
PHYSICS INFORMED EXTREME LEARNING MACHINE (PIELM)– A RAPID METHOD FOR THE NUMERICAL SOLUTION OF PARTIAL DIFFERENTIAL EQUATIONS

A PREPRINT

Vikas Dwivedi*

Department of Mechanical Engineering
Indian Institute of Technology, Madras
Chennai-600036, India
me15d080@smail.iitm.ac.in

Balaji Srinivasan

Department of Mechanical Engineering
Indian Institute of Technology, Madras
Chennai-600036, India
sbalaji@iitm.ac.in

July 9, 2019

ABSTRACT

There has been rapid progress recently on the application of deep networks to solution of partial differential equations, collectively labelled as Physics Informed Neural Networks (PINNs). In this paper, we develop Physics Informed Extreme Learning Machine (PIELM), a rapid version of PINNs which can be applied to stationary and time dependent linear partial differential equations. We demonstrate that PIELM matches or exceeds the accuracy of PINNs on a range of problems. We also discuss the limitations of neural network based approaches, including our PIELM, in the solution of PDEs on large domains and suggest an extension, a distributed version of our algorithm -- DPIELM. We show that DPIELM produces excellent results comparable to conventional numerical techniques in the solution of time-dependent problems. Collectively, this work contributes towards making the use of neural networks in the solution of partial differential equations in complex domains as a competitive alternative to conventional discretization techniques.

Keywords Partial differential equations · Physics informed neural networks · Extreme learning machine · Advection-diffusion equation

1 Introduction

Partial differential equations (PDEs) are extensively used in the mathematical modelling of various problems in physics, engineering and finance. In practical situations, these equations typically lack analytical solutions and are solved numerically. In current practice, most numerical approaches to solve PDEs like finite element method (FEM), finite difference method (FDM) and finite volume method (FVM) are mesh based. A typical implementation of a mesh based approach involves three steps: (1) Grid generation, (2) Discretization of governing equation and (3) Solution of the discretized equations with some iterative method.

However, there are limitations to these approaches. Some of the limitations of these methods are as follows:

1. They cannot be used to solve PDEs in complex computational domains because grid generation (step 1) itself becomes infeasible.
2. The process of discretization (step 2) creates a discrepancy between the mathematical nature of actual PDE and its approximate difference equation [1]. Sometimes this can lead to quite serious problems [2].

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

One of the options to fix these issues is to use neural networks. In this approach, the data set consists of some randomly selected points in the domain and on the boundary. The governing equations and the boundary conditions are fitted using neural network. There are two main motivations for this approach. First, being universal approximators, neural networks can potentially represent any PDE. So this avoids the discretization step and thus discretization based physics errors too. Second, it is meshfree and therefore complex geometries can be easily handled [3]. Initial work in this direction can be credited to Lagaris et al. [4, 5]. Firstly, they solved the initial boundary value problem using neural networks and later they extended their work to handle irregular boundaries. Since then, a lot of work has been done in this field [6, 7, 8, 9, 10, 11, 12, 13]. In particular, we refer to the physics-informed neural networks (PINN) approach by Raissi and Karniadakis [11] and Raissi et. al [12, 13]. This approach has produced promising results for a series of benchmark nonlinear problems.

Recently, Berg et al. [3] have developed a PINN based method to solve PDEs on complex domains and produced several results. However, in spite of various advantages of using deep networks for solving PDEs, PINNs have several problems [13]. Firstly, there is no theoretical basis to know the size of neural network architecture and the amount of data needed. Then, there is no guarantee that the algorithm will not hit upon a local minima. Finally, their learning time is slower than the traditional numerical methods making them very expensive for practical problems.

We show that some of the problems mentioned above can be easily handled by using an alternative network called the extreme learning machine (ELM). The basic ELM was proposed by Huang et al. [14] for a single hidden layer feed forward networks (SLFNs) and later it was extended to generalized SLFNs. The essence of ELM is that the weights of the hidden layer of SLFNs need not to be learnt. It is much faster than the traditional gradient based optimization methods alleviating the learning time problem. Previously, ELMs have been used in approximating functions [15] and solving ordinary differential equations (ODEs) and stationary PDEs [16, 17] using Legendre and Bernstein polynomial basis functions respectively.

In this paper, we propose a new machine learning algorithm to solve stationary and time dependent PDEs in complex geometries. We have named it physics informed extreme learning machine (PIELM) because it is a combination of two algorithms namely ELM and PINN. Theoretically, there is no question over the employment of ELM as a PDE solver because it is a universal approximator [18] and therefore it can approximate any PDE. We have made our ELM “physics informed” by incorporating the information about the physics of PDE as the cost function. In addition to this, we have also proposed an extension to original PIELM called distributed PIELM that enhances the representation power of PIELM without adding any extra hidden layers. We demonstrate that both PIELM and DPIELM exhibit superior performance on a range of stationary and time-dependent problems in comparison to existing methods.

This paper proceeds as follows. We give a brief review of PINN and ELM in Section 2. The proposed PIELM is described in Section 3. In Section 4, we evaluate the performance of PIELM in solving various stationary and time-dependent PDEs. To our knowledge, this is the first application of an ELM based algorithm to solve a 2D unsteady PDE. In Section 5, we discuss the limitation of PIELM to represent discontinuous functions and the functions with sharp gradient. We also illustrate a test case where even a deep PINN fails to represent a complicated function. In Section 6, DPIELM, the distributed version of PIELM for enhanced representation is described. In Section 7, the results of implementation of DPIELM algorithm in test cases involving representation of functions with sharp gradients have been presented. We have also shown that DPIELM outperforms the deep PINN in representing the complicated function described earlier. Finally, conclusion and future work are given in Section 8.

2 Brief review of ELM and PINN

The PIELM is combination of two learning algorithms: ELM and PINN. In this section, we review these two algorithms in brief.

2.1 Extreme learning machine

Traditional gradient-based learning algorithms [19] have prohibitively slow learning speed and they suffer from various problems like improper learning rate, local minima etc. Huang et al. [14] originally proposed a novel learning algorithm called ELM to fix these issues. ELM is extremely fast and mostly it shows better generalization performance than gradient-based learning approaches like back-propagation. A typical implementation of the algorithm involves the following steps:

1. Select a shallow neural network.
2. Fix the hidden layer weights and biases randomly. These parameters will not be learned and therefore no iterative optimization is required for them.

3. Apply a nonlinear transformation to the input data set. This gives the input to the final layer.
4. Take the linear combination of all the inputs of the final layer. This is the output of ELM.
5. Learn the output layer weights using the least squares method.

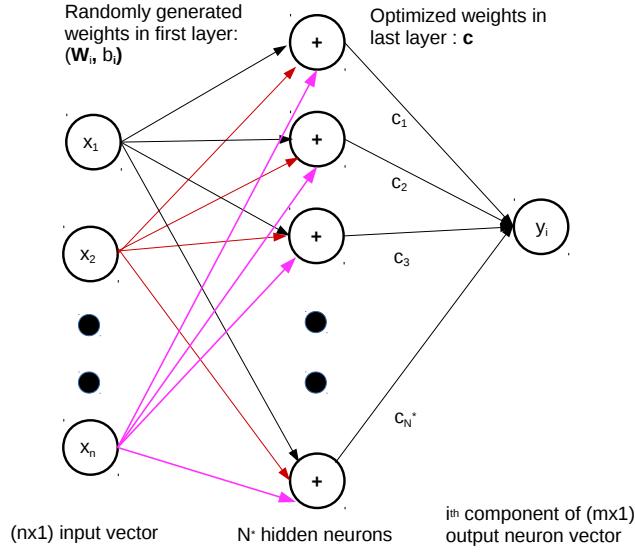


Figure 1: Basic structure of ELM

Mathematical formulation

Consider the basic ELM shown in Fig (1). It is a single layer feed forward neural network with N^* neurons in the hidden layer. Input is a vector of size n and output is the i^{th} component of the output vector of size m . We denote the non-linear activation by φ and the weights and biases of j^{th} node of hidden layer by $\vec{a}_j^{(i)}$ and $b_j^{(i)}$ respectively. The output of the hidden node j is $\varphi(\vec{x}; \vec{a}_j^{(i)}, b_j^{(i)})$. For a given data set $\{(x_k, y_k)\}_{k=1}^N \subset \mathbb{R}^n \times \mathbb{R}^m$ with N distinct samples, the ELM output is given by

$$\vec{y}_{ELM}^{(i)} = \mathbf{H}^{(i)}(\vec{x}) \vec{c} \quad (1)$$

where,

$$\begin{aligned} \mathbf{H}^{(i)} &= [\vec{h}^{(i)}(\vec{x}_1), \vec{h}^{(i)}(\vec{x}_2), \dots, \vec{h}^{(i)}(\vec{x}_N)]^T, \\ \vec{h}^{(i)}(\vec{x}_k) &= [\varphi(\vec{x}_k; \vec{a}_1^{(i)}, b_1^{(i)}), \varphi(\vec{x}_k; \vec{a}_2^{(i)}, b_2^{(i)}), \dots, \varphi(\vec{x}_k; \vec{a}_{N^*}^{(i)}, b_{N^*}^{(i)})], \end{aligned}$$

and $\vec{c} = [c_1, c_2, \dots, c_{N^*}]^T$ is vector of output layer weights.

On writing the Eq. (1) for all the m components, the resulting ELM is given by

$$\mathbf{H}^{(i)} \vec{c} = \vec{y}^{(i)} - \vec{\xi}^{(i)}, \quad i = 1, 2, \dots, m, \quad (2)$$

where $\vec{\xi}$ is the training error vector. The ELM tends to reach the smallest training error together with the smallest norm of the output weights. Mathematically saying, the loss function to be minimized for the ELM is given by

$$J = \frac{1}{2} \|\vec{c}\|^2 + \frac{1}{2N} \lambda \sum_{i=1}^m \vec{\xi}^{(i)T} \vec{\xi}^{(i)}, \quad (3)$$

where λ is the regularization parameter. The correct weights that minimize J can be calculated by solving the normal equations as given below.

$$\frac{\partial J}{\partial c_k} = 0, \quad k = 1, 2, \dots, N^* \quad (4)$$

2.2 Physics informed neural network

Raissi et al. [13] proposed a data efficient PINN for approximating solutions to general non-linear PDEs and validated it with a series of benchmark test cases. The main feature of the PINN is the inclusion of the prior knowledge of physics in the learning algorithm as cost function. As a result, the algorithm imposes penalty for any non-physical solution and quickly directs it towards the correct solution. This physics informed approach enhances the information content of the data. As a result, the algorithm has good generalization property even in the small data set regime.

Mathematical formulation

Consider a PDE of the following form

$$\frac{\partial}{\partial t} u(\vec{x}, t) + \mathcal{N}u(\vec{x}, t) = R(\vec{x}, t), (\vec{x}, t) \in \Omega \times [0, T], \quad (5)$$

$$u(\vec{x}, t) = B(\vec{x}, t), (\vec{x}, t) \in \partial \Omega \times [0, T], \quad (6)$$

$$u(\vec{x}, 0) = F(\vec{x}), \vec{x} \in \Omega, \quad (7)$$

where \mathcal{N} may be a linear or nonlinear differential operator and $\partial \Omega$ is the boundary of computational domain Ω . We approximate $u(\vec{x}, t)$ with the output $f(\vec{x}, t)$ of PINN. The network architecture may be shallow or deep depending upon the non-linearity \mathcal{N} . The essence of PINN lies in the definition of its loss function. In order to make the neural network “physics informed”, the loss function is defined such that a penalty is imposed whenever the network output doesn’t respect the physics of the problem. If we denote the training errors in approximating the PDE, BCs and IC by $\vec{\xi}_f$, $\vec{\xi}_{bc}$ and $\vec{\xi}_{ic}$ respectively. Then, the expressions for these errors are as follows:

$$\vec{\xi}_f = \frac{\partial \vec{f}}{\partial t} + \mathcal{N} \vec{f} - \vec{R}, (\vec{x}, t) \in \Omega \times [0, T], \quad (8)$$

$$\vec{\xi}_{bc} = \vec{f} - \vec{B}, (\vec{x}, t) \in \partial \Omega \times [0, T], \quad (9)$$

$$\vec{\xi}_{ic} = \vec{f}(., 0) - \vec{F}, \vec{x} \in \Omega. \quad (10)$$

For shallow networks, $\frac{\partial \vec{f}}{\partial t}$ and $\mathcal{N} \vec{f}$ can be determined using hand calculations. However, for deep networks, we have to use automatic differentiation [20]. The loss function J to be minimized for a PINN is given by

$$J = \frac{\vec{\xi}_f^T \vec{\xi}_f}{2N_f} + \frac{\vec{\xi}_{bc}^T \vec{\xi}_{bc}}{2N_{bc}} + \frac{\vec{\xi}_{ic}^T \vec{\xi}_{ic}}{2N_{ic}}, \quad (11)$$

where N_f , N_{bc} and N_{ic} refer to number of collocation points, boundary condition points and initial condition points respectively. Finally, any gradient based optimization routine may be used to minimize J .

This completes the mathematical formulation of PINN. The key steps in its implementation are as follows:

1. Identify the PDE to be solved along with the initial and boundary conditions.
2. Decide the architecture of PINN.
3. Approximate the correct solution with PINN.
4. Find expressions for the PDE, BCs and IC in terms of PINN and its derivatives.
5. Define a loss function which penalizes for error in PDE, BCs and IC.
6. Minimize the loss with gradient based algorithms.

3 Proposed PIELM

Consider the following unsteady linear PDE

$$\frac{\partial}{\partial t} u(\vec{x}, t) + \mathcal{L}u(\vec{x}, t) = R(\vec{x}, t), (\vec{x}, t) \in \Omega \times [0, T], \quad (12)$$

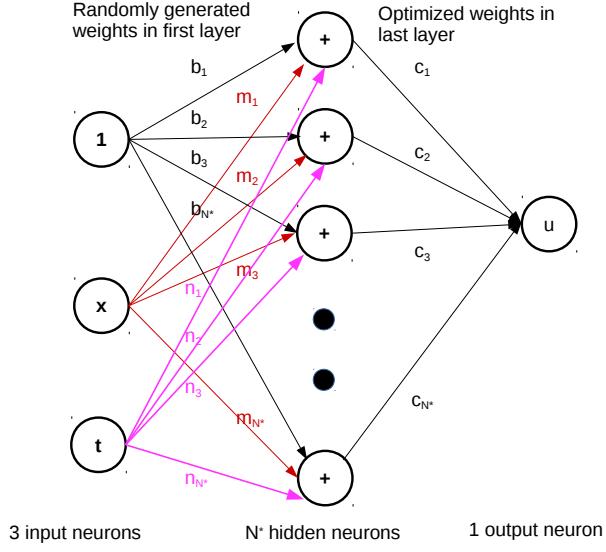


Figure 2: PIELM for 1D unsteady problems

$$u(\vec{x}, t) = B(\vec{x}, t), (\vec{x}, t) \in \Omega \times [0, T], \quad (13)$$

$$u(\vec{x}, 0) = F(\vec{x}), \vec{x} \in \Omega, \quad (14)$$

where \mathcal{L} is a linear differential operator and $\partial\Omega$ is the boundary of computational domain Ω . We approximate $u(\vec{x}, t)$ with the output $f(\vec{x}, t)$ of PIELM. For simplicity, we consider the 1D unsteady version of Eqns (12, 13, 14). The extension to higher dimensional problems is straightforward. The PIELM for 1D unsteady problem is schematically shown in Fig (2). The number of neurons in the hidden layers is N^* . If we define $\vec{\chi} = [x, t, 1]^T$, $\vec{m} = [m_1, m_2, \dots, m_{N^*}]^T$, $\vec{n} = [n_1, n_2, \dots, n_{N^*}]^T$, $\vec{b} = [b_1, b_2, \dots, b_{N^*}]^T$ and $\vec{c} = [c_1, c_2, \dots, c_{N^*}]^T$ then, the output of the k^{th} hidden neuron is

$$h_k = \varphi(z_k),$$

where $z_k = [m_k, n_k, b_k] \vec{\chi}$ and $\varphi = \tanh$ is the nonlinear activation function. The PIELM output is given by

$$f(\vec{\chi}) = \vec{h} \cdot \vec{c}. \quad (15)$$

Similarly, the formulae for $\frac{\partial^p f}{\partial x^p}$ and $\frac{\partial f}{\partial t}$ are given by

$$\frac{\partial^p f_k}{\partial x^p} = m_k^p \frac{\partial^p \varphi}{\partial z^p}, \quad (16)$$

$$\frac{\partial f_k}{\partial t} = n_k \frac{\partial \varphi}{\partial z}. \quad (17)$$

We denote the training errors in approximating the PDE, BCs and IC by $\vec{\xi}_f$, $\vec{\xi}_{bc}$ and $\vec{\xi}_{ic}$ respectively. The expressions for these errors are as follows:

$$\vec{\xi}_f = \frac{\partial \vec{f}}{\partial t} + \mathcal{L} \vec{f} - \vec{R}, \quad (\vec{x}, t) \in \Omega \times [0, T], \quad (18)$$

$$\vec{\xi}_{bc} = \vec{f} - \vec{B}, \quad (\vec{x}, t) \in \partial\Omega \times [0, T], \quad (19)$$

$$\vec{\xi}_{ic} = \vec{f}(., 0) - \vec{F}, \quad \vec{x} \in \Omega. \quad (20)$$

Next, we put a hard constraint on \vec{c} to solve the PDE exactly with zero error by setting

$$\vec{\xi}_f = \vec{0}, \quad (21)$$

$$\vec{\xi}_{bc} = \vec{0}, \quad (22)$$

$$\vec{\xi}_{ic} = \vec{0}. \quad (23)$$

Eqns (21 to 23) lead to the a system of linear equations which can be represented as

$$\mathbf{H}\vec{c} = \vec{K}. \quad (24)$$

The form of \mathbf{H} and \vec{K} depends on the \mathcal{L} , B and F i.e. on the type of PDE, boundary condition and initial condition. In order to find \vec{c} , Moore–Penrose generalized inverse [21] (also called pseudo-inverse) should be used as it works well for singular and non square \mathbf{H} too. An additional advantage with this formulation is that we have a basis to guess the scale of the PIELM architecture. When we are solving Eqns (21 to 23) simultaneously, we know that a unique solution will exist when number of unknowns are equal to number of equations which means that $N^* = N_f + N_{bc} + N_{ic}$. This gives us an idea of the size of the hidden layer. However, in practice we get the correct solution even with lesser number of neurons. For example, if we supply a large number of points to approximate a linear function, the PIELM would not require the same number of neurons for learning.

This completes the mathematical formulation of PIELM. The key steps in its implementation are as follows:

1. Assign the input layer weights randomly.
2. Depending on the PDE and the initial and boundary conditions, find the expressions for $\vec{\xi}_f$, $\vec{\xi}_{bc}$ and $\vec{\xi}_{ic}$.
3. Assemble the three sets of equations in the form of $\mathbf{H}\vec{c} = \vec{K}$.
4. Output layer weight vector is given by $\text{pinv}(\mathbf{H})\vec{K}$, where pinv refers to pseudo-inverse.

It is to be noted that unlike conventional ELMs , we are not solving an optimization problem. The loss function J to be minimized for a conventional physics informed ELM would be given by

$$J = \frac{1}{2} \|\vec{c}\|^2 + \frac{1}{2} \lambda \left(\frac{\vec{\xi}_f^T \vec{\xi}_f}{2N_f} + \frac{\vec{\xi}_{bc}^T \vec{\xi}_{bc}}{2N_{bc}} + \frac{\vec{\xi}_{ic}^T \vec{\xi}_{ic}}{2N_{ic}} \right), \quad (25)$$

where λ is a regularization parameter and N_f , N_{bc} and N_{ic} refer to number of collocation points, boundary condition points and initial condition points respectively. The correct ELM weights that minimize J can be calculated by solving the normal equations which is given below.

$$\frac{\partial J}{\partial c_k} = 0, \quad k = 1, 2, \dots, N^* \quad (26)$$

Although a PIELM can be made with this minimization approach, we have opted for the direct approach due to the following reasons:

1. The direct approach is straightforward to formulate and code. It saves the effort of calculating loss function and setting the derivatives equal to zero.
2. The learning of the minimization approach is comparatively less “physics informed” because physics is not being imposed in an exact sense.

Steady/ Unsteady	1D/2D	Test case ID	Description
Steady	1D	TC-1	Linear advection
		TC-2	Linear diffusion
		TC-3	Linear advection-diffusion
	2D	TC-4	Linear advection in a star shaped computational domain
		TC-5	Linear diffusion in a star shaped computational domain
		TC-6	Linear diffusion in a complex computational domain
Unsteady	1D	TC-7	Linear advection
		TC-8	Quasi-linear advection
		TC-9	Linear advection-diffusion
	2D	TC-10	Linear advection-diffusion

Table 1: List of test cases for PIELM.

4 Performance evaluation of PIELM

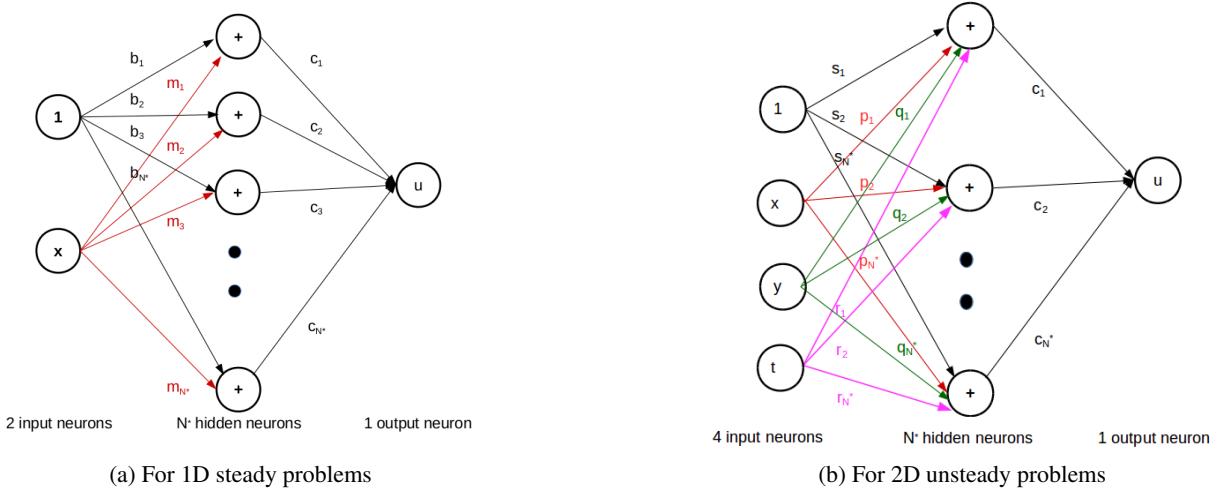


Figure 3: PIELMs for steady 1D and unsteady 2D problems

To evaluate the performance of PIELM, we rigorously test it on various linear and quasi-linear PDEs described in Table(1). TC-1, TC-2, TC-4, TC-5, TC-6 are taken from Berg et al. [3]. TC-8 is taken from Kopriva et al. [22]. TC-9 and TC-10 are taken from Borker et al. [23]. All the experiments are conducted in Matlab 2017b environment running in an Intel Core i5 2.20GHz CPU and 8GB RAM Dell laptop. The error is defined as the difference between the PIELM prediction and the exact solution.

4.1 1D steady cases [TC-1, TC-2, TC-3]

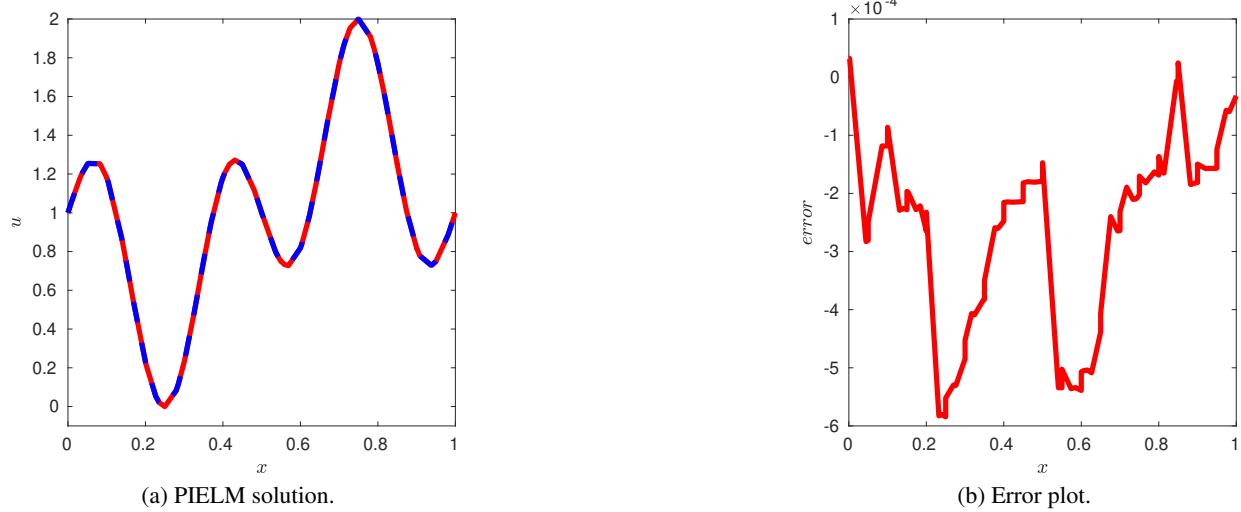


Figure 4: Solution and error for steady 1D advection. Red: PIELM, Blue: Exact.

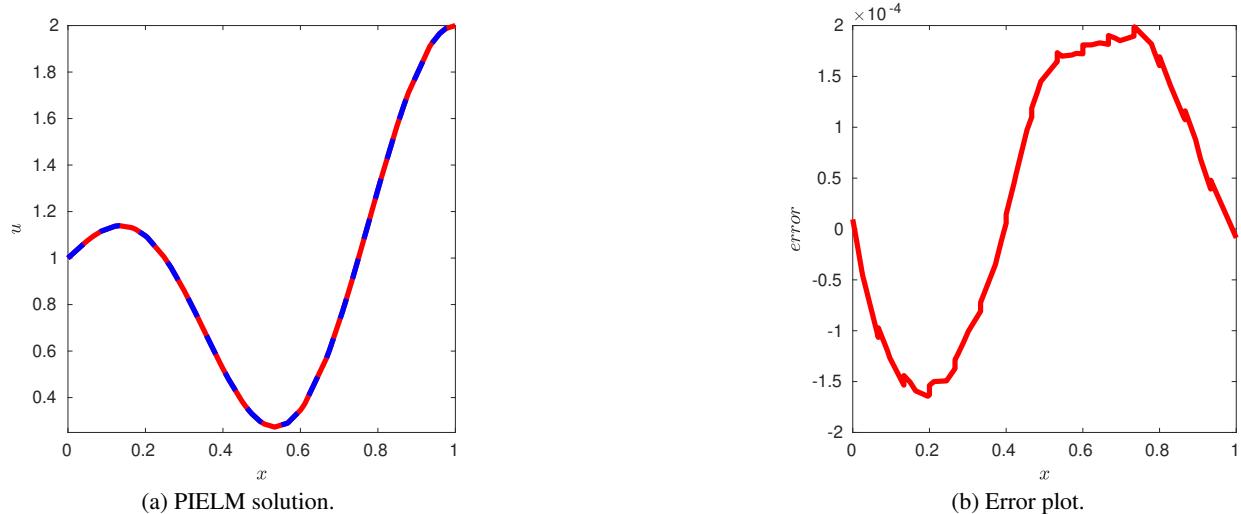


Figure 5: Solution and error for steady 1D diffusion. Red: PIELM, Blue: Exact.

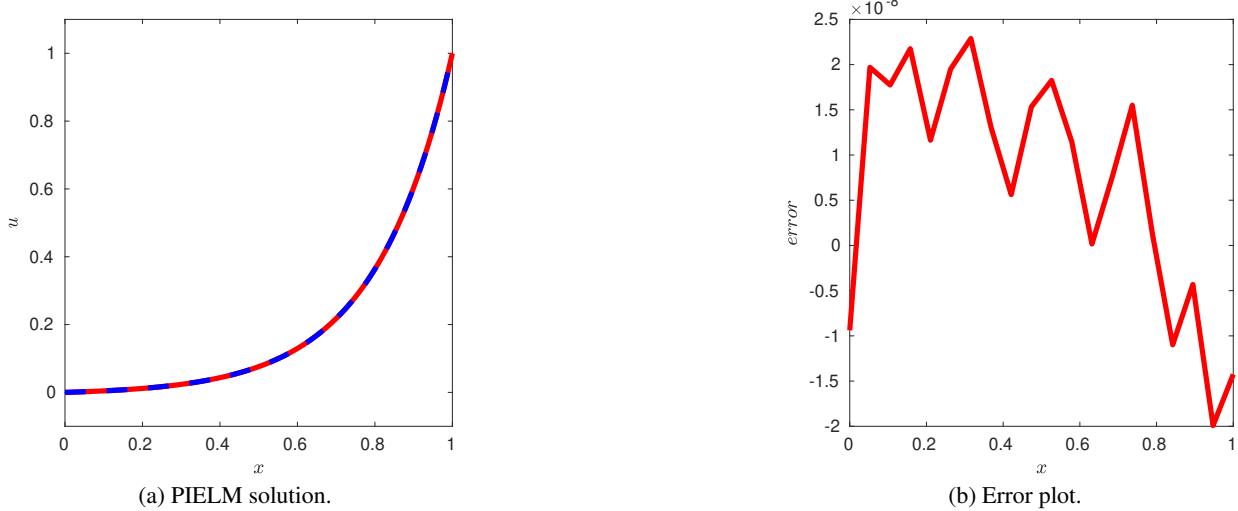


Figure 6: Solution and error for 1D steady advection diffusion at $\nu = 0.2$. Red: PIELM, Blue: Exact.

The 1D stationary advection, diffusion and advection-diffusion equations are given by

$$u_x = R, 0 < x \leq 1, \quad (27)$$

$$u_{xx} = R, 0 < x \leq 1, \quad (28)$$

$$u_x - \nu u_{xx} = R, 0 < x < 1 \quad (29)$$

respectively. The expressions for R and the Dirichlet boundary conditions for these cases are calculated by assuming the following exact solutions

$$\hat{u} = \sin(2\pi x)\cos(4\pi x) + 1, \quad (30)$$

$$\hat{u} = \sin\left(\frac{\pi x}{2}\right)\cos(2\pi x) + 1, \quad (31)$$

$$\hat{u} = \frac{e^{\frac{x}{\nu}} - 1}{e^{\frac{1}{\nu}} - 1} \quad (32)$$

respectively. In order to solve these equations in PIELM framework, we have to solve Eqn(21 to 23). The expression for $\vec{\xi}_f$ depends on the linear differential operator \mathcal{L} . The definitions of \mathcal{L} in these cases are $\frac{\partial}{\partial x}$, $\frac{\partial^2}{\partial x^2}$ and $\frac{\partial}{\partial x} - \nu \frac{\partial^2}{\partial x^2}$ respectively. When \mathcal{L} acts on u , the corresponding expressions for $\vec{\xi}_f = \vec{0}$ may be written as follows:

$$\varphi'(\mathbf{X}_f \mathbf{W}^T) \odot \vec{c} \odot \vec{m} = R(\vec{x}_f), \quad (33)$$

$$\varphi''(\mathbf{X}_f \mathbf{W}^T) \odot \vec{c} \odot \vec{m} \odot \vec{m} = R(\vec{x}_f), \quad (34)$$

$$\varphi'(\mathbf{X}_f \mathbf{W}^T) \odot \vec{c} \odot \vec{m} - \nu \varphi''(\mathbf{X}_f \mathbf{W}^T) \odot \vec{c} \odot \vec{m} \odot \vec{m} = \vec{0}. \quad (35)$$

where \vec{x}_f is collocation points vector, \vec{I} is bias vector, $\mathbf{X}_f = [\vec{x}_f, \vec{I}]$ and ' \odot ' refers to Hadamardt product. Referring to Fig (3a), $\mathbf{W} = [\vec{m}, \vec{b}]$. Similarly, expression for $\vec{\xi}_{bc} = \vec{0}$ is given by

$$\varphi(\mathbf{X}_{bc} \mathbf{W}^T) \vec{c} = B(\vec{x}_{bc}) \quad (36)$$

where \vec{x}_{bc} is boundary points vector, $\mathbf{X}_{bc} = [\vec{x}_{bc}, \vec{I}]$ and B is the boundary condition.

The results for these test cases are given in Fig(4), Fig(5) and Fig (6) respectively and the summary of the experiments is given in Tab(2).

TC	$[N_f, N_{bc}, N^*]$	$\mathcal{O}(\text{Error})$
TC-1	$[40, 2, 42]$	10^{-4}
TC-2	$[40, 2, 42]$	10^{-4}
TC-3	$[20, 2, 22]$	10^{-6}

Table 2: Summary of experiments for 1D steady test cases.

Remark

1. It should be noted that the unified deep ANN algorithm [3] took 100 points to achieve an order of accuracy of 10^{-5} and 10^{-3} in TC-1 and TC-2 respectively. In comparison, PIELM took less than half of the points and still achieved an order of accuracy of 10^{-4} in both the cases.

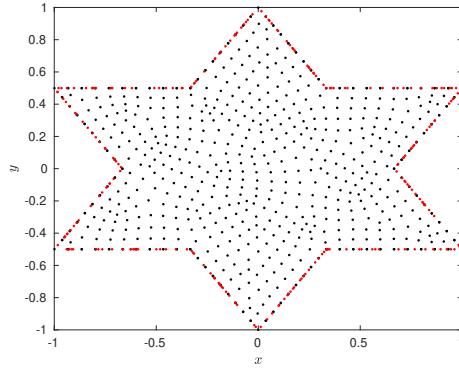
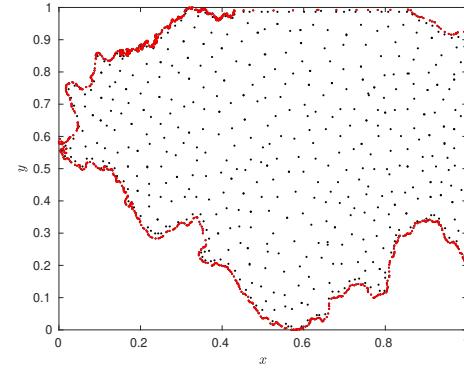
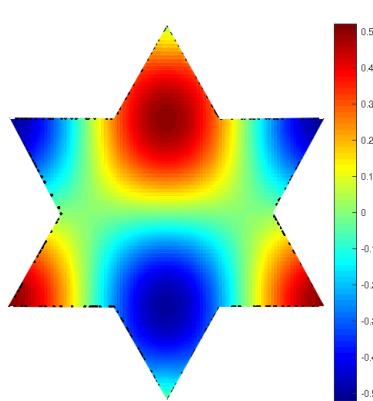
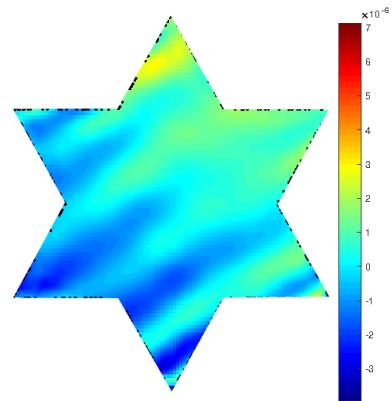
4.2 2D steady cases [TC-4, TC-5, TC-6](a) Ω_1 (b) Ω_2

Figure 7: Computational domains for 2D steady problems



(a) PIELM solution.



(b) Error plot.

Figure 8: Solution and error for 2D steady advection equation on Ω_1 .

Figure 9: Solution and error for 2D steady diffusion equation on Ω_1 .

Figure 10: Solution and error for 2D diffusion equation in a complex 2D geometry.

The stationary 2D advection and diffusion equations for the three cases are given by

$$au_x + bu_y = R, (x, y) \in \Omega_1 \quad (37)$$

$$u_{xx} + u_{yy} = R, (x, y) \in \Omega_1 \quad (38)$$

$$u_{xx} + u_{yy} = R, (x, y) \in \Omega_2 \quad (39)$$

where $a = 1$ and $b = \frac{1}{2}$ are advection coefficients. The computational domains Ω_1 and Ω_2 are shown in respectively. The expressions for R and the B for these cases are constructed by choosing the following exact solutions

$$\hat{u} = \frac{1}{2} \cos(\pi x) \sin(\pi y), \quad (40)$$

$$\hat{u} = \frac{1}{2} + e^{-(2x^2+4y^2)}, \quad (41)$$

$$\hat{u} = \frac{1}{2} + e^{-((x-0.6)^2+(y-0.6)^2)} \quad (42)$$

respectively. PIELM equations for these problems are as follows:

$$1. \vec{\xi}_f = \vec{0}$$

- Case 1: Advection

$$\varphi'(\mathbf{X}_f \mathbf{W}^T) \odot \vec{c} \odot (a\vec{m} + b\vec{n}) = R(\vec{x}_f, \vec{y}_f) \quad (43)$$

- Case 2: Diffusion

$$\varphi''(\mathbf{X}_f \mathbf{W}^T) \odot \vec{c} \odot (\vec{m} \odot \vec{m} + \vec{n} \odot \vec{n}) = R(\vec{x}_f, \vec{y}_f) \quad (44)$$

where $\mathbf{X}_f = [\vec{x}_f, \vec{y}_f, \vec{I}]$ and $\mathbf{W} = [\vec{m}, \vec{n}, \vec{b}]$ (refer Fig (2)).

$$2. \vec{\xi}_{bc} = \vec{0}$$

$$\varphi(\mathbf{X}_{bc} \mathbf{W}^T) \vec{c} = B(\vec{x}_{bc}, \vec{y}_{bc}) \quad (45)$$

where $\mathbf{X}_{bc} = [\vec{x}_{bc}, \vec{y}_{bc}, \vec{I}]$.

The results for the advection and diffusion cases on Ω_1 are shown in Fig (8) and Fig (9) respectively. The result for diffusion case on Ω_2 is given in Fig (10). Summary of the experiments is given in

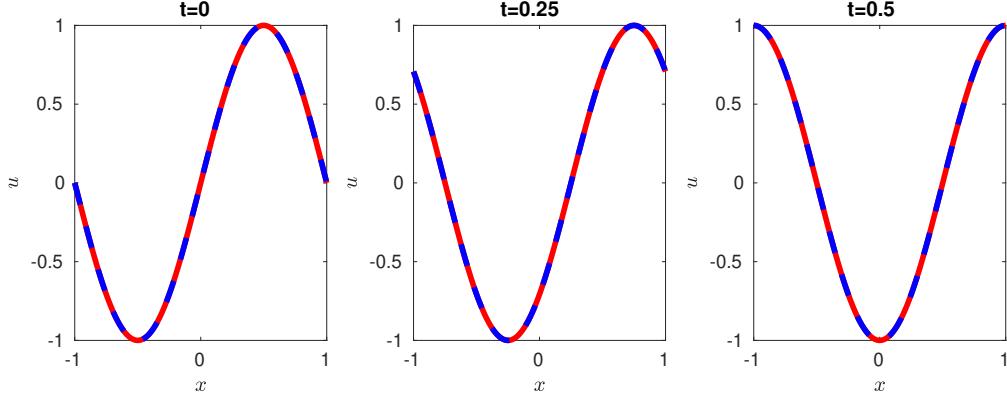
TC	$[N_f, N_{BC}, N^*]$	$\mathcal{O}(Error)$
TC-4	[921, 240, 2000]	10^{-6}
TC-5	[921, 240, 2000]	10^{-4}
TC-6	[1489, 881, 5000]	10^{-7}

Table 3: Summary of experiments for 2D steady test cases.

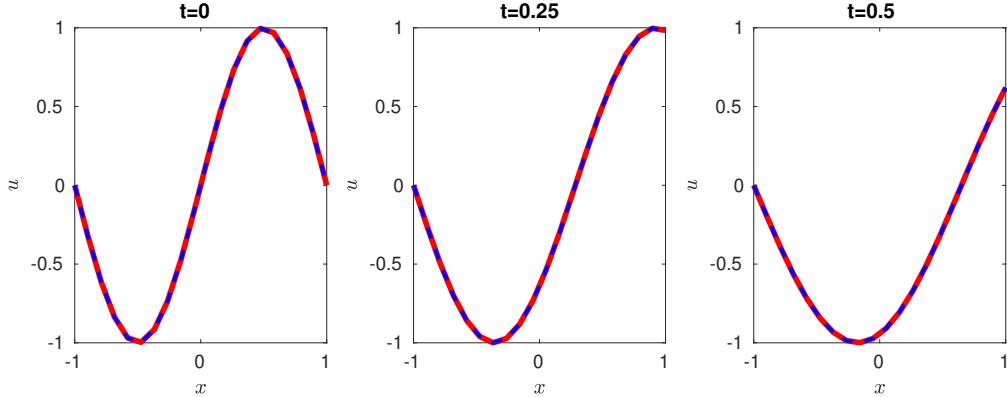
Remarks

1. The unified deep ANN algorithm [3] took 5500 points to achieve an order of accuracy of 10^{-3} in TC-4 and TC-5. In comparison, PIELM solved these cases with merely 1161 points and still achieved an order of accuracy of 10^{-6} and 10^{-4} respectively.
2. False diffusion is an error which gives diffusion like appearance to solution of pure advection equation when it is solved using upwind discretization. In Fig (8) these errors can be seen flowing along streamlines. However, PIELM reduces the order of these errors to an insignificant level.
3. Computational domain Ω_2 is the map of state of Illinois in USA. This is an example of a complicated polygon which has very short line segments and fine grained details in various regions. Conventional mesh based methods are not feasible for these kind of geometries. The latitudes and longitudes of the boundary are available in MATLAB's in-built function "usamap". We have re-scaled the data in the range 0 – 1. PIELM solved this case with just 2370 points to an accuracy level of 10^{-7} in just 10 seconds.

4.3 1D unsteady advection cases [TC-7, TC-8]



(a) Constant coefficient unsteady advection.



(b) Variable coefficient unsteady advection.

Figure 11: Exact and PIELM solution for 1D unsteady advection with constant and variable coefficients. Red: PIELM, Blue: Exact.

The unsteady 1D advection equation is given by

$$u_t + a(x)u_x = 0, \quad (x, t) \in [-1, 1] \times [0, 0.5], \quad (46)$$

$$u(x, 0) = F(x), \quad x \in [-1, 1], \quad (47)$$

where $a(x, t)$ is the advection coefficient. In this problem, we consider two cases: (1) constant coefficient case with $a(x, t) = 1$ and (2) variable coefficient case [22] with $a(x, t) = 1 + x$. The two cases have periodic and inflow boundary conditions respectively. The value of F is $\sin(\pi x)$. The exact solutions to the 1D unsteady advection problems with constant and variable coefficient are respectively given by

$$\hat{u} = F(x - t), \quad (48)$$

$$\hat{u} = F((1 + x)e^{-t} - 1). \quad (49)$$

PIELM equations

$$1. \quad \vec{\xi}_f = \vec{0}$$

- TC-7: Linear case

$$\varphi'(\mathbf{X}_f \mathbf{W}^T) \odot \vec{c} \odot (\vec{m} + \vec{n}) = \vec{0} \quad (50)$$

- TC-8: Quasi-linear case

$$\varphi'(\mathbf{X}_f \mathbf{W}^T) \vec{c} \odot \vec{n} + (\varphi'(\mathbf{X}_f \mathbf{W}^T) \odot \mathbf{A}_f) \vec{c} \odot \vec{m} = \vec{0} \quad (51)$$

where \vec{x}_f, \vec{t}_f are collocation point vectors, $\mathbf{X}_f = [\vec{x}_f, \vec{t}_f, \vec{l}]$, $\mathbf{W} = [\vec{m}, \vec{n}, \vec{b}]$ and $\mathbf{A}_f = [a(\vec{x}_f, \vec{t}_f), \dots, a(\vec{x}_f, \vec{t}_f)]_{N \times N^*}$.

2. $\vec{\xi}_{bc} = \vec{0}$

$$\varphi(\mathbf{X}_{lbc} \mathbf{W}^T) \vec{c} = \varphi(\mathbf{X}_{rbc} \mathbf{W}^T) \vec{c} \quad (52)$$

where $\vec{x}_{lbc}, \vec{t}_{lbc}$ are left boundary points vectors, $\vec{x}_{rbc}, \vec{t}_{rbc}$ are right boundary points vectors, $\mathbf{X}_{lbc} = [\vec{x}_{lbc}, \vec{t}_{lbc}, \vec{l}]$ and $\mathbf{X}_{rbc} = [\vec{x}_{rbc}, \vec{t}_{rbc}, \vec{l}]$.

3. $\vec{\xi}_{ic} = \vec{0}$

$$\varphi(\mathbf{X}_{ic} \mathbf{W}^T) \vec{c} = F(\vec{x}_{ic}, \vec{t}_{ic}) \quad (53)$$

where $\vec{x}_{ic}, \vec{t}_{ic}$ are initial condition vectors, $\mathbf{X}_{ic} = [\vec{x}_{ic}, \vec{t}_{ic}, \vec{l}]$ and F is initial condition.

The results are shown in Fig (11). PIELM predicts the exact solution correctly for both linear and quasi-linear advection. In this case, we took $N_f = 420$, $N_{bc} = 21$, $N_{ic} = 20$ and $N^* = 440$. The total learning time is within 2-3 seconds.

Remark

1. For advection problems, the time step of the traditional numerical schemes like upwinding can not exceed the mesh size due to stability issues. However, PIELM doesn't impose any such restriction and we can take larger time steps.

4.3.1 1D unsteady advection-diffusion [TC-9]

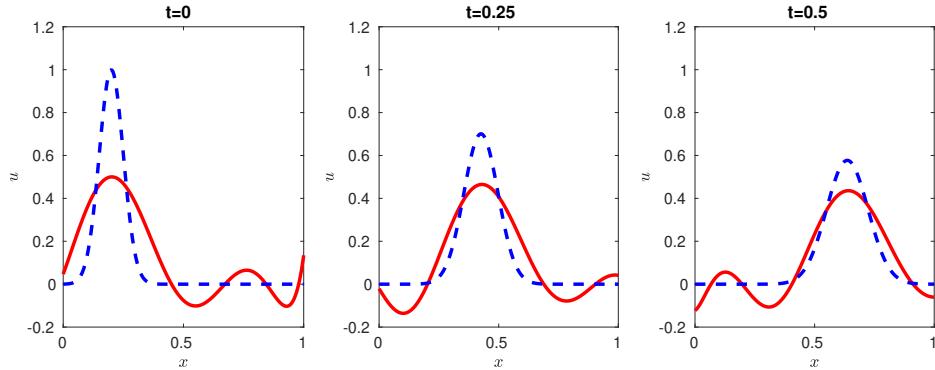


Figure 12: Exact and PIELM solution for unsteady 1D convection diffusion at $\nu = 0.005$. Red: PIELM, Blue: Exact.

The 1D equivalent of the unsteady 2D advection-diffusion equation solved by Borker et al. [23] is given by

$$u_t + au_x = \nu u_{xx}, (x, t) \in [0, 1] \times [0, 0.5], \quad (54)$$

where a is the advection coefficient. The expressions for the initial condition F and the boundary condition B are constructed on the basis of the following exact solution

$$\hat{u} = \frac{1}{\sqrt{4t+1}} e^{-\frac{1}{\nu(4t+1)}(x-0.2-at)^2} \quad (55)$$

The PIELM equations are as follows:

1. $\vec{\xi}_f = \vec{0}$

$$\varphi'(\mathbf{X}_f \mathbf{W}^T) \vec{c} \odot \vec{n} + a\varphi'(\mathbf{X}_f \mathbf{W}^T) \vec{c} \odot \vec{m} = \nu\varphi''(\mathbf{X}_f \mathbf{W}^T) \odot \vec{c} \odot \vec{m} \odot \vec{m} \quad (56)$$

where \vec{x}_f, \vec{t}_f are collocation point vectors, $\mathbf{X}_f = [\vec{x}_f, \vec{t}_f, \vec{l}]$ and $\mathbf{W} = [\vec{m}, \vec{n}, \vec{b}]$.

2. $\vec{\xi}_{bc} = \vec{0}$

$$\varphi(\mathbf{X}_{bc}\mathbf{W}^T)\vec{c} = B(\vec{x}_{bc}, \vec{t}_{bc}) \quad (57)$$

where \vec{x}_{bc} , \vec{t}_{bc} are boundary points vectors, $\mathbf{X}_{bc} = [\vec{x}_{bc}, \vec{t}_{bc}, \vec{I}]$ and B is the boundary condition.

3. $\vec{\xi}_{ic} = \vec{0}$

$$\varphi(\mathbf{X}_{ic}\mathbf{W}^T)\vec{c} = F(\vec{x}_{ic}, \vec{t}_{ic}) \quad (58)$$

where \vec{x}_{ic} , \vec{t}_{ic} are initial condition vectors, $\mathbf{X}_{ic} = [\vec{x}_{ic}, \vec{t}_{ic}, \vec{I}]$ and F is initial condition.

The results are shown in Fig (12). In this case, PIELM prediction clearly goes wrong in the following grounds:

1. Initial and boundary conditions aren't captured correctly.
2. Solution has unphysical oscillations throughout the domain.
3. The hump of the Gaussian decays a lot slower than the correct rate.

We increased the size of hidden layer but didn't see any improvement in results. For example, we took 250x30 points in the computational domain and put as many as 7780 neurons in the hidden layer. Still the PIELM predictions were poor.

4.3.2 2D unsteady advection-diffusion [TC-10]

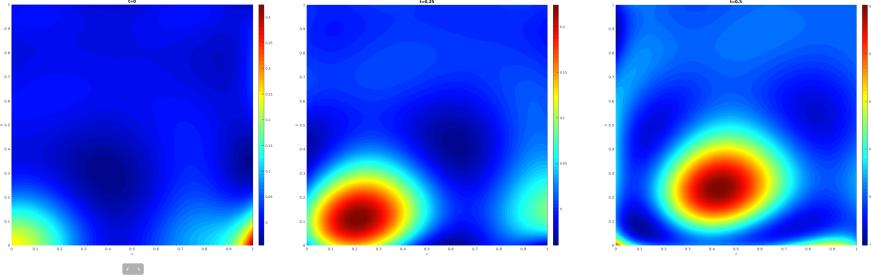


Figure 13: PIELM solution for 2D unsteady advection diffusion

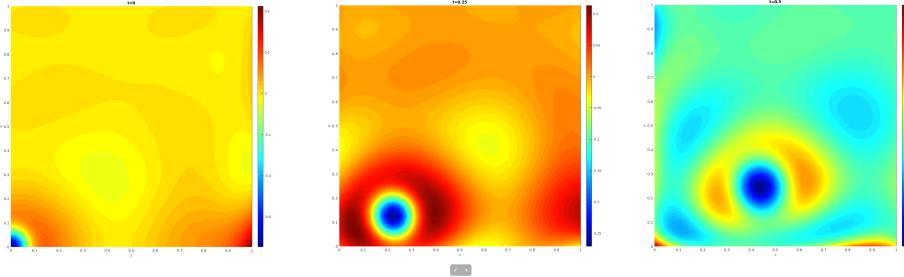


Figure 14: Error for 2D unsteady advection diffusion

We solve the following unsteady 2D advection-diffusion equation[23]

$$u_t + au_x + bu_y = \nu(u_{xx} + u_{yy}), (x, y, t) \in \Omega_{xy} \times [0, 0.2], \quad (59)$$

$$u(x, y, 0) = F(x, y), (x, y) \in \Omega_{xy}, \quad (60)$$

$$u(x, y, t) = B(x, y, t), (x, y, t) \in \partial\Omega_{xy} \times [0, 0.5], \quad (61)$$

where $\Omega_{xy} = [0, 1] \times [0, 1]$. $a = \cos(22.5)$ and $b = \sin(22.5)$ are advection coefficients in x and y directions and $\nu = 0.005$ is the diffusion coefficient. The expressions for the initial and the boundary conditions are constructed by choosing the following exact solution:

$$\hat{u} = \frac{1}{(4t+1)} e^{-\frac{(x-at)^2+(y-bt)^2}{\nu(4t+1)}} \quad (62)$$

PIELM equations to be solved are as follows:

$$1. \vec{\xi}_f = \vec{0}$$

$$\varphi'(\mathbf{X}_f \mathbf{W}^T) \vec{c} \odot \vec{r}' + \varphi'(\mathbf{X}_f \mathbf{W}^T) \vec{c} \odot (a \vec{p} + b \vec{q}) = \nu \varphi''(\mathbf{X}_f \mathbf{W}^T) \odot \vec{c} \odot (\vec{p} \odot \vec{p} + \vec{q} \odot \vec{q}) \quad (63)$$

where $\vec{x}_f, \vec{y}_f, \vec{t}_f$ are collocation point vectors, $\mathbf{X}_f = [\vec{x}_f, \vec{y}_f, \vec{t}_f, \vec{l}]$. Referring to Fig (3b), $\mathbf{W} = [\vec{p}, \vec{q}, \vec{r}, \vec{s}]$.

$$2. \vec{\xi}_{bc} = \vec{0}$$

$$\varphi(\mathbf{X}_{bc} \mathbf{W}^T) \vec{c} = B(\vec{x}_{bc}, \vec{y}_{bc}, \vec{t}_{bc}) \quad (64)$$

where $\vec{x}_{bc}, \vec{y}_{bc}, \vec{t}_{bc}$ are boundary points vectors and $\mathbf{X}_{bc} = [\vec{x}_{bc}, \vec{y}_{bc}, \vec{t}_{bc}, \vec{l}]$.

$$3. \vec{\xi}_{ic} = \vec{0}$$

$$\varphi(\mathbf{X}_{ic} \mathbf{W}^T) \vec{c} = F(\vec{x}_{ic}, \vec{y}_{ic}, \vec{t}_{ic}) \quad (65)$$

where $\vec{x}_{ic}, \vec{y}_{ic}, \vec{t}_{ic}$ are initial condition vectors and $\mathbf{X}_{ic} = [\vec{x}_{ic}, \vec{y}_{ic}, \vec{t}_{ic}, \vec{l}]$.

The results are shown in Fig (13-14). They are 2D equivalents of 1D results. The solution is diffusive and contains unphysical oscillations throughout the domain. In spite of all these issues, it should be noted that:

1. The advection component of the equation is captured correctly. The two humps are moving at the same speed.
2. The non-physical oscillations don't grow with time.

For this case, we have taken a total of 125000 data points and 1000 neurons in the hidden layer. The results show signs of improvement if we keep increasing the number of points. However, that takes a lot of time which makes the option impractical. The limitations of the PIEML will be further investigated in the next section. We close this section by summarizing the advantages of the PIEML which are as follows:

1. It is extremely fast as well as data efficient.(TC-1 to TC-6)
2. It can be seamlessly extended to higher dimensions. (TC-10)
3. It reduces numerical artefacts like false diffusion.(TC-4)
4. It is meshfree method and can handle the complex geometries.(TC-6)
5. It reduces the arbitrariness of the number of neurons in the hidden layer.

5 Limitations of PIEML

	Test case ID	Description
Representation of functions with PIEML	TC-11	Representation of sharp and discontinuous functions in 1D
	TC-12	Representation of a sharp peaked 2D Gaussian
Solution of PDEs with PIEML	TC-13	TC-7 with $F = e^{-100x^2}$
	TC-14	TC-3 with $\nu = 0.02$
Solution of PDEs with PINN	TC-15	TC-7 with $F = e^{-5x^2} \sin(10\pi x)$

Table 4: List of numerical experiments that show limitations of PIEML and PINN

A potential reason for the failure of PIEML in solving advection-diffusion equation [23] could be the limited representation capacity of PIEML to represent a complex function. A PDE consists of function and its derivatives. If a neural network cannot represent the function itself, then calculation of derivatives only adds to the error.

Representation of functions with sharp gradients [TC-11, TC-12]

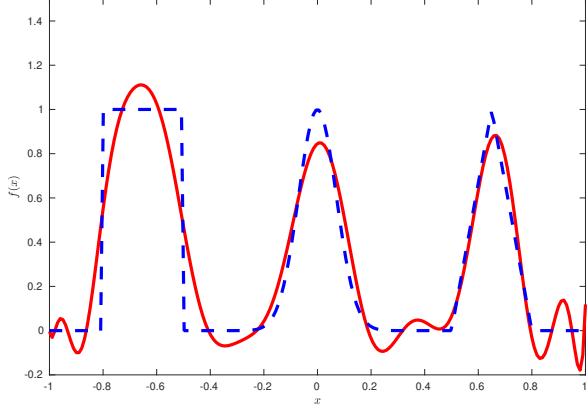


Figure 15: Representation of 1D non smooth functions with PIELM. Red: PIELM solution, Blue: exact solution.

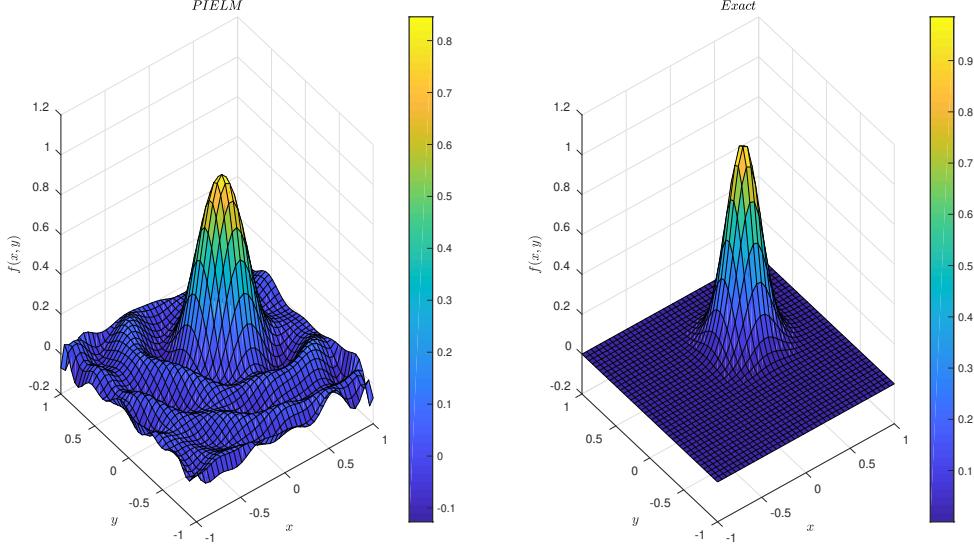


Figure 16: Representation of a sharp peaked 2D Gaussian with PIELM

To put the representation power of PIELM to test we choose the following two functions: (1) A 1D composite function that contains both discontinuous functions and functions with sharp gradients and (2) A 2D Gaussian function with a sharp peak. The expressions for 2D Gaussian function and 1D composite function are $f(x, y, t) = e^{-20\{(x-0.25)^2 + (y-0.25)^2\}}$ and

$$f(x) = \begin{cases} \frac{1}{2}\{sgn(x+0.8) - sgn(x+0.5)\} & x \leq -\frac{1}{2} \\ e^{-100x^2} & -\frac{1}{2} < x \leq \frac{1}{2} \\ \frac{20}{3}x - \frac{10}{3} & \frac{1}{2} < x \leq \frac{13}{20} \\ -\frac{20}{3}x + \frac{16}{3} & \frac{13}{20} < x \leq \frac{4}{5} \\ 0 & x > \frac{4}{5} \end{cases} \quad (66)$$

respectively. The results are shown in figure (15)and Fig (16) respectively. These cases clearly expose the limitation of PIELM in representing profiles with sharp gradients and corners.

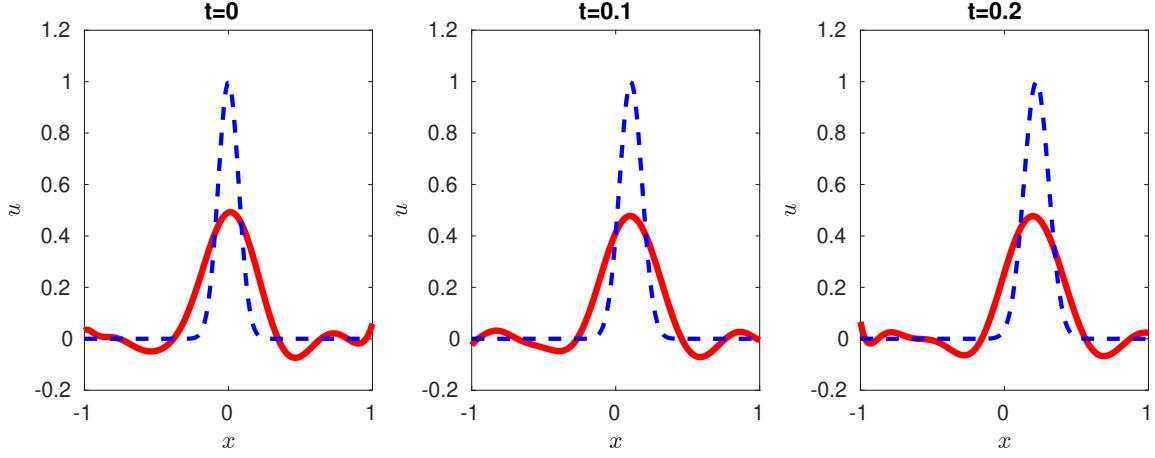
PDEs with sharp solutions [TC-13, TC-14]

Figure 17: Exact and PIELM solution of 1D advection of a sharp peaked Gaussian with PIELM. Red: PIELM, Blue: exact.

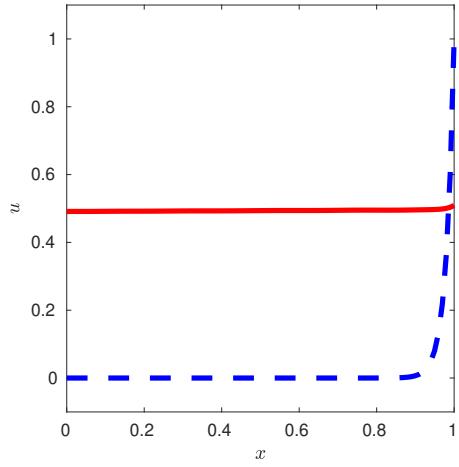


Figure 18: Steady 1D convection diffusion at $\nu = 0.02$. Red: PIELM, Blue: Exact.

Due to this limitation, PIELM fails to solve any PDE which admits functions with sharp gradients. For example, we have already seen the failure of our algorithm in solving 1D and 2D advection-diffusion equation. We further illustrate the impact of this limitation on two simpler equations. Firstly, we consider pure advection of a sharp peaked Gaussian. This equation has been solved in TC-7 for a smooth sine function. Next, we take steady advection-diffusion equation (TC-3) with a low value of diffusion coefficient.

The exact and PIELM solutions for these problems are shown in Fig (17) and Fig (18) respectively.

Representation of a high frequency wavelet with PINN [TC-15]

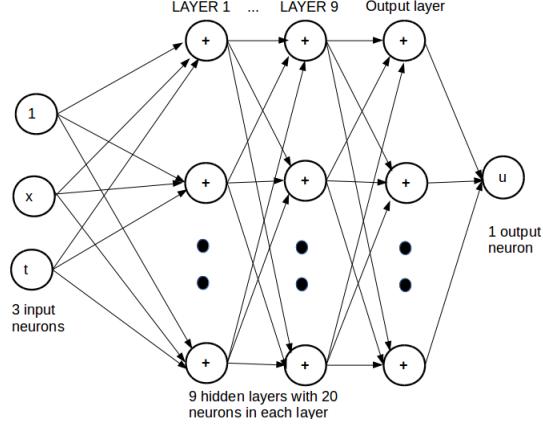


Figure 19: Deep PINN architecture

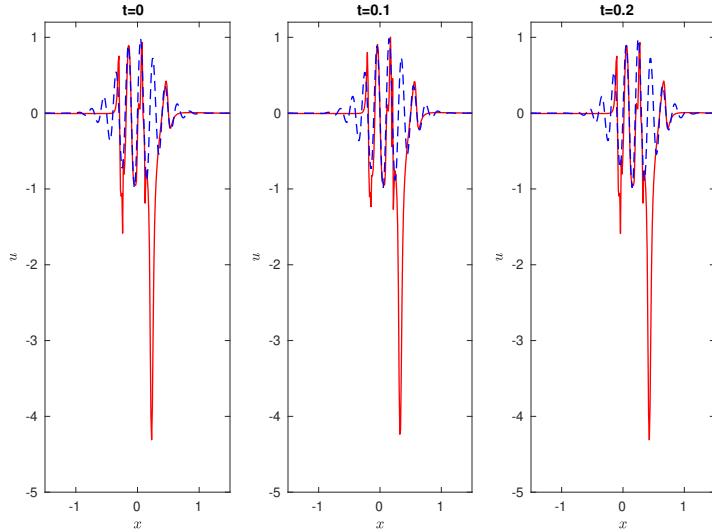


Figure 20: Exact and PINN solution of pure advection of a high frequency wave packet. Red: PINN, Blue: exact

An obvious idea to improve the representation capacity of a neural network is to make it deep. Therefore, we test the performance of PINN in TC-7 with $F = e^{-5x^2} \sin(10\pi x)$ i.e. pure advection of a high frequency wavelet. The PINN code is freely available at <https://github.com/maziarraissi/PINNs>. We modified the original code by replacing the Burgers equation with pure linear advection equation. Fig(19) shows the architecture of PINN. It consists of 9 hidden layers with 20 neurons each. Each layer is activated by tanh functions.

The exact and the PINN solutions are shown in Fig (20). We can see that even a deep network is unable to capture the sharp gradients of this wavelet.

List of the experiments conducted in this section are given in Tab 4. We summarize this section as follows:

1. The main limitation of a PIELM is its inability to represent complex functions.
2. This limitation restricts the PIELM to solve the PDEs with sharp exact solutions.
3. Adding extra layers is not the practical solution to this problem. (TC-15)

6 Distributed PIELM

In this section, we propose a distributed version of PIELM called DPIELM. This algorithm takes motivation from finite volume methods in which the whole computational domain is partitioned into multiple cells and governing equations are solved at each cell. The solutions of these individual cells are stitched together with additional convective and diffusive fluxes conditions at the cell interfaces. We adopt a similar strategy in DPIELM. As representation of a complex function is very hard for a single PIELM or PINN in the whole domain, we divide the domain into multiple cells and install a PIELM in each cell. Therefore, each PIELM uses different representations in different portions of domain while satisfying some additional constraints of continuity and differentiability.

6.1 Mathematical formulation

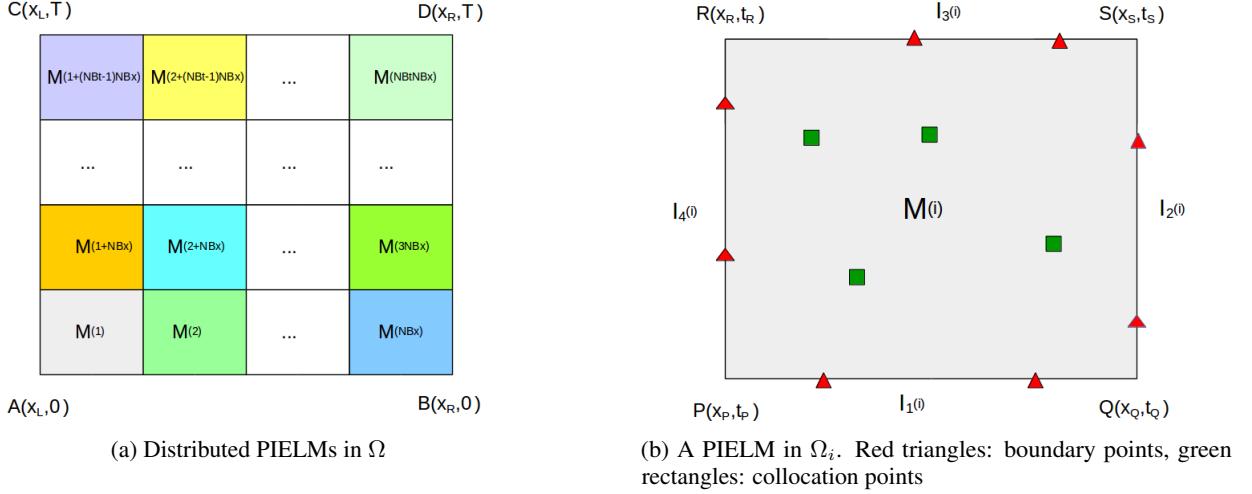


Figure 21: DPIELM architecture for full domain and an individual cell.

Consider the following 1D unsteady problem

$$\frac{\partial}{\partial t} u(x, t) + \mathcal{L}u(x, t) = R(x, t), (x, t) \in \Omega \quad (67)$$

$$u(x, t) = B(x, t), (x, t) \in \partial\Omega, \quad (68)$$

$$u(x, 0) = F(x), x \in [x_L, x_R], \quad (69)$$

where \mathcal{L} is a linear differential operator and $\partial\Omega$ is the boundary of computational domain Ω . In this problem, the rectangular domain Ω is given by $\Omega = [x_L, x_R] \times [0, T]$. On uniformly dividing Ω into N_c non-overlapping rectangular cells, Ω may be written as

$$\Omega = \bigcup_{i=1}^{N_c} \Omega_i. \quad (70)$$

The boundary of the cell Ω_i is denoted by $\partial\Omega_i$. For rectangular cells,

$$\partial\Omega_i = \bigcup_{m=1}^4 I_m^{(i)} \quad (71)$$

where $I_m^{(i)}$ represents the m^{th} interface of Ω_i .

Fig (21a) shows the distribution of PIELMs in a rectangular computational domain with $N_B \times N_B$ cells (i.e. $N_c = N_B \times N_B$). We denote the PIELM on the i^{th} cell by $M^{(i)}$. Fig (21b) shows a PIELM with collocation points at

the interior and the boundary points at the four interfaces. The weights and output corresponding to a given $M^{(i)}$ are denoted by $[\vec{m}^{(i)}, \vec{n}^{(i)}, \vec{b}^{(i)}, \vec{c}^{(i)}]$ and $f^{(i)}$ respectively.

At each $M^{(i)}$, we enforce additional constraints of continuity (or smoothness) of solution at the cell interfaces depending on the differential operator \mathcal{L} . For example, continuity of solution is sufficient for advection problems. For diffusion problem, the solution should be continuously differentiable. For the computational domain shown in the Fig (21), the system of equations to be solved in the DPIELM framework are given below.

6.1.1 Regular PIELM equations

$$1. \quad \vec{\xi}_f^{(i)} = \vec{0} \quad \left(\frac{\partial \vec{f}^{(i)}}{\partial t} + \mathcal{L} \vec{f}^{(i)} - \vec{R}^{(i)} \right)_{\Omega_i} = \vec{0}, i = 1, 2, \dots, N_c \quad (72)$$

$$2. \quad \vec{\xi}_{bc}^{(i)} = \vec{0} \quad (\vec{f}^{(i)} - \vec{B}^{(i)})_{I_4} = \vec{0}, i = 1, (1 + NB_x), \dots, (1 + (NB_t - 1)NB_x) \quad (73)$$

$$(\vec{f}^{(i)} - \vec{B}^{(i)})_{I_2} = \vec{0}, i = NB_x, 2NB_x, \dots, NB_t \times NB_x \quad (74)$$

$$3. \quad \vec{\xi}_{ic}^{(i)} = \vec{0} \quad (\vec{f}^{(i)} - \vec{F}^{(i)})_{I_1} = \vec{0}, i = 1, 2, \dots, NB_x \quad (75)$$

6.1.2 Additional interface equations

$$1. \text{ Constraints for } C^0 \text{ solutions i.e. } \vec{\xi}_{C^0}^{(i)} = \vec{0}.$$

- Continuity along x direction

$$\left\{ \begin{array}{c} \vec{f}^{(\kappa(1)+i-1)} \\ \vec{f}^{(\kappa(2)+i-1)} \\ \dots \\ \vec{f}^{(\kappa(NB_t)+i-1)} \end{array} \right\}_{I_2} = \left\{ \begin{array}{c} \vec{f}^{(\kappa(1)+i)} \\ \vec{f}^{(\kappa(2)+i)} \\ \dots \\ \vec{f}^{(\kappa(NB_t)+i)} \end{array} \right\}_{I_4}, i = 1, 2, \dots, NB_x - 1 \quad (76)$$

where $\kappa = [1, (1 + NB_x), \dots, 1 + (NB_t - 1)NB_x]^T$.

- Continuity along t direction

$$\left\{ \begin{array}{c} \vec{f}^{(\kappa(1)+(i-1)NB_x)} \\ \vec{f}^{(\kappa(2)+(i-1)NB_x)} \\ \dots \\ \vec{f}^{(\kappa(NB_x)+(i-1)NB_x)} \end{array} \right\}_{I_3} = \left\{ \begin{array}{c} \vec{f}^{(\kappa(1)+iNB_x)} \\ \vec{f}^{(\kappa(2)+iNB_x)} \\ \dots \\ \vec{f}^{(\kappa(NB_x)+iNB_x)} \end{array} \right\}_{I_1}, i = 1, 2, \dots, NB_t - 1 \quad (77)$$

where $\kappa = [1, 2, \dots, NB_x]^T$.

$$2. \text{ Constraints for } C^1 \text{ solutions i.e. } \vec{\xi}_{C^1}^{(i)} = \vec{0}.$$

- Smooth solutions along x direction

$$\frac{\partial}{\partial x} \left\{ \begin{array}{c} \vec{f}^{(\kappa(1)+i-1)} \\ \vec{f}^{(\kappa(2)+i-1)} \\ \dots \\ \vec{f}^{(\kappa(NB_t)+i-1)} \end{array} \right\}_{I_2} = \frac{\partial}{\partial x} \left\{ \begin{array}{c} \vec{f}^{(\kappa(1)+i)} \\ \vec{f}^{(\kappa(2)+i)} \\ \dots \\ \vec{f}^{(\kappa(NB_t)+i)} \end{array} \right\}_{I_4}, i = 1, 2, \dots, NB_x - 1 \quad (78)$$

where $\kappa = [1, (1 + NB_x), \dots, 1 + (NB_t - 1)NB_x]^T$.

Assembly of Eqns (72 to 78) leads to a system of linear equations which can be represented as

$$\mathbf{H} \vec{c} = \vec{K}, \quad (79)$$

where $\vec{c} = [\vec{c}^{(1)}, \vec{c}^{(2)}, \dots, \vec{c}^{(N_c)}]^T$. The form of \mathbf{H} and \vec{K} depends on the \mathcal{L} , B and F . Finally \vec{c} can be found using pseudo inverse. It is to be noted that although we have shown the formulation for 1D unsteady problems, no special adjustment is needed to extend the formulation to higher dimensional problems.

This completes the mathematical formulation of DPIELM. The main steps in its implementation are as follows:

1. Divide the computational domain into uniformly distributed non overlapping cells and install a PIELM in each cell.
2. Depending on the cell location, PDE and the initial and boundary conditions, find the expressions for $\vec{\xi}_f^{(i)}$, $\vec{\xi}_{bc}^{(i)}$ and $\vec{\xi}_{ic}^{(i)}$ at each cell.
3. Depending on the PDE, find the expressions for $\vec{\xi}_{C^0}^{(i)}$ and $\vec{\xi}_{C^1}^{(i)}$ at each cell interface.
4. Assemble these equations in the form of $\mathbf{H}\vec{c} = \vec{K}$, where $\vec{c} = [\vec{c}^{(1)}, \vec{c}^{(2)}, \dots, \vec{c}^{(N_c)}]^T$.
5. Find the value of \vec{c} using pseudo inverse.

7 Performance evaluation of DPIELM

Test Case ID	Description	Architecture
TC-9	1D unsteady linear advection-diffusion	[10, 10, 5, 5, 30]
TC-10	2D unsteady linear advection-diffusion equation	[20, 20, 50, 3, 3, 3, 30]
TC-11	Representation of 1D sharp and discontinuous functions	[50, 5, 5]
TC-12	Representation of a sharp peaked 2D Gaussian with DPIELM	[15, 15, 5, 5, 15]
TC-13	Advection of a sharp peaked Gaussian	[15, 10, 5, 5, 30]
TC-14	1D steady advection-diffusion	[20, 5, 20]
TC-15	Advection of a high frequency wave packet	[15, 10, 5, 5, 30]

Table 5: Details of DPIELM architecture for the test cases. For 1D steady problems, architecture is given by $[NB_x, nb_x, N_{cell}^*]$, where NB_x , nb_x and N_{cell}^* refer to number of cells, number of points in the cell and size of hidden layer of the PIELM. Similarly, for 1D and 2D unsteady problems, it is given by $[NB_x, NB_t, nb_x, nb_t, N_{cell}^*]$ and $[NB_x, NB_y, NB_t, nb_x, nb_y, nb_t, N_{cell}^*]$ respectively.

In this section, we evaluate the performance of DPIELMs by testing it on all the cases in which regular PIELM and PINN failed to perform. The details of the architecture is given in Tab(5).

Representation of functions with sharp gradient [TC-11, TC-12]

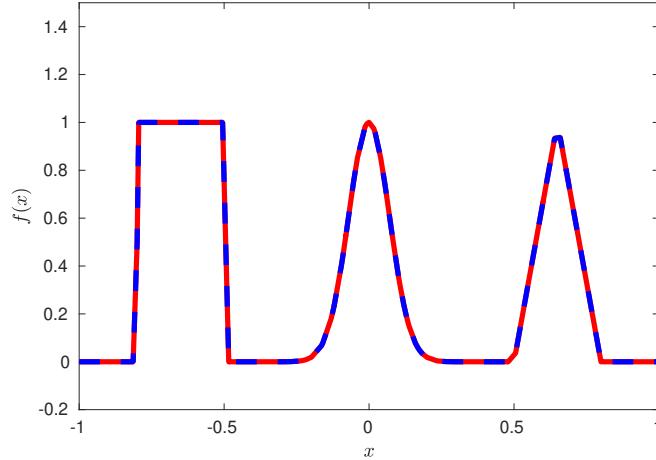


Figure 22: Representation of 1D non smooth functions with DPIELM. Red: DPIELM, Blue: Exact.

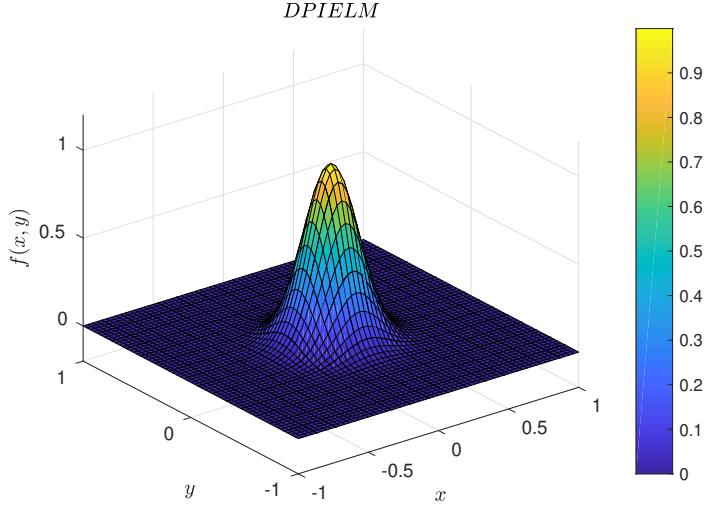


Figure 23: Representation of a sharp peaked 2D Gaussian with DPIELM

The mathematical formulation of DPIELM equations for 1D case are as follows:

$$1. \quad \vec{\xi}_f^{(i)} = \vec{0} \\ \varphi(\mathbf{X}_f^{(i)} \mathbf{W}^{(i)^T}) \odot \vec{c}^{(i)} = f(\vec{x}_f^{(i)}), i = 1, 2, \dots, NB_x \quad (80)$$

$$2. \quad \vec{\xi}_{bc}^{(i)} = \vec{0} \\ \left\{ \begin{array}{l} \varphi(\mathbf{X}_{bc, I_1}^{(1)} \mathbf{W}^{(1)^T}) \vec{c}^{(1)} \\ \varphi(\mathbf{X}_{bc, I_2}^{(NB_x)} \mathbf{W}^{(NB_x)^T}) \vec{c}^{(NB_x)} \end{array} \right\} = \left\{ \begin{array}{l} f(\vec{x}_{bc, I_1}^{(1)}) \\ f(\vec{x}_{bc, I_2}^{(NB_x)}) \end{array} \right\} \quad (81)$$

where I_1, I_2 refer to left and right cell interfaces respectively.

$$3. \quad \vec{\xi}_{C^0}^{(i)} = \vec{0} \\ \varphi(\mathbf{X}_{bc, I_2}^{(i)} \mathbf{W}^{(i)^T}) \vec{c}^{(i)} = \varphi(\mathbf{X}_{bc, I_1}^{(i+1)} \mathbf{W}^{(i+1)^T}) \vec{c}^{(i+1)}, i = 1, 2, \dots, NB_x - 1 \quad (82)$$

The equations for the 3D case can be written in a similar fashion. The exact and DPIELM solutions for these cases are shown in Figs (22) and (23) respectively.

1D steady advection-diffusion equation with low value of diffusion constant [TC-14]

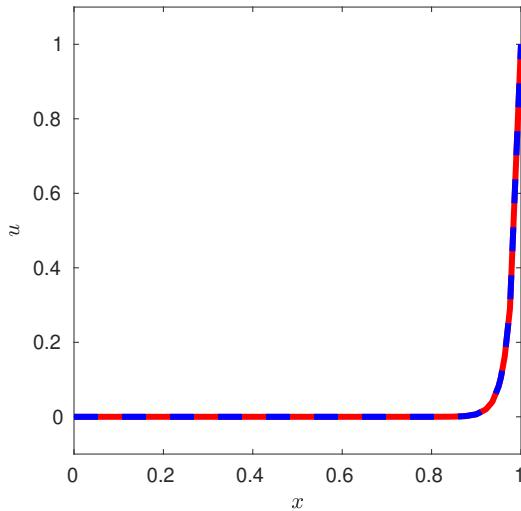


Figure 24: Exact and DPIELM solution of unsteady 1D convection diffusion. Red: DPIELM, Blue: exact

The DPIELM equations are as follows:

$$1. \quad \vec{\xi}_f^{(i)} = \vec{0} \\ \varphi'(\mathbf{X}_f^{(i)} \mathbf{W}^{(i)^T}) \odot \vec{c}^{(i)} \odot \vec{m} - \nu \varphi''(\mathbf{X}_f^{(i)} \mathbf{W}^{(i)^T}) \odot \vec{c} \odot \vec{m} \odot \vec{m} = \vec{0}, i = 1, 2, \dots, NB_x \quad (83)$$

$$2. \quad \vec{\xi}_{bc}^{(i)} = \vec{0} \\ \left\{ \begin{array}{l} \varphi(\mathbf{X}_{bc, I_1}^{(1)} \mathbf{W}^{(1)^T}) \vec{c}^{(1)} \\ \varphi(\mathbf{X}_{bc, I_2}^{(NB_x)} \mathbf{W}^{(NB_x)^T}) \vec{c}^{(NB_x)} \end{array} \right\} = \left\{ \begin{array}{l} B(\vec{x}_{bc, I_1}^{(1)}) \\ B(\vec{x}_{bc, I_2}^{(NB_x)}) \end{array} \right\} \quad (84)$$

$$3. \quad \vec{\xi}_{C^0}^{(i)} = \vec{0} \\ \varphi(\mathbf{X}_{bc, I_2}^{(i)} \mathbf{W}^{(i)^T}) \vec{c}^{(i)} = \varphi(\mathbf{X}_{bc, I_1}^{(i+1)} \mathbf{W}^{(i+1)^T}) \vec{c}^{(i+1)}, i = 1, 2, \dots, NB_x - 1 \quad (85)$$

$$4. \quad \vec{\xi}_{C^1}^{(i)} = \vec{0} \\ \varphi'(\mathbf{X}_{bc, I_2}^{(i)} \mathbf{W}^{(i)^T}) \vec{c}^{(i)} \odot \vec{m}^{(i)} = \varphi'(\mathbf{X}_{bc, I_1}^{(i+1)} \mathbf{W}^{(i+1)^T}) \vec{c}^{(i+1)} \odot \vec{m}^{(i+1)}, i = 1, 2, \dots, NB_x - 1 \quad (86)$$

The results of the exact and DPIELM solution is given in Fig(24).

7.1 1D unsteady advection of a sharp peaked Gaussian and a high frequency wavelet and [TC-13, TC-15]

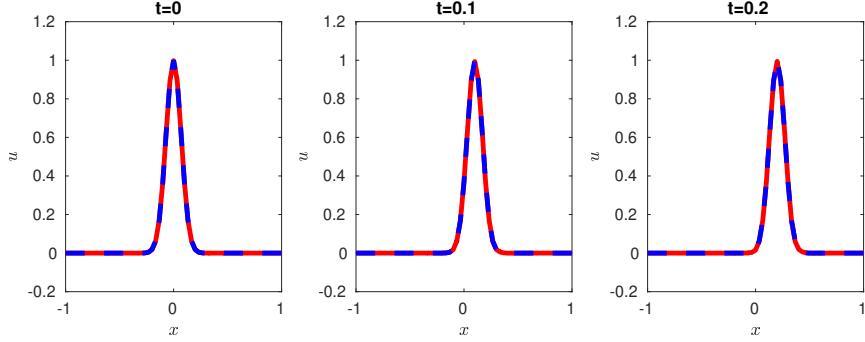


Figure 25: Exact and DPIELM solution advection of a sharp peaked Gaussian with DPIELM. Red: DPIELM, Blue: Exact.

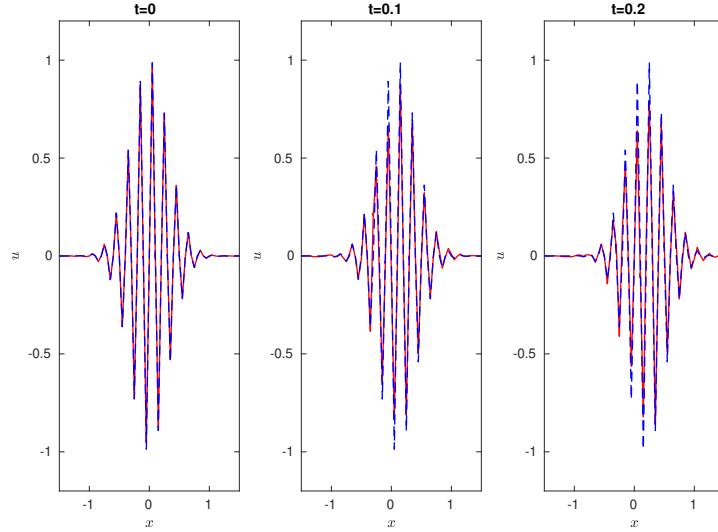


Figure 26: Exact and DPIELM solution of pure advection of a high frequency wave packet. Red: DPIELM, Blue: exact

The DPIELM equation to be solved are as follows:

$$1. \quad \vec{\xi}_f^{(i)} = \vec{0}$$

$$\varphi'(\mathbf{X}_{\mathbf{f}}^{(i)} \mathbf{W}^{(i)^T}) \odot \vec{\mathcal{C}}^{(i)} \odot (\vec{m}^{(i)} + \vec{n}^{(i)}) = \vec{0}, i = 1, 2, \dots, N_c \quad (87)$$

$$2. \quad \vec{\xi}_{bc}^{(i)} = \vec{0}$$

$$\varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_4}^{(i)} \mathbf{W}^{(i)^T}) \vec{\mathcal{C}}^{(i)} = \vec{0}, i = 1, (1 + NB_x), \dots, (1 + (NB_t - 1)NB_x) \quad (88)$$

$$\varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_2}^{(i)} \mathbf{W}^{(i)^T}) \vec{\mathcal{C}}^{(i)} = \vec{0}, i = NB_x, 2NB_x, \dots, NB_t \times NB_x \quad (89)$$

$$3. \quad \vec{\xi}_{ic}^{(i)} = \vec{0}$$

$$\varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_1}^{(i)} \mathbf{W}^{(i)^T}) \vec{\mathcal{C}}^{(i)} = \vec{F}(\vec{x}_{bc, I_1}^{(i)}), i = 1, 2, \dots, NB_x \quad (90)$$

$$4. \quad \vec{\xi}_{C^0}^{(i)} = \vec{0}$$

- Continuity along x direction

$$\left\{ \begin{array}{l} \varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_2}^{(j(1,i)-1)} \mathbf{W}^{(j(1,i)-1)^T}) \vec{\mathcal{C}}^{(j(1,i)-1)} \\ \varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_2}^{(j(2,i)-1)} \mathbf{W}^{(j(2,i)-1)^T}) \vec{\mathcal{C}}^{(j(2,i)-1)} \\ \dots \\ \varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_2}^{(j(NB_t,i)-1)} \mathbf{W}^{(j(NB_t,i)-1)^T}) \vec{\mathcal{C}}^{(j(NB_t,i)-1)} \\ \left\{ \begin{array}{l} \varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_4}^{(j(1,i))} \mathbf{W}^{(j(1,i))^T}) \vec{\mathcal{C}}^{(j(1,i))} \\ \varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_4}^{(j(2,i))} \mathbf{W}^{(j(2,i))^T}) \vec{\mathcal{C}}^{(j(2,i))} \\ \dots \\ \varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_4}^{(j(NB_t,i))} \mathbf{W}^{(j(NB_t,i))^T}) \vec{\mathcal{C}}^{(j(NB_t,i))} \end{array} \right\} \end{array} \right\} = \quad (91)$$

where $\rho = 1, 2, \dots, NB_t$, $j(\rho, i) = \kappa(\rho) + i$, $i = 1, 2, \dots, NB_x - 1$ and $\kappa = [1, (1 + NB_x), \dots, 1 + (NB_t - 1)NB_x]^T$.

- Continuity along t direction

$$\left\{ \begin{array}{l} \varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_3}^{(j(1,i-1)} \mathbf{W}^{(j(1,i-1)^T}) \vec{\mathcal{C}}^{(j(1,i-1)} \\ \varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_3}^{(j(2,i-1)} \mathbf{W}^{(j(2,i-1)^T}) \vec{\mathcal{C}}^{(j(2,i-1)} \\ \dots \\ \varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_3}^{(j(NB_t,i-1)} \mathbf{W}^{(j(NB_t,i-1)^T}) \vec{\mathcal{C}}^{(j(NB_t,i-1)} \\ \left\{ \begin{array}{l} \varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_1}^{(j(1,i))} \mathbf{W}^{(j(1,i))^T}) \vec{\mathcal{C}}^{(j(1,i))} \\ \varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_1}^{(j(2,i))} \mathbf{W}^{(j(2,i))^T}) \vec{\mathcal{C}}^{(j(2,i))} \\ \dots \\ \varphi(\mathbf{X}_{\mathbf{bc}, \mathbf{I}_1}^{(j(NB_t,i))} \mathbf{W}^{(j(NB_t,i))^T}) \vec{\mathcal{C}}^{(j(NB_t,i))} \end{array} \right\} \end{array} \right\} = \quad (92)$$

where $\rho = 1, 2, \dots, NB_x$, $j(\rho, i) = \kappa(\rho) + iNB_x$, $i = 1, 2, \dots, NB_t - 1$ and $\kappa = [1, 2, \dots, NB_x]^T$.

The results for the TC-13 and TC-15 are given in Fig (25) and Fig (26) respectively.

7.2 1D and 2D unsteady advection-diffusion equations [TC-9, TC-10]

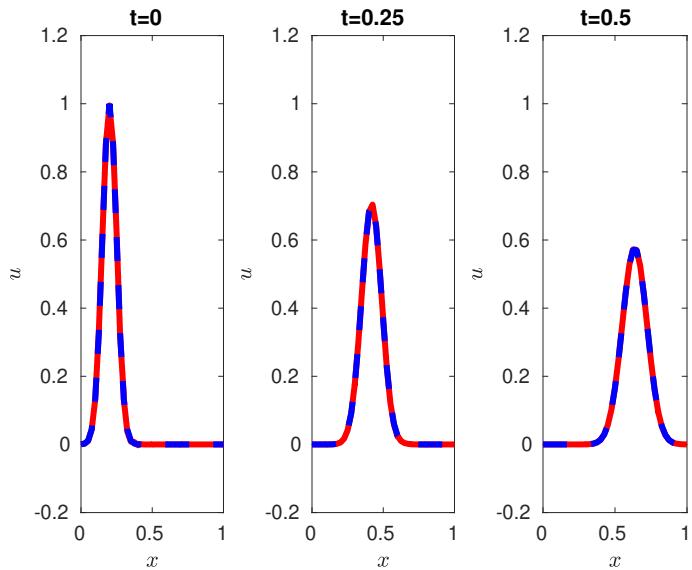


Figure 27: Exact and DPIELM solution of unsteady 1D convection diffusion. Red: DPIELM, Blue: exact

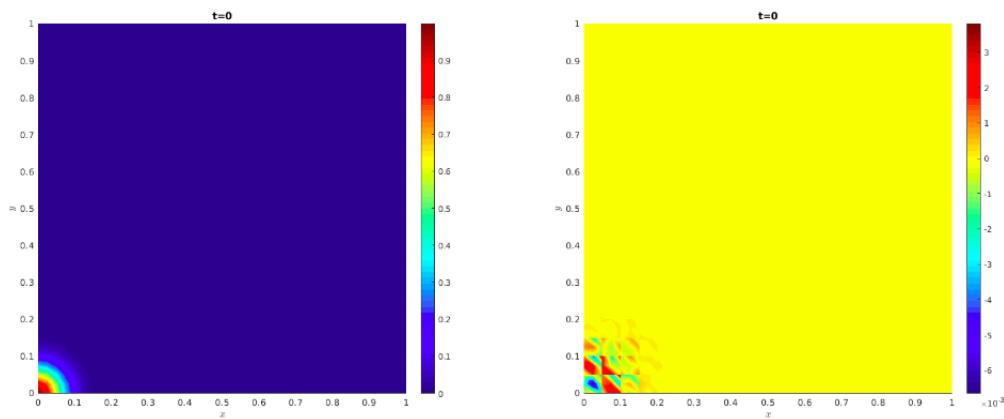
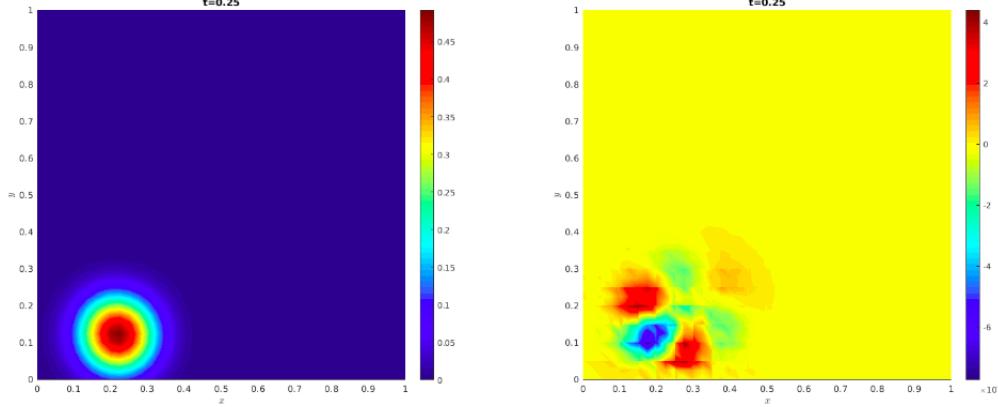
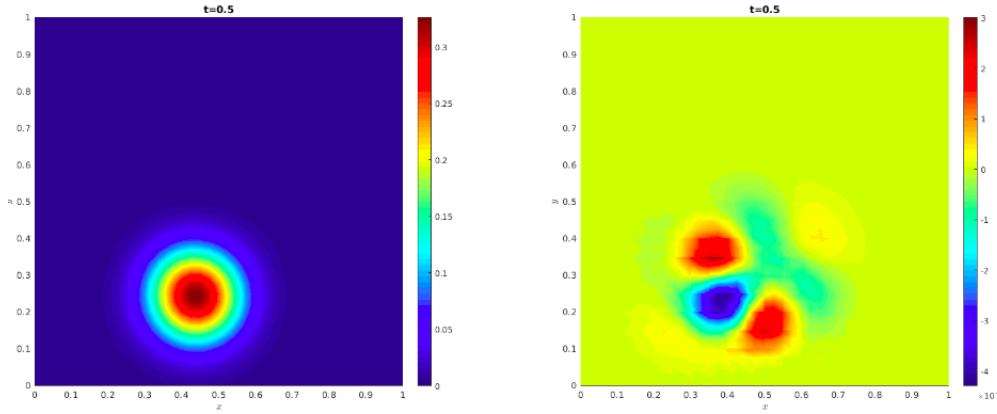


Figure 28: DPIELM solution and error for unsteady 2D convection diffusion at $t = 0$

Figure 29: DPIELM solution and error for unsteady 2D convection diffusion at $t = 0.25$ Figure 30: DPIELM solution and error for unsteady 2D convection diffusion at $t = 0.5$

In this section, we present the DPIELM equation for the 1D case. The equations for the 2D case can be formulated in a similar fashion. The equations to be solved for 1D unsteady advection-diffusion equation are as follows:

$$1. \quad \vec{\xi}_f^{(i)} = \vec{0}$$

$$\varphi'(\mathbf{X}_f^{(i)} \mathbf{W}^{(i)^T}) \odot \vec{c}^{(i)} \odot (\vec{n}^{(i)} + a \vec{m}^{(i)} - \nu \vec{m}^{(i)} \odot \vec{m}^{(i)}) = \vec{0}, i = 1, 2, \dots, N_c \quad (93)$$

$$2. \quad \vec{\xi}_{C^1}^{(i)} = \vec{0}$$

$$\left\{ \begin{array}{l} \varphi'(\mathbf{X}_{bc, I_2}^{(j(1,i)-1)} \mathbf{W}^{(j(1,i)-1)^T}) \vec{c}^{(j(1,i)-1)} \odot \vec{m}^{(j(1,i)-1)} \\ \varphi'(\mathbf{X}_{bc, I_2}^{(j(2,i)-1)} \mathbf{W}^{(j(2,i)-1)^T}) \vec{c}^{(j(2,i)-1)} \odot \vec{m}^{(j(2,i)-1)} \\ \cdots \\ \varphi'(\mathbf{X}_{bc, I_2}^{(j(NB_t,i)-1)} \mathbf{W}^{(j(NB_t,i)-1)^T}) \vec{c}^{(j(NB_t,i)-1)} \odot \vec{m}^{(j(NB_t,i)-1)} \end{array} \right\} = \left\{ \begin{array}{l} \varphi'(\mathbf{X}_{bc, I_4}^{(j(1,i))} \mathbf{W}^{(j(1,i))^T}) \vec{c}^{(j(1,i))} \odot \vec{m}^{(j(1,i))} \\ \varphi'(\mathbf{X}_{bc, I_4}^{(j(2,i))} \mathbf{W}^{(j(2,i))^T}) \vec{c}^{(j(2,i))} \odot \vec{m}^{(j(2,i))} \\ \cdots \\ \varphi'(\mathbf{X}_{bc, I_4}^{(j(NB_t,i))} \mathbf{W}^{(j(NB_t,i))^T}) \vec{c}^{(j(NB_t,i))} \odot \vec{m}^{(j(NB_t,i))} \end{array} \right\} \quad (94)$$

where $\rho = 1, 2, \dots, NB_t$, $j(\rho, i) = \kappa(\rho) + i$, $i = 1, 2, \dots, NB_x - 1$ and $\kappa = [1, (1 + NB_x), \dots, 1 + (NB_t - 1)NB_x]^T$.

Rest of the equations i.e. $\vec{\xi}_{bc}^{(i)} = \vec{0}$, $\vec{\xi}_{ic}^{(i)} = \vec{0}$ and $\vec{\xi}_{C^0}^{(i)} = \vec{0}$ are same as that used in 1D unsteady linear advection. The results for the 1D and 2D cases are given in Fig (27) and Fig (28 to 30) respectively.

This brings us to the end of our numerical experiments. We close this section by highlighting the key points which are as follows:

1. The process of partitioning of the whole computational domain into multiple cells simplifies the representation of the complicated function (and thus PDE) in the individual cells. As a result, local PIELMs are able to capture not just the functions with sharp gradients, but also discontinuous functions (TC-11).
2. To our knowledge, this is the first demonstration of capability of ELM based algorithms to solve 2D unsteady PDEs and produce results comparable to sophisticated numerical methods. (TC-10)
3. The distributed version of PIELM exhibits better representation ability than a deep PINN (TC-15).

8 Conclusion and future work

We have presented in this paper PIELM -- an efficient method to utilize physics informed neural networks to solve stationary and time dependent linear PDEs. As PIELM inherits the unique qualities of its parent algorithms (PINN and ELM), it works very well on complex geometries, respects the inherent physics of the PDEs and is extremely fast as well. This leads to several advantages over existing conventional numerical methods; PIELM reduces numerical artefacts such as false diffusion as well can handle complex geometries in a meshfree approach. PIELM also reduces, to a certain extent, the arbitrariness of the number of neurons in typical deep PINNs. Our numerical tests also confirm that, for a fixed problem, our minimal PIELM is more accurate than prior deep NN results [3] while being faster.

We have also presented in this paper the limitations in representing complex functions using a single PIELM or PINN for the whole domain. For practical problems using PINNs can lead to very deep networks with the concomitant training problems and efficiency issues. Our proposed solution is to use a distributed PIELM (DPIELM) which uses different representations in different portions of the domain while imposing some continuity and differentiability constraints. The resultant DPIELM easily captures profiles that PINNs have difficulty with. Further, on time-dependent problems, DPIELM gives results that are comparable to sophisticated conventional numerical techniques as seen in Section 7.2.

We believe that the method, as formulated, is already very powerful for linear PDEs with constant or space-varying coefficients. Two primary areas of development remain, in our opinion. Our preliminary tests show that, for linear PDEs, unlike deep PINNs [13], our method may actually be competitive with conventional techniques in terms of speed and accuracy. However, a firm conclusion on this cannot be reached until a more thorough and fair study is done. Theoretical and numerical evidence for this efficacy is the first area which deserves attention, as the number of practical applications (such as heat conduction, etc) where numerical methods for linear equations are a staple is large. Having an efficient neural network framework for such problems can be tremendously beneficial to practitioners.

Even more importantly, PIELM's efficacy is thanks to the linear nature of the final problem. We have, therefore, limited our present study to linear problems. Extending this method to nonlinear equations is the obvious next frontier. This may be approached in one of two ways. The first approach would be to use the PIELM structure as is and then solve the resultant, non-convex optimization problem. Another approach would be to linearize the equation around the current time step to predict a future time step. This would be the equivalent of a linearized, explicit time-stepping method in conventional time marching techniques. We are currently investigating these approaches and will report progress in future publications.

References

- [1] Versteeg, Henk Kaarle, and Weeratunge Malalasekera. An introduction to computational fluid dynamics: the finite volume method. Pearson education, 2007.
- [2] Quirk, James J. "A contribution to the great Riemann solver debate." Upwind and High-Resolution Schemes. Springer, Berlin, Heidelberg, 1997. 550-569.
- [3] Berg, Jens, and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. Neurocomputing 317 (2018): 28-41.
- [4] Lagaris, Isaac E., Aristidis Likas, and Dimitrios I. Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations." IEEE transactions on neural networks 9.5 (1998): 987-1000.
- [5] Lagaris, Isaac E., Aristidis C. Likas, and Dimitris G. Papageorgiou. "Neural-network methods for boundary value problems with irregular boundaries." IEEE Transactions on Neural Networks 11.5 (2000): 1041-1049.

- [6] van Milligen, B. Ph, V. Tribaldos, and J. A. Jiménez. " Neural network differential equation and plasma equilibrium solver." Physical review letters 75.20 (1995): 3594.
- [7] McFall, K. S., & Mahan, J. R. (2009). 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.
- [8] Kumar, Manoj, and Neha Yadav. "Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: a survey." Computers & Mathematics with Applications 62.10 (2011): 3796-3811.
- [9] Mall, Susmita, and Snehashish Chakraverty. " Application of Legendre Neural Network for solving ordinary differential equations." Applied Soft Computing 43 (2016): 347-356.
- [10] Sirignano, Justin, and Konstantinos Spiliopoulos. " DGM: A deep learning algorithm for solving partial differential equations." Journal of Computational Physics 375 (2018): 1339-1364.
- [11] Raissi, Maziar, and George Em Karniadakis. " Hidden physics models: Machine learning of nonlinear partial differential equations." Journal of Computational Physics 357 (2018): 125-141.
- [12] Raissi, Maziar, Paris Perdikaris, and George Em Karniadakis. " Numerical gaussian processes for time-dependent and nonlinear partial differential equations." SIAM Journal on Scientific Computing 40.1 (2018): A172-A198.
- [13] Raissi, Maziar, 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 (2019): 686-707.
- [14] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (2006) 489–501.
- [15] Balasundaram, S. " Application of error minimized extreme learning machine for simultaneous learning of a function and its derivatives." Neurocomputing 74.16 (2011): 2511-2519.
- [16] Yang, Yunlei, Muzhou Hou, and Jianshu Luo. " A novel improved extreme learning machine algorithm in solving ordinary differential equations by Legendre neural network methods." Advances in Difference Equations 2018.1 (2018): 469.
- [17] Sun, Hongli, et al. " Solving partial differential equation based on Bernstein neural network and extreme learning machine algorithm." Neural Processing Letters (2018): 1-20.
- [18] G.-B. Huang, L. Chen, C.-K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Transactions on Neural Networks 17 (4) (2006) 879–892.
- [19] D. Rumelhart, G. Hintont, R. Williams, Learning representations by backpropagating errors, Nature 323 (6088) (1986) 533–536.
- [20] Baydin, Atilim Gunes, et al. " Automatic differentiation in machine learning: a survey." Journal of Machine Learning Research 18 (2018): 1-43.
- [21] D. Serre, Matrices: Theory and Applications, Springer, New York, 2002.
- [22] Kopriva, David A., and Gregor J. Gassner. " An energy stable discontinuous Galerkin spectral element discretization for variable coefficient advection problems." SIAM Journal on Scientific Computing 36.4 (2014): A2076-A2099.
- [23] Borker, Raunak, Charbel Farhat, and Radek Tezaur. "A high-order discontinuous Galerkin method for unsteady advection–diffusion problems." Journal of Computational Physics 332 (2017): 520-537.