_i''(x_j) + \gamma \sum_{i=1}^{n} w_i \sigma_i'(x_j) + \lambda \sum_{i=1}^{n} w_i \sigma_i(x_j) = f(x_j) \text{ for } j = 2, \ldots, M-1.$$

Finally, we observe that we can write separately the external weights $\boldsymbol{w}$ in a matrix form as:

$$\mathbb{S}\boldsymbol{w} := (-\mu \mathbb{S}^{(2)} + \gamma \mathbb{S}^{(1)} + \lambda \mathbb{S}^{(0)})\boldsymbol{w} = \boldsymbol{f}, \tag{3.5}$$

where $\mathbb{S}_2$, $\mathbb{S}_1$ and $\mathbb{S}_0$ are matrices of elements:

$$\mathbb{S}^{(2)} = \left(\sigma_i''(x_j)\right)_{i,j}, \;\; \mathbb{S}^{(1)} = \left(\sigma_i'(x_j)\right)_{i,j} \text{ and } \mathbb{S}^{(0)} = \left(\sigma_i(x_j)\right)_{i,j} \qquad i = 1, \ldots, n\,, \;\; j = 2, \ldots, M-1. \tag{3.6}$$

For the boundary conditions defined in (3.1), we augment the system by two equations collocating on the boundary points 0 and 1 as:

$$\nu \sum_{i=1}^{n} w_i \sigma_i'(x_k) + \rho \sum_{i=1}^{n} w_i \sigma_i(x_k) = g(x_k) \qquad k = 0, M\,; \;\; x_1 = 0, x_M = 1. \,.$$

As before, we rewrite the above in a matrix form:

$$\mathbb{B}\boldsymbol{w} := (\nu \mathbb{B}^{(1)} + \rho \mathbb{B}^{(0)})\boldsymbol{w} = \boldsymbol{g}\,, \tag{3.7}$$

where $\mathbb{B}^{(1)}$ and $\mathbb{B}^{(0)}$ are matrices of elements:

$$\mathbb{B}^{(1)} = \left(\sigma_i'(x_k)\right)_{i,k} \text{ and } \mathbb{B}^{(0)} = \left(\sigma_i(x_k)\right)_{i,k} \qquad i = 1, \ldots, n\,, \;\; k = 0, M\,. \tag{3.8}$$

[1] The randomness of the biases β_i is strictly related to the value of internal weights α_i and centers C_i, in order to force functions σ_i to be "active" and "non-trivial" in the considered domain, i.e. the S-shape amplitude I_S is not disjoint to the considered domain.

To this end, considering Eq. (3.5)–(3.8), we have a linear system of M equations and n unknowns, which can be over-determined ($M > n$), under-determined ($M < n$) or square ($M = n$). In the over- and under-determined cases, we consider the solution to be the one obtained in the least-square sense (in the under-determined problem, we obtain the minimum-norm regularized solution). In the next section, we show that the under-determined solutions reach the same accuracy when compared with the square or over-determined cases. Moreover, we observe that the square case can lead to ill-conditioned problems, thus we propose an under-determined collocation. A pseudo-code summarizing the above procedure is given below.

Input: The values of the parameters of the PDE: $\mu, \gamma, \lambda; \nu_i, \rho_i$; the number of neurons n, the number of collocation points M, the function evaluations $[\boldsymbol{f}, \boldsymbol{g}] \in \mathbb{R}^M$.
ELM initialization:
1. Set $\boldsymbol{\alpha} \in \mathbb{R}^n$ randomly according to (3.3) ;
2. Set $\boldsymbol{\beta} \in \mathbb{R}^n$ such that C_i are uniformly distributed, i.e. $\beta_i = -\alpha_i * C_i$;
Numerical Solution (ELM Collocation)
3. Compute the matrix $\mathbb{S}$ according to (3.5)–(3.6) ;
4. Compute the matrix $\mathbb{B}$ according to (3.7)–(3.8) ;
5. Solve the (eventually under/over-determined) linear system $[\mathbb{S}; \mathbb{B}]\boldsymbol{w} = [\boldsymbol{f}; \boldsymbol{g}]$;
Output: The external ELM weights $\boldsymbol{w} \in \mathbb{R}^n$ that provide the collocated solution $\tilde{u}(x)$ in (3.2);

Algorithm 1: Pseudo-code for the proposed ELM collocation method

4. Analysis of the ELM collocation approximation

In this section, we report the results that guarantee convergence of the proposed ELM collocation method. First, based on the Theorem 2.1 presented in [42], we state the following theorem that fits to our proposed framework.

Theorem 4.1. *Let $(x_i, y_i)\,,\ i = 1, \ldots, M$ be a set of points such that $x_i < x_{i+1}$, and take the ELM network with $n < M$ neurons $\tilde{u}(x; \boldsymbol{w})$ in (3.2) such that the internal weights $\boldsymbol{\alpha}$ and the biases $\boldsymbol{\beta}$ are randomly generated independently from the data according to any continuous probability distribution. Then, $\forall \varepsilon > 0$, there exists a choice of $\boldsymbol{w}$ such that $\|(\tilde{u}(x_i; \boldsymbol{w}) - y_i)_i\| < \varepsilon$. Here $\|\cdot\|$ denotes the L^2 Euclidean norm of vectors, the analogous of the Frobenius norm.*

Moreover, if $n = M$ then $\boldsymbol{w}$ can be found such that $\|(\tilde{u}(x_i; \boldsymbol{w}) - y_i)_i\| = 0$.

The above theorem states that if the number of hidden neurons is at least equal to the number of data points, then the interpolation error is zero. Starting from the above theorem, we detail the properties of the ELM network for the approximation of functions. First, we consider the construction of a sequence of nested ELM families: $\mathbb{E}^{(n)} := \{\tilde{u}(x; \boldsymbol{w})\,,\ \boldsymbol{w} \in \mathbb{R}^n\} \subset \mathbb{E}^{(n+1)} := \{\tilde{u}(x; \boldsymbol{w})\,,\ \boldsymbol{w} \in \mathbb{R}^{n+1}\}$, where we add one neuron at a time.

The interpolation property of the above theorem extends to a convergence result, as proved in [45] (see also Theorem 2 in [25]).

Theorem 4.2. *Let $\phi(x)$ be a continuous function. Then, there exist a sequence of ELM network functions $\tilde{u}^{(n)} \in \mathbb{E}^{(n)}$ such that:*

$$\|\phi - \tilde{u}^{(n)}\|_{L^2} \to 0\,,$$

where by $\|\cdot\|_2$ we denote the L^2 norm.

Theorem 4.2 extends the convergence to a general continuous function ϕ in the L^2 sense; thus it states that the finite-dimensional discrete space of ELM networks is consistent in the L^2 setting.

Theorems 4.1 and 4.2 also imply that the collocation matrices $\mathbb{S}^{(0)}$ are of full rank.

We should note that for the problems that we consider here, we could obtain ill-conditioned matrices. In these cases, the solution of the under-determined collocation problem (i.e. when the number of neurons is larger than the number of collocation points), a regularization method can applied.

The collocation solution can now be sought in a variational formulation tested against delta-type functions or, equivalently, a discretized formulation of the problem with C^2 test functions. We follow the classic approach (see

e.g. [46,47]) and write the weak formulation for Eq. (3.1) that seeks a $u(x)$ in the trial space (denoted by U) such that:

$$\int_0^1 \left[-\mu u''(x) + \gamma u'(x) + \lambda u(x)\right] v(x)\, dx = \int_0^1 f(x) v(x)\, dx$$

for all test functions $v \in V$. For simplicity, we suppose that the boundary conditions are imposed on the space U and are satisfied in all the corresponding subspaces.

The general Galerkin method is a weak formulation as above where one looks for a function $\tilde{u}_n \in \mathbb{U}_n$ for which the equations hold true $\forall v_m \in \mathbb{V}_m$ and both $\mathbb{U}_n$ and $\mathbb{V}_m$ are finite-dimensional. In our case, we fix $\mathbb{U}_n = \mathbb{E}^{(n)}$, i.e. the ELM space with n neurons so that, the degrees of freedom of the finite dimensional space are the weights $\boldsymbol{w} \in \mathbb{R}^n$.

Now, consider the test space to be $\mathbb{V}_n = span\{\delta_{x_i,\varepsilon}(x),\ i = 1, \dots, n\}$ defined by a set of n distinct points $\{x_i\}$. We also consider that the function f is approximated by its interpolant in the subspace of ELM functions, i.e. $\tilde{f} \approx f$, $\tilde{f} \in \mathbb{E}^{(n)}$, $\tilde{f}(x_i) = f(x_i)$. Then, the Galerkin problem is given by:

$$\int_0^1 \left[-\mu \tilde{u}''_\varepsilon(x) + \gamma \tilde{u}'_\varepsilon(x) + \lambda \tilde{u}_\varepsilon(x)\right] \delta_{x_i,\varepsilon}(x)\, dx = \int_0^1 \tilde{f}(x) \delta_{x_i,\varepsilon}(x)\, dx \qquad \forall i = 1, \dots, n, \tag{4.1}$$

where the functions δ are defined by:

$$\delta_{x_i,\varepsilon}(x) = \phi(|x - x_i|/\varepsilon),$$
$$\phi(\zeta) = \begin{cases} \dfrac{exp\{1/(\zeta^2 - 1)\}}{\int exp\{1/(\zeta^2 - 1)\}\, d\zeta} & \zeta \in [0, 1) \\ 0 & \zeta \in [1, +\infty) \end{cases}.$$

This Galerkin problem is related to the above collocation problem by the following theorem.

Theorem 4.3. *Let $\tilde{u}_{n,\varepsilon}(x)$ be a solution to the Galerkin problem (4.1). Then, $\lim_{\varepsilon \to 0} \tilde{u}_{n,\varepsilon}(x) = \tilde{u}_n(x)$, where $\tilde{u}_n(x)$ is the solution in the ELM subspace of the collocation problem (3.4) on the points $\{x_i\}$:*

$$-\mu \tilde{u}''_n(x_i) + \gamma \tilde{u}'_n(x_i) + \lambda \tilde{u}_n(x_i) = \tilde{f}(x_i) \qquad for\ i = 1, \dots, n. \tag{4.2}$$

The proof is straightforward as noticed in the Appendix A of [46]: because $\delta_{x_i,\varepsilon}$ functions converge to δ distributions, this provides that the integrals converge to the evaluation of the integrand function on x_i.

The above result gives, for our problem, an asymptotic stability property. With all this machinery, what we expect from the numerical solution is that non only one can obtain the usual convergence on collation points, but also, based on Theorem 2.2 presented in [47], convergence to the true solution in the L^2 norm. This is what we observe also numerically in the next section.

We remark that the collocation equations can be derived also via quadrature rules applied to Eqs. (4.1). In this case there is no need of Theorem 4.3, and the relation with collocation is obtained by the requirement that the quadrature rule is supported on the collocation points. In this case, the functions δ can be substituted with any family of locally supported functions whose support includes only one collocation point.

5. Numerical results

In this section, we apply the proposed algorithm for the solution of benchmark linear PDEs exhibiting steep gradients as suggested in [33]. More particularly, in Section 5.1, we first consider regular problems, in Section 5.2 boundary layer problems, and in Section 5.3 internal layer problems. For our computations, we have used Matlab2020b. In order to get reproducible results, we have used rng(5) to create the random values for the vectors $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ before calling the random value generator of Matlab (rand). The solution of the least squares problems is achieved via the backslash command.

For each problem, the number n of neurons ranged from 10 to 1280, doubling the number of neurons at each execution. For each fixed n, we compute solutions with various numbers M of collocation points. The computational time for all the above problems was of the order of 0.5 s on a single intel core i7-10750H with 16 GB RAM 2.60 Ghz. In particular, the maximum computational time was 0.6 s for $n = 1280$. To evaluate the approximation error,

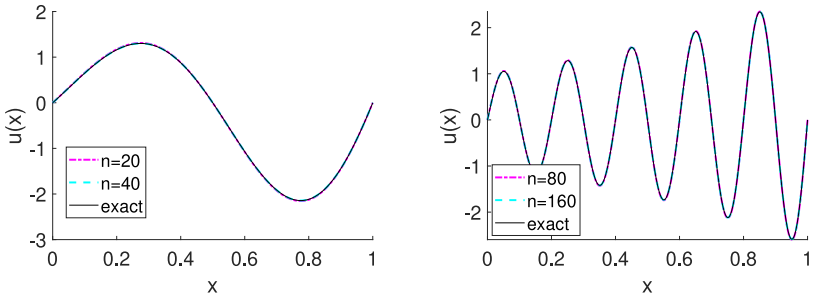


Fig. 2. Applying the proposed ELM network for the numerical solution of the problem given in (5.1): approximation with n neurons, the number of collocation points M fixed to be $n/2$ (on the left panel, $k = 1$, on the right panel, $k = 5$).

we used two different metrics: the absolute difference between the computed and the exact-analytical solution, denoted by E_u, and the residual of the equation, denoted by R_f:

$$E_u(x) = |u(x) - \tilde{u}(x)|$$
$$R_f(x) = |f(x) + \mu\tilde{u}''(x) - \gamma\tilde{u}'(x) - \lambda\tilde{u}(x)| \ .$$

Note, that all errors are absolute, not normalized. Then, we computed the L^2 norm of E_u and R_f; when computing the L^2-norm, we used 5000 equispaced points and the trapezoidal rule for integration. As a reference numerical method, we considered the Finite Difference (FD) scheme obtained with a maximal order on 7 nodes computed on M equispaced points. In order to compute the solution also on points that are not grid-points, the FD function is defined as the piecewise linear polynomial that interpolates the computed values. We emphasize that in all computations the number of points considered for the FD solution is equal to M, i.e. the number of collocation equations considered for the ELM implementation. In two cases, we also considered the solution only at collocation points, thus considering L^∞ errors between vectors in $\mathbb{R}^M$.

5.1. Regular problems

In this section, we analyze sinusoidal-bump problems as well as a high-order polynomial problem to show that the proposed approach can provide a high numerical approximation accuracy.

The first example is a simple 1D boundary value problem with homogeneous Dirichlet boundary conditions:

$$\begin{cases} u'' + (4k^2\pi^2 - 1)u = 4k\pi e^x \cos(2k\pi x), & 0 < x < 1 \\ u(0) = 0\,, \ u(1) = 0 \qquad \text{with } k \in \mathbb{N} \end{cases} \tag{5.1}$$

The above has the exact analytical solution:

$$u = \exp(x)\sin(2k\pi x)$$

The coefficient k represents the number of oscillations in the domain. We consider the simple case with $k = 1$ and a more challenging one with $k = 5$ (see Fig. 2). The computed errors with respect to the exact solution are reported in Fig. 3 and in Table 1. In both cases, we note that the proposed ELM network results to a good approximation with a modest number of neurons. In Fig. 3 on the right panels one can notice what pointed out in Section 3: when we fix the number of neurons and increase the number of collocation points the convergence is slow so that the choice $M = n/2$ is proposed.

Finally, for this case, we have also computed the L^∞-norm of the error only at the collocation points, see Fig. 4, where the ELM turns out to have a "super-convergence" (as also the FD scheme) but provides an accuracy of at least 1 order more compared to the FD scheme.

The second boundary value problem that we consider is a problem characterized by the presence of a high-order polynomial at the right-hand side:

$$\begin{cases} u'' = 2^{2p}p(1-x)^{p-2}x^{p-2}(-1 + 2x - 2x^2 + p(1 - 4x + 4x^2)), & 0 < x < 1 \\ u(0) = 0\,, \ u(1) = 0 \end{cases} \tag{5.2}$$

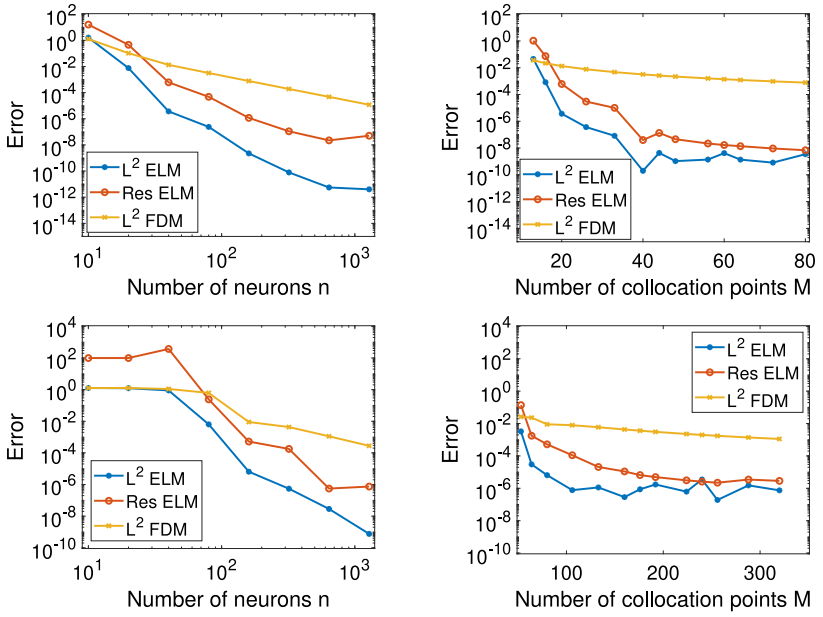


Fig. 3. Error and residual convergence for the ELM numerical solution of the problem (5.1). On the top panels, $k = 1$: on the left top panel $M = n/2$, on the right top panel $n = 40$. On the bottom panels, $k = 5$: on the left bottom panel, $M = n/2$, on the right bottom panel, $n = 160$.

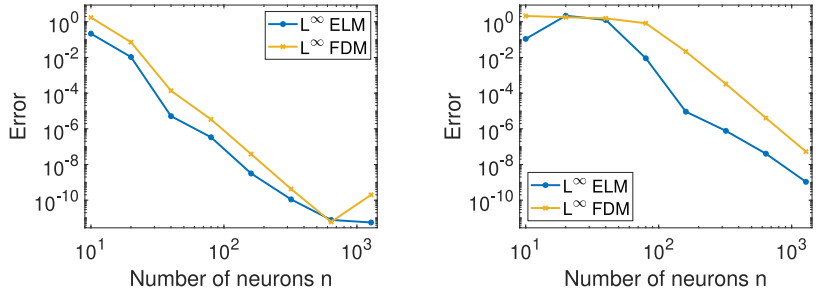


Fig. 4. Converge plots of the L^∞ norm at the collocation points for the solution of the problem (5.1): on the left panel, $k = 1$, on the right panel, $k = 5$.

Table 1
Error and residuals for the ELM numerical solution of the problem (5.1): on the left table, $k = 1$, on the right table $k = 5$.

k=1				k=5			
Neurons	Nodes	Error_L2	Residual	Neurons	Nodes	Error_L2	Residual
10	3	1.2825e+00	1.7566e+01	40	13	1.0291e+00	5.4833e+02
	4	1.1336e+00	1.4846e+01		16	9.0266e−01	2.8111e+02
	5	1.6084e+00	1.5970e+01		20	8.9477e−01	3.7287e+02
	6	6.4758e+00	4.7969e+01		26	1.0524e+00	2.7019e+01
	8	1.9045e−01	6.7843e+00		33	1.1745e+00	1.1760e+00
	10	1.1092e−02	1.5801e−01		40	3.0679e−01	8.3835e+00

(*continued on next page*)

The above BV problem has the exact solution:

$$u(x) = 2^{2p} x^p (1 - x)^p \tag{5.3}$$

Here, we consider the case $p = 10$, for which the FD scheme fails to accurately approximate the correct solution.

Table 1 (*continued*).

k=1 Neurons	Nodes	Error_L2	Residual	k=5 Neurons	Nodes	Error_L2	Residual
20	6	4.1912e−01	2.1314e+01	80	26	1.0283e+00	2.7422e+01
	8	4.1948e−01	7.0691e+00		32	2.3444e−02	1.4636e+00
	10	7.5525e−03	4.5484e−01		40	6.3788e−03	2.4238e−01
	13	2.1932e−03	4.7934e−02		53	6.1162e−04	3.7449e−02
	16	9.4571e−05	4.7279e−03		66	2.7654e−06	1.3766e−03
	20	3.4250e−07	2.3290e−05		80	1.0742e−06	1.9191e−04
40	13	4.5810e−02	1.0488e+00	160	53	3.2723e−03	1.3777e−01
	16	8.2667e−04	7.3745e−02		64	3.0106e−05	1.7641e−03
	20	3.6740e−06	6.1176e−04		80	6.4715e−06	5.2040e−04
	26	3.6832e−07	2.9772e−05		106	7.7111e−07	1.1040e−04
	33	8.2306e−08	1.0047e−05		133	1.1303e−06	2.0768e−05
	40	1.9954e−10	3.9535e−08		160	2.8612e−07	1.0990e−05
80	26	1.4968e−04	1.1856e−02	320	106	5.1808e−06	9.5524e−04
	32	1.4893e−05	1.9670e−03		128	6.5518e−06	1.1001e−03
	40	2.3953e−07	4.7799e−05		160	5.4403e−07	1.7879e−04
	53	1.9385e−10	5.7391e−08		213	1.3397e−07	1.0850e−05
	66	3.0444e−10	6.2147e−08		266	9.3901e−08	4.1553e−06
	80	2.3971e−11	5.9213e−09		320	2.6074e−09	8.7342e−08

In Fig. 5, we depict the computed errors of the approximated ELM solutions with respect to the exact-analytical ones.

5.2. Boundary layer problems

In this section, we consider two benchmark boundary layer problems, namely an advection-dominated and a reaction-dominated problem that lead to sharp gradients near the boundary (see also [34]). We show how the proposed scheme can deal properly with the steep gradients that arise in these cases.

When $\lambda = 0$ and $f \equiv 0$ in (3.1), we have a diffusion–advection problem. If we impose Dirichlet boundary conditions with $g_0 = 0\,,\ g_1 = 1$, the solution is given by:

$$u(x) = \frac{\exp\left(\frac{\gamma}{\mu}x\right) - 1}{\exp\left(\frac{\gamma}{\mu}\right) - 1}\,. \tag{5.4}$$

If $|\frac{\gamma}{\mu}| \ll 1$, the solution approaches the line connecting the boundary conditions while if $|\frac{\gamma}{\mu}| \gg 1$ the solution is close to zero in almost the whole domain, except in a neighborhood of the right boundary where the function has a steep gradient. In this last case, the solution has a *boundary layer* with a width of order $\mathcal{O}\left(\frac{\mu}{\gamma}\right)$. From a numerical point of view, for the particular problem, in which advection dominates over diffusion, we need to catch the behavior on a small scale $\frac{\mu}{\gamma}$, but it is well known that using Finite Differences or even Finite Element methods, the approximate solution may oscillate while the exact solution is monotonic (see [34]).

In the advection-dominated problem, the presence of the boundary layer is usually related to the so-called global Péclet number:

$$\mathbb{P}e_g = \frac{|\gamma| \cdot |I|}{2\mu}, \tag{5.5}$$

where $|I|$ is the domain length.

We considered the case with $\gamma = 100$ and $\mu = 1$, with a Péclet number $\mathbb{P}e_g = 50$. Fig. 6 depicts the exact-analytical solution and some of the ELM numerical solutions along with the corresponding approximation errors.

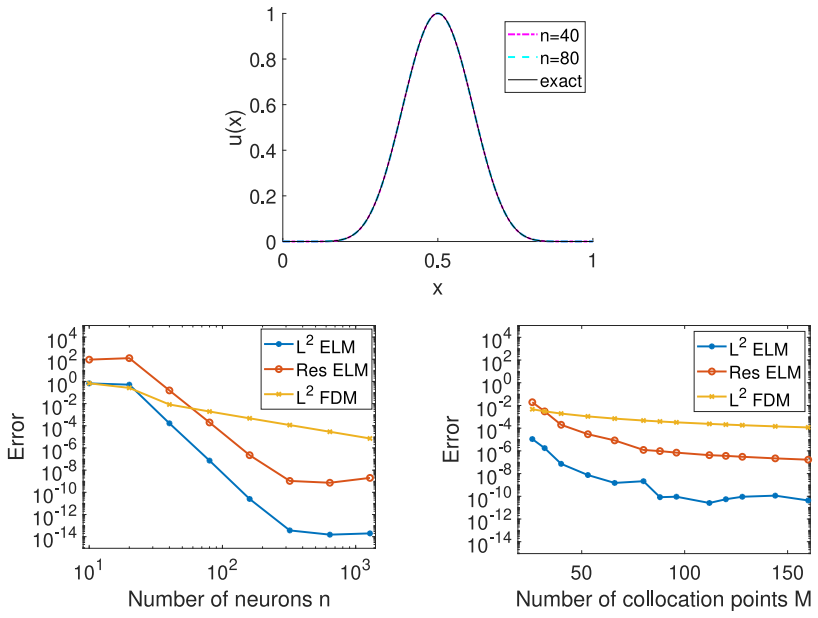


Fig. 5. Numerical tests for the solution of problem (5.2) with $p = 10$. On the top panel are depicted the exact-analytical (5.3) and the ELM numerical solutions with $n = 40, 80$; the number of collocation points M was set to $n/2$. On the bottom panels are shown the error and residual convergence: on the left panel, we have fixed $M = n/2$ and varied n and on the right panel we have fixed $n = 80$ and varied M.

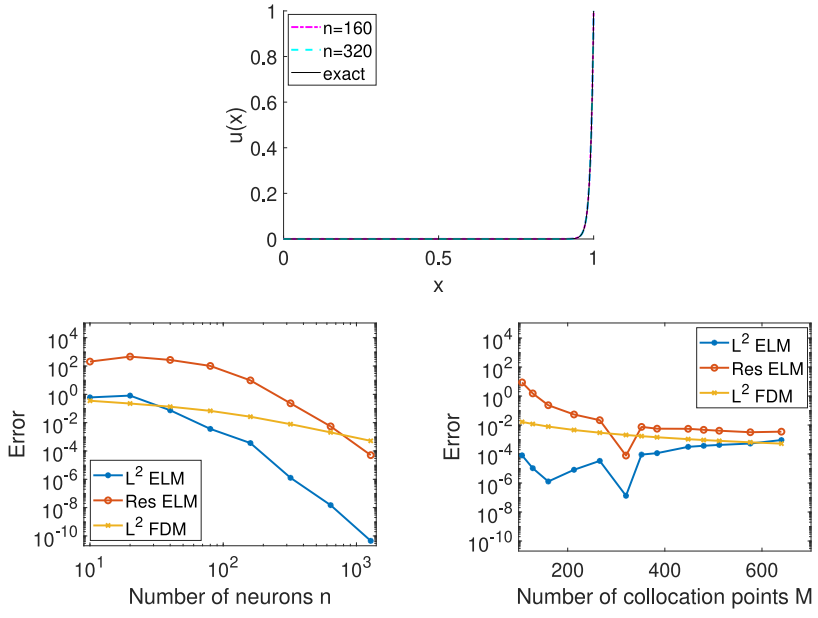
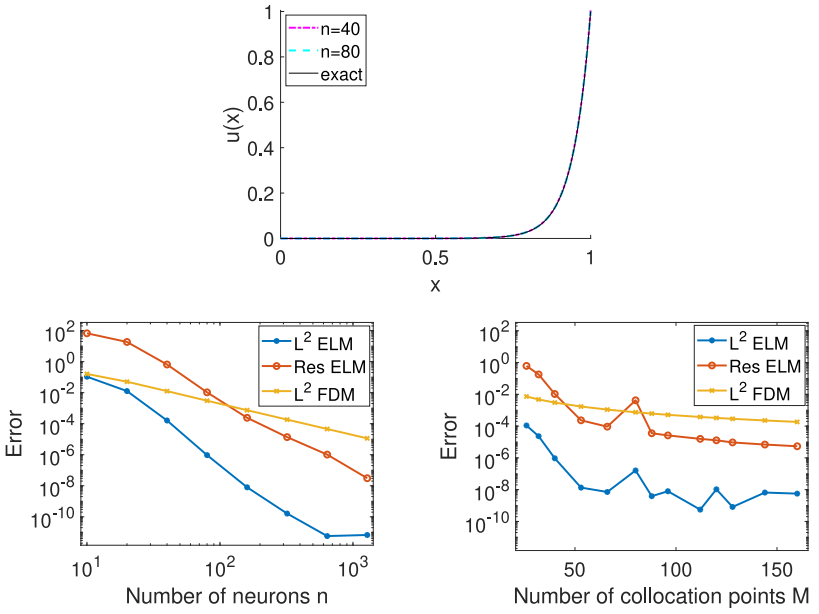


Fig. 6. Numerical tests for the solution of the problem (3.1) with $\mu = 1, \gamma = 100, \lambda = 0$. In this case, the Péclet number is $\mathbb{P}e_g = 50$, the boundary layer is of the order of 10^{-2}. On the top panel are shown both the exact-analytical (5.4) and the ELM numerical solutions with $n = 40, 80$; the number of collocation points was set to $M = n/2$. On the bottom panels are shown the error and residual convergence; on the left panel, we have fixed $M = n/2$ and varied n and on the right panel we have fixed $n = 320$ and varied M.

When $\gamma = 0$, $\lambda > 0$ and $f \equiv 0$, in the problem (3.1), one gets a diffusion–reaction problem. As before, we imposed Dirichlet boundary conditions with $g_0 = 0\,,\ g_1 = 1$. The problem has analogous difficulties as the previous



edited both bottom panels

Fig. 7. Numerical tests for the solution of the problem (3.1) with $\mu = 1, \gamma = 0, \lambda = 300$. In this case, the Péclet number is $\mathbb{P}e_g = 50$, the boundary layer is of the order of 10^{-1}. On the top panel are shown the exact-analytical (5.6) and the ELM numerical solutions for $n = 40, 80$; the number of collocation points was set to $M = n/2$. On the bottom panels are depicted the error and the residual convergence: on the left panel we have fixed $M = n/2$ and varied n and on the right panel we have fixed $n = 80$ and varied M.

one. The exact-analytical solution is given by:

$$u(x) = \frac{\sinh(\theta x)}{\sinh(\theta)}, \qquad \text{where } \theta = \sqrt{\lambda/\mu}. \tag{5.6}$$

Also in this case, the solution exhibits steep gradients if $\lambda/\mu \gg 1$ and there is a *boundary layer* of amplitude $\mathcal{O}(\sqrt{\mu/\lambda})$ at the right boundary. In this case, the usual definition of the global Péclet number is:

$$\mathbb{P}e_g = \frac{|\lambda| \cdot |I|^2}{6\mu} \ . \tag{5.7}$$

We have set $\lambda = 300$ and $\mu = 1$ so that the Péclet number is $\mathbb{P}e_g = 50$ as before. The computed errors with respect to the exact-analytical solution are reported in Fig. 7.

Finally, we considered both the advection- and the reaction-dominated problems with different values for the parameters γ and λ so that we get appropriate values of the Péclet number. In Fig. 8, we plot the L^2 errors of the computed ELM solutions versus the Péclet numbers for two different choices of number of neurons.

5.3. Internal layer problems

In this section, we consider highly transient problems that lead to an internal layer. For the solutions of these problems an adaptive mesh is usually proposed. For a description of the numerical problems that can arise, we refer to [33]. In this section, we show that the proposed ELM scheme can provide good approximations. First, we consider the *atan* problem:

$$\begin{cases} -u'' = \frac{(2\alpha^3(x-x_0))}{(1+\alpha^2(x-x_0)^2)^2}, & 0 < x < 1 \\ u(0) = \theta_0\,, \ u(1) = \theta_1 \end{cases} \tag{5.8}$$

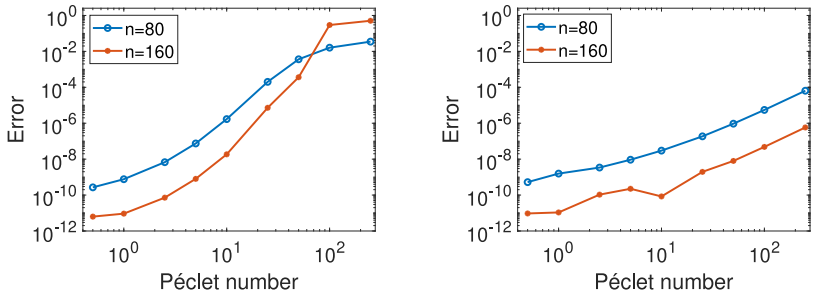


Fig. 8. L^2 errors of the computed ELM solutions with fixed n and $M = n/2$ with respect to the Péclet number as computed with Eq. (5.5) (left panel) and Eq. (5.7) (right panel). In particular, on the left panel, we have taken $\gamma = [0, 1, 2, 5, 10, 20, 50, 100, 200, 500]$ and on the right panel $\lambda = [0, 3, 6, 15, 30, 60, 150, 300, 600, 1500]$.

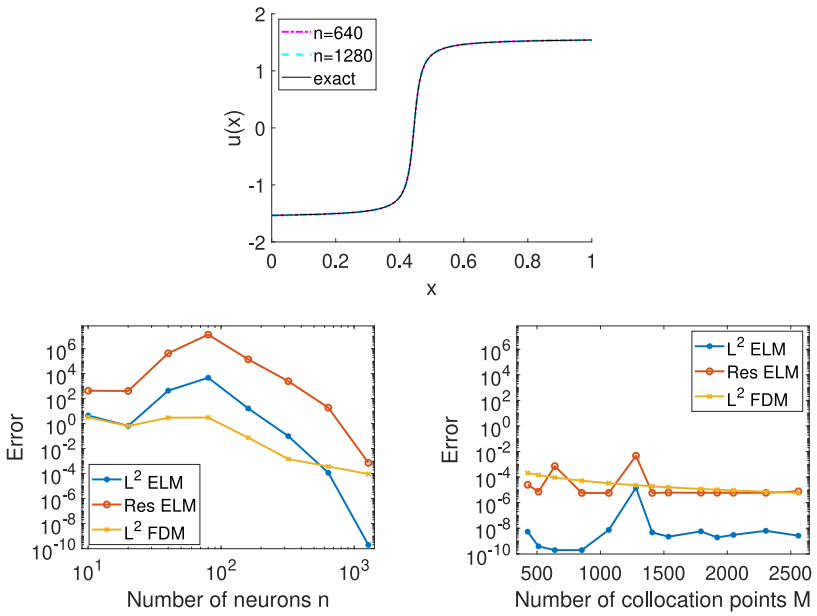


Fig. 9. Numerical tests for the solution of problem (5.8) with $\alpha = 60$ and $x_0 = 4/9$. On the top panel, we depict the exact-analytical (5.9) and the ELM numerical solutions for $n = 640, 1280$; the number of collocation points M was fixed to $n/2$. On the bottom panel, we depict the error and residual convergence: on the left panel, we have fixed $M = n/2$ and varied n and on the right panel we have fixed $n = 1280$ and varied M.

θ_0, θ_1 are fixed in order to have an exact solution that reads:

$$u(x) = \text{atan}(\alpha(x - x_0)) \tag{5.9}$$

In our tests, we have fixed $\alpha = 60$ and $x_0 = 4/9$ that leads to a non-symmetric internal layer as in [48]. Fig. 9 depicts the exact-analytical solution and some of the numerical ones. The approximation errors with respect to the exact-analytical solution are also given. The second test with internal peak is a rescaled sinusoidal problem suggested in [49]:

$$\begin{cases} -u'' = \left(\frac{-4x^2}{\varepsilon^2} + \frac{2}{\varepsilon}\right)e^{-x^2/\varepsilon}, & 0 < x < 1 \\ u(0) = \theta_0\,, \;\; u(1) = \theta_1 \end{cases} \tag{5.10}$$

θ_0, θ_1 are fixed in order to have as exact solution:

$$u(x) = e^{-x^2/\varepsilon}\ . \tag{5.11}$$

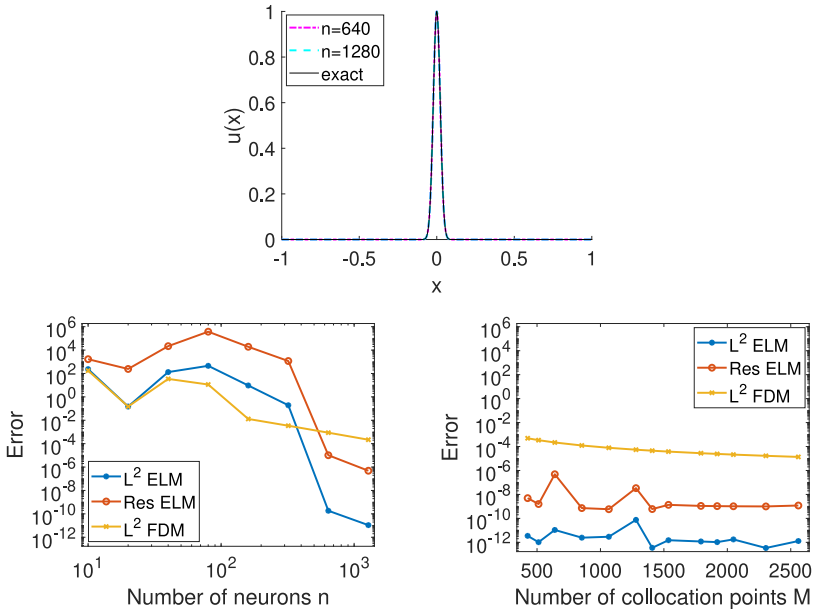


Fig. 10. Numerical results for the solution of the problem (5.10) with $\varepsilon = 10^{-3}$. On the top panel are depicted the exact-analytical (5.11) and the computed solutions based on the proposed scheme with $n = 640, 1280$ neurons and $M = n/2$ collocations points. On the bottom panel are depicted the error and residual convergence: on the left panel, we have fixed $M = n/2$ and varied n and on the right panel, we have fixed $n = 1280$ and varied M.

We consider the case $\varepsilon = 10^{-3}$. Fig. 10 depicts the exact-analytical solution and some of the numerical ones. The approximation errors with respect to the exact-analytical solution are also given. Finally, we considered a very difficult numerical problem that has as exact-analytical solution a comb-like profile. For this problem, we compute the approximation errors only on collocation points using several numbers of neurons. The equation we consider is the following:

$$\begin{cases} -u'' = -\frac{2(\varepsilon+x)\cos(\frac{1}{\varepsilon+x}) - \sin(\frac{1}{\varepsilon+x})}{(\varepsilon+x)^4}, & 0 < x < 1 \\ u(0) = \theta_0\,, \;\; u(1) = \theta_1 \end{cases} \tag{5.12}$$

We have set θ_0, θ_1 in order to have the following exact-analytical solution:

$$u(x) = \sin\left(\frac{1}{\varepsilon + x}\right). \tag{5.13}$$

Then, we considered the case $\varepsilon = 1/10\pi$ that exhibits a solution with 10 zeros and 5 oscillations in the domain. Fig. 11 depicts the exact-analytical solution as well as the numerical solution obtained with the proposed machine learning scheme for $n = 1280, 2560$ and the resulting approximation errors with respect to the exact-analytical solution considering only the approximation errors at the collocation points using the L^∞ norm.

6. Conclusions and future work

In the present work, we address a numerical scheme based on the concept of ELM networks for the numerical approximation of 1D elliptic linear PDEs which solutions exhibit steep gradients. The weights between the input layer and the hidden layer as well as the biases of the neurons in the hidden layer are fixed appropriately in advance and kept fixed. In this way, the only unknown parameters are the weights of the connections between the hidden layer and the linear activation transfer function of the output layer of the ELM. First, we have explored the underlying subspace, so as to construct it in a random but proper way. Then, the numerical solution to the boundary value problems was reduced to the computation of the n output weights of the ELM network via the solution of M collocation equations. We have justified the convergence of the scheme in the square-matrix case, and we proposed the use of an under-determined collocation scheme to avoid ill-conditioning.

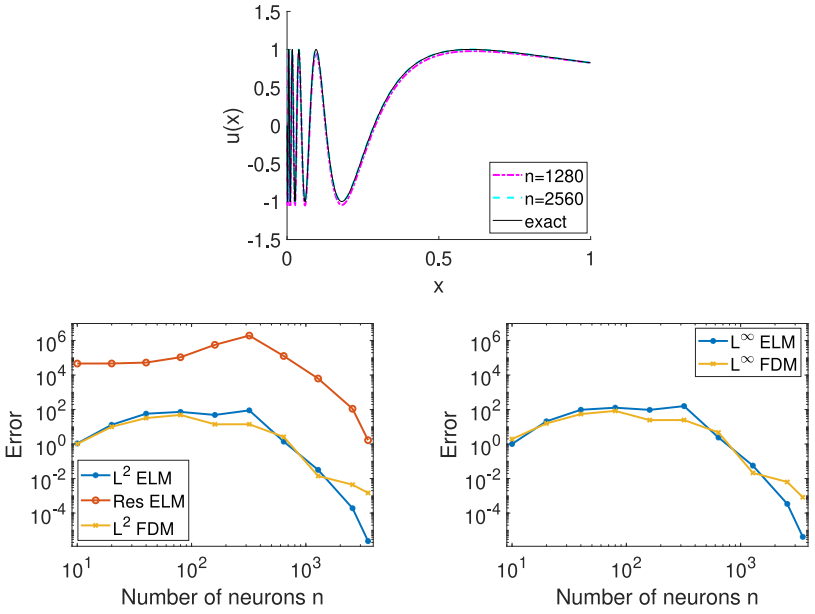


Fig. 11. Numerical results for the solution of the problem (5.12) with $\varepsilon = 1/10\pi$. On the top panels it is depicted the exact-analytical (5.13) and the numerical solutions with $n = 1280, 2560$ neurons and the number of collocation points fixed at $M = n/2$. On the bottom panels are shown the error and residual convergence: on the left panel, it is shown the L^2 norms with fixed $M = n/2$ and on the right panel it is shown the L^∞ on the collocation points.

Numerical tests were performed for the evaluation of the approximation accuracy of the proposed method with the aid of benchmark boundary value problems exhibiting sharp gradients. We also compared the proposed approach with a high order Finite Difference (FD) scheme. We showed that for both internal and boundary layer problems, the proposed machine learning method outperforms FD, when the number of neurons is taken to be large enough in order to catch the steep gradient of the layer. We emphasize also that in our case, the solutions are C^∞ functions.

The proposed numerical method can be further developed by exploring some alternatives, such as the use of optimal activation functions, as proposed in [22], or the selection of the solution of the under-determined system by using the properties of null rules, as explored in [50]. Finally, the application of the proposed machine learning collocation method can be extended in various ways. Among others, we are currently considering in developing numerical schemes for the solution of:

- Non linear differential problems;
- Multidimensional problems;
- Time-dependent problems;
- Inverse problems with overfitting.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank Prof. Ferdinando Auricchio, Prof. Bert Jüttler for fruitful discussions on the subject of the paper. Francesco Calabrò and Constantinos Siettos are partially supported by INdAM, through GNCS research projects. This support is gratefully acknowledged.

References

[1] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000.

[2] I.E. Lagaris, A.C. Likas, D.G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw. 11 (5) (2000) 1041–1049.

[3] C. Anitescu, E. Atroshchenko, N. Alajlan, T. Rabczuk, Artificial neural network methods for the solution of second order boundary value problems, Comput. Mater. Contin. 59 (1) (2019) 345–359.

[4] H. Arbabi, J.E. Bunder, G. Samaey, A.J. Roberts, I.G. Kevrekidis, Linking machine learning with multiscale numerics: Data-driven discovery of homogenized equations, JOM (2020) 1–14.

[5] J. Berg, K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018) 28–41.

[6] Q. Chan-Wai-Nam, J. Mikael, X. Warin, Machine learning for semi linear PDEs, J. Sci. Comput. 79 (3) (2019) 1667–1712.

[7] V. Dwivedi, B. Srinivasan, Physics informed extreme learning machine (PIELM) - a rapid method for the numerical solution of partial differential equations, Neurocomputing 391 (2020) 96–118, URL http://www.sciencedirect.com/science/article/pii/S0925231219318144.

[8] H. Guo, X. Zhuang, X. Meng, T. Rabczuk, Analysis of three dimensional potential problems in non-homogeneous media with deep learning based collocation method, 2020, arXiv preprint arXiv:2010.12060.

[9] H. Guo, X. Zhuang, T. Rabczuk, A deep collocation method for the bending analysis of Kirchhoff plate, Comput. Mater. Contin. 59 (2) (2019) 433–456, URL http://www.techscience.com/cmc/v59n2/27944.

[10] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (34) (2018) 8505–8510.

[11] J. Han, M. Nica, A.R. Stinchcombe, A derivative-free method for solving elliptic partial differential equations with deep neural networks, J. Comput. Phys. 419 (2020) 109672, URL http://www.sciencedirect.com/science/article/pii/S0021999120304460.

[12] A.D. Jagtap, E. Kharazmi, G.E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, Comput. Methods Appl. Mech. Engrg. 365 (2020) 113028.

[13] S. Lee, M. Kooshkbaghi, K. Spiliotis, C.I. Siettos, I.G. Kevrekidis, Coarse-scale PDEs from fine-scale observations via machine learning, Chaos 30 (1) (2020) 013141.

[14] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, DeepXDE: A deep learning library for solving differential equations, SIAM Review 63 (1) (2021) 208–228.

[15] C. Michoski, M. Milosavljević, T. Oliver, D. Hatch, Solving differential equations using deep neural networks, Neurocomputing (2020).

[16] G. Pang, L. Yang, G.E. Karniadakis, Neural-net-induced Gaussian process regression for function approximation and PDE solution, J. Comput. Phys. 384 (2019) 270–288.

[17] K. Rudd, S. Ferrari, A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks, Neurocomputing 155 (2015) 277–285.

[18] J. Sirignano, J.F. MacArt, J.B. Freund, DPM: A deep learning PDE augmentation method with application to large-eddy simulation, J. Comput. Phys. 423 (2020) 109811, URL http://www.sciencedirect.com/science/article/pii/S0021999120305854.

[19] J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.

[20] J. Almira, P. Lopez-de Teruel, D. Romero-Lopez, F. Voigtlaender, Negative results for approximation using single layer and multilayer feedforward neural networks, J. Math. Anal. Appl. (2020) 124584.

[21] D. Costarelli, R. Spigler, Approximation results for neural network operators activated by sigmoidal functions, Neural Netw. 44 (2013) 101–106.

[22] N.J. Guliyev, V.E. Ismailov, On the approximation by single hidden layer feedforward neural networks with fixed weights, Neural Netw. 98 (2018) 296–304.

[23] A. Pinkus, Approximation theory of the MLP model in neural networks, Acta Numer. 8 (1) (1999) 143–195.

[24] K. Akyol, Comparing of deep neural networks and extreme learning machines based on growing and pruning approach, Expert Syst. Appl. 140 (2020) 112875.

[25] G. Huang, G.-B. Huang, S. Song, K. You, Trends in extreme learning machines: A review, Neural Netw. 61 (2015) 32–48, URL http://www.sciencedirect.com/science/article/pii/S0893608014002214.

[26] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: a new learning scheme of feedforward neural networks, in: 2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541), 2, IEEE, 2004, pp. 985–990.

[27] G.-B. Huang, What are extreme learning machines? Filling the gap between Frank Rosenblatt's dream and John von Neumann's puzzle, Cognitive Computation 7 (3) (2015) 263–278.

[28] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychol. Rev. 65 (6) (1958) 386.

[29] W.F. Schmidt, M.A. Kraaijveld, R.P. Duin, et al., Feed forward neural networks with random weights, in: International Conference on Pattern Recognition, IEEE Computer Society Press, 1992, p. 1.

[30] Y.-H. Pao, G.-H. Park, D.J. Sobajic, Learning and generalization characteristics of the random vector functional-link net, Neurocomputing 6 (2) (1994) 163–180.

[31] C. De Falco, E. O'Riordan, Interior layers in a reaction-diffusion equation with a discontinuous diffusion coefficient, Int. J. Numer. Anal. Model 7 (3) (2010) 444–461.

[32] E. Oñate, J. Miquel, P. Nadukandi, An accurate FIC-FEM formulation for the 1D advection–diffusion–reaction equation, Comput. Methods Appl. Mech. Engrg. 298 (2016) 373–406.

[33] W.F. Mitchell, A collection of 2D elliptic problems for testing adaptive grid refinement algorithms, Appl. Math. Comput. 220 (2013) 350–364, URL http://www.sciencedirect.com/science/article/pii/S0096300313006449.
[34] A. Quarteroni, Diffusion-transport-reaction equations, in: Numerical Models for Differential Problems, Springer, 2017, pp. 315–365.
[35] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Systems 2 (4) (1989) 303–314.
[36] K. Hornik, M. Stinchcombe, H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, Neural Netw. 3 (5) (1990) 551–560.
[37] K. Hornik, M. Stinchcombe, H. White, et al., Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366.
[38] J.-G. Attali, G. Pagès, Approximations of functions by a multilayer perceptron: a new approach, Neural Netw. 10 (6) (1997) 1069–1081.
[39] S. Dong, Z. Li, Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations, 2020, arXiv preprint arXiv:2012.02895.
[40] Y. Ito, Nonlinearity creates linear independence, Adv. Comput. Math. 5 (1) (1996) 189–203.
[41] G.-B. Huang, D.H. Wang, Y. Lan, Extreme learning machines: a survey, Int. J. Mach. Learn. Cybern. 2 (2) (2011) 107–122.
[42] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1–3) (2006) 489–501.
[43] N. Hahm, B.I. Hong, An approximation by neural networks with a fixed weight, Comput. Math. Appl. 47 (12) (2004) 1897–1903.
[44] S. Lin, X. Guo, F. Cao, Z. Xu, Approximation by neural networks with scattered data, Appl. Math. Comput. 224 (2013) 29–35.
[45] G.-B. Huang, L. Chen, C.K. Siew, et al., Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Netw. 17 (4) (2006) 879–892.
[46] F. Auricchio, L.B. Da Veiga, T.J. Hughes, A. Reali, G. Sangalli, Isogeometric collocation for elastostatics and explicit dynamics, Comput. Methods Appl. Mech. Engrg. 249 (2012) 2–14.
[47] G. Strang, G.J. Fix, An Analysis of the Finite Element Method, Prentice-hall, 1973.
[48] L. Demkowicz, W. Rachowicz, P. Devloo, A fully automatic hp-adaptivity, J. Sci. Comput. 17 (1–4) (2002) 117–142.
[49] A. Schmidt, K.G. Siebert, A posteriori estimators for the h–p version of the finite element method in 1d, Appl. Numer. Math. 35 (1) (2000) 43–66.
[50] F. Calabrò, D. Bravo, C. Carissimo, E. Di Fazio, A. Di Pasquale, A. Eldray, C. Fabrizi, J. Gerges, S. Palazzo, J. Wassef, Null rules for the detection of lower regularity of functions, J. Comput. Appl. Math. 361 (2019) 547–553.