Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/MBE

MBE, 18(1): 22–56.
DOI: 10.3934/mbe.2021002
Received: 09 August 2020
Accepted: 02 November 2020
Published: 18 November 2020

Research article

An immersed boundary neural network for solving elliptic equations with singular forces on arbitrary domains

Reymundo Itzá Balam[1], **Francisco Hernandez-Lopez**[1,2], **Joel Trejo-Sánchez**[1,2] **and Miguel Uh Zapata**[1,2,*]

[1] Centro de Investigación en Matemáticas A.C, CIMAT-Mérida, México
[2] Consejo Nacional de Ciencia y Tecnología, CONACYT, México

* **Correspondence:** Email: angeluh@cimat.mx; Tel: +52 1 99 93 32 86 26.

Abstract: In this paper, we present a deep learning framework for solving two-dimensional elliptic equations with singular forces on arbitrary domains. This work follows the ideas of the physical-inform neural networks to approximate the solutions and the immersed boundary method to deal with the singularity on an interface. Numerical simulations of elliptic equations with regular solutions are initially analyzed in order to deeply investigate the performance of such methods on rectangular and irregular domains. We study the deep neural network solutions for different number of training and collocation points as well as different neural network architectures. The accuracy is also compared with standard schemes based on finite differences. In the case of singular forces, the analytical solution is continuous but the normal derivative on the interface has a discontinuity. This discontinuity is incorporated into the equations as a source term with a delta function which is approximated using a Peskin's approach. The performance of the proposed method is analyzed for different interface shapes and domains. Results demonstrate that the immersed boundary neural network can approximate accurately the analytical solution for elliptic problems with and without singularity.

Keywords: elliptic equation; immersed boundary method; singular forces; neural networks; deep learning; continuous time model; interface; two-phase flow

1. Introduction

Nowadays, deep neural networks (DNN) have been used to solve a wide variety of problems [1]. For instance, there exist several algorithms based on DNN for weather forecasting [2], detecting defaulter users [3], financial forecast [4], COVID-19 prediction models [5], among others. These algorithms infer information about the behavior of these phenomena when a set of historical data is given. Recently, deep learning methods have been also applied to solve partial differential equations (PDEs) modeling

different phenomena in physics, engineering, and the biological sciences by using residual-minimizing formulations [6–10]. The idea is to replace usual discretization strategies, such as finite-difference, finite-element or finite-volume methods, with neural networks in order to find the numerical solution using the known data coming from boundary conditions, initial conditions, or the differential equation itself.

Deep neural networks are inspired by biological neuron interconnections [11]. Basically, an artificial neural network is a function resulting from the composition of a finite set of elementary operations and coefficients set as a computational graph. Usually these elementary operations include the binary arithmetic operations and transcendental functions [12]. The coefficients are the values to determine using a set of known data, named training data, that receives as input. Thus, the deep neural network consists of an input layer, an output layer, and a set of hidden layers (two or more), each one containing a set or arbitrary neurons [13–15]. The idea is updating the coefficients of the neurons by minimizing the error between the known value of training data and the corresponding predicted values [16]. This iterative procedure is known as the training process.

In mathematical models based on PDEs, data is mostly coming from the boundary and initial conditions. Moreover, in real applications, the knowledge of this data is generally limited and very costly to get. However, PDEs are based on conservation laws, physical principles, and/or phenomenological behaviors that can be incorporated into the training process of the DNN. Raissi et al. [9] propose a class of universal function approximators named *physical-informed neural networks* to deal with these cases. Their neural network formulation differs from the standard DNN by involving the differential equations used to model the problem. This technique has shown benefits in solving partial differential equations with a lack of information. However, although a series of promising results were presented in [9], the authors remark that their work creates many questions concerning the range of applicability of the DNN and more work is needed collectively to set the foundations in this field. Moreover, results show that the performance of these techniques are directly related to the particular equation which is analyzed.

In the recent years, physical-informed neural networks have been applied in fluid mechanics [17, 18], biomedical problems [19], among others. For instance, Sun et al. [18] provide a deep learning approach, using a fully-connected neural network, that models a fluid flow without using any labeled data, allowing reducing the computational costs derived from using several simulation data. Few works have been presented to solve two-dimensional (2D) elliptic equations with artificial neural networks on arbitrary domains [20–26]. For example, Koryagin et al. [24] provides PyDENs, an open-source easy mechanism to solve PDEs using neural networks. PyDENs allows to define and configure the solution of several PDE problems, that includes heat and wave equations. However, the analysis of the performance of their proposed DNN is still very limited. Motivated by this, in this paper we analyze the capacity of a deep learning framework for solving 2D elliptic equations. Extensive numerical simulations are presented in order to deeply investigate the performance of the DNN on rectangular and arbitrary domains. The accuracy is also compared with standard schemes based on finite differences. Furthermore, this paper shows that a classical method such as the immersed boundary method (IBM) can coexist in harmony with deep neural networks to solve elliptic equations with singularities. To the best of the authors' knowledge, there are not other works proposing a similar approach.

Elliptic equations with singular forces arise in problems where discontinuities in the solution and derivatives are expected across immersed interfaces such as fluid-structure interactions and two-phase

flow systems. A well-known technique developed to solve problems with these characteristics is the immersed boundary method. It was initially introduced by Peskin [27, 28] and developed to simulate cardiac mechanics and associated blood flow. However, the IBM has been extended to simulate a wide range of problems. For instance, some applications are: massive boundaries using penalty IBM [29] or D'Alembert force [30], porous boundaries [31], generalized IBM for torque in flexible rods [32], membrane transport and osmosis [33], stochastic IBM [34], variable density and viscosity fluids [35], among others [36, 37]. The idea of the IBM is to regularize discontinuous interface problems, which typically produces low-order accuracy approximation at the interface.

The present work is aimed to solve 2D elliptic equations as data-driven solutions of PDEs. Thus, we seek for the unknown hidden solution of an equation with fixed parameters. The IBM is also introduced by the Peskin's approach to deal with equations with singular forces. One of the main advantages of this method is the low number of known data required on arbitrary locations of the boundaries. This flexibility can not be obtained using standard algorithms such as finite-difference, finite-element or finite-volume methods. Moreover, applications with arbitrary domains are directly implemented with minor modifications to the code. For rectangular domains, a random number of points are selected from uniform grids. However, in order to deal with arbitrary geometries, the location of the points are obtained as the vertex points of unstructured grids which has more flexibility in fitting complicated boundaries. All code and data-sets accompanying this manuscript are available on GitHub at https://github.com/ibmcimatmerida/IBMNN.

This paper is organized as follows. The second section presents the governing equations. The third section introduces to the numerical method including a brief description of the physics-informed neural network and the IBM by Peskin's approach. Next, details of the implementation of the DNN algorithm is included in Section 4. The numerical results are presented in Section 5. Finally, last section includes the conclusions and future work.

2. Governing equations

Let us consider elliptic equations of the form

$$\nabla \cdot (\beta(\boldsymbol{x})\nabla u(\boldsymbol{x})) + \kappa(\boldsymbol{x})u(\boldsymbol{x}) = \tilde{g}(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \tag{2.1}$$

where Ω is an arbitrary domain and the boundary, $\partial\Omega$, is composed of the union of sections with Dirichlet, Neumann or Robin boundary conditions. There is also an irregular interface Γ contained in Ω across which source $\tilde{g}$ may have a delta function singularity. We remark that for discontinuities to arise in the exact solution or its derivatives, there must be discontinuities or singularities present in the coefficients of Eq (2.1). Another possibility is that β and κ are continuous but the source term $\tilde{g}$ has a delta function singularity along the interface [38].

This paper focuses on a deep neural network method for solving a two-dimensional elliptic equation given by

$$(\beta(x,y)u_x)_x + (\beta(x,y)u_y)_y + \kappa(x,y)u(x,y) = g(x,y) + \mathfrak{J}(x,y), \quad (x,y) \in \Omega \subset \mathbb{R}^2 \tag{2.2}$$

where function $\mathfrak{J}$ is defined as

$$\mathfrak{J}(x,y) = \int_\Gamma C(s)\delta(x - X(s))\delta(y - Y(s))ds, \tag{2.3}$$

where $X(s)$ and $Y(s)$ are the parametric equations of a rectifiable curve Γ and $C(s)$ is the source strength. By this we mean that $\mathfrak{J}(x, y)$ is a distribution with the property that

$$\iint \mathfrak{J}(x, y)\phi(x, y)dxdy = \int_{\Gamma} C(s)\phi(X(s), Y(s))ds,$$

for any smooth test function $\phi(x, y)$. Thus, the analytical solution u will be continuous but the normal derivative will have a jump at the interface of magnitude $C(s)$, thus

$$\left[\frac{du}{dn}\right]_{\Gamma} = \frac{du^{+}}{dn} - \frac{du^{-}}{dn} = C(s).$$

where the superscript indicates the limit of the normal derivative at each side of the interface. We remark that the interface Γ can be an arbitrary piecewise smooth curve lying in Ω.

Figure 1 shows an example of an immersed interface in an arbitrary domain. It is important to remark that Γ is not necessarily closed or even connected. Moreover, we do not require an explicit parametrization of this curved as a numerical discretization over the points will be directly applied.

3. DNN approximation

Our goal is to develop a DNN that can be used together with the known boundary data to obtain accurate approximations of Eq (2.2). For equations with singular forces, we follow the ideas of the immersed boundary method together with physical-inform neural networks to reach this goal.

3.1. Physics-informed neural networks

In this paper, elliptic equations are solved by employing a DNN based on the physical-informed neural networks proposed by Raissi et al. [9]. It is based on a model of data-driven solutions of partial differential equations which all parameters are given and u is considered as the unknown hidden solution of the system.

We proceed by approximating $u(x, y)$ by a deep neural network as follows

$$u(x, y) \approx \tilde{u}(x, y; W, b), \tag{3.1}$$

where W and b are two sets of parameters of the neural network to determine. More details about this DNN will be given in the next section. We also consider f as the left-hand-side of Eq (2.2) as follows

$$f := (\beta u_x)_x + (\beta u_y)_y + \kappa u - g - \mathfrak{J}, \tag{3.2}$$

which results in a physics-informed neural network $\tilde{f}(x, y; W, b)$. Note that $\tilde{f}$ not only includes the solution $\tilde{u}$ but also its partial derivatives. These derivatives are obtained by automatic differentiation [12]. In this procedure, given a set of parameters (W, b), the derivatives of the neural network can be known by combining the derivatives of the constituent operations through the chain rule. This gives the derivative of the overall composition. An efficient algorithm to obtain these derivatives is referred as backpropagation [16].

In the absence of any interface, parameters W and b in Eq (3.1) can be obtained by using only two finite sets of points: the training and collocation points.

- First, the training points $\{x_u^i, y_u^i\}_{i=1}^{N_u}$ are located at the boundary $\partial\Omega_u$ which the data is already available for Dirichlet boundary conditions: $\{u_u^i\}_{i=1}^{N_u}$.

- Second, the collocation points $\{x_f^j, y_f^j\}_{j=1}^{N_f}$ are located inside the domain Ω (boundary included) which the differential equation represented by f is known: $\{f^j\}_{j=1}^{N_f}$.

Here N_u and N_f are the number of training and collocation points, respectively. We remark that these points can be arbitrarily selected as shown in Figure 1. We denote a particular point as $\boldsymbol{x}_D$, $\boldsymbol{x}_N$ and $\boldsymbol{x}_f$.

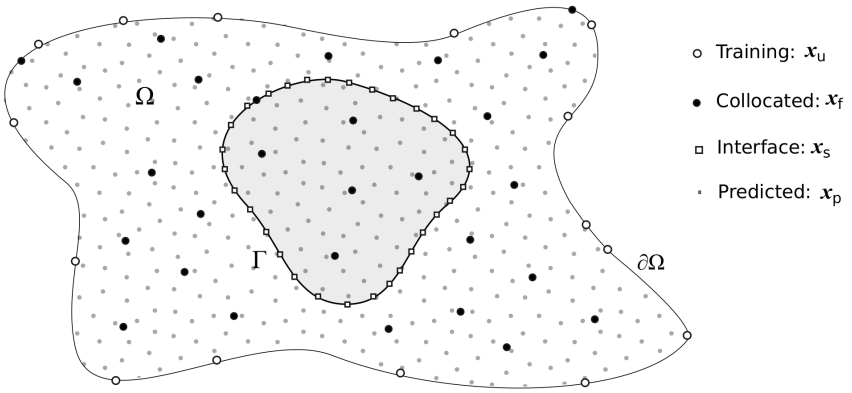

Figure 1. Domain Ω with an interface Γ. Location of the four different type of points required for the DNN simulations: training, collocated, interface and predicted.

Finally, parameters W and b are learned by minimizing the mean squared error (MSE) loss

$$E = E_u + E_f, \tag{3.3}$$

where E_u corresponds to the error of the training boundary data $\{x_u^i, y_u^i, u^i\}_{i=1}^{N_u}$ for Dirichlet boundary conditions given by

$$E_u = \frac{1}{N_u}\sum_{i=1}^{N_u}\left|\tilde{u}(x_u^i, y_u^i; W, b) - u^i\right|^2, \tag{3.4}$$

and E_f enforces the solution of the elliptic equation by using Eq (3.2) at the collocation points $\{x_f^j, y_f^j\}_{j=1}^{N_f}$ and taking $f^j = 0$ as follows

$$E_f = \frac{1}{N_f}\sum_{j=1}^{N_f}\left|\tilde{f}(x_f^j, y_f^j; W, b)\right|^2. \tag{3.5}$$

It is important to remark that Neumann and Robin boundary conditions can be also considered in our domain by including extra errors to the loss function Eq (3.3) as done for Dirichlet conditions in Eq (3.4). For instance, Neumann boundary conditions at the set of training points $\{x_{u_n}^i, y_{u_n}^i\}_{i=1}^{N_{u_n}}$ are included as

$$E = E_u + E_{u_n} + E_f, \tag{3.6}$$

where E_{u_n} corresponds to the error of the training boundary data $\{x^i_{u_n}, y^i_{u_n}, u^i_n\}_{i=1}^{N_{u_n}}$ given by

$$E_{u_n} = \frac{1}{N_{u_n}} \sum_{i=1}^{N_{u_n}} \left| \frac{du}{dn}(x^i_{u_n}, y^i_{u_n}; W, b) - u^i_n \right|^2 . \tag{3.7}$$

and where u^i_n corresponds to the known values of the normal derivative at the boundary.

3.2. Immersed boundary method by Peskin's approach

In the case of elliptic equations with singular forces, note that the singularity is already incorporated at the DNN by the definition of f in Eq (3.2). Thus, the corresponding right-hand side values at collocation points (x^j_f, y^j_f) are evaluated as follows

$$g(x^j_f, y^j_f) + J(x^j_f, y^j_f), \tag{3.8}$$

where J is an approximation of $\mathfrak{J}$. In this paper, this approximation follows the IBM by Peskin's approach.

The main idea of IBM is to replace the delta function by some discrete approximation d_h with support related to a discretization parameter h. In a standard uniform rectangular mesh, h would correspond to the mesh width. In this paper, we employ the Peskin's discrete delta function

$$d_h(x) = \begin{cases} \frac{1}{4h}(1 + \cos\left(\frac{\pi x}{2h}\right) & \text{if } |x| < 2h, \\ 0 & \text{if } |x| \geq 2h. \end{cases} \tag{3.9}$$

Thus, the delta is replaced by a smoothed function which can be easier to implement. However, this function becomes sharper as h becomes smaller making it harder to approximate.

This formulation also approximates the interface by a set of N_s points

$$(\boldsymbol{x}_s)_k = (X(s_k), Y(s_k)), \quad k = 1, 2, \ldots, N_s,$$

and replace the integrals in Eq (2.3) by a discrete sum over the interface. If an arc-length parametrization is given, then this approximation can be done as follows

$$\mathfrak{J}(x, y) \approx J(x, y) = \sum_{k=1}^{N_s} C(s_k) d_h(x - X_k) d_h(y - Y_k) \Delta s_k, \tag{3.10}$$

where $\Delta s_k = \Delta s$ can be directly calculated from the discretization interval. Otherwise, Δs_k representing the discretization step on the interface can be calculated as

$$\Delta s_k = \sqrt{(X_{k+1} - X_k)^2 + (Y_{k+1} - Y_k)^2}. \tag{3.11}$$

3.3. Predicted solution and errors

Once the optimization process to obtain W and b is finished, the resulting pair $\tilde{u}(x, y; W, b)$ and $\tilde{f}(x, y; W, b)$ must approximate elliptic Eq (2.2) with the corresponding boundary and interface conditions. It is important to remark that during training (optimization process) residual of the Eq (3.3) will not be exactly zero, and therefore our approximation will have the accuracy of the residual loss.

The DNN methodology incorporates many decisions to obtain an accurate numerical solution. The source of errors are coming from the DNN configuration such as the number of layers, neurons and activation function, the number and distribution of training and collocation points, the optimizer, and the choice of the loss function. The approximation of elliptic equations with singular forces will also depend on the number and location of interface points $\boldsymbol{x}_s$ as well as parameter h. In this work, we take enough interface points such that the integral approximation does not significantly contribute to the global precision of the problem. Thus, the major influence in the precision of IBM will be given by the selection of parameter h.

Finally, we remark that the predicted solution $\tilde{u}$ is given as a function and not as a set of approximated points. Thus, an arbitrary set of predicted points can be selected to test the precision of the proposed method. In this paper, we refer to this set as $\boldsymbol{x}_p$. They are usually different from $\boldsymbol{x}_u$ and $\boldsymbol{x}_f$ and they can be selected at any position of the domain, see Figure 1.

In the next section, we will describe the DNN and the minimization algorithm.

4. Methodology

We are interested in approximating the solutions to Eq (2.2) with deep neural networks. In terms of implementation, there are several decisions about the DNN formulation that should be taken into account. In this section, we briefly describe two of the most important parts: the neural network and the optimization algorithm.

4.1. Deep neural network

The DNN model of L layers consists of an input layer, an output layer, and a set of $L-1$ hidden layers, each one containing N_ℓ arbitrary neurons or nodes. The output layer is not counted in L. The received input traverses the network from the input layer to the output layer, through the hidden layers. When the signals arrive in each node, an activation function $\phi : \mathbb{R} \to \mathbb{R}$ is used to produce the node output [13–15]. Neural networks with many layers (two or more) are called deep neural networks.

In this work, the neural network is described in terms of the input $\boldsymbol{x} \in \mathbb{R}^2$, the output $\tilde{u} \in \mathbb{R}$, and an input-to-output mapping $\boldsymbol{x} \mapsto \tilde{u}$, where $\boldsymbol{x} = (x, y)$ is any selected point in Ω. For any hidden layer ℓ, we consider the pre-activation $X^\ell \in \mathbb{R}^{N_\ell}$ and post-activation $U^\ell \in \mathbb{R}^{N_{\ell+1}}$ as

$$X^\ell(\boldsymbol{x}) = \left[X_1^\ell(\boldsymbol{x}), \dots X_k^\ell(\boldsymbol{x}), \dots, X_{N_\ell}^\ell(\boldsymbol{x})\right]^T, \tag{4.1}$$

and

$$U^\ell(\boldsymbol{x}) = \left[U_1^\ell(\boldsymbol{x}), \dots U_j^\ell(\boldsymbol{x}), \dots, U_{N_{\ell+1}}^\ell(\boldsymbol{x})\right]^T, \tag{4.2}$$

respectively. Thus, the activation in the ℓ-th hidden layer of the network for $j = 1, \dots, N_{\ell+1}$, is given by [10]:

$$U_j^\ell(\boldsymbol{x}) = b_j^\ell + \sum_{k=1}^{N_\ell} W_k^\ell X_k^\ell(\boldsymbol{x}), \qquad \ell = 1, \dots, L \tag{4.3}$$

where

$$\begin{aligned} X_k^1(\boldsymbol{x}) &= \boldsymbol{x}, \\ X_k^\ell(\boldsymbol{x}) &= \phi(U_k^{\ell-1}(\boldsymbol{x})), \qquad \ell = 2, \dots, L \end{aligned} \tag{4.4}$$

for $k = 1, \ldots, N_\ell$. Here, W_k^ℓ and b^ℓ are the weights and bias parameters of layer ℓ. Activation functions ϕ must be chosen such that the differential operators can be readily and robustly evaluated using reverse mode automatic differentiation [12, 39]. Throughout this work,we have been using relatively simple deep feed-forward neural networks architectures with hyperbolic tangent activation functions. Results show that this function is robust for the proposed formulation.

It is important to remark that as more layers and neurons are incorporated into the DNN the number of parameters significantly increases. Thus the optimization process becomes less efficient. The final size of the bias and weights vector are

$$N_b = \sum_{\ell=1}^{L} N_{\ell+1} \quad \text{and} \quad N_W = \sum_{\ell=1}^{L} N_\ell N_{\ell+1}$$

respectively, where $N_{L+1} = 1$.

Figure 2 shows an example of the computational graph representing a DNN as described in Eqs (4.1)–(4.4). This graph has the weights or biases on each edge labeled. When one node's value is the input of another node, an arrow goes from one to another. In this particular example, we have

$$layers = \left(N_\ell\right)_{\ell=1}^{L+1} = \left(2, 3, 3, 3, 3, 3, 1\right).$$

This is, the total number of layers is $L = 6$. The first layer ($l = 1$) has only two elements (dark gray nodes), $N_\ell = 3$ for $\ell = 2, 3, \ldots, L$, and the last layer (not counted in L) has a single neuron (only one solution). Bias is also considered (light grey nodes), there is a bias node in each layer, which has a value equal to the unit and is only connected to the nodes of the next layer. Although, the number of nodes for each layer can be different; the same number has been employed in this paper for simplicity.

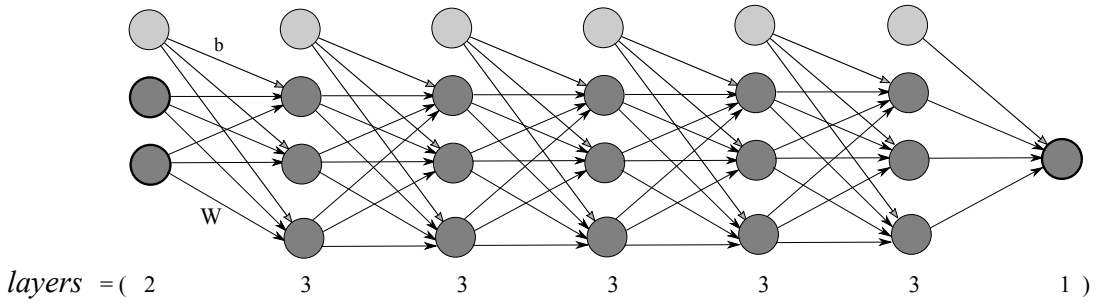

Figure 2. Deep neural network with 6 layers, two inputs and one output.

For the residual network, the derivatives of $\tilde{f}$ are obtained by applying automatic differentiation. The code has been implemented in Python using Tensorflow [40]; currently one of the most popular and well documented open-source libraries for machine learning computations.

4.2. Optimization algorithm

Deep neural networks (see Eqs (4.1)–(4.4)) are trained iteratively, updating the coefficient of the neurons by minimizing the error (see Eq (3.3)) between the target value of the inputs and the predicted values. Thus, the network parameters can be obtained by minimizing the loss function as follows

$$\min_{\boldsymbol{W_b}} E(\boldsymbol{W_b}), \tag{4.5}$$

where the vector $\boldsymbol{W_b} = [W, b]$ represents the total weights and bias of unknown parameters.

There are many algorithms used to solve this minimization problem. Moreover, the accuracy of the final numerical solution significantly depends on the residual loss of the chosen optimization algorithm. In this work, we use the Limited-memory Broyden-Fletcher-Goldfarb-Shanno with Boundaries (L-BFGS-B) method [41]. It is an optimization algorithm in the family of quasi-Newton methods using a limited amount of computer memory. The algorithm starts with an initial estimate of the optimal value, $\boldsymbol{W_b^0} = [W^0, b^0]$, and proceeds iteratively to refine that estimate with a sequence of better estimates $\boldsymbol{W_b^1}$, $\boldsymbol{W_b^2}$, $\boldsymbol{W_b^3}, \ldots$ until tolerance is reached. We remark that each iteration requires one o more evaluations of the loss function.

Similar to the $\tilde{f}$ function, the required gradients for the optimization process are obtained using backpropagation provided by TensorFlow.

5. Results

Now, we present several numerical tests to analyze the applicability and accuracy of the proposed neural network on elliptic equations without any singular force. These tests allow us to analyze in detail the DNN predictions without IBM. In this section, we include the following three examples:

- Example 1 studies the Laplace equation with a smooth solution. We perform an analysis of the number of training and collocation points, as well as the number of hidden layers and neurons required to obtain accurate solutions. The predicted solution is initially calculated on rectangular domains and later applied to arbitrary ones. Finally, the predicted solutions are compared with the numerical solutions of standard methods such as finite difference and finite element.

- Example 2 is designed to test the DNN using different boundary conditions. Thid example studies a mixed boundary value problem with Dirichlet and Neumann conditions. The network parameters are also studied here; however, we limit our analysis to rectangular domains.

- Example 3 analyzes the capacity of the DNN for elliptic equations with more complex configurations. This example studies the predicted solution of a Helmholtz equation with variable coefficients $\beta(x, y)$ and $\kappa(x, y)$ on a rectangular domain. Here we vary the number of training and collocation points as well as the network architecture.

For each case, analytical solutions are used to quantify the numerical error of the proposed formulation. Besides particular sections which the DNN configurations are analyzed, in the rest of our experiments, we always learn the latent solution u by training all 921 parameters corresponding to a 4-layer deep neural network. The first layer contains 2 neurons and each hidden layer contains 20 neurons. The hyperbolic tangent activation function is applied for these simulations. The experiments were executed on a server with Intel Xeon E5-2690 CPU 3.0 GHz, Ubuntu 18.04 (64-bits), 20 cores, 256 GB RAM and a video card NVIDIA Tesla K40. The software was developed using the Python 3.6 programming language and the TensorFlow 1.14 library.

5.1. Example 1: Laplace equation

As a first example, we proceed by solving the two-dimensional Laplace equation with smooth solution and Dirichlet boundary conditions as follows

$$\begin{aligned} u_{xx} + u_{yy} &= 0, \quad (x, y) \in \Omega, \\ u(x, y) &= u_0(x, y), \quad (x, y) \in \partial\Omega, \end{aligned} \tag{5.1}$$

where u_0 is calculated from the analytical solution given by

$$u(x, y) = e^x \cos(y). \tag{5.2}$$

The predicted solution is initially analyzed in the rectangular domain $\Omega = [-1, 1] \times [-1, 1]$, and later on arbitrary domains.

5.1.1. Predicted solution on rectangular domains

The predicted solution is initially calculated in the domain $\Omega = [-1, 1]\times[-1, 1]$ using a $(N+1)\times(N+1)$ rectangular grid with different number of subdivisions N. The total number of training points (N_u) is randomly chosen from the boundary data, and the number of collocation points (N_f) is randomly sampled using a space filling Latin Hypercube Sampling (LHS) strategy inside the domain. Figure 3 shows the analytical and predicted solution using $N_u = 10$ and $N_f = 25$. The results demonstrate that the proposed method approximates accurately the exact solution even for few number of training and collocation points. This is expected as previous works have shown that physics-informed neural networks gives accurate results for smooth solutions of PDEs [9].

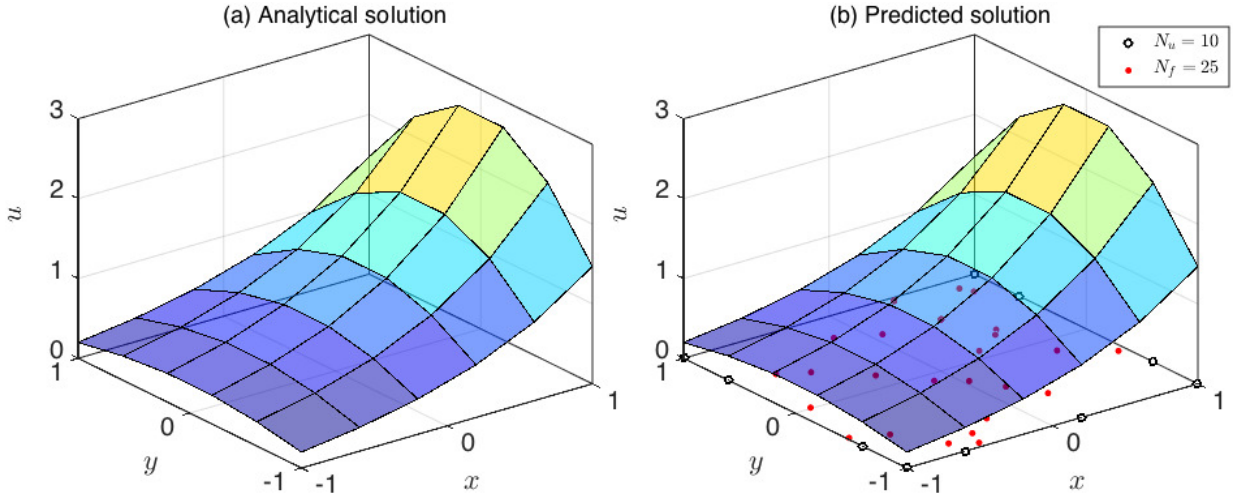

Figure 3. Example 1: (a) Analytical and (b) predicted solution using $N_u = 10$ and $N_f = 25$. The predicted solution is calculated using the rectangular grid of 5×5 subdivisions.

It is important to remark that the results in Figure 3 were obtained using the 36 points of the 6×6 rectangular grid. It corresponds to $N = 5$ subdivisions in each direction of the domain. However, the predicted solution and the corresponding error measurements should be independent of this choice

once the training has been performed. Table 1 shows the relative L_2-norm and L_∞-norm error of the same latent solution $\tilde{u}(x, y; W, b)$ using different grid resolutions N. Results demonstrate that there is not a significant difference between the norm errors of two different grids, as expected.

Table 1. Example 1: Relative L_2-norm and L_∞-norm error between the predicted and the exact solution for $N_u = 10$ and $N_f = 25$ varying the grid resolution of the predicted solution.

N	5	10	15	20	25	30	35	40
L_2-norm	2.89e-03	2.69e-03	2.60e-03	2.56e-03	2.53e-03	2.51e-03	2.49e-03	2.48e-03
L_∞-norm	1.20e-02	1.44e-02	1.41e-02	1.44e-02	1.43e-02	1.44e-02	1.43e-02	1.44e-02

The performance of the deep neural network is further analyzed for different number of training and collocation points. These numbers are chosen as $N_u = 2N$ and $N_f = N^2$. The absolute errors are displayed in Figure 4 for $N = 5$, 10, and 20. Stable and accurate results are obtained in each case. As expected, the absolute errors decrease as the total number of training data is increased. Note that the largest errors are not necessarily located close to the highest values of the exact solution. Moreover, the maximum error is located in regions close to the boundary. Although Dirichlet boundary conditions are applied, non-zero errors can be expected at the boundaries. This is related to the minimization of the loss function and DNN architecture which residual E_u may not be exactly zero. These errors are also related to the low number of training points and its random location selected for each test case. Table 2 shows the relative L_2- and L_∞-norm error for different number of training and collocation points using the current 4-layer network architecture. In general, the prediction is more accurate as the total number of training data N_u increased. Note that for many collocation points N_f, such as 400 and 1600, the errors stop decreasing for N_u close to N as shown in Figure 5. On the other hand, few number of collocation points requires more training data to reach a similar precision, as expected.

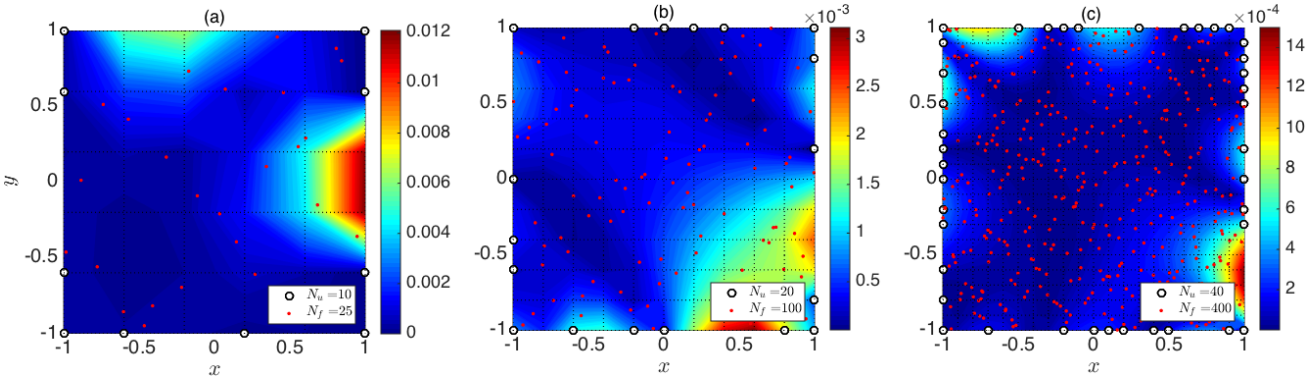

Figure 4. Example 1: Absolute errors between the analytical and predicted solution using different numbers of training and collocation points. The predicted solution was calculated using rectangular grids of (a) 5×5, (b) 10×10, and (c) 20×20 subdivisions.

We remark that the performance of the neural networks also depends on the selected configuration. Now, the DNN is analyzed for different architectures including different number of hidden layers and neurons. We keep the same hyperbolic tangent activation function for all these simulations. Figure 6 shows the resulting relative L_2-norm and L_∞-norm for different number of hidden layers, and different number of neurons per layer, while the total number of training and collocation points is kept fixed to

Table 2. Example 1: Relative L_2-norm and L_∞-norm error between the predicted and the exact solution for different boundary training data N_u, and collocation points N_f.

	$N_f = 25$		$N_f = 100$		$N_f = 400$		$N_f = 1600$		$N_f = 6400$	
N_u	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm
5	2.91e-01	9.42e-01	1.07e-01	2.82e-01	2.50e-01	1.01e+00	2.02e-01	7.81e-01	1.42e-01	3.71e-01
10	2.89e-03	1.20e-02	3.05e-03	1.34e-02	3.86e-02	1.71e-01	3.62e-02	2.23e-01	2.01e-03	1.04e-02
20	5.68e-04	2.99e-03	7.38e-04	3.11e-03	4.01e-04	1.73e-03	1.00e-03	9.06e-03	4.29e-04	2.34e-03
40	5.76e-04	3.28e-03	2.99e-04	1.27e-03	2.36e-04	1.49e-03	1.94e-04	1.11e-03	2.31e-04	2.72e-03
80	2.36e-04	9.78e-04	3.04e-04	1.59e-03	3.33e-04	1.46e-03	3.78e-04	1.81e-03	3.16e-04	1.50e-03
160	4.80e-04	3.20e-03	2.87e-04	1.58e-03	3.05e-04	1.51e-03	3.93e-04	2.37e-03	8.17e-05	5.12e-04

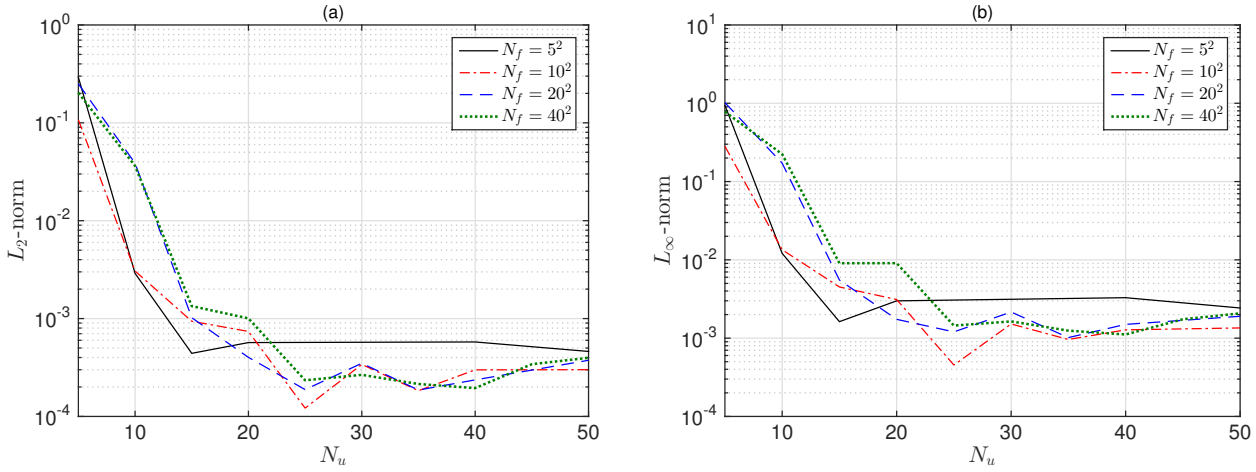

Figure 5. Example 1: (a) Relative L_2- and (b) L_∞-norm error using different numbers of collocation points $N_f = N^2$ and varying training data N_u.

$N_u = 40$ and $N_f = 1600$, respectively. We observe that accurate approximations are obtained for all the network architectures. Moreover, as the number of neurons is increased, the predictive accuracy is increased; there is not a significant change after 20 neurons. Note that the number of layers seems to stabilize the accuracy in this example (not many fluctuations). The number of layers plays an important role in the computer power required for the algorithm as shown in Table 3. Note that the solution is not only more accurate using two layers and 20, 30 or 40 neurons than using eight layers and 10 neurons; but also the simulation of the first case is five times faster the second one.

5.1.2. Predicted solution on arbitrary domains

The following experiments are designed to show the capacity of the DNN for solving the elliptic equations on arbitrary domains. Two different domains are selected as shown in Figure 7. The first domain is a circle of radius one and center $(0, 0)$. The second domain considers an arbitrary domain inspired on a realistic configuration in flows such as rivers, fluvial or estuary zones. The boundaries of the square are deformed by cutting the sides using two circles of radius 1 and $1/2$ located at the top-right and top-left corners, and an ellipse of mayor and minor axis equal to $7/4$ and 1 at the bottom-right corner. The training and collocation points are located at the vertex points of unstructured meshes. A

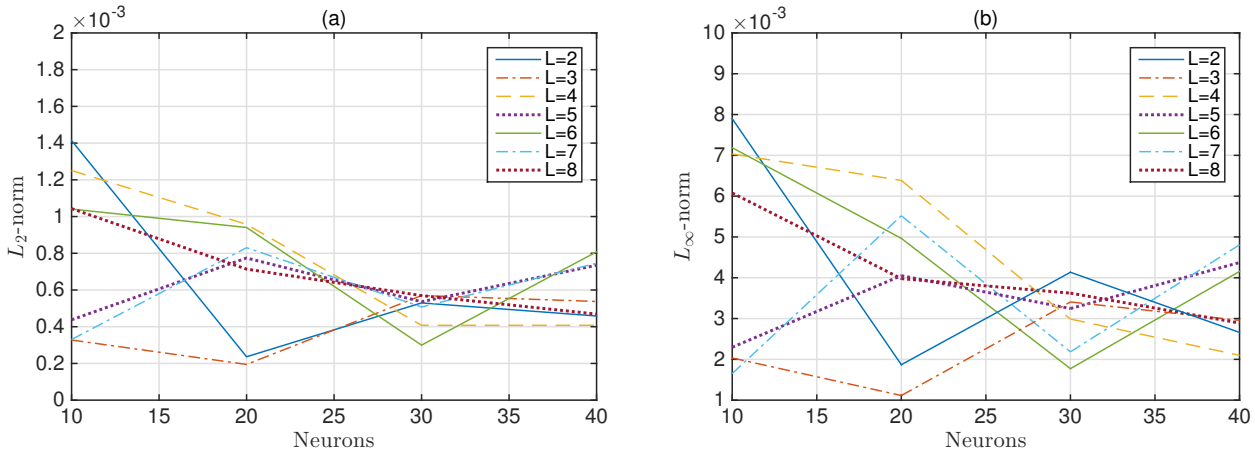

Figure 6. Example 1: (a) Relative L_2-norm and (b) L_∞-norm error using different numbers of layers (L) and varying the number of neurons (N_e). Here, the total number of training and collocation points is fixed to $N_u = 40$ and $N_f = 1600$, respectively.

Table 3. Example 1: Relative L_2-norm error and simulation time for different number of hidden layers and different number of neurons per layer. Here, the total number of training and collocation points is fixed to $N_u = 40$ and $N_f = 1600$, respectively.

	Neurons (L_2-norm)			
Layers	10	20	30	40
2	1.41e-03	2.36e-04	5.29e-04	4.58e-04
3	3.27e-04	1.94e-04	5.69e-04	5.37e-04
4	1.25e-03	9.57e-04	4.07e-04	4.07e-04
5	4.37e-04	7.74e-04	5.36e-04	7.34e-04
6	1.41e-03	2.36e-04	5.29e-04	4.58e-04
7	3.27e-04	1.94e-04	5.69e-04	5.37e-04
8	1.25e-03	9.57e-04	4.07e-04	4.07e-04

	Neurons (Simulation time in seconds)			
Layers	10	20	30	40
2	13.64	6.66	4.76	6.19
3	18.97	6.16	8.28	6.07
4	23.52	9.45	22.68	15.29
5	16.30	19.35	14.60	13.28
6	25.05	11.89	15.01	26.50
7	20.21	21.07	31.91	23.89
8	30.42	30.56	32.81	37.15

first mesh is used to fix the number of collocation points N_f and randomly select the number of training points N_u. A second mesh of higher resolution is employed to the predicted solution. Thus, Mesh 1 was selected such that the mean interior length is close to $\Delta = 2/32 = 0.0625$. It results in 1126 and 4337 nodes for the circular and general domain, respectively. For Mesh 2, the mean interior is $\Delta = 2/64 = 0.03125$ resulting in 512 and 1961 nodes for the circular and general domain, respectively, see Table 4. For these examples, Dirichlet boundary conditions are selected according to Eq (5.2) such that the same exact solution holds.

Table 4. Number of nodes used in the numerical experiments for Example 1.

	Circle				General			
Mesh	Triangles	Nodes	Int. nodes	Bdy. nodes	Triangles	Nodes	Int. nodes	Bdy. nodes
Mesh 1	2149	1126	1025	101	906	512	396	116
Mesh 2	4337	8470	8268	202	3695	1961	1736	225

Figure 8 shows the analytical and numerical solution as well as the absolute errors. The number of

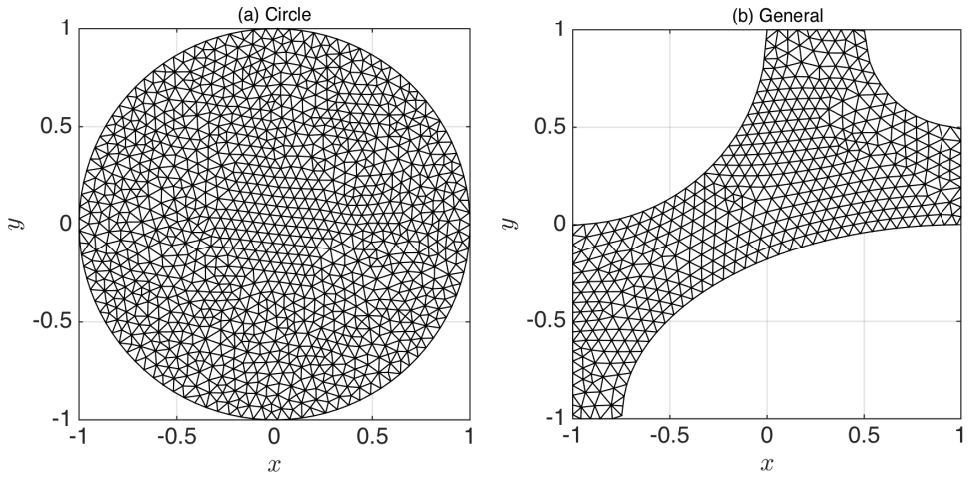

Figure 7. Example 1: (a) Circle and (b) general domains with a triangular mesh discretization used to obtain the training and collocation points.

training points is $N_u = 40$ for all cases and the number of collocation points is $N_f = 1126$ and 512 for the circle and general domain, respectively. We also approximate the solution using the same 4-layer deep neural network used for the rectangular domains. As expected, the results demonstrate that the proposed method approximates accurately the exact solution. Note that the largest errors are located in regions close to the boundary. Although not shown here, more collocation points do not contributes to obtain smaller errors. However, the behavior of the error is related not only to the number of training points, but also to its specific location. It is because the arbitrary geometry of the problem plays an important role in the approximation. If there is not boundary information in some particular region, then the error will become larger there. For instance, this can be observed in the top boundary of the general case.

The DNN is also analyzed for different number of training points. Table 5 shows the relative L_2- and L_∞- norm error for the two cases. The solution is calculated by fixing N_f on Mesh 1 and varying N_u. The errors between the predicted and analytical solution are calculated using Mesh 2. Results are more accurate as N_u increases until reaching its maximum precision. It is expected as more data at boundary describes more precise the irregular configuration of the domain. Note that more data N_u is required to obtain more accurate solutions as the complexity of the domain increases. Finally, it is important to remark that the maximum precision for all domains are similar, including the rectangular domain, as shown in Figure 9. It means that if enough information is provided by the training and collocation points, then the DNN can approximate the same solution independently of the domain shape.

5.1.3. Accuracy

As previously commented, there is a maximum precision of the predicted solution. Thus, the algorithm does not obtain more accurate solutions either increasing N_u or N_f or changing the network architecture. Thus, the main source of error is coming from other components of the neural network approximation such as the loss configuration, activation function or the optimization algorithm. In the current simulations, the minimum norm errors are close to 10^{-4} and 5×10^{-4} using the L_2- and L_∞-norm, respectively. These results hold for either rectangular or arbitrary domains, see Tables 5

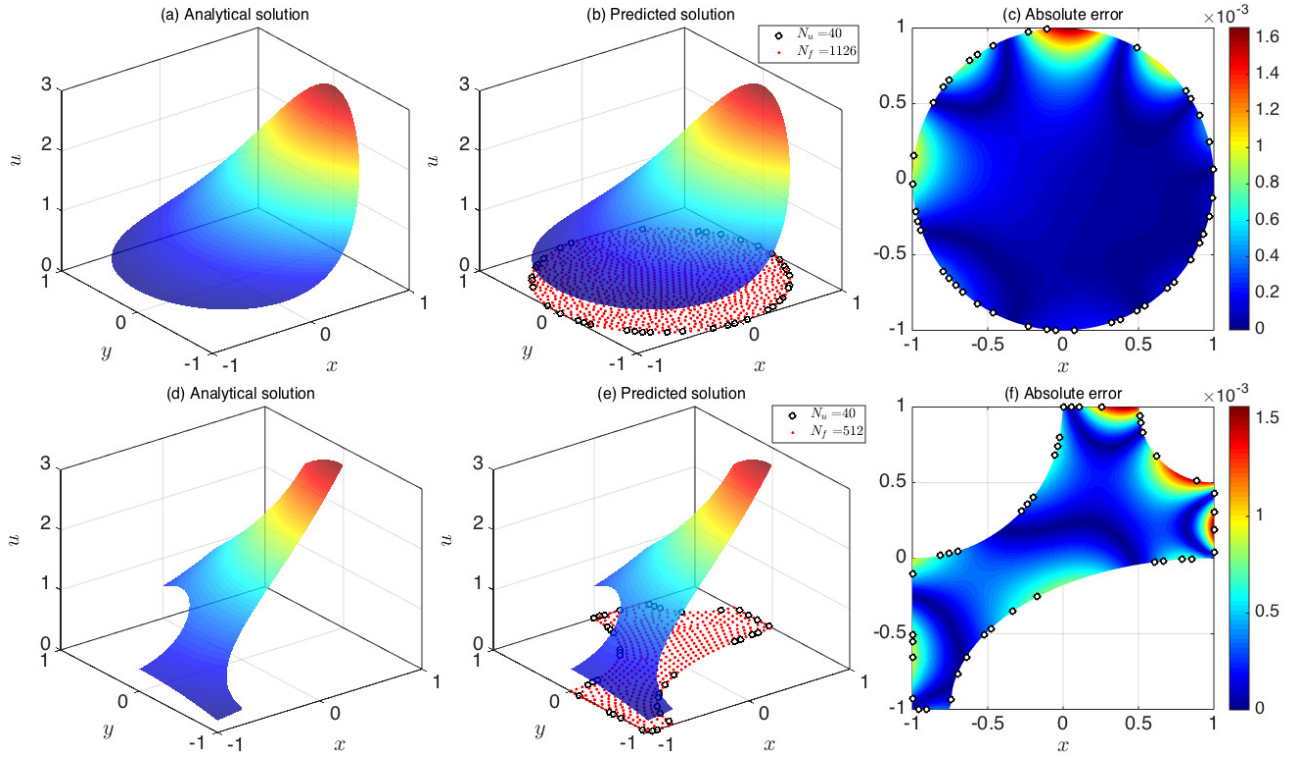

Figure 8. Example 1: Analytical solution, predicted solution and absolute errors using different domains. The number of training points is fixed to $N_u = 40$ and the number of collocation points N_f is given by the vertex points of Mesh 1.

Table 5. Example 1: Relative L_2-norm and L_∞-norm error between the predicted and the exact solution for different boundary training data N_u.

	Circle ($N_f = 1126$)		General ($N_f = 512$)	
N_u	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm
10	2.01e-01	8.21e-01	7.53e-03	3.38e-02
20	1.05e-04	6.07e-04	3.66e-04	1.99e-03
40	2.60e-04	1.65e-03	3.78e-04	1.56e-03
60	1.48e-04	7.49e-04	1.12e-04	7.21e-04
80	3.03e-04	1.62e-03	1.07e-04	4.62e-04
100	1.93e-04	9.95e-04	1.35e-04	6.21e-04

and 9. In order to get an idea of how much accurate is the approximation of the DNN, we compare our results with the numerical solutions of the standard schemes such as finite-difference (FDM) and finite-element (FEM) methods.

First, we compare our results on rectangular domains using the numerical solution of the standard second-order central FDM as shown in Table 6. An estimated order is computed as Order $= \log\left(E_{N_1}/E_{N_2}\right) / \log\left(N_1/N_2\right)$, where E_{N_1} and E_{N_2} are the errors for resolutions N_1 and N_2, respectively. Note that the accuracy of the deep neural network is close to the one given by the FDM using a 15×15 mesh grid. The neural network solution does not reach a precision similar to a second-order accurate method. This is a clear disadvantage of this methodology. However, the number of known data re-

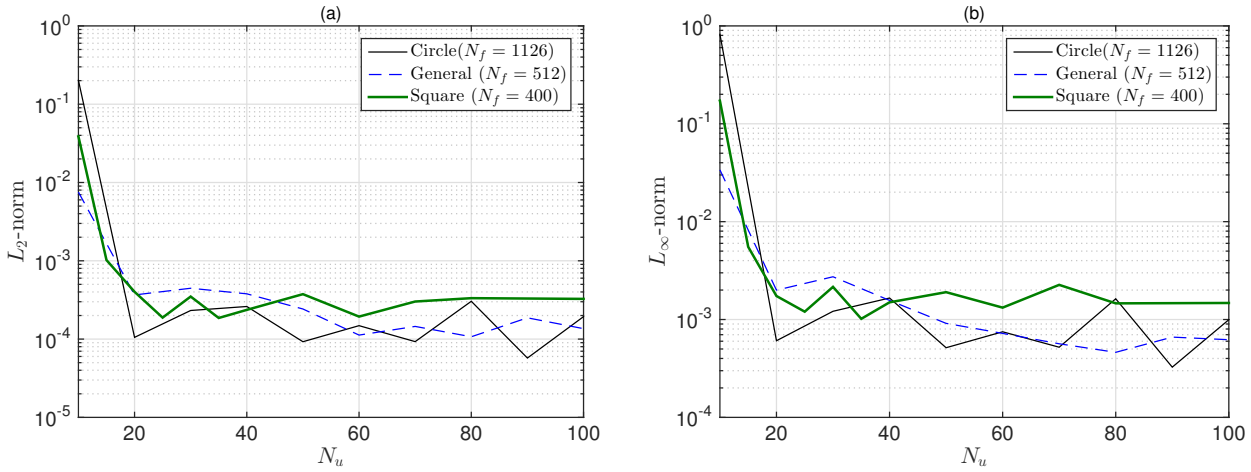

Figure 9. Example 1: (a) L_2- and (b) L_∞-norm error using different numbers of collocation points N_f at different domains and varying training data N_u.

quired at the boundary is significantly low. Moreover, this prior data knowledge is not fixed at some determinate location. This is definitely an advantage of this methodology that is not found using standard methods such as the finite-difference method. Furthermore, the deep neural network framework can be applied either to rectangular or arbitrary domains without any extra effort.

Table 6. Example 1: Error analysis on a rectangular domain using a central finite-difference method.

	Mesh		Numerical solution			
N	Total points	Bdy. points	L_2-norm	Order	L_∞-norm	Order
5	36	22	3.01e-03	—	7.80e-03	—
10	121	42	8.59e-04	1.81	2.08e-03	1.91
15	256	62	3.98e-04	1.89	9.25e-04	2.00
20	441	82	2.29e-04	1.93	5.25e-04	1.97
25	676	102	1.48e-04	1.94	3.36e-04	2.00
30	961	122	1.04e-04	1.95	2.34e-04	1.99

We also compare the accuracy of the DNN over the proposed circular domain with the numerical solution of the FEM with linear elements as implemented by Alberty et al. [44]. The minimum DNN errors for this case are 1.05×10^{-4} and 6.07×10^{-4} using the L_2- and L_∞-norm, respectively (see Table 5). Table 7 shows the same norms of the FEM using triangular meshes of different mean interior lengths (Δ). Note that the accuracy of the DNN is close to the one given by the FEM using Mesh 1. As previously commented, the neural network solution does not have the consistency property of a scheme such as the proposed FEM. Thus DNN errors do not continue decreasing as more points are used; it is again a disadvantage of this methodology. However, prior known data used here is not limited to a well-distributed triangular discretization over the whole domain as in the FEM. This is a notable advantage of DNN methodology in comparison with standard methods for arbitrary geometries such as finite-element or finite-volume method. Although the results are not shown here, we have similar conclusions when we compare errors between the DNN and FEM for the general domain.

To improve the precision of the algorithm, we analyze the performance of the neural network by

Table 7. Example 1: Errors analysis on a circular domain using the FEM.

Δ	Elements	Nodes	L_2-norm	Order	L_∞-norm	Order
0.125	561	307	3.10e-04	—	1.39e-03	—
0.0625 (Mesh 1)	2149	1126	8.16e-05	1.92	4.29e-04	1.69
0.03125 (Mesh 2)	8470	4337	1.91e-05	2.09	1.42e-04	1.59
0.015625	33339	16872	5.24e-06	1.87	4.20e-05	1.76

optimizing the error using different loss functions during the training of the neural network. Several loss functions were tested; however, most of them do not converge to a feasible solution. Overall, only four of them give us feasible solutions: the present mean square error (MSE), the error based on the L_1-norm, based on the L_1-norm and the Logarithm of the hyperbolic Cosine (LogCosH) [42]. Although the results are not presented here, there is not a significant difference between the results of the tested loss functions.

On the other hand, the optimization algorithm plays a more significant role. Let us consider the case with $N_u = 80$, $N_f = 6400$ and the initial 4-layer network. After 7.77 seconds of training with L-BFGS-B, the numerical solution is obtained with an error of 3.16×10^{-4} and 1.50×10^{-3} for the L_2- and L_∞-norm, respectively. If we change the optimizer to the Sequential Least Square Quadratic Programming (SLSQP) [43], the L_2- and L_∞-norm errors are significantly reduced to 2.32×10^{-5} and 1.33×10^{-4}, respectively. However, the training time is around 343.25 seconds. It means that the SLSQP optimizer is almost 45 times slower than L-BFGS-B method.

5.2. Example 2: Laplace equation with mix boundary conditions

For this test, we consider the following mixed problem for the Laplace equation over the region $\Omega = [0, 1] \times [0, 1]$ under the boundary conditions:

$$u_{xx} + u_{yy} = 0, \quad 0 \le x \le 1, \tag{5.3}$$

$$u(x, 0) = 0, \quad 0 \le x \le 1 \tag{5.4}$$

$$u(x, 1) = \sin(\pi x/2), \quad 0 \le x \le 1 \tag{5.5}$$

$$u(0, y) = 0, \quad 0 < y < 1, \tag{5.6}$$

$$u_x(1, y) = 0, \quad 0 < y < 1. \tag{5.7}$$

Note that besides the east edge of the square which Neumann conditions are used, Dirichlet conditions are applied at all boundaries. Both Neumann and Dirichlet training points (N_{u_n} and N_u) are randomly selected from the boundary data. The exact solution of the considered problem is given in [45] as:

$$u(x, y) = \sinh(\pi y/2)\frac{\sin(\pi x/2)}{\sinh(\pi/2)}. \tag{5.8}$$

Results using $N_{u_n} = 30$ (Dirichlet) and $N_u = 10$ (Neumann) training points are plotted in Figure 10. We apply 1600 collocation points and the same DNN arquitechture as previous examples: 3 hidden layers and 20 neurons per layer. As expected, the proposed method approximates accurately the exact solution. Note that the maximum error is located close to one of the bottom corners where Dirichlet boundary conditions are applied. Thus Neumann boundary approximations are more accurate than the

Dirichlet ones. Results also show the errors do not significantly decrease as N_{u_n} increases, as shown in Figure 11. Although, the results are not shown here, a similar behavior between Neumann and Dirichlet conditions was also observed using different elliptic equations such as the Laplace equation with exact solution $u(x, y) = e^x \cos(y)$ of Example 1.

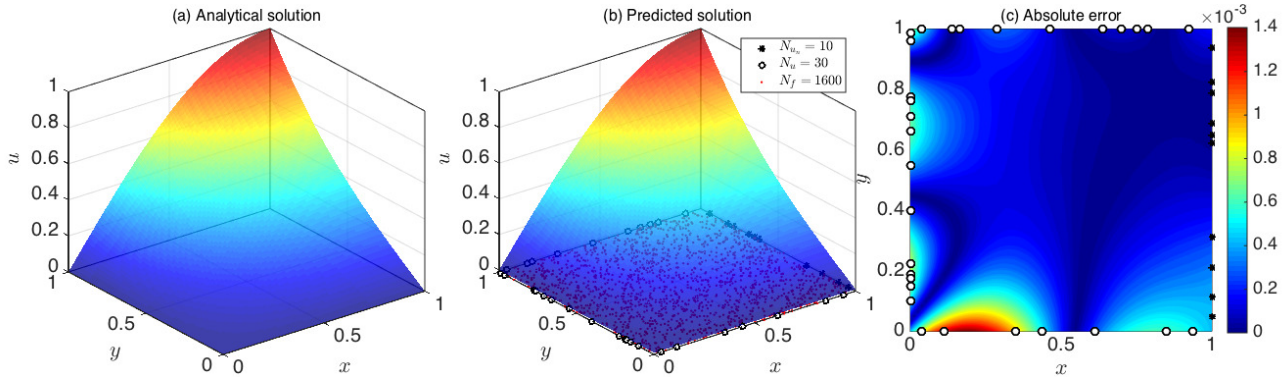

Figure 10. Example 2: Results of the Laplace equation with mix boundary conditions. The predicted solution and absolute errors are calculated using $N_{u_n} = 10$, $N_u = 30$ and $N_f = 1600$.

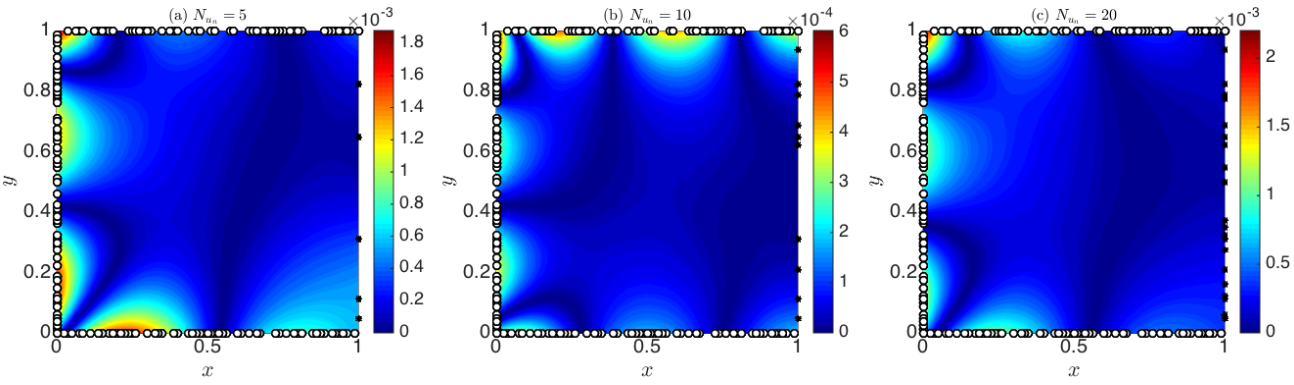

Figure 11. Example 2: Absolute errors between analytical and predicted solution using different numbers of Neumann training points, N_{u_n}, $N_u = 160$, and $N_f = 1600$.

More details of the norm errors varying the number of Neumann and Dirichlet training points are shown in Table 8. For this example, the minimum L_2- and L_∞-norm errors are 2.95×10^{-4} and 6.02×10^{-4}, respectively. They are obtained using $N_u = 160$ and $N_{u_n} = 10$. We remark that the performance of the DNN is not significantly improved with more number of collocation points. Table 9 shows the errors for $N_u = 160$ but $N_f = 6400$ instead of $N_f = 1600$ used in Table 8. Note that both tables give similar approximations. Results also confirm that few number of Neumann training points are required to reach the maximum accuracy of the method. For instance, accurate results are obtained using $N_u = 160$ and only $N_{u_n} = 5$ training points.

The performance of the DNN is also analyzed in terms of number of hidden layers and neurons and the results are shown in Table 10. We keep the same hyperbolic tangent activation function and fix $N_{u_n} = 10$, $N_u = 160$ and $N_f = 1600$ for all these simulations. Accurate solutions are obtained for all

Table 8. Example 2: Relative L_2-norm and L_∞-norm error for different number of Dirichlet and Neumann training data using $N_f = 1600$.

	$N_{u_n} = 5$		$N_{u_n} = 10$		$N_{u_n} = 20$		$N_{u_n} = 30$		$N_{u_n} = 40$	
N_u	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm
5	1.70e-02	2.48e-02	1.93e-02	2.68e-02	1.83e-02	2.53e-02	1.65e-02	2.38e-02	2.13e-02	2.73e-02
10	5.77e-03	8.25e-03	2.26e-03	2.27e-03	4.62e-03	6.35e-03	1.89e-03	3.30e-03	7.02e-03	1.03e-02
20	2.07e-03	3.31e-03	7.04e-04	1.03e-03	1.36e-03	2.32e-03	2.40e-03	4.06e-03	5.85e-04	8.67e-04
40	1.36e-03	1.93e-03	9.91e-04	1.45e-03	8.62e-04	1.20e-03	6.29e-04	1.10e-03	6.22e-04	1.01e-03
80	2.66e-03	4.09e-03	6.68e-04	1.22e-03	1.86e-03	2.77e-03	9.71e-04	1.71e-03	7.81e-04	1.41e-03
160	1.09e-03	1.87e-03	2.95e-04	6.02e-04	9.52e-04	2.20e-03	2.13e-03	3.16e-03	5.70e-04	1.33e-03

Table 9. Example 2: Relative L_2-norm and L_∞-norm error for different number of Dirichlet and Neumann training data using $N_f = 6400$.

	$N_{u_n} = 5$		$N_{u_n} = 10$		$N_{u_n} = 20$		$N_{u_n} = 30$		$N_{u_n} = 40$	
N_u	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm
160	9.47e-04	1.41e-03	2.71e-04	4.68e-04	7.67e-04	1.35e-03	7.17e-04	1.25e-03	3.29e-04	5.03e-04

cases. Note that the predicted solution using 3 hidden layers and 20 neurons is still the most accurate solution from all network configurations in this table. As previous cases, increasing the number of layers and neurons do not necessary improve the accuracy of the DNN. However, the computational power increases as these parameters become larger.

Table 10. Example 2: Relative L_2-norm and L_∞-norm error for different number of hidden layers and neurons per layer using $N_u = 160$, $N_{u_n} = 10$ and $N_f = 1600$.

	$N_e = 10$		$N_e = 20$		$N_e = 30$		$N_e = 40$	
H. layers	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm
2	7.71e-04	1.53e-03	3.20e-04	5.00e-04	4.78e-04	1.22e-03	1.05e-03	1.70e-03
3	8.36e-04	1.44e-03	2.95e-04	6.02e-04	7.31e-04	1.30e-03	1.32e-03	2.10e-03
4	3.48e-03	5.10e-03	2.51e-03	3.57e-03	1.26e-03	2.03e-03	1.55e-03	2.56e-03
5	4.11e-03	6.25e-03	1.60e-03	2.85e-03	7.02e-04	1.23e-03	1.45e-03	2.32e-03
6	3.64e-03	4.86e-03	2.15e-03	3.05e-03	2.25e-03	3.30e-03	2.31e-03	3.22e-03
7	3.55e-03	4.75e-03	4.09e-03	5.25e-03	1.94e-03	3.04e-03	2.91e-03	4.12e-03
8	4.08e-03	5.79e-03	3.08e-03	5.15e-03	1.14e-03	1.81e-03	3.68e-03	5.11e-03

5.3. Example 3: Elliptic equation with variable coefficients

For this test, we consider the elliptic equation with variable coefficients and Homogeneous Dirichlet boundary conditions over the region $\Omega = [-1, 1] \times [-1, 1]$ given by

$$\begin{aligned} (\beta(x,y)u_x)_x + (\beta(x,y)u_y)_y + \kappa(x,y)u = g(x,y), \quad (x,y) \in \Omega, \\ u(x,y) = 0, \quad (x,y) \in \partial\Omega, \end{aligned} \tag{5.9}$$

where $\beta(x, y) = \sin(x + y)$, $\kappa(x, y) = -(x^2 + y^2)$. In this example, $g(x, y)$ is selected according to the exact solution given by

$$u(x,y) = \sin(\pi x)\sin(\pi y). \tag{5.10}$$

Note that κ is always negative, thus this equation corresponds to a monotone Helmholtz-type equation with variable coefficients. The diffusion coefficient was selected such that β varies for each point of the domain.

The analytical solution, predicted solution, and absolute errors using $N_u = 40$, $N_f = 1600$ and the current 4-layer network architecture are displayed in Figure 12. As expected, accurate predictions are also obtained for elliptic equations with variable coefficients. Note that the maximum errors is located at the left-top corner with none training points. In comparison with Example 1, the predicted solution of this example requires more training points at the boundary to obtain accurate solutions.

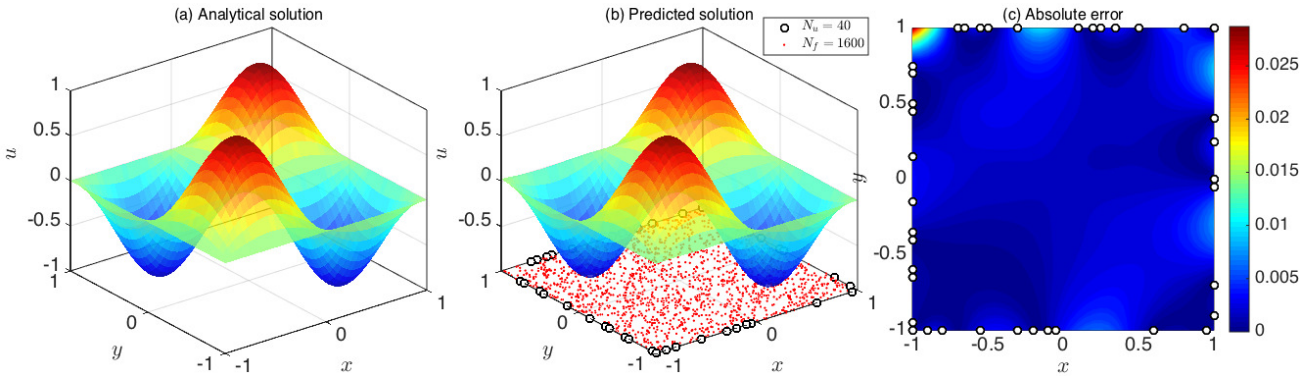

Figure 12. Example 3: (a) Analytical solution $u(x, y) = e^x \cos(y)$, (b) predicted solution, and (c) absolute errors using the training data $N_u = 40$ and collocation points $N_f = 400$. The predicted solution is calculated using the rectangular grid of 40×40 subdivisions.

Table 11 shows the relative L_2- and L_∞-norm error for different number of training and collocation points. As expected, the prediction is more accurate as the total number of training data and collocation points increased. Opposite to Example 1, inaccurate approximations are obtained using only 25 collocation points. One of the most accurate solutions are obtained using $N_u = 40$ and $N_f = 1600$. The errors are 7.44×10^{-4} for the L_2-norm and 2.6×10^{-3} for the L_∞-norm.

Table 11. Example 3: Relative L_2-norm and L_∞-norm error for different boundary training data and collocation points using 3 hidden layers and 20 neurons per layer.

	$N_f = 100$		$N_f = 400$		$N_f = 1600$		$N_f = 6400$	
N_u	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm
5	2.08e-01	4.82e-01	1.78e-01	2.65e-01	7.17e-02	2.35e-01	1.56e-01	2.63e-01
10	1.57e-01	7.59e-01	1.81e-02	9.58e-02	3.43e-02	1.68e-01	3.77e-02	1.34e-01
20	1.00e-01	6.00e-01	4.83e-03	2.41e-02	2.28e-03	1.04e-02	2.83e-03	8.75e-03
40	1.72e-03	1.82e-02	3.20e-03	9.47e-03	7.44e-04	2.60e-03	1.75e-03	4.77e-03
80	2.70e-03	9.77e-03	1.13e-03	3.55e-03	1.58e-03	6.23e-03	1.15e-03	6.47e-03
160	4.14e-03	1.29e-02	1.97e-03	9.84e-03	1.09e-03	4.54e-03	8.48e-04	2.29e-03
320	5.57e-03	1.73e-02	9.48e-04	2.98e-03	1.15e-03	3.64e-03	1.19e-03	5.36e-03

On the other hand, Table 12 shows the performance of the DNN using different number of hidden layers and neurons. We keep the same hyperbolic tangent activation function and fix $N_u = 40$ and $N_f = 1600$ for all these simulations. Note that accurate solutions are obtained even for 2 hidden layers and 10 neurons. There is not a significant improvement of the solution if we increase the number

of these parameters. However, precision may be lost if many layers are applied, such as the results obtained using eight hidden layers and $N_e = 30$ per layer.

Table 12. Example 3: Relative L_2-norm and L_∞-norm error for different number of hidden layers and neurons per layer using $N_u = 40$ and $N_f = 1600$.

	$N_e = 10$		$N_e = 20$		$N_e = 30$		$N_e = 40$	
H. layers	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm
2	1.18e-02	3.00e-02	4.57e-03	1.95e-02	1.30e-03	2.86e-03	2.61e-03	7.94e-03
3	6.63e-03	1.68e-02	7.44e-04	2.60e-03	3.42e-03	1.14e-02	1.87e-03	5.48e-03
4	1.22e-02	4.01e-02	1.75e-03	5.76e-03	1.04e-03	3.34e-03	1.17e-03	4.42e-03
5	1.31e-02	4.80e-02	1.67e-03	4.74e-03	1.02e-03	3.95e-03	2.07e-03	9.37e-03
6	5.91e-03	1.53e-02	2.31e-03	7.80e-03	2.15e-03	6.72e-03	2.25e-03	6.78e-03
7	9.67e-03	2.25e-02	5.77e-03	1.79e-02	4.47e-03	1.81e-02	1.42e-03	7.07e-03
8	6.46e-03	1.64e-02	3.30e-03	6.12e-03	8.87e-03	2.77e-02	1.39e-03	4.48e-03

6. Results with singular forces

In this section, we present several numerical tests to analyze the applicability and accuracy of the proposed DNN with the IBM on elliptic equations with singular forces. As described in Section 2, this kind of examples includes the term

$$\Im(x, y) = \int_\Gamma \left[\frac{\partial u}{\partial n}\right]_\Gamma \delta(x - X(s))\delta(y - Y(s))ds,$$

at the right-hand side of the elliptic equation which allows solutions with discontinuous derivatives. Thus, the solution of the problem is continuous at interface Γ; however, the derivative is discontinuous in the normal direction. Moreover, the magnitude of this discontinuity is known and is given as the jump condition: $C(s) = \left[\frac{\partial u}{\partial n}\right]_\Gamma$. Next, we describe the predicted solutions obtained after performing simulations for a different number of training and collocation points. Discussion about the experiments are presented in the following two examples:

- Example 4 presents a Poisson equation with a circular interface Γ. In this example, the derivative in the normal direction is discontinuous with a constant jump. The predicted solution is analyzed not only in terms of N_u and N_f but also of the number of interface points, N_s. Moreover analysis of Peskin's parameter h is done. Rectangular and arbitrary domains are tested here. Comparisons between predicted and numerical solutions of standard methods are also given at the end of this example.

- Example 5 is designed to test the capacity of the DNN to solve more challenging test cases with IBM. Here, the interface has a bubble shape solution and the jump condition is not constant.

6.1. Example 4: Poisson equation with a circular interface

This example deals with an elliptical equation with singular source given by

$$u_{xx} + u_{yy} = \int_\Gamma \frac{1}{r_0}\delta(x - X(s))\delta(y - Y(s))ds. \tag{6.1}$$

where Γ is the circle $(x - x_0)^2 + (y - y_0)^2 = r_0^2$. Dirichlet boundary conditions are applied according to the exact solution given by

$$u(x, y) = \begin{cases} 1 & \text{if } r \leq r_0, \\ 1 + \log(2r) & \text{if } r > r_0, \end{cases} \tag{6.2}$$

where $r = \sqrt{(x - x_0)^2 + (y - y_0)^2}$. Note that the solution is continuous at the interface. However, the derivative in the normal direction is discontinuous satisfying a constant jump condition: $\left[\frac{\partial u}{\partial n}\right]_\Gamma = \frac{1}{r_0} = 2$, where n is the outward normal unit vector at the interface.

An arc-length parametrization is considered for the interface. For the interface discretization, we take N_s points on Γ and $\Delta s = (2\pi/N_s)r_0$. It is important to remark that we do not have a direct reference to set the Peskin's parameter h. In the standard IBM based on finite differences, $h = \Delta x$ has shown to be a proper choice [38] where Δx is the step size of the discretization. However, there is not a mesh grid in the DNN. Instead, we have a set of points randomly distributed in the domain. In order to deal with this issue, h is initially chosen according to Δs which depends on the number of interface points. Thus, the forcing function will become sharper as more points on the interface are applied. An analysis about the selection of h will be given later in this section.

6.1.1. Predicted solution on rectangular domains

For the following simulations, we consider $\Omega = [-1, 1] \times [-1, 1]$ and the circumference with centre $(x_0, y_0) = (0, 0)$ and radius $r_0 = 1/2$. We select a number N as a reference resolution of the problem. Thus for the number of training points, $N_u = N$ is randomly chosen from the boundary data. For number of collocation points, $N_f = N^2$ is sampled using a space filling Latin Hypercube Sampling strategy inside the domain as previously done in Example 1. The predicted solution is always calculated using the points of the $(N + 1) \times (N + 1)$ rectangular grid. For the interface points, we take $N_s = N$. In the numerical experiments, we have found that beyond this point, increasing the number of points on the interface gives little improvement in the solution. For the discrete delta function, we take $h = \Delta s$. The analytical solution, predicted solution, and forcing function for $N = 80$ is shown in Figure 13. It is found using the same 4-layer DNN of previous examples and $h \approx 0.04$. The results demonstrate that the proposed method approximates accurately the exact solution.

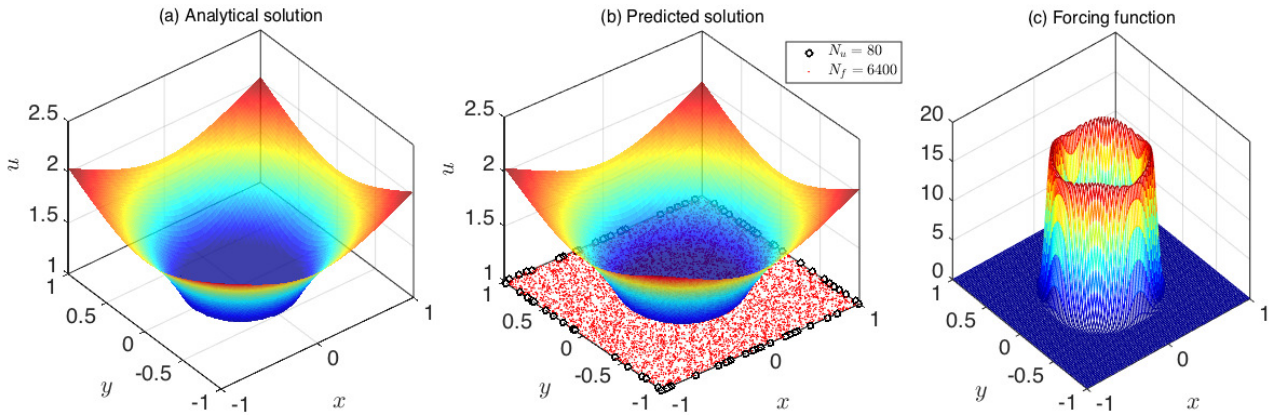

Figure 13. Example 4: (a) Analytical solution, (b) predicted solution, and (c) forcing function with singular force using the Peskin's approach with $h \approx 0.04$ ($N = 80$).

Figure 14 shows the results for three sets of points corresponding to $N = 40$, 80 and 160. Note that the IBM initially dominates the global accuracy of the method because the largest errors are located in regions close to the interface. However, as more points are used, the errors on the interface become smaller and the global error is mainly determine by the number of training points and its random location at the boundary. A comparison of the predicted and exact solutions at $y = 0$ is presented in Figure 15(a). Note that the DNN accurately approximates the solution close to the interface as more points are considered. Figure 15(b) presents the norm errors for resolutions ranging from $N = 10$ to $N = 160$. The effect in the global accuracy of IBM and DNN can be clearly noticed here. The behavior of the error before and after $N = 90$ is different. The first section has a smooth decreasing tendency; meanwhile, the second part becomes stable and oscillating as reported in previous examples without interface. It means that the error will not significantly decrease even if more points are considered. Moreover, results show that resolutions at the first section follow a first-order approximation as the standard IBM based on finite differences, as shown in Table 13.

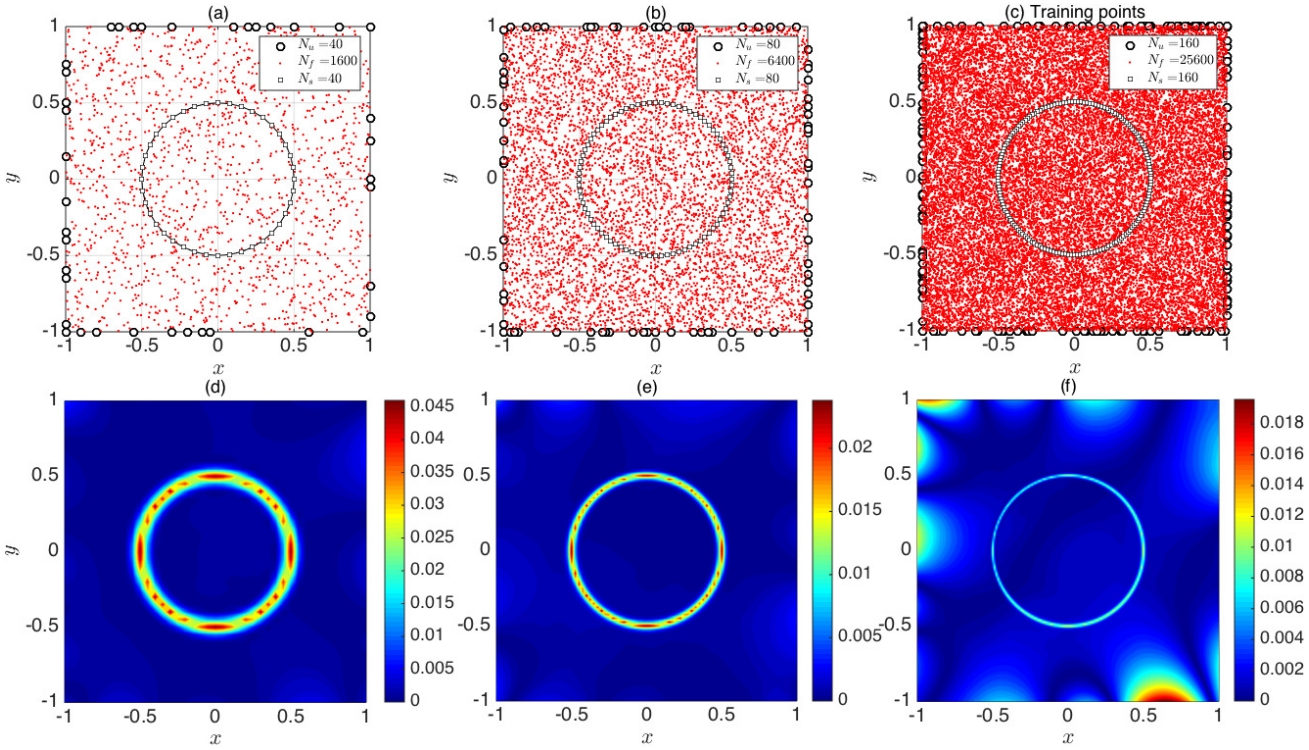

Figure 14. Example 4: (a)–(c) Training, collocation, and interface points, (d)–(f) absolute errors using $N_u = N$, $N_f = N^2$, $N_s = N$ and $h = \Delta s$ for $N = 40$, 80 and 160 as reference resolutions.

Table 13. Example 4: Error analysis for different values of h.

N	$h = \pi/N$	L_∞-norm	Order
20	0.1571	9.44e-02	—
40	0.0785	4.64e-02	1.02
80	0.0393	2.40e-02	0.95

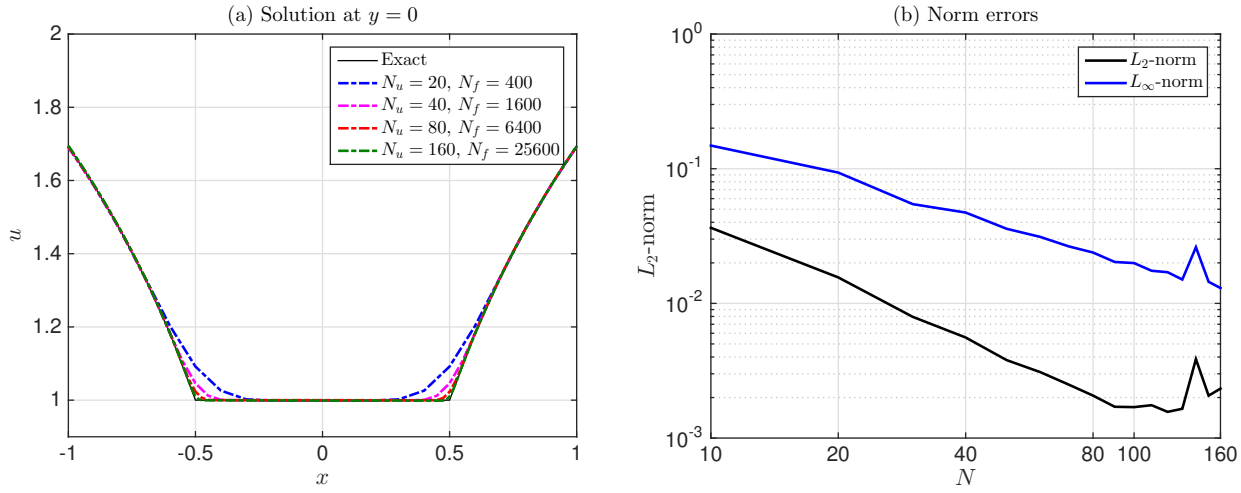

Figure 15. Example 4: (a) Predicted solution at $y = 0$ along with $N = 20$, 40, 80 and 160 as reference resolutions. (b) Norm errors varying the resolution of the problem; both axis are in logarithm scale. Here, we consider $N_u = N$, $N_f = N^2$, $N_s = N$ and $h = \Delta s$.

More details about these errors can be found in Table 14. Results show that the error stabilizes at values close to 2×10^{-3} and 2×10^{-2} for the relative L_2- and L_∞-norm, respectively. Note that $h = 0.3141$ ($N = 10$) does not obtain an accurate approximation to the problem as the forcing function is not sharp enough. On the other hand, if we keep increasing N, inaccurate solutions are obtained because h is too small. Finally, we remark that as the number of points is increased, more computational time is required in the training process. For this example, a single simulation takes around 100 seconds of training. We can compare this with the 6 seconds of CPU time used for the simulations of Example 1.

Table 14. Example 4: Error analysis of the predicted solution of the deep neural network.

N_u	N_f	N_s	$h = \Delta s$	L_2-norm	L_∞-norm	iterations	Loss	CPU time (s)
10	100	10	0.3141	3.88e-02	1.55e-01	3799	0.00001	53.28
20	400	20	0.1571	1.60e-02	9.44e-02	8077	0.0001	109.26
30	900	30	0.1047	7.83e-03	5.43e-02	8311	0.0003	115.03
40	1600	40	0.0785	5.51e-03	4.64e-02	9383	0.0007	133.46
50	2500	50	0.0628	4.05e-03	3.74e-02	9050	0.0014	131.84
60	3600	60	0.0524	3.25e-03	3.12e-02	14862	0.0012	216.12
70	4900	70	0.0448	2.50e-03	2.69e-02	13365	0.0017	194.66
80	6400	80	0.0393	2.11e-03	2.40e-02	14502	0.0021	214.21
90	8100	90	0.0349	2.21e-03	2.10e-02	10873	0.0025	161.41
160	25600	160	0.0196	2.15e-03	1.98e-02	20635	0.0056	436.84

The performance of the DNN with IBM is also analyzed for different number of training and collocation points. Table 15 shows the relative L_2-norm and L_∞-norm error for this analysis using the 4-layer network architecture. The latent solution is calculated by fixing all parameters except N_u. Four cases are presented: $N = 20$, $N = 40$, $N = 80$ and $N = 160$. We remark that the number of interface points, N_s, and h are also fixed according to N. Results show that there is not a significant difference in the errors for $N_u \geq N$. Moreover, the maximum precision of the DNN is already reached for $N = 80$ ($N_f = 6400$, $N_u \geq 80$). Note that the results are already very similar for $N = 160$ ($N_f = 25600$,

$N_u \geq 80$). We observe that although the errors are bigger, the DNN still calculates correctly the solution of the differential equation using only $N_u = 10$. Figure 16 shows the case for $N = 80$. Note that accurate results are obtained even if there is not any boundary point at the west boundary. This is a clear advantage of this methodology; in particular, if limited data is available.

Table 15. Example 4: Error analysis varying the number of training data N_u. Here, we consider $N_f = N^2$, $N_s = N$ and $h = \Delta s$.

	$N = 20$		$N = 40$		$N = 80$		$N = 160$	
N_u	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm
10	2.61e-02	1.47e-01	1.52e-02	9.98e-02	4.43e-03	2.39e-02	2.27e-02	1.43e-01
20	1.60e-02	9.44e-02	6.27e-03	4.60e-02	2.60e-03	2.55e-02	3.76e-03	2.74e-02
40	1.58e-02	9.35e-02	5.52e-03	4.64e-02	2.25e-03	2.36e-02	4.55e-03	3.50e-02
80	1.59e-02	9.41e-02	5.99e-03	4.83e-02	2.11e-03	2.40e-02	2.87e-03	2.06e-02
160	1.57e-02	9.34e-02	5.68e-03	4.72e-02	2.06e-03	2.38e-02	2.15e-03	1.97e-02
320	1.58e-02	9.40e-02	5.62e-03	4.70e-02	2.07e-03	2.35e-02	2.27e-03	1.38e-02

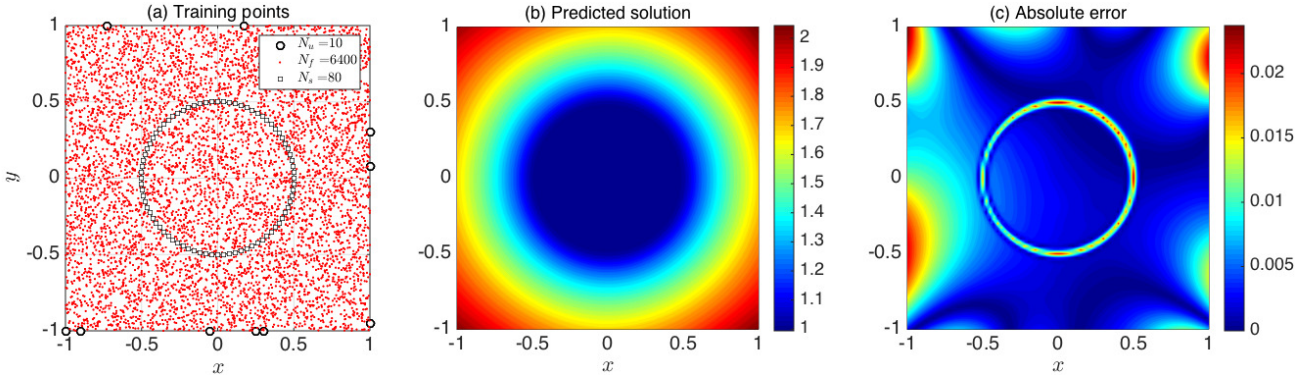

Figure 16. Example 4: Training points, predicted solution and absolute errors for $N_u = 10$, $N_f = 6400$, $N_s = 40$ and $h = \Delta s$.

Now, we investigate the effect of the number of interface points and the size of h in the approximation. As discussed in Section 3.2, the interface points is mainly to approximate the integral over the interface. In this case, the interface has a circular shape and few points are enough to obtain accurate approximations. Table 16 shows the error analysis varying the number of interface points. In all simulations, besides the number of training and collocation points, parameter h is fixed according to the example N by taking $h = (2\pi/N)r_0$. Numerical results show that beyond $N_s = N$, increasing the number of points on the interface gives little improvement in the solution.

Table 16. Example 4: Error analysis varying the number of interface points, N_s.

	$N = 20$		$N = 40$		$N = 80$		$N = 160$	
N_s	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm	L_2-norm	L_∞-norm
20	1.60e-02	9.45e-02	6.17e-03	5.18e-02	3.23e-02	1.40e-01	2.41e-02	1.37e-01
40	1.58e-02	9.53e-02	5.51e-03	4.64e-02	3.62e-03	2.82e-02	2.71e-02	1.43e-01
80	1.56e-02	9.30e-02	5.66e-03	4.71e-02	2.11e-03	2.40e-02	2.67e-03	1.45e-02
160	1.56e-02	9.33e-02	5.70e-03	4.75e-02	2.28e-03	2.46e-02	2.16e-03	1.98e-02

As previously commented, there is not an obvious choice to select Peskin's parameter h. Figure 17 shows the predicted at $y = 0$ for different h values. It is clear that $h = 0.2$ is too big to capture the singularity close to the interface and $h = 0.006$ is too small to give accurate results. Note that h equal to $\Delta x = 2/N$ and $\Delta s = \pi/N_s$ give accurate results; however the solution for $h = \Delta x$ fails to approximate values inside the interface. For more details, Figure 18 shows the norm error analysis by varying h for different N cases ($N_u = N$, $N_f = N^2$, and $N_s = N$). Results confirm that the correct selection of h plays an important role in the final approximation. The error decreases as h becomes smaller; however, there is a breaking point where the simulations begin to give inaccurate results. As the DNN accuracy depends on the number of training and collocation points, both Δx and Δs look like plausible choices for h. However, once the minimum error of DNN is reached, the choice $h = \Delta x$ is located in the region where the errors begin to increase. On the other hand, $h = \Delta s$ is located closer to minimum error.

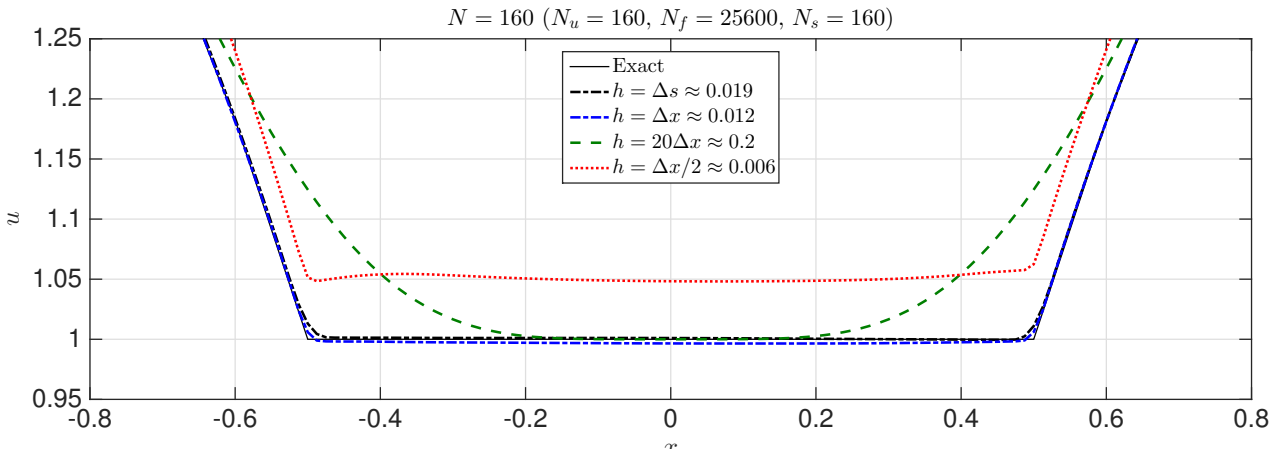

Figure 17. Example 4: Predicted and the exact solution at $y = 0$ for $N_u = 160$, $N_f = 25600$, $N_s = 160$ and varying h.

6.1.2. Predicted solution on arbitrary domains

To demonstrate the capacity of the DNN and IBM on arbitrary domains, let us consider the interface problem given by Eqs (6.1) and (6.2) with two irregular domains. First, the problem is simulated in the circle with radius 1 and Γ is a circle of $r_0 = 1/2$ and $(x_0, y_0) = (0, 0)$. Second, we use the general domain given in Example 2 and Γ is the circle of $r_0 = 1/4$ and $(x_0, y_0) = (0.3, 0.3)$. In these simulations, N_f is selected according to Mesh 1 and $N_u = 32$. As the previous example, we should decide how to choose N_s and h. In this case, they are selected according to the mean interior length of the triangular mesh. For Mesh 1, this value is close to $\Delta = 0.0625$. Thus, the number of interface points and the Peskin's parameter are chosen as $N_s = 2/\Delta = 32$ and $h = \Delta$, respectively. For the second case, we expect that half value of h will give more accurate results as the interface radius is only half of the first case.

The predicted solution and absolute errors for the two examples are shown in Figure 19. For $h = \Delta$, large errors are accumulated around the interface, thus the maximum error is due to IBM. For the circular case, inaccurate solutions are obtained for $h = \Delta/2$. On the other hand, for the second case, the absolute errors are larger than the first case using the same $h = \Delta$. Moreover, the numerical solution is

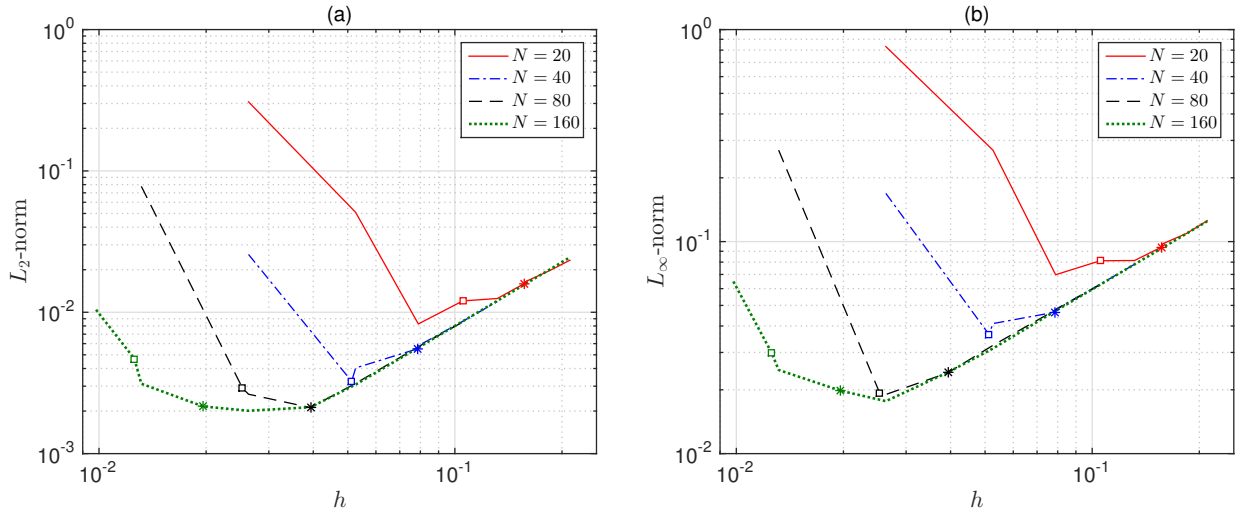

Figure 18. Example 4: Error analysis by varying Peskin's parameter h for different N cases. The asterisk markers correspond to the value $h = \Delta s$ and the square markers to $h = \Delta x$.

still accurate for $\Delta/2$. The interface is reduced to half of radius, thus these results are expected.

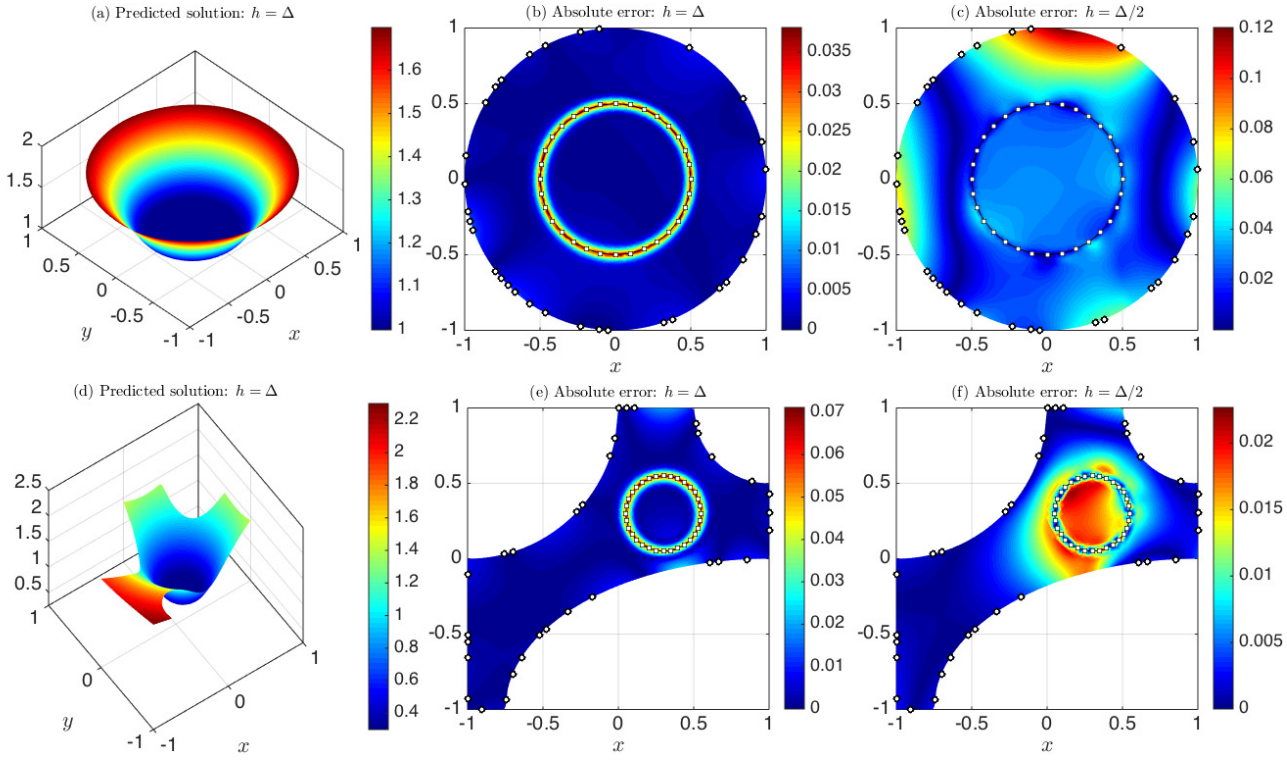

Figure 19. Example 4: Predicted solution and absolute errors using two different h values. The number of collocation points are according to Table 4.

More details about the predicted solution can be found in Figure 20 and Table 17. The figure shows

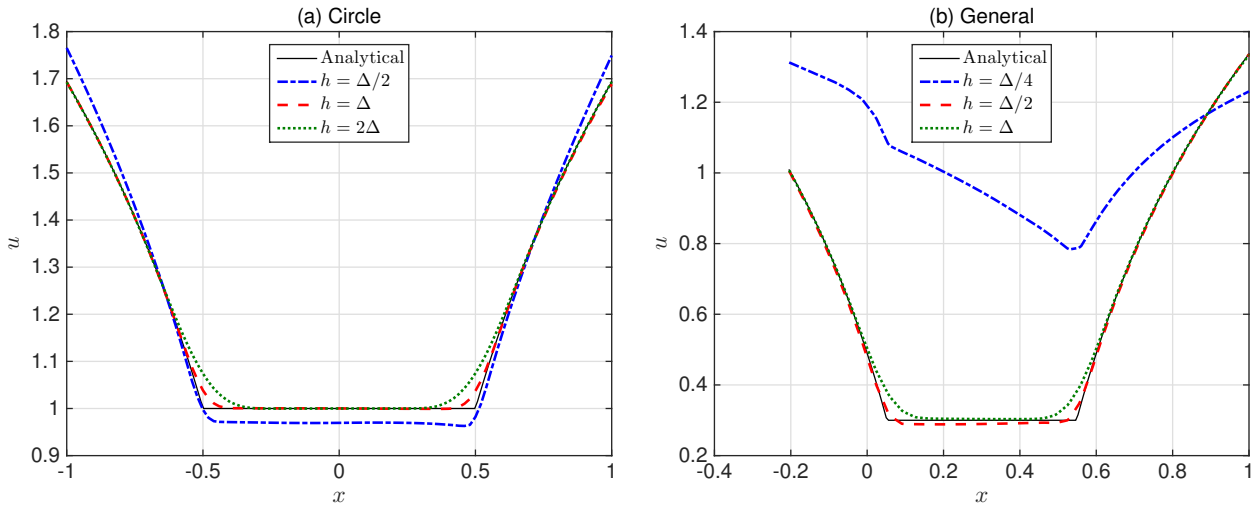

Figure 20. Example 4: Predicted solution using two different h values at $y = y_0$.

the predicted at $y = y_0$ for different h values. Note that for the general domain, the precision is lost for $h = \Delta/4$. Table 17 shows an error analysis for Example 4 using different values of h. In all simulations, the number of training, collocated, interface and predicted points are fixed. Results confirm that the DNN can recover the solution of the problem with high precision for $h = \Delta$ and $h = \Delta/2$ for the circular and general domain, respectively. As expected, the norm values firstly decrease when the value of h decrease; however, the norm error begin to increase when h goes toward zero. Note that for the circular domain, the errors are close to the ones in the rectangular domain with a similar number of training and collocation points.

Table 17. Example 4: Error analysis with different h values, $N_u = 32$ and $N_s = 32$. The number of collocation points corresponds to Mesh 1.

	Circle				General			
	$h = \Delta/4$	$h = \Delta/2$	$h = \Delta$	$h = 2\Delta$	$h = \Delta/8$	$h = \Delta/4$	$h = \Delta/2$	$h = \Delta$
L_2-norm	5.31e-01	2.72e-02	5.03e-03	1.36e-02	5.28e-01	2.63e-01	6.00e-03	9.53e-03
L_∞-norm	1.03e+00	1.20e-01	3.81e-02	7.39e-02	1.24e+00	7.99e-01	2.26e-02	7.11e-02

6.1.3. Accuracy

On the other hand, the maximum precision of the DNN is reached with an error of 2×10^{-3} and 2×10^{-2} for the relative L_2- and L_∞-norm, respectively. To get an idea of how much accurate is the approximation of the DNN, we compare our results with the numerical solution of the standard IBM based on a finite difference method using a rectangular domain. Table 18 shows the results presented for Leveque and Li [38] for the same problem. For the discrete delta function method, they consider N_s points on the interface Γ, where $N_s = 2/\Delta x = 2/\Delta y$ is the also number of uniform grid points in each direction; and $h = \Delta x$. Note that the standard IBM for $N = 20$ is not accurate enough to reach the first order of accuracy of the model; however, the DNN does. Both formulations have similar accuracy for $N = 40$ and $N = 80$. However, as the limit of the accuracy of the predicted solution is reached, the

performed of the classical approximation is more accurate than the proposed solution, as expected.

Table 18. Example 4: Error analysis of the L_∞-norm using the standard IBM and DNN.

N_x	h	Finite difference IBM	Order	DNN
20	0.10526	3.61e-01	—	8.118e-02
40	0.05128	2.64e-02	3.77	3.649e-02
80	0.02531	1.32e-02	1.00	1.923e-02
160	0.01257	6.68e-03	0.98	2.986e-02

6.2. Example 5

Finally, we propose a new and more challenge example to test the proposed immersed boundary neural network. Here, the solution is continuous with a non-constant normal jump condition on the interface. Moreover, none circular interfaces can be chosen with an available analytical solution. In this example, the interface geometry resembles the bubble shape transition of the rising air bubble immersed in water [46, 47].

In this example, the Poisson equation is selected with a singular source term along Γ as follows

$$u_{xx} + u_{yy} = g + \int_\Gamma C(s)\delta(x - X(s))\delta(y - Y(s))ds. \tag{6.3}$$

The exact solution is defined by a four degree polynomial given by

$$u(x,t) = \begin{cases} z_0^2, & \text{if } r \le r_0, \\ (x^2 - a)(x^2 - b) + (y^2 - a)(y^2 - b), & \text{if } r > r_0. \end{cases} \tag{6.4}$$

Thus, the right-hand side g is given by

$$g(x,y) = \begin{cases} 0 & \text{if } r \le r_0, \\ 12(x^2 + y^2) - 4(a + b) & \text{if } r > r_0. \end{cases} \tag{6.5}$$

The interface, Γ, corresponds to the closed curve

$$r = r_0, \tag{6.6}$$

where

$$r = \left(x^2 - \frac{a+b}{2}\right)^2 + \left(y^2 - \frac{a+b}{2}\right)^2, \tag{6.7}$$

and

$$r_0 = z_0^2 - 2ab - (a + b)^2/2. \tag{6.8}$$

The right-hand side integral now contains a non-constant C function which is defined according to the exact solution. We remark that the solution is continuous at the interface. However, the derivative in the normal direction is discontinuous satisfying the following jump condition

$$C(s) = \left[\frac{\partial u}{\partial n}\right]_\Gamma = \sqrt{(4x^3 - 2(a + b)x)^2 + (4y^3 - 2(a + b)y)^2}. \tag{6.9}$$

In this example, we fix the parameters $a = 1/2$ and $b = 1/4$. Note that different interface shapes can be obtained by varying z_0. This curve can be parametrized as follows

$$(x(\theta), y(\theta)) = \left(\sqrt{\sqrt{r_0}\cos(\theta) + (a+b)/2},\ \sqrt{\sqrt{r_0}\sin(\theta) + (a+b)/2} \right). \tag{6.10}$$

where $0 \le \theta < 2\pi$. Note that this is not an arc-length parametrization. A square domain is selected with different interface configurations. Dirichlet boundary conditions are applied according to the exact solution.

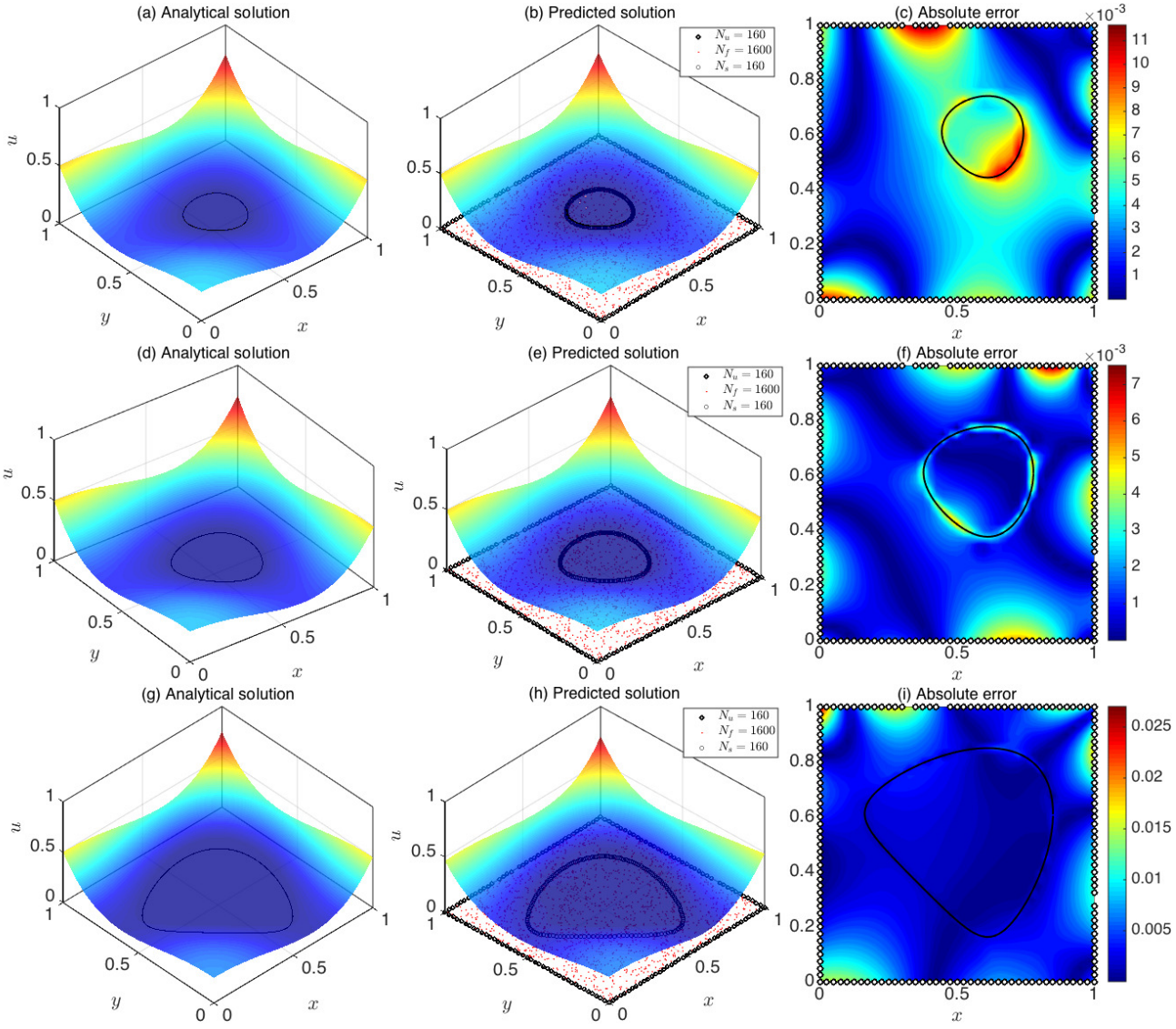

Figure 21. Example 5: Exact solution, predicted solution and absolute errors using $z_0 = 0$ (top), 0.15 (middle) and 0.3 (bottom) for $N_u = 160$, $N_f = 1600$, $N_s = 160$.

In the present simulations, three different interfaces are used. They correspond to $z_0 = 0$, 0.15 and 0.3. As the value of z_0 increases, the round shape is lost and the interface has a flat region as

shown in Figure 21. Thus, to correctly approximate the integral over Γ, we consider 160 points on the interface. The number of training and collocation points are set as $N_u = 160$ and $N_f = 1600$, respectively. The DNN architecture is fixed to 4 layers with 40 neurons per hidden layer. Figure 21 shows the numerical solution and absolute errors for the three different interfaces. In all simulations, the DNN is capable to capture correctly the solution. Results also show that the numerical solution becomes less accurate as the interface becomes more irregular. The global accuracy of the problem is affected for several parameters of the DNN formulation. First, as analyzed in the previous examples, the choice of h significantly contributes to the precision of the method. We expect different h values for different interfaces. For instance, the Peskin's parameter for the first two cases is $h = 1/40 = 0.025$ and for the third case $h = 1/80 = 0.0125$. Second, the number of training points also plays an important role in the approximation of this example. In particular, more errors are found for the cases with an interface close to the boundary. Note that more N_u points than other examples are used here. In the first two cases, the largest absolute errors are equally distributed between the interface and the boundary. It shows that the DNN recover the solution of the problem with high precision. However, for the third case, the errors are bigger and located in regions close to the boundary.

More details about the precision of the numerical solution can be observed in Figure 22. This shows the exact solution, predicted solution, and absolute error at $x = 0.5$ and $y = 0.5$ for $z_0 = 1.5$. Note how the maximum error comes from the interface and boundary approximation as expected. However, the numerical approximation fits nicely the exact solution.

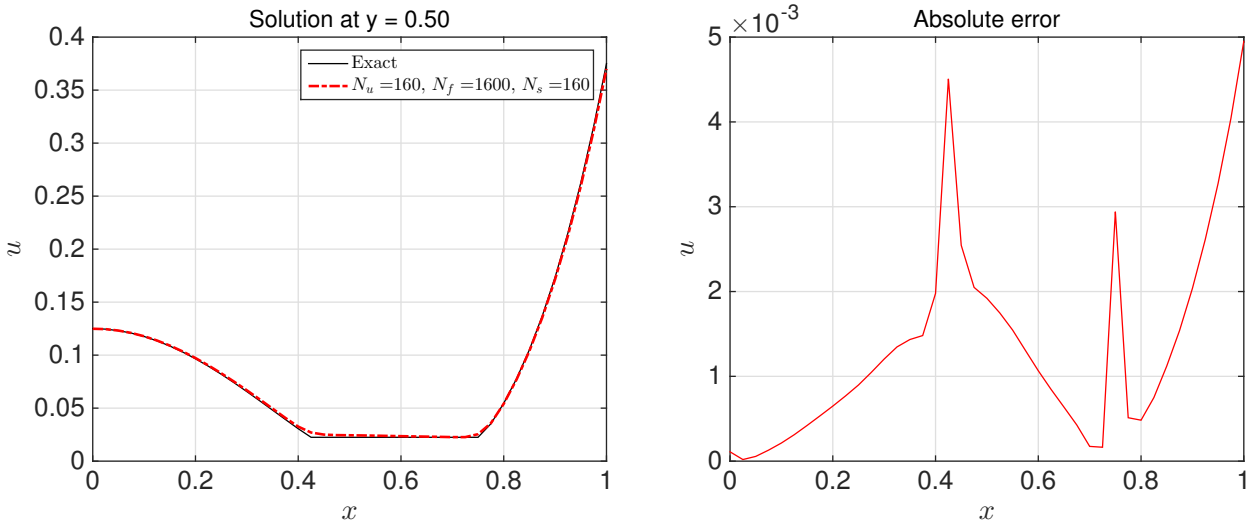

Figure 22. Example 5: Exact solution, predicted solution and absolute errors at $y = 0.5$ for $N_u = 160$, $N_f = 1600$, $N_s = 160$ using $z_0 = 1.5$.

7. Conclusions

In this paper we have analyzed the capacity of a deep learning framework for solving two-dimensional elliptic equations on arbitrary domains. The source term considers singularities with a delta function on an interface. This work follows the ideas of the physical-inform neural networks to

approximate the solutions on arbitrary domains and the immersed boundary method to deal with the singularity. The results demonstrate that the proposed method approximates accurately the analytical solutions for elliptic problems with and without singularity. For singular forces, the choice of the Peskin's parameter significantly contributes obtaining accurate solutions. As expected, the method is more accurate as the number of training and collocation points are increased. However, there is a limit in the precision of the method even if more points are included. The source of this error are coming from other components of the approximation such as the optimization algorithm. Although the numerical results are not as precise as other classical methods, such as finite-difference scheme, one of the main advantages of this method is the low number of known data and their arbitrary location required at the boundaries. This flexibility can not be obtained using finite-difference, finite-element or finite-volume methods. Furthermore, the deep neural network framework can be applied to arbitrary domains without any extra effort as we analyzed in this paper. Future work involves further improvement of the present algorithm. We also plan to study the behavior of this methodology on elliptic equations with discontinuities solutions arising from discontinuities present in the coefficients of this equation.

Acknowledgments

This work was partially supported by the National Council of Science and Technology, Mexico under the program Cátedras CONACYT.

Conflict of interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

1. I. J. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, Cambridge, 2016.
2. G. Shrivastava, S. Karmakar, M. K. Kowar, P. Guhathakurta, Application of artificial neural networks in weather forecasting: a comprehensive literature review, *Int. J. Comput. Appl.*, **51** (2012), 17–29.
3. Q. Zhong, Y. Liu, X. Ao, B. Hu, J. Feng, J. Tang, et al., Financial Defaulter Detection on Online Credit Payment via Multi-view Attributed Heterogeneous Information Network, *Proc. Web Conf. 2020*, (2020), 785–795.
4. M. Lam, Neural network techniques for financial performance prediction: integrating fundamental and technical analysis, *Decis. Support Syst.*, **37** (2004), 567–581.
5. G. Youyang, COVID-19 Projections Using Machine Learning, 2020. Available from: https://covid19-projections.com/about.
6. J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, *Proc. Natl. Acad. Sci.*, **115** (2018), 8505–8510.
7. J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, *J. Comput. Phys.*, **375** (2018), 1339–1364.

8. K. Xu, E. Darve, The neural network approach to inverse problems in differential equations, preprint, arXiv:1901.07758.
9. 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.
10. C. Michoski, M. Milosavljevic, T. Oliver, D. Hatch, Solving irregular and data-enriched differential equations using deep neural networks, preprint, arXiv:1905.04351.
11. K. Gurney, An introduction to neural networks, CRC press, Boca Raton, 1997.
12. A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, preprint, arXiv:1502.05767.
13. S. Marsland, Machine learning: an algorithmic perspective, CRC press, Boca Raton, 2015.
14. S. Pattanayak, Pro deep learning with tensorflow: A mathematical approach to advanced artificial intelligence in python, Apress, New York, 2017.
15. G. Zaccone, R. Karim, Deep learning with tensorFlow: Explore neural networks and build intelligent systems with python, Packt Publishing Ltd, Birmingham, 2018.
16. M. A. Nielsen, Neural networks and deep learning, Determination press, San Francisco, 2015.
17. Z. Mao, A. D. Jagtap, G. E. Karniadakis, Physics-informed neural networks for high-speed flows, *Comput. Methods Appl. Mech. Eng.*, **360** (2020), 112789.
18. L. Sun, H. Gao, S. Pan, J. X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Comput. Methods Appl. Mech. Eng.*, **361** (2020),112732.
19. M. Raissi, A. Yazdani, G. E. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, *Sci.*, **367** (2020), 1026–1030.
20. J. Berg, K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, *Neurocomputing*, **317** (2018), 28–41.
21. E. Weinan, B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, *Commun. Math. Stat.*, **6** (2018), 1–12.
22. C. Anitescu, E. Atroshchenko, N. Alajlan, T. Rabczuk, Artificial neural network methods for the solution of second order boundary value problems, *Comput. Mater. Continua*, **59** (2019), 345–359.
23. L. Lu, X. Meng, Z. Mao, G. E. Karniadakis, DeepXDE: A deep learning library for solving differential equations, preprint, arXiv:1907.04502.
24. A. Koryagin, R. Khudorozkov, S. Tsimfer, PyDEns: A python framework for solving differential equations with neural networks, preprint, arXiv:1909.11544.
25. J. Han, M. Nica, A.R. Stinchcombe, A derivative-free method for solving elliptic partial differential equations with deep neural networks, preprint, arXiv:2001.06145.
26. Y. Zang, G. Bao, X. Ye, H. Zhou, Weak adversarial networks for high-dimensional partial differential equations, *J. Comput. Phys.*, (2020), 109409.
27. C. S. Peskin, Flow patterns around heart valves: a numerical method, *J. Comput. Phys.*, **10** (1972), 252–271.

28.C. S. Peskin, Numerical analysis of blood flow in the heart, *J. Comput. Phys.*, **25** (1977), 220–252.

29.Y. Kim, C. S. Peskin, Penalty immersed boundary method for an elastic boundary with mass, *Physics of Fluids*, **19(5)** (2007), 053103.

30.Y. Mori, C. S. Peskin, Implicit second-order immersed boundary methods with boundary mass, *Comput. Methods Appl. Mech. Eng.*, **197** (2008), 2049–2067.

31.Y. Kim, C. S. Peskin, 2-D parachute simulation by the immersed boundary method, *SIAM J. Sci. Comput.*, **28** (2006), 2294–2312.

32.S. Lim, A. Ferent, X. S. Wang, C. S. Peskin, Dynamics of a closed rod with twist and bend in fluid, *SIAM J. Sci. Comput.*, **31** (2008), 273–302.

33.P. J. Atzberger, C. S. Peskin, A Brownian dynamics model of kinesin in three dimensions incorporating the force-extension profile of the coiled-coil cargo tether, *Bull. Math. Biol.*, **68** (2006), 131.

34.P. J. Atzberger, P. R. Kramer, C. S. Peskin, Stochastic immersed boundary method incorporating thermal fluctuations, *Proc. Appl. Math. Mech.*, **7** (2007), 1121401–1121402.

35.T. G. Fai, B. E. Griffith,Y. Mori, C. S. Peskin, Immersed boundary method for variable viscosity and variable density problems using fast constant-coefficient linear solvers I: Numerical method and results, *SIAM J. Sci. Comput.*, **35** (2013), B1132–B1161.

36.C. S. Peskin, The immersed boundary method, *Acta Numerica*, **11** (2002), 479–517.

37.R. Mittal, G. Iaccarino, Immersed boundary methods, *Annu. Rev. Fluid Mech.*, **37** (2005), 239–261.

38.R. J. Leveque, Z. Li, The immersed interface method for elliptic equations with discontinuous coefficients and singular sources, *SIAM J. Numer. Anal.*, **31** (1994), 1019–1044.

39.D. A. Fournier, H. J. Skaug, J. Ancheta, J. Ianelli, A. Magnusson, M. N. Maunder, et al., Ad model builder: using automatic differentiation for statistical inference of highly parameterized complex nonlinear models, *Optim. Methods Software*, **27** (2012), 233–249.

40.M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, et al, Tensorflow: Large-scale machine learning on heterogeneous distributed systems, preprint, `arXiv:1603.04467`.

41.R. Byrd, P. Lu, J. Nocedal, C. Zhu, A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, **16** (1995), 1190–1208.

42.W. Qi, M. Yue, Z. Kun, T. Yingjie, A Comprehensive Survey of Loss Functions in Machine Learning, *Ann. Data Sci.*, **1** (2020), 1–26.

43.F. Zhengqing, L. Goulin, G. Lanlan, Sequential quadratic programming method for nonlinear least squares estimation and its application, *Math. Probl. Eng.*, **2019**(2019).

44.J. Alberty, C. Carstensen, S. A. Funken, Remarks around 50 lines of Matlab: short finite element implementation. *Numer. algorithms*, **20** (1999), 117–137.

45.F. Civan, C. M. Sliepcevich, Solution of the Poisson equation by differential quadrature. *Int. J. Numer. Methods Eng.*, **19** (1983), 711–724.

46.J. Hua, J. Lou, Numerical simulation of bubble rising in viscous liquid. *J. Comput. Phys.*, **222** (2007), 769–795.

47.M. Uh, S. Xu, The immersed interface method for simulating two-fluid flows. *Numer. Math. Theory, Methods Appl.*, **7** (2014), 447–472.

AIMS Press