Journal of Computational and Applied Mathematics 436 (2024) 115396

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

Physics-Informed Neural Networks for 2nd order ODEs with sharp gradients

Mario De Florio [a], Enrico Schiassi [b], Francesco Calabrò [c,*], Roberto Furfaro [b,d]

[a] Division of Applied Mathematics, Brown University, 170 Hope St, Providence, 02906, RI, USA
[b] Department of Systems & Industrial Engineering, The University of Arizona, Tucson, AZ, USA
[c] Dipartimento di Matematica e Applicazioni "Renato Caccioppoli", Università degli Studi di Napoli "Federico II", Naples, Italy
[d] Department of Aerospace & Mechanical Engineering, The University of Arizona, Tucson, AZ, USA

ARTICLE INFO

Article history:
Received 3 February 2023
Received in revised form 19 May 2023

Keywords:
Extreme learning machine
Functional interpolation
Least-squares
Physics-Informed Neural Networks

ABSTRACT

In this work, four different methods based on Physics-Informed Neural Networks (PINNs) for solving Differential Equations (DE) are compared: *Classic-PINN* that makes use of Deep Neural Networks (DNNs) to approximate the DE solution;*Deep-TFC* improves the efficiency of classic-PINN by employing the constrained expression from the Theory of Functional Connections (TFC) so to analytically satisfy the DE constraints;*PIELM* that improves the accuracy of classic-PINN by employing a single-layer NN trained via Extreme Learning Machine (ELM) algorithm;*X-TFC*, which makes use of both constrained expression and ELM. The last has been recently introduced to solve challenging problems affected by discontinuity, learning solutions in cases where the other three methods fail. The four methods are compared by solving the boundary value problem arising from the 1D Steady-State Advection–Diffusion Equation for different values of the diffusion coefficient. The solutions of the DEs exhibit steep gradients as the value of the diffusion coefficient decreases, increasing the challenge of the problem.

1. Introduction

Machine Learning, in particular, Neural Networks (NNs), have become widespread across many scientific fields in recent years. This requires designing them to satisfy application-specific constraints, such as conservation laws, symmetries, or other domain-specific knowledge. These constraints are usually imposed as soft penalties during network training and act as regularizers within the loss function. An example of this philosophy is the recently introduced *Physics-Informed Neural Networks* (PINNs), see [1].

Within the PINN paradigm, the network's outputs are constrained to approximately satisfy specific physics laws, modeled as a set of Differential Equations (DEs). PINNs are introduced to add the DEs, modeling the physics of the experimental dataset, as a penalty to the loss function. This additional term acts as a regularizer that penalizes the training when the DE and its constraints (e.g., initial conditions ICs and/or boundary conditions BCs) are violated. Overall, one aims to ensure that the process's physics is not violated. This approach is called *data-physics-driven solution* of DEs. Conversely, when the goal is to estimate parameters governing some physical phenomena modeled through a DE (e.g., the thermal conductivity in the heat equation), one usually talks about *data-physics-driven parameter discovery* of DEs (e.g., inverse problems). When data is unavailable, and therefore the loss function contains only the residual of the DEs and the

* Corresponding author.
E-mail address: francesco.calabro@unina.it (F. Calabrò).

https://doi.org/10.1016/j.cam.2023.115396
0377-0427/

corresponding DE's constraints, PINNs are used for approximating the solutions of problems involving DEs, solely in a *physics-driven* fashion.

In the authors' opinion, classic PINN frameworks present two important drawbacks: (1) the use of Deep NNs (DNNs) to approximate the DE solutions and (2) the fact that the equation's constraints are not analytically satisfied.

The use of standard DNNs introduces four main issues:

- Gradient-based methods are required to minimize the scalar loss function. These methods are computationally expensive and sensitive to the algorithm chosen for the minimization.
- The complexity of the DNN may cause the divergence of the training. Indeed a non-convex optimization problem is solved.
- Linear DEs are treated as nonlinear ones, as the network parameters appear non-linearly to approximate the DE solutions.
- For nonlinear DEs, it is prohibitive to provide an informative initial guess on the solution to be used as starting point of the optimization.

Having the equation's constraints not analytically satisfied introduces the issues of having competing objectives during the PINN training: learning the DE solution within the domain and learning its constraints. This can lead to unbalanced gradients when training the networks via gradient-based techniques that cause PINNs to fail to approximate the DE solution accurately [2].

Thus, a significant advance in the research area of PINNs would be to introduce a PINN framework able to remove both the issues discussed above.

In [2], to overcome the issue caused by having competing objectives during the PINN training, the authors proposed a learning rate annealing algorithm. It uses gradient statistics to adaptively assign proper weights to the different terms forming the PINN loss function (e.g., DE residuals within the domain and DE residuals on the boundaries) during the network training. With the algorithm proposed in [2], the issue of having competing objectives during the PINN training is mitigated, but the issue of using standard DNN remains.

In [3], the authors remove the issue of having the DE constraints included in the loss function by merging, for the first time, DNNs and the *Theory of Functional Connections* [4]. This PINN framework is called *Deep Theory of Functional Connections* (Deep-TFC). TFC is a functional interpolation methodology, where any mathematical problem solution can be represented via the so-called Constrained Expressions (CEs). The CEs are a sum of a free function and a functional that analytically satisfies the problem constraints [5]. That is,

$$u(\mathbf{x}) \simeq u_{CE}(\mathbf{x}, g(\mathbf{x})) = A(\mathbf{x}) + B(\mathbf{x}, g(\mathbf{x})) \tag{1}$$

where $u(\mathbf{x})$ is the unknown function to be interpolated, $u_{CE}(\mathbf{x}, g(\mathbf{x}))$ is the CE, $A(\mathbf{x})$ is the functional that analytically satisfies any given linear constraint, and $B(\mathbf{x}, g(\mathbf{x}))$ projects the free function $g(\mathbf{x})$, which is a real function that must exist on the constraints, onto the space of functions that vanish at the constraints [3]. When applied to problems involving DEs, the mathematical problem is represented by the DE itself, where the constraints are the initial and/or boundary conditions [6]. Nevertheless, Deep-TFC does not remove the issues caused by using DNNs.

Shallow NN gives an alternative that has been considered so as not to consider deep structure. In [7], the authors for the first time introduce a PINN framework with shallow NN trained via Extreme Learning Machine (ELM) called *Physics-Informed Extreme Learning Machines* (PIELM). The advantage of the ELM algorithm is that input weights and biases are randomly selected: the training is solely necessary for the output weights, with a significant reduction in the computational effort [8]. Other authors have considered the use of NN trained via ELM for differential problems, see e.g. [9–11]. Nevertheless, none of these approaches remove the issue of having the DE constraints included in the loss function.

In [12], for the first time, the authors merge ELM and TFC, generating a PINN framework that overcomes all the main issues of the classic one. Indeed, using TFC removes the issue of having the DE constraints included in the loss function. Hence, this PINN framework is called *Extreme Theory of Functional Connections* (X-TFC). Using ELM removes the issues caused by using DNNs.

Our aim is to present and compare such approaches using as a test problem a second-order differential equation. This manuscript is organized as follows. Section 2 briefly describes the four PINNs methods under study. Section 3 provides the numerical results for the 1D Steady-State Advection–Diffusion Equation for different values of the diffusion coefficient, with discussions. The article concludes with Section 4.

2. PINNs for differential equations

Consider a generic single PDE,

$$\mathcal{D}\left(u(\boldsymbol{x}); \boldsymbol{\Gamma}\right) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega \tag{2}$$

$$\mathcal{B}\left(u(\boldsymbol{x})\right) = g(\boldsymbol{x}), \quad \boldsymbol{x} \in \delta\Omega \tag{3}$$

where $\mathcal{D}(\cdot)$ and $\mathcal{B}(\cdot)$ are some linear or nonlinear differential operators, $u : \mathbb{R}^n \to \mathbb{R}$ is the unknown exact DE solution, $\boldsymbol{x} = \{x_1, \ldots, x_n\}^T$ are the independent variables (with $x_i \in \mathbb{R}\, \forall\, i = 1, \ldots, n$), $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}$ are some known

functions, Ω is a bounded open domain with boundary $\delta\Omega$, and $\mathbf{\Gamma} \in \mathbb{R}^m$ are some parameters governing the DE. When dealing with forward problems, $\mathbf{\Gamma}$ is known.

In the *classic PINN* frameworks, as presented by Raissi et al. [1], u is represented through a NN. That is,

$$u(\boldsymbol{x}) \simeq u(\boldsymbol{x};\theta) = u_\theta(\boldsymbol{x}) \tag{4}$$

where θ are the NN parameters (e.g., weights and bias). Considering, as an example, a Multiple-Input Single-Output (MISO) single layer (e.g., shallow) NN as represented in Fig. 1, $u_\theta(\boldsymbol{x})$ is expressed as follows,

$$u_\theta(\boldsymbol{x}) = \sum_{j=1}^{L} \beta_j \sigma\left(\boldsymbol{w}_j^T \boldsymbol{x} + b_j\right) \tag{5}$$

where σ's are some nonlinear activation functions, $W = \{\boldsymbol{w}_1, \dots, \boldsymbol{w}_L\} \in \mathbb{R}^{L\times n}$ (with $\boldsymbol{w}_j = \{w_{j,1}, \dots, w_{j,n}\}^T \in \mathbb{R}^L$, $\forall j = 1, \dots L$) is the input weights matrix, $\boldsymbol{b} = \{b_1, \dots, b_L\}^T \in \mathbb{R}^L$ is the bias vector, $\boldsymbol{\beta} = \{\beta_1, \dots, \beta_L\}^T \in \mathbb{R}^L$ is the output weights vector, and $\theta = \{W|\,\boldsymbol{b}|\,\boldsymbol{\beta}\} \in \mathbb{R}^{L\times(n+2)}$ is the NN parameters matrix. $u_\theta(\boldsymbol{x})$ is plugged into the DE residual and into the boundary conditions (BC), which are then collocated on the collocation (or training) points $\{\boldsymbol{x}_{\Omega,i}\}_{i=1}^{N_\Omega}$ and $\{\boldsymbol{x}_{\delta\Omega,i}\}_{i=1}^{N_{\delta\Omega}}$, that can be chosen from any collocation scheme. Finally, the following optimization problem is solved via gradient-based techniques (e.g., ADAM optimizer [13]) or LBFGS optimizer,

$$\min_\theta \quad \sum_{i=1}^{N_\Omega} \left(\|\mathcal{D}(u_\theta(\boldsymbol{x}_{\Omega,i};\mathbf{\Gamma})) - f(\boldsymbol{x}_{\Omega,i})\|^2\right) + \sum_{i=1}^{N_{\delta\Omega}} \left(\|\mathcal{B}\left(u_\theta(\boldsymbol{x}_{\delta\Omega,i})\right) - g(\boldsymbol{x}_{\delta\Omega,i})\|^2\right) \tag{6}$$

Once the training is completed, the optimal NN parameters, θ^*, are learned and plugged back into Eq. (5). That is,

$$u_{\theta^*}(\boldsymbol{x}) = \sum_{j=1}^{L} \beta_j^* \sigma\left(\boldsymbol{w}_j^{*,T} \boldsymbol{x} + b_j^*\right) \tag{7}$$

which is a closed-form analytical approximation of the DE true unknown solution u.

Within *PIELM*, (5) is replaced with an ELM. Hence, input weights and bias are randomly selected and not tuned during the training, leaving the output weights the only trainable parameters. Thus (6) is solved with a single pass least-square if the DE is linear, or with an iterative least-squares procedure, if the DE is nonlinear. More details regarding PIELM can be found in [7] and references within.

Within *Deep-TFC*, u is approximated with the TFC CE, where the free function is a DNN. Hence the DE constraints are analytically satisfied and (6) reduces to,

$$\min_\theta \quad \sum_{i=1}^{N_\Omega} \left(\|\mathcal{D}(u_\theta(\boldsymbol{x}_{\Omega,i};\mathbf{\Gamma})) - f(\boldsymbol{x}_{\Omega,i})\|^2\right) \tag{8}$$

which is solved via gradient-based methods, as for classic PINNs. More details regarding Deep-TFC can be found in [3] and references within.

Within *X-TFC*, u is approximated with the TFC CE, where the free function is an ELM. Therefore, (6) reduces to (8). Moreover, as the free function is an ELM, (8) is solved with a single pass least-square, if the DE is linear, or with an iterative least-squares procedure, if the DE is nonlinear. More details regarding X-TFC can be found in [12] and references within.

3. Numerical results and discussions

We tested the PINN frameworks explained above in solving the 1D Steady-State Advection–Diffusion Equation for different values of the diffusion coefficient ν.

The equation under study is the following,

$$u_{xx} - P_e u_x = 0 \qquad \text{subject to} \qquad \begin{cases} u(0) &= 1 \\ u(1) &= 0 \end{cases}$$

where $x \in [0; 1]$ and $P_e = \frac{1}{\nu}$ is the Peclet number.

The analytical solution is,

$$u(x) = \frac{1 - e^{P_e(x-1)}}{1 - e^{-P_e}}$$

The sharpness of the solution increases for decreasing values of ν. That is, for high values of ν the solution is smooth, while for low values of ν the solution tends to have a steep gradient, thus miming a discontinuous behavior.

The problem considered has been coded in MATLAB R2022a and all the tests were run with an Intel Core i7 - 9700 CPU PC with 64 GB of RAM.

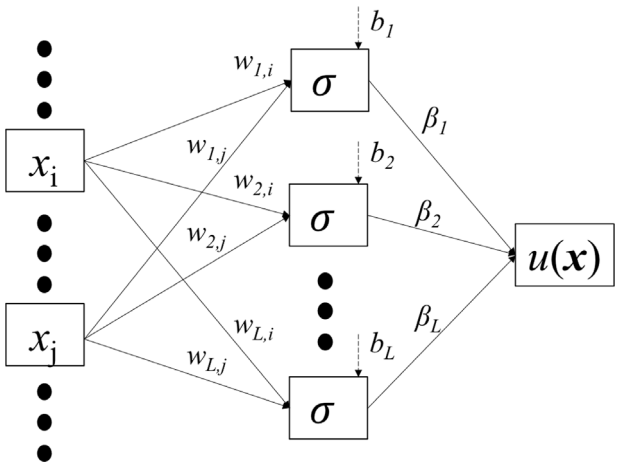

Fig. 1. MISO NN schematic.

Table 1
Simulations hyperparameters.

	PINN	PIELM	Deep-TFC	X-TFC
Number of layers	10	1	10	1
Number of neurons per layer	Problem dep.	Problem dep.	Problem dep.	Problem dep.
Activation function	$e^{-(\bullet)^2}$	$e^{-(\bullet)^2}$	$e^{-(\bullet)^2}$	$e^{-(\bullet)^2}$
Optimizer	ADAM/LBFGS	Least-Squares	ADAM/LBFGS	Least-Squares
Learning-rate	Adaptive	–	Adaptive	–
N_{int}	10000	10000	10000	10000
N_{BC}	2	2	0	0
Epochs	15000	1	15000	1
Mini-batch size (for ADAM)	2000	10000	2000	10000
Number of batches (for ADAM)	5	1	5	1

We run the tests using the hyperparameters reported in Table 1. For PINN and Deep-TFC the DNN parameters were randomly initialized. For PIELM and X-TFC input weights and bias were sampled according to the sampling technique proposed in [9].

For Deep-TFC and X-TFC the CE is the following,

$$u_{\boldsymbol{\beta}}(x) = g(x) + \frac{x_f - x}{x_f - x_0}\left(u(0) - g(0)\right) + \frac{x - x_0}{x_f - x_0}\left(u(1) - g(1)\right) \tag{9}$$

where $g(x)$ is a DNN for Deep-TFC and an ELM for X-TFC. From (9) the reader can see how the DE constraints are analytically satisfied.

Results and performances are summarized in Tables 2–6 and Figs. 2–6. The definition of training error ε_T and generalization error are given in detail in [14] and references within.

It is straightforward to note that ELM-based methods (PIELM and X-TFC) achieve the best accuracy both in terms of absolute difference compared with analytical solution and training error, for each value of the diffusion coefficient ν. This is due to the employment of a single-layer NN trained via ELM. Also, among the two ELM-based methods, X-TFC outperforms PIELM, thanks to its capability to analytically satisfy the boundary conditions to which the problem is subject.

In detail, we can see that in the simplest case for $\nu = 1$, all the four methods are able to learn the solution of the DE under study (Fig. 2), with mean absolute errors of the order of e-05 and e-08 for PINN and Deep-TFC, respectively, and e-15 and e-16, for PIELM and X-TFC, respectively. Similar results are obtained for the case $\nu = 0.1$ (Fig. 3). The first failure on the part of PINN is seen for the case $\nu = 0.01$, where the solution is missing to be learned, and a slight deviation from the analytical solution is seen for Deep-TFC (Fig. 4). Here, both PIELM and X-TFC keep a great accuracy in the mean absolute error (e-08 and e-14, respectively). By decreasing the value of the diffusion coefficient at $\nu = 0.001$, the failure of Deep-TFC appears (Fig. 5), while PIELM and X-TFC keep the accuracy of e-08 and e-14, respectively. In the last, more complex case, $\nu = 0.0001$, all the methods fail, but X-TFC, which learns the solution with a mean absolute error of e-03. One can see that the training error is high even for X-TFC (4.87e2). This is due to the fact that all these methods collocate the DE residuals on the training points. Hence, when the salutation gets almost discontinuous, the derivative is not well computed. This issue can be possibly overcome by decomposing the domain into smaller sub-domains, as done in domain decomposition methods. Such an approach needs more investigation.

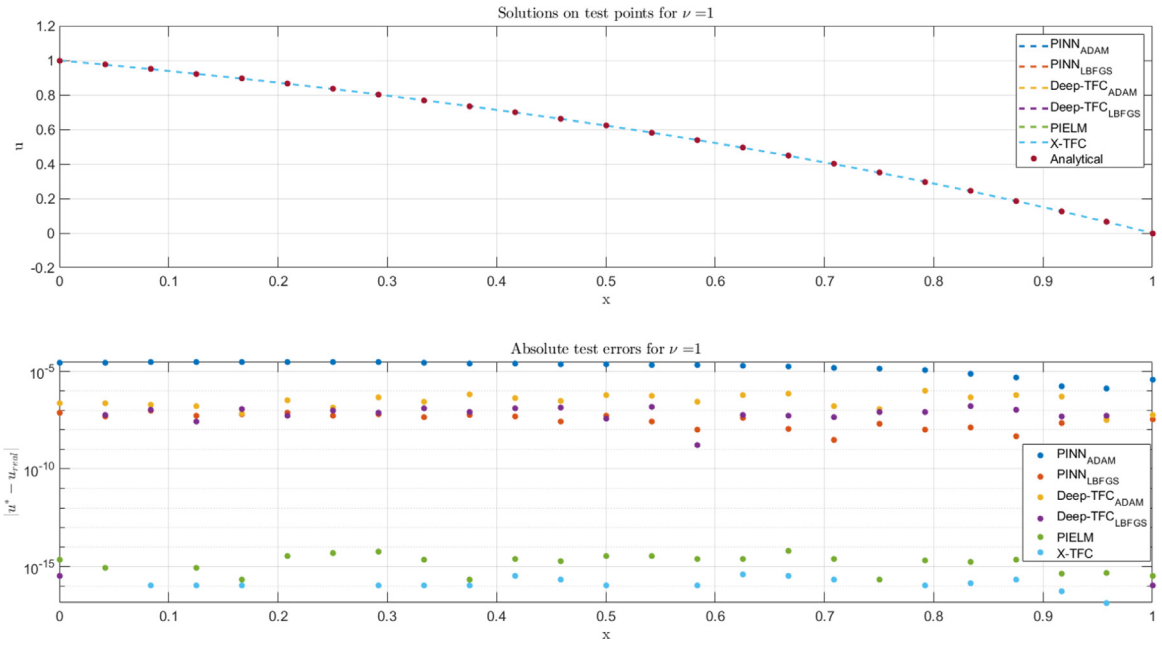

Fig. 2. $\nu = 1$.

Table 2
Results for $\nu = 1$. The superscript indicates the number of neurons per layer. The subscript indicates the gradient-based optimizer used. The same applies to all the other tables.

	PINN$_A^{2e1}$	PINN$_L^{2e1}$	Deep-TFC$_A^{2e1}$	Deep-TFC$_L^{2e1}$	PIELM^{1e2}	X-TFC^{1e2}
Training time [s]	10192.3	4855.51	11018.3	5019.69	0.02	0.02
ε_T	2.7e−04	4.0845e−05	6.10e−04	2.11e−04	5.26e−11	2.44e−12
ε_G	7.65e−06	3.34739e−08	2.35557e−07	6.49304e−08	2.99e−15	1.57e−16
max$\|u^* - u\|$	3.15e−05	9.96934e−08	1.04022e−06	1.619e−07	6.11e−15	3.89e−16
mean$\|u^* - u\|$	2.01e−05	4.02397e−08	3.70183e−07	7.62862e−08	2.10e−15	1.17e−16
std$\|u^* - u\|$	9.99e−06	2.52352e−08	2.5048e−07	4.67391e−08	1.68e−15	1.134e−16
$\|u^*(x_0) - u(x_0)\|$	2.65e−05	7.56815e−08	0	0	2.22e−15	0
$\|u^*(x_f) - u(x_f)\|$	3.64e−06	3.58488e−08	0	0	3.29e−16	0

Table 3
Results for $\nu = 0.1$.

	PINN$_A^{2e1}$	PINN$_L^{2e1}$	Deep-TFC$_A^{2e1}$	Deep-TFC$_L^{2e1}$	PIELM^{1e2}	X-TFC^{1e2}
Training time [s]	10483.3	4556.97	11756.7	5170.77	0.02	0.02
ε_T	1.03e−03	3.28e−04	4.35e−02	3.01e−02	3.33e−09	1.46e−09
ε_G	1.35075e−05	9.38758e−06	5.08616e−05	3.97102e−05	1.90e−13	7.31e−14
max$\|u^* - u\|$	5.7474e−05	1.47079e−05	1.03e−04	1.10e−04	3.18e−13	1.49e−13
mean$\|u^* - u\|$	4.95858e−05	1.27206e−05	6.35822e−05	4.55148e−05	1.64e−13	5.93e−14
std$\|u^* - u\|$	1.11992e−05	3.46304e−06	2.96194e−05	3.13723e−05	9.15e−14	4.29e−14
$\|u^*(x_0) - u(x_0)\|$	5.28097e−05	1.43572e−05	0	0	1.36e−13	0
$\|u^*(x_f) - u(x_f)\|$	1.3113e−05	2.40891e−07	0	0	5.89e−14	0

4. Conclusions

In this work, we explained four different PINN frameworks and tested them on a benchmark second-order ODE which presents sharp gradients while varying the DE parameter. Two of the PINN frameworks, classic PINN and Deep-TFC, employ standard DNNs; while two, PIELM and X-TFC use shallow NN trained via ELMs.

The frameworks employing ELMs outperform the ones using standard DNNs. In particular, X-TFC was able to solve the DE even when the solution was highly challenging because of vanishing diffusion.

X-TFC has been tested also in other different engineering and science fields leading to good performance, see e.g. [15–21]. Nevertheless, the reported tests in the standard linear 1d case lead to new insight into the strength of such a method.

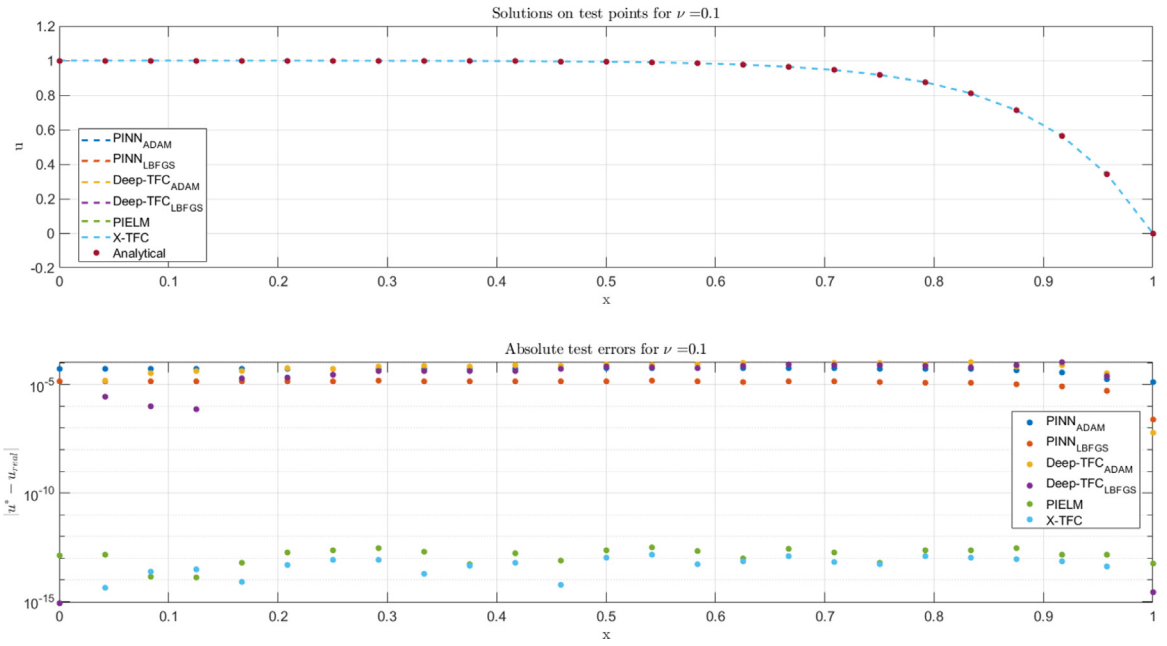

Fig. 3. $\nu = 0.1$.

Table 4
Results for $\nu = 0.01$.

	$\text{PINN}_\text{A}^{2e2}$	$\text{PINN}_\text{L}^{2e2}$	$\text{Deep-TFC}_\text{A}^{2e2}$	$\text{Deep-TFC}_\text{L}^{2e2}$	PIELM^{1e3}	X-TFC^{1e3}
Training time [s]	12298.9	43318	12400.3	35946.7	2.11	2.14
ε_T	3.53e−01	3.53e−01	3.58e0	1.85e−01	1.38e−06	1.23e−06
ε_G	3.50e−01	3.50e−01	1.08e−02	2.20906e−05	2.16e−08	6.70e−13
$\max\|u^* - u\|$	5.00e−01	5.00e−01	2.59e−02	6.18332e−05	2.18e−08	1.73e−12
$\text{mean}\|u^* - u\|$	5.00e−01	5.00e−01	1.26e−02	2.47546e−05	2.18e−08	9.45e−14
$\text{std}\|u^* - u\|$	3.10e−03	5.00e−01	8.34e−03	2.08108e−05	1.36e−10	3.43e−13
$\|u^*(x_0) - u(x_0)\|$	5.00e−01	5.00e−01	0	0	2.18e−08	0
$\|u^*(x_f) - u(x_f)\|$	5.00e−01	5.00e−01	0	0	2.18e−08	0

Table 5
Results for $\nu = 0.001$.

	$\text{PINN}_\text{A}^{1e3}$	$\text{PINN}_\text{L}^{1e3}$	$\text{Deep-TFC}_\text{A}^{1e3}$	$\text{Deep-TFC}_\text{L}^{1e3}$	PIELM^{1e4}	X-TFC^{1e4}
Training time [s]	19030.6	181125	19491.6	91380.8	156.06	153.78
ε_T	3.53e−01	3.53e−01	7.07e2	7.07e2	1.48e−04	6.83284e−06
ε_G	3.53e−01	3.53e−01	4.07e−01	4.07e−01	4.88e−05	1.18e−12
$\max\|u^* - u\|$	5.00e−01	5.00e−01	9.58e−01	9.58e−01	4.89e−05	2.83e−12
$\text{mean}\|u^* - u\|$	5.00e−01	5.00e−01	4.6e−01	4.59e−01	4.89e−05	7.09e−13
$\text{std}\|u^* - u\|$	1.33015e−05	1.00701e−05	3.04e−01	3.04e−01	4.27e−09	7.80e−13
$\|u^*(x_0) - u(x_0)\|$	5.00e−01	5.00e−01	0	0	4.89e−05	0
$\|u^*(x_f) - u(x_f)\|$	5.00e−01	5.00e−01	0	0	4.89e−05	0

Table 6
Results for $\nu = 0.0001$.

	$\text{PINN}_\text{A}^{1e3}$	$\text{PINN}_\text{L}^{1e3}$	$\text{Deep-TFC}_\text{A}^{1e3}$	$\text{Deep-TFC}_\text{L}^{1e3}$	PIELM^{1e4}	X-TFC^{1e4}
Training time [s]	19030.6	181125	19491.6	91380.8	155.60	153.81
ε_T	3.53e−01	3.53e−01	7.07e2	7.07e2	1e0	4.87e2
ε_G	3.53e−01	3.53e−01	4.07e−01	4.07e−01	5.0e−01	1.74e−03
$\max\|u^* - u\|$	5.00e−01	5.00e−01	9.58e−01	9.58e−01	5.0e−01	2.89e−03
$\text{mean}\|u^* - u\|$	5.00e−01	5.00e−01	4.6e−01	4.59e−01	5.0e−01	1.39e−03
$\text{std}\|u^* - u\|$	1.33015e−05	1.00701e−05	3.04e−01	3.04e−01	1.53e−13	9.18e−04
$\|u^*(x_0) - u(x_0)\|$	5.00e−01	5.00e−01	0	0	5.0e−01	0
$\|u^*(x_f) - u(x_f)\|$	5.00e−01	5.00e−01	0	0	5.0e−01	0

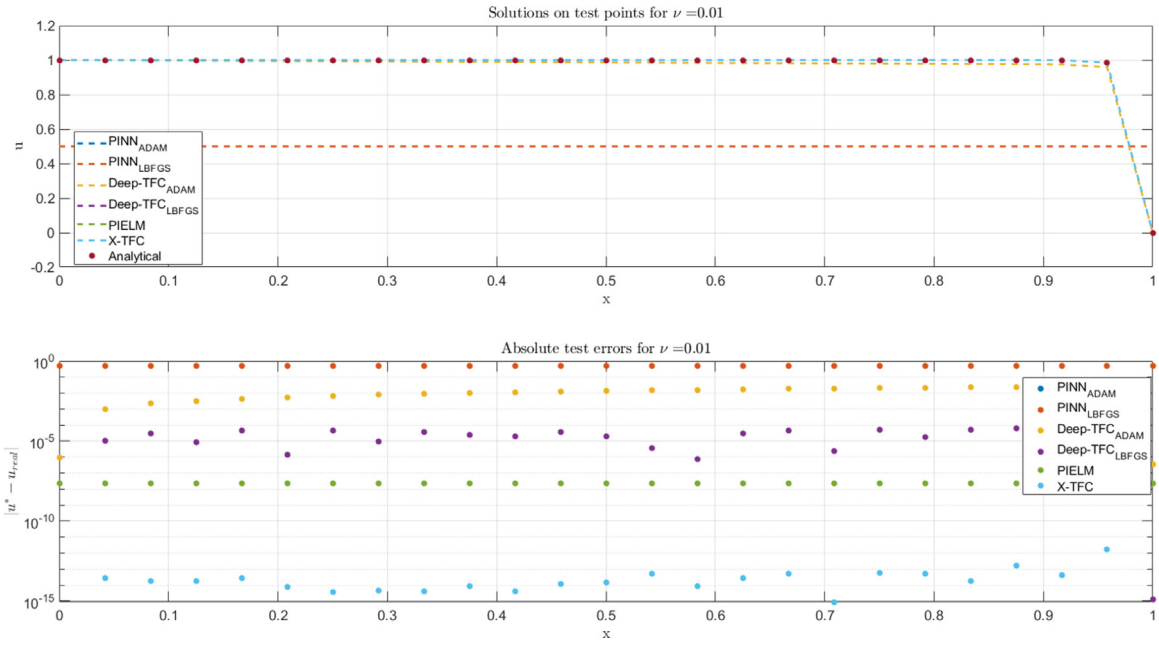

Fig. 4. $\nu = 0.01$.

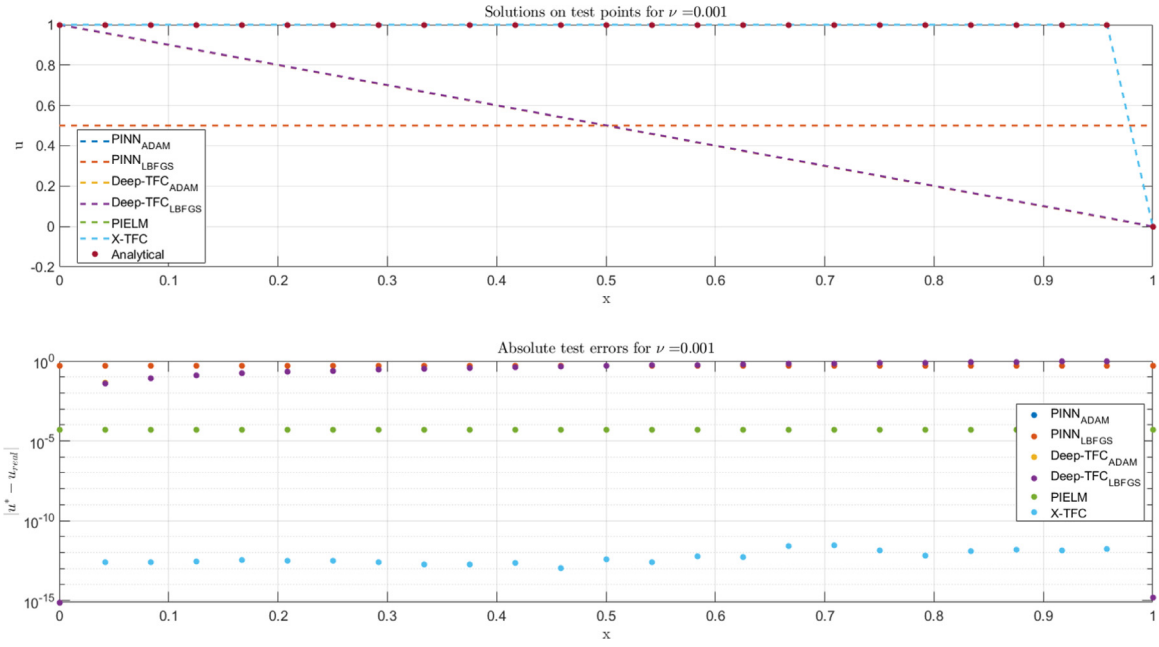

Fig. 5. $\nu = 0.001$.

Data availability

No data was used for the research described in the article.

Acknowledgments

F.C. was partially supported by the Istituto Nazionale di Alta Matematica - Gruppo Nazionale per il Calcolo Scientifico (INdAM-GNCS), Italy.

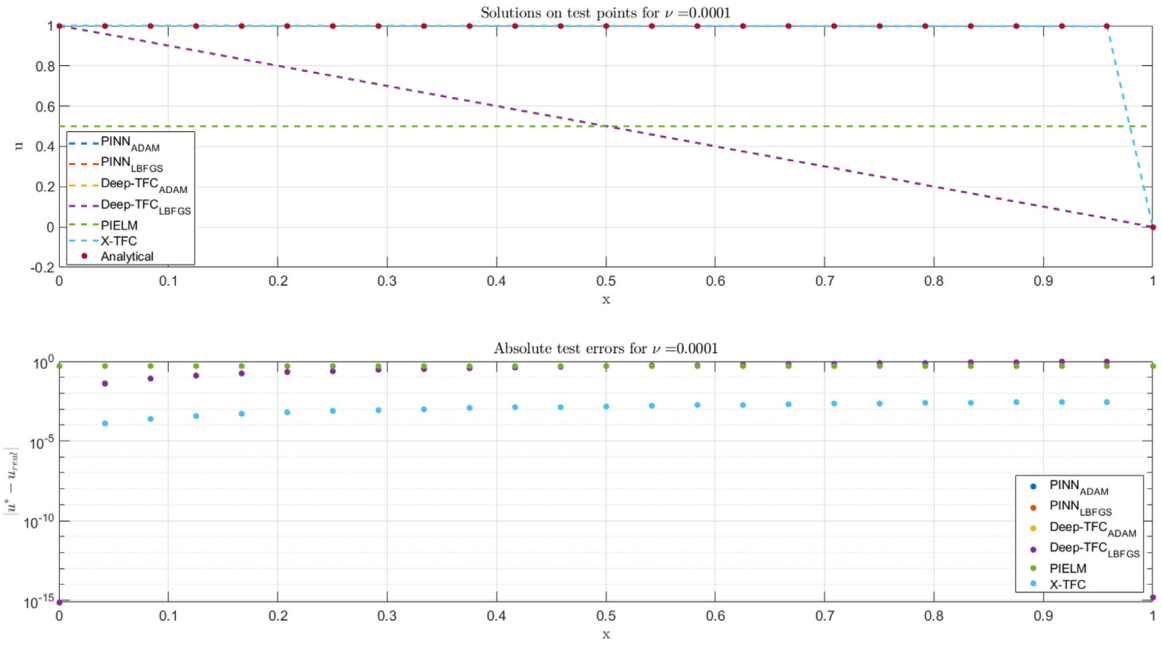

Fig. 6. $\nu = 0.0001$.

References

[1] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.

[2] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM J. Sci. Comput. 43 (5) (2021) A3055–A3081.

[3] C. Leake, D. Mortari, Deep theory of functional connections: A new method for estimating the solutions of partial differential equations, Mach. Learn. Knowl. Extr. 2 (1) (2020) 37–55.

[4] D. Mortari, The theory of connections: Connecting points, Mathematics 5 (4) (2017) 57.

[5] C. Leake, H. Johnston, D. Mortari, The Theory of Functional Connections: A Functional Interpolation. Framework with Applications, Lulu, Morrisville NC, 2022.

[6] C. Leake, H. Johnston, D. Mortari, The multivariate theory of functional connections: Theory, proofs, and application in partial differential equations, Mathematics 8 (8) (2020) 1303.

[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.

[8] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1–3) (2006) 489–501.

[9] F. Calabrò, G. Fabiani, C. Siettos, Extreme learning machine collocation for the numerical solution of elliptic PDEs with sharp gradients, Comput. Methods Appl. Mech. Engrg. 387 (2021) 114188.

[10] G. Fabiani, F. Calabrò, L. Russo, C. Siettos, Numerical solution and bifurcation analysis of nonlinear partial differential equations with extreme learning machines, J. Sci. Comput. 89 (2) (2021) 1–35.

[11] S. Dong, Z. Li, Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations, Comput. Methods Appl. Mech. Engrg. 387 (2021) 114129.

[12] E. Schiassi, R. Furfaro, C. Leake, M. De Florio, H. Johnston, D. Mortari, Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations, Neurocomputing 457 (2021) 334–356.

[13] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[14] S. Mishra, R. Molinaro, Estimates on the generalization error of physics-informed neural networks for approximating PDEs, IMA J. Numer. Anal. (2022).

[15] E. Schiassi, M. De Florio, B.D. Ganapol, P. Picca, R. Furfaro, Physics-informed neural networks for the point kinetics equations for nuclear reactor dynamics, Ann. Nucl. Energy 167 (2022) 108833.

[16] E. Schiassi, A. D'Ambrosio, K. Drozd, F. Curti, R. Furfaro, Physics-informed neural networks for optimal planar orbit transfers, J. Spacecr. Rockets 59 (3) (2022) 834–849.

[17] E. Schiassi, M. De Florio, A. D'ambrosio, D. Mortari, R. Furfaro, Physics-informed neural networks and functional interpolation for data-driven parameters discovery of epidemiological compartmental models, Mathematics 9 (17) (2021) 2069.

[18] M. De Florio, E. Schiassi, B.D. Ganapol, R. Furfaro, Physics-informed neural networks for rarefied-gas dynamics: Thermal creep flow in the Bhatnagar–Gross–Krook approximation, Phys. Fluids 33 (4) (2021) 047110.

[19] M. De Florio, E. Schiassi, R. Furfaro, Physics-informed neural networks and functional interpolation for stiff chemical kinetics, Chaos 32 (6) (2022) 063107.

[20] M. De Florio, E. Schiassi, B.D. Ganapol, R. Furfaro, Physics-Informed Neural Networks for rarefied-gas dynamics: Poiseuille flow in the BGK approximation, Z. Angew. Math. Phys. 73 (3) (2022) 1–18.

[21] E. Schiassi, A. D'Ambrosio, R. Furfaro, Bellman neural networks for the class of optimal control problems with integral quadratic cost, IEEE Trans. Artif. Intell. (2022).