Copyright © 2019 Tech Science Press CMC, vol.59, no.1, pp.345-359, 2019

# Artificial Neural Network Methods for the Solution of Second Order Boundary Value Problems

**Cosmin Anitescu[1], Elena Atroshchenko[2], Naif Alajlan[3] and Timon Rabczuk[3, *]**

**Abstract:** We present a method for solving partial differential equations using artificial neural networks and an adaptive collocation strategy. In this procedure, a coarse grid of training points is used at the initial training stages, while more points are added at later stages based on the value of the residual at a larger set of evaluation points. This method increases the robustness of the neural network approximation and can result in significant computational savings, particularly when the solution is non-smooth. Numerical results are presented for benchmark problems for scalar-valued PDEs, namely Poisson and Helmholtz equations, as well as for an inverse acoustics problem.

**Keywords:** Deep learning, adaptive collocation, inverse problems, artificial neural networks.

## 1 Introduction

Artificial neural networks (ANNs) have been a topic of great interest in the machine learning community due to their ability to solve very difficult problems, particularly in the fields of image processing and object recognition, speech recognition, medical diagnosis, etc. More recently, applications have been found in engineering, especially where large data sets are involved. From a mathematical point of view, neural networks are also interesting due to their ability to efficiently approximate arbitrary functions [Cybenko (1989)].

A natural question is to determine whether ANNs can be used to approximate the solution of partial differential equations which commonly appear in physics, engineering and mathematical problems. Several articles and even a book [Yadav (2015)] have been very recently devoted to this topic. In most of the approaches considered, a collocation-type method is employed which attempts to fit the governing equations and the boundary conditions at randomly selected points in the domain and on the boundary. Among these methods we mention the Deep Galerkin Method [Sirignano and Spiliopoulos (2018)], Physics Informed Neural Networks [Raissi, Perdikaris, and Karniadakis (2019)], as well as the earlier works in Lagaris et al. [Lagaris, Likas and Fotiadis (1998); Lagaris, Likas

[1] Institute of Structural Mechanics, Bauhaus-Universität Weimar, 99423, Weimar, Germany.

[2] School of Civil & Environmental Engineering, University of New South Wales, Sydney, Australia.

[3] Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia.

[*] Corresponding Author: Timon Rabczuk. Email: timon.rabczuk@uni-weimar.de.

and Papageorgiou (2000); van Milligen, Tribaldos and Jiménez (1995); Kumar and Yadav (2011); McFall and Mahan (2009)]. This method appears to produce reasonably accurate results, particularly for high-dimensional domains [Han, Jentzen and Weinan (2018)] and domains with complex geometries [Berg and Nyström (2018)], where the meshfree character of these methods makes them competitive with established discretization methods.

Another related approach is to use an energy minimization formulation of the governing equation as in Weinan et al. [Weinan and Yu (2018); Wang and Zhang (2019)]. This formulation has the advantage that only the first derivatives need to be computed for a 2nd order problem, however it requires a more precise integration procedure and not all governing equations can be cast in an energy-minimization framework.

In this work, we employ a collocation formulation for solving 2nd order boundary value problems such as Poisson's equation and Helmholtz equation. Different from existing methods which typically use a randomly scattered set of collocation points, we present an adaptive approach for selecting the collocation points based on the value of the residual at previous training steps. This method can improve the robustness of collocation method, particularly in cases when the solution has a non-smooth region where increasing the number of training points is beneficial.

The paper is structured as follows: in Section 2 we give an overview of artificial neural networks and briefly discuss their approximation properties. The application of ANNs to forward and inverse boundary-value problems is discussed in Section 3. Detailed numerical results are presented in Section 4, followed by concluding remarks.

## 2 Structure of neural network

In this section, we briefly describe the anatomy of a neural network and its properties for approximating arbitrary functions. We focus in particular on simple feed-forward neural networks which are used in the subsequent sections.

### *2.1 Feed-forward networks*

A feed-forward network can be seen as a computational graph consisting of an *input layer*, an *output layer* and an arbitrary number of intermediary *hidden layers*, where all the neurons (units) in adjacent layers are connected with each other. It can be used to represent a function $u: \mathbb{R}^n \rightarrow \mathbb{R}^m$ by using *n* neurons in input layer and *m* neurons in the output layer, see Fig. 1. We index the layers, starting with the input layer at 0, and the output layer as $L$, and we denote the number of neurons in each layer by $k_0 = n, k_1, \dots, k_L = m$. To each connection between the *i*-th neuron in layer $l-1$ and the *j*-th neuron in layer $l$, with $0 < l \leq L$, we associate a weight $w_{ji}^l$ and to each neuron in the layers $0 < l \leq L$ we associate a bias $b_i, i = 1, \dots, k_l$. Moreover, we define an activation function $\sigma^l: \mathbb{R} \rightarrow \mathbb{R}$ between the layers $l$ and $l-1$. Then the values at each neuron can be written in terms of the activation function applied to a linear combination of the neurons in the previous layer given by the corresponding weights and biases, i.e.,

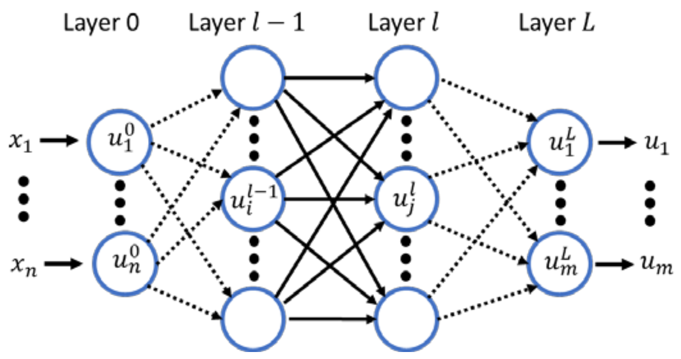


**Figure 1:** Schematic of a feed-forward neural network with $\boldsymbol{L-1}$ hidden layers

$$u_k^l = \sigma^l\left(\sum_{j=1}^{k_{l-1}} w_{kj}^l u_j^{l-1} + b_k^l\right) \tag{1}$$

This can be written more compactly in matrix form as:

$$\boldsymbol{u}^l = \sigma^l(\boldsymbol{W}^l\boldsymbol{u}^{l-1} + \boldsymbol{b}^l) \ \text{ for } \ l = 1,2,\dots,L, \tag{2}$$

where $\boldsymbol{W}^l$ is a matrix of weights corresponding to the connections between layers $l-1$ and $l$, $\boldsymbol{u}^l = [u_j^l]$ and $\boldsymbol{b}^l = [b_j^l]$ are column vectors and the activation function is applied element-wise.

Existing computational frameworks such as Tensorflow or Pytorch perform very efficient parallel execution, also on GPUs if available, of computational graphs like the ones defined by (2). Moreover, the input values can be defined as multi-dimensional arrays (tensors) and the computation of the corresponding outputs is efficiently vectorized and distributed across the available computational resources.

A key observation is that, for a given a neural network, the partial derivatives of the outputs with respect to the weights and biases can also be efficiently computed by the back-propagation algorithm. The idea is to perform the chain rule starting with the last layer and store the intermediary values in a computational graph where the order of the layers is reversed. The back-propagation algorithm enables the application of gradient-based minimization algorithms were a *loss function* based on the output of the neural network is to be minimized. Moreover, the partial derivatives of the outputs with respect to the inputs or with respect to some other prescribed parameters can be calculated in a similar way.

In typical applications of deep learning, a neural network is *trained* on a set of matching inputs and outputs by seeking to minimize the difference between the predicted values and some known correct outputs. This often requires large data sets that often must be manually processed and are themselves subject to different types of errors. We avoid this by defining a loss function which minimizes the residuals of the governing equations at a chosen set of training points. The training points can be simply generated as randomly scattered points in the domain as in Raissi et al. [Raissi, Perdikaris and Karniadakis (2019)]. In this work, we adopt instead an adaptive procedure which iteratively adds more points to the training set where the residual values are higher than some prescribed threshold, as detailed in Section 3.

Aside from the choice of the size of the neural networks (the number of hidden layers and the number of neurons in each layers) and that of the training points, other important parameters are related to the selection of the activation function and the choice of the minimization algorithm. Typical activation functions used are ramp functions like ReLU, sigmoid (logistic) function, and the hyperbolic tangent function (tanh). In this work, we use the tanh activation function which is preferable due to its smoothness. For optimization, we use the Adam (adaptive momentum) optimizer which based on stochastic gradient descent followed by a quasi-Newton method (L-BFGS) which builds an approximated Hessian at each gradient-descent step.

### *2.2 Theoretical approximation properties of neural networks*

It is well-known that artificial neural networks have very good approximation properties when the function to be approximated is continuous. This has been established since the 1980s, where it was shown (Hornik, Stinchcombe and White 1989) that under mild assumptions on the activation function, a given function $\boldsymbol{f}(\boldsymbol{x})$ can be approximated within any chosen tolerance by a network with single hidden layer and a finite number of neurons.

It was later shown, e.g., [Lu, Pu, Wang et. al. (2017)] that by choosing an non-linear activation function and forming deeper networks, the number of neurons can be significantly reduced. Various estimates of the approximation properties of neural networks for approximating PDEs have been more recently derived [Sirignano and Spiliopoulos (2018)].

We note however, that while the neural networks are in theory capable to represent complex functions in a very compact way, finding the actual parameters (weights and biases) which solve a given differential equation within a given approximation tolerance can be quite difficult. In the following, we present a method for selecting the training data which gives some control over the performance of the solver and optimization algorithms.

## 3 Collocation solver for PDEs

In the collocation method, the main idea is to define a loss function which is based on the strong form of the governing equations and the boundary conditions. The loss function is evaluated at chosen sets of points in the interior on the domain as well as on the boundary. More specifically, let us assume that a general boundary value problem can be written as:

$$\mathcal{L}\big(\mathbf{u}(\mathbf{x})\big) = f(\mathbf{x}) \text{ for } \mathbf{x} \in \Omega, \tag{3}$$

$$\mathcal{G}\big(\mathbf{u}(\mathbf{x})\big) = g(\mathbf{x}) \text{ for } \mathbf{x} \in \partial\Omega, \tag{4}$$

where $\Omega \in \mathbb{R}^d$ is the problem domain with boundary $\partial\Omega$, $\mathcal{L}$, $\mathcal{G}$ are interior and boundary differential operators, and $f,\ g$ are prescribed functions (e.g., loading data).

### *3.1 Poisson equation*

We consider equations of the form:

$$-\Delta u(\mathbf{x}) + ku(\mathbf{x}) = f(\mathbf{x}), \text{for } \mathbf{x} \in \Omega \tag{5}$$

$$u(\mathbf{x}) = \bar{u}(\mathbf{x}) \text{for } \boldsymbol{x} \in \partial\Omega_D, \tag{6}$$

$$\frac{\partial u(\mathbf{x})}{\partial \mathbf{n}} = g(\mathbf{x}), \text{for } \mathbf{x} \in \partial\Omega_{\mathrm{N}}, \tag{7}$$

where $u(\mathbf{x})$ is the unknown solution, $\Omega \subset \mathbb{R}^d$ is the computational domain, $k \geq 0$ is a given constant, $f$ is the given source term, $\bar{u}$ is the prescribed Dirichlet data on the Dirichlet boundary $\partial\Omega_D$, $\mathbf{n}$ is the outer normal vector, and $g$ is the Neumann data. When $k = 0$, the problem is a standard Poisson equation; we also consider a more general setting when e.g., $k = 1$.

We define a loss function, $\mathbb{C}(u; k, f, \bar{u}, g)$ using a mean squared error (MSE) for the interior and boundary governing equations as:

$$\begin{aligned}\mathbb{C}(u; k, f, \bar{u}, g) :=& \frac{1}{N_{int}} \sum_{i=1}^{N_{int}} [\Delta u(x_i^*, y_i^*) - ku(x_i^*, y_i^*) + f(x_i^*, y_i^*)]^2 \\ &+ \frac{\gamma_{dir}}{N_{dir}} \sum_{j=1}^{N_{dir}} \left[\bar{u}\left(x_j^{dir}, y_j^{dir}\right) - u\left(x_j^{dir}, y_j^{dir}\right)\right]^2 \\ &+ \frac{\gamma_{neu}}{N_{neu}} \sum_{k=1}^{N_{neu}} \left[g(x_k^{neu}, y_k^{neu}) - \frac{\partial u}{\partial \mathbf{n}}(x_k^{neu}, y_k^{neu})\right]^2,\end{aligned}$$

where $N_{int}$, $N_{dir}$ and $N_{neu}$ represent the number of points $(x_i^*, y_i^*)$, $\left(x_j^{dir}, y_j^{dir}\right)$, and $(x_k^{neu}, y_k^{neu})$ in the interior domain, on the Dirichlet boundary and the Neumann boundary respectively. Moreover, $\gamma_{dir}$ and $\gamma_{neu}$ represent penalty terms for the Dirichlet and Neumann boundaries. While in some cases it is enough to set $\gamma_{dir} = \gamma_{neu} = 1$, the convergence of the loss function can be improved by increasing one or both of these values to ensure that the boundary conditions are satisfied. During the minimization process, the terms in the cost function can be monitored and usually more accurate results can be obtained when the loss for the interior points is of the same order as the losses corresponding to the Dirichlet and Neumann boundaries.

### 3.2 Helmholtz equation

The governing equation for the homogeneous Helmholtz equation is of the form:

$$\Delta u(x, y) + k^2 u(x, y) = 0 \text{ for } (x, y) \in \Omega. \tag{8}$$

Here $u(x, y)$ can be complex-valued function. This equation is a time-independent form of the wave equation and it has applications in the study of various physical phenomena, such as acoustics, seismology and electromagnetic radiation. For many problems, the domain $\Omega$ is not bounded and the solution $u(x, y)$ can be highly oscillatory, which creates difficulties in standard finite element analysis.

Different types of boundary conditions, usually of the Neumann type can be imposed depending on the problem. As for Poisson's equation, we construct a loss function which seeks to minimize the residual of the governing equation at collocation points. While in theory neural networks are defined in $\mathbb{R}^n$, where $n$ is the number of neurons in the input layer and can capture unbounded domains, we limit ourselves here to finite domains.

### 3.3 Inverse problems

In this class of problems, one is given a particular solution $\mathbf{u}^*(\mathbf{x})$, which satisfies the governing equations of the form:

$$\mathcal{L}(\mathbf{u}^*(\mathbf{x});\boldsymbol{\lambda}^*) = f(\mathbf{x}) \text{ for } \mathbf{x} \in \Omega, \tag{9}$$

$$\mathcal{G}(\mathbf{u}^*(\mathbf{x});\boldsymbol{\lambda}^*) = g(\mathbf{x}) \text{ for } \mathbf{x} \in \partial\Omega, \tag{10}$$

where $\boldsymbol{\lambda}^*$ is unknown. The problem is then to determine the unknown parameter or vector of parameters $\boldsymbol{\lambda}^*$. This can be reformulated as an optimization problem, where we start with an initial guess $\boldsymbol{\lambda}$, and we approximate the solution $\mathbf{u}$ which satisfies the governing equations. In the framework of neural networks, we can use gradient descent to minimize $||\mathbf{u}^* - \mathbf{u}||$ under some suitable norm. Formally, we define a cost function of the form:

$$\mathbb{C}(\mathbf{u},\boldsymbol{\lambda}) := \frac{1}{N_{int}}\sum_{i=1}^{N_{int}}\left(\mathcal{L}(\mathbf{u}(\mathbf{x}_i^*);\boldsymbol{\lambda}) - f(\mathbf{x}_i^*)\right)^2 + \frac{\gamma_{bnd}}{N_{bnd}}\sum_{j=1}^{N_{bnd}}\left(\mathcal{G}\left(\mathbf{u}\left(\mathbf{x}_j^{bnd}\right);\boldsymbol{\lambda}\right) - g\left(\mathbf{x}_j^{bnd}\right)\right)^2 + \frac{1}{N_{int}}\sum_{i=1}^{N_{int}}\left(\mathbf{u}^*(\mathbf{x}_i^*) - \mathbf{u}(\mathbf{x}_i^*)\right)^2,$$

where $\mathbf{x}_i^*$ are interior collocation points, $\mathbf{x}_j^{bnd}$ are boundary collocation points, $\gamma_{bnd}$ is a penalty term for enforcing boundary conditions. This method seeks to find both $\mathbf{u}$ and $\boldsymbol{\lambda}$ simultaneously which approach $\mathbf{u}^*$ and $\boldsymbol{\lambda}^*$ respectively. In this work, we investigate applying this method to the Helmholtz equation where the wave number $k$ is unknown.

## *3.4 Adaptive collocation*

Several methods can be proposed to select the collocation points in the interior and the boundary of the domain. Here we propose a method where we start with a coarse grid and then select additional points based on the evaluation of the residual. We apply this method for the selection of the interior points, for which evaluating the governing equations requires the most computational effort, although in principle it can also be applied to the boundary points.

The main idea of the method is described in Fig. 2. The blue dots represent the training (collocation points) in the interior, the green dots represent the model evaluation points after the first training is complete, and the purple dots represent the additional training points. We note that evaluating the model at a larger number of points is quite inexpensive computationally, while the number of training points impacts the performance much more significantly as the governing equations need to be evaluated at the training points at each gradient descent step. Therefore, this method provides a criterion for selecting the collocation points in an efficient manner. Once the training is completed for one step, the network weights and biases can be carried over to the subsequent step, resulting in faster convergence.

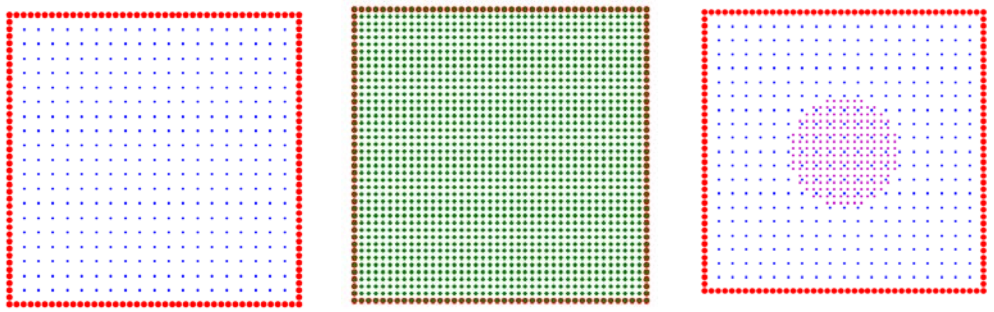


**Figure 2:** The steps of the adaptive collocation method, assuming the residual values are higher in the center of the domain

## 4 Numerical results

### *4.1 Poisson equation on the unit square*

We first consider a Poisson equation with Dirichlet and Neumann boundary conditions:

$$-\Delta u(x,y) = 8\sin(2\pi x)\cos(2\pi y) \text{ for } (x,y) \in (0,1)^2,$$
$$u(x,y) = 0 \text{ for } x = 0,$$
$$\frac{du}{dy} = 0 \text{ for } y = 0 \text{ and } y = 1, \text{and}$$
$$\frac{du}{dx} = 2\pi\cos(2\pi x)\cos(2\pi y) \text{ for } x = 1.$$

The exact solution of this equation is $u(x,y) = \sin(2\pi x)\cos(2\pi y)$. We consider a loss function of the form:

$$\mathbb{C}(u) := \frac{1}{N_{int}}\sum_{i=1}^{N_{int}}[\Delta u(x_i^*, y_i^*) + 2\pi^2\sin(\pi x_i)\cos(\pi y_i)]^2 + \frac{1}{N_{left}}\sum_{j=1}^{N_{left}} u\left(x_j^{left}, y_j^{left}\right)^2 +$$
$$+\frac{1}{N_{bottom}}\sum_{j=1}^{N_{bottom}}\left[\frac{\partial u}{\partial y}\left(x_j^{bottom}, y_j^{bottom}\right)\right]^2 + \frac{1}{N_{top}}\sum_{j=1}^{N_{top}}\left[\frac{\partial u}{\partial y}\left(x_j^{top}, y_j^{top}\right)\right]^2 +$$
$$+\frac{1}{N_{right}}\sum_{j=1}^{N_{right}}\left[\frac{\partial u}{\partial x}\left(x_j^{right}, y_j^{right}\right) - \pi\cos\left(\pi x_j^{right}\right)\cos\left(\pi y_j^{right}\right)\right]^2,$$

where $(x_i^*.y_i^*)$ are interior collocation points, and $(x_j^{left}, y_j^{left})$, $(x_j^{bottom}, y_j^{bottom})$, $(x_j^{top}, y_j^{top})$, and $(x_j^{right}, y_j^{right})$ are boundary collocation points. Moreover, $N_{int}, N_{left}$, $N_{bottom}$, $N_{top}$ and $N_{right}$ are the number of collocation points in the interior and the four boundaries. For this example, in the initial training stage we have used $N_{int} = 19^2$ equally spaced points in the interior of the domain and $N_{left} = N_{top} = N_{bottom} = N_{right} = 41$ uniformly distributed points on each of the four edges. The test (evaluation set) consists of a grid of $29^2$ equally-spaced points. The training (collocation) points at subsequent iterations were chosen by selecting the top 30% of the points with the highest.

The results obtained for a shallow network with one hidden layer of 10 neurons are shown in Figure 3. The blue dots represent the interior collocation points, while the red and green dots represent the points corresponding to the Dirichlet and Neumann boundary conditions respectively. We note that even for this simple network with 41 parameters (30 weights and 11 biases), an accurate solution can be obtained. Because the solution is smooth throughout the domain, the training points are generally evenly distributed. However, more points are selected near the corners and boundaries since the residuals and the actual errors are higher there.

The relative $L^2$ errors obtained by increasing the number of layers while keeping the number of neurons per layer fixed and using the same refinement strategy are shown in Tab. 1. It can be observed that except for the single-layer network, the error decreases significantly as more training points are used. Moreover, the error for deeper networks is greatly reduced compared to the single-layer network, although the number of parameters and computational cost increases as well.

**Table 1:** Relative $\boldsymbol{L^2}$ errors for different levels of refinement and different numbers of layers for the Poisson equation on the unit square

| | Number of training points | Relative $L^2$ error for 1 layer | Relative $L^2$ error for 2 layers | Relative $L^2$ error for 3 layers |
|---|---|---|---|---|
| Refinement 1 | 361 | 0.07475152 | 0.00486584 | 0.00116222 |
| Refinement 2 | 816 | 0.03761188 | 0.00119753 | 0.00043185 |
| Refinement 3 | 1271 | 0.08465629 | 0.00026268 | 0.00026697 |

### *4.2 Source problem on a quarter-annulus domain*

We now consider a 2nd order problem with pure Dirichlet boundary conditions on a non-rectangular domain. The governing equation is given by:

$$-\Delta u(x,y) + u(x,y) = f(x,y), \text{for } (x,y) \in \Omega$$

$$u(x,y) = 0, \text{for } (x,y) \in \partial\Omega,$$

where Ω is quarter of an annulus located in the first quadrant and centered at the origin, with inner radius $r_{int} = 1$ and outer radius $r_{ext} = 4$. Here $f(x,y)$ is chosen such that the exact solution is $u(x,y) = (x^2 + y^2 - 1)(x^2 + y^2 - 16)\sin(x)\sin(y)$, i.e.,

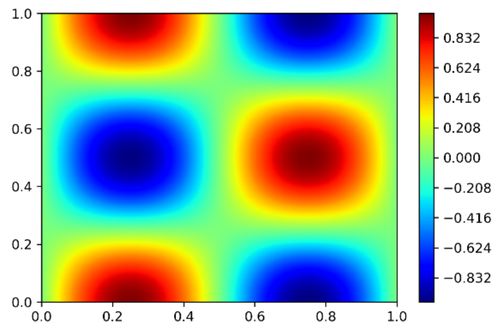
$$f(x,y) = (3x^4 - 67x^2 - 67y^2 + 3y^4 + 6x^2y^2 + 116)\sin(x)\sin(y) +$$

$$(68x - 8x^3 - 8xy^2)\cos(x)\sin(y) + (68y - 8y^3 - 8yx^2)\cos(x)\sin(y).$$

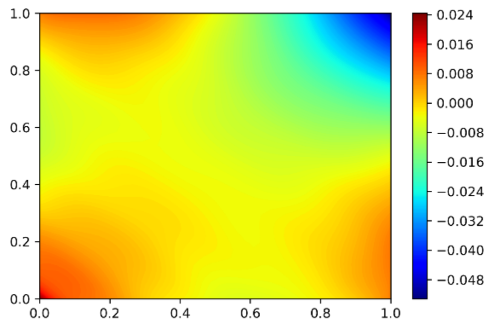
As before, we define a cost function consisting of term corresponding to the interior governing equation and another term corresponding to the boundary conditions. To ensure that the boundary conditions are satisfied during the minimization process, we apply a penalty factor $\gamma = 100$, so that the cost function becomes:

$$\mathbb{C}(u) \coloneqq \frac{1}{N_{int}}\sum_{i=1}^{N_{int}}[\Delta u(x_i^*, y_i^*) - u(x_i^*, y_i^*) + f(x_i^*, y_i^*)]^2 + \frac{\gamma}{N_{bnd}}\sum_{j=1}^{N_{bnd}} u\left(x_j^{bnd}, y_j^{bnd}\right)^2$$

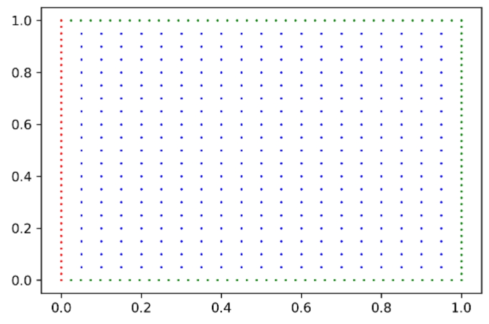
We first choose a neural network with 2 hidden layers with 10 neurons each and the tanh activation function. The initial set of collocation points consists of $N_{int} = 19^2$ points in the interior and $N_{bnd} = 168$ points on the boundary, spaced uniformly as shown in Fig. 4. Subsequent refinements are done according to the same procedure as in the first example. The relative $L^2$ error is calculated as 0.00053738 for the initial training and decreases to 0.00036766 and 0.00043207 as the number of collocation points is increased.



a) Computed solution $u_{comp}$



b) Error $u_{ex} - u_{comp}$


c) Initial training set

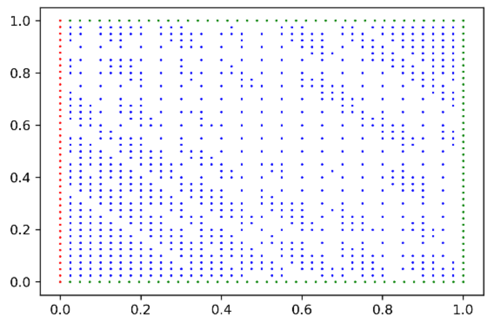

d) Final training set

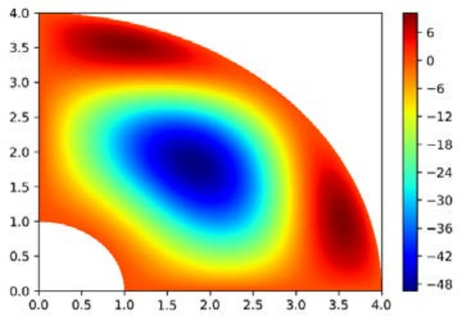
**Figure 3:** Computed solution with error and the training sets of collocation points using a network with one hidden layer and 10 neurons for the Poisson equation on a unit square

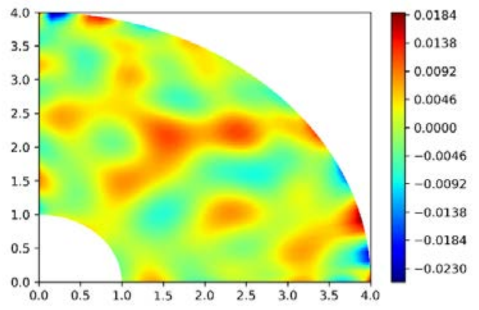
### *4.3 Poisson equation with a corner singularity*

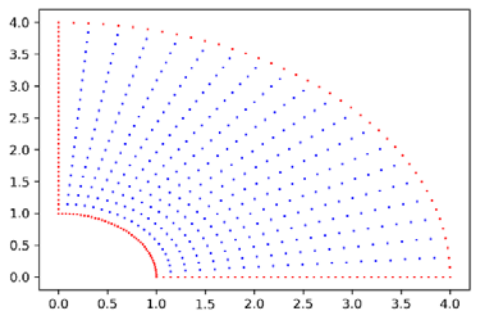
To investigate the ability of the proposed refinement scheme to approximate solutions with sharp gradients, we next consider the Poisson equation on a domain with an internal boundary which results in a corner-type singularity appearing in the solution. The domain considered is given by $\Omega \coloneqq (-1,1)^2 - [0,1)$ in Cartesian coordinates and we seek an unknown solution which satisfies:

$$-\Delta u(x, y) = 0, \text{for } (x, y) \in \Omega$$

$$u(r, \theta) = r^{1/2}\sin(\theta/2) \text{ for } (r\,, \theta) = \left(\sqrt{x^2 + y^2}, \arctan(y/x)\right) \in \partial\Omega.$$


a) Computed solution $u_{comp}$


b) Error $u_{ex} - u_{comp}$


c) Initial training set

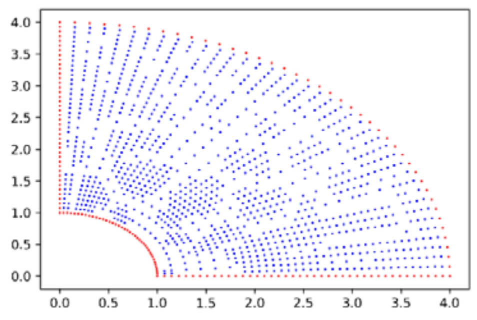

d) Final training set

**Figure 4:** Computed solution with error and the training sets of collocation points using a network with two hidden layer and 10 neurons each for the source equation on a quarter-annulus domain

The exact solution in polar coordinates is $u_{ex}(r,\theta) = r^{1/2}\sin(\theta/2)$, which has the singular term $r^{1/2}$ creating approximation difficulties near the origin. In finite elements methods, a more refined mesh is typically required to obtain a good approximation. This problem was also investigated in Weinan et al. [Weinan and Yu (2018)] using an energy minimization method.

The geometry is modelled by considering 3 rectangular subdomains $(-1,0) \times (-1,1)$, $(0,1) \times (-1,0)$, and $(0,1) \times (0,1)$. We define a loss function of the form:

$$\mathbb{C}(u) := \frac{1}{N_{int}} \sum_{i=1}^{N_{int}} [\Delta u(x_i^*, y_i^*)]^2 + \frac{\gamma}{N_{bnd}} \sum_{j=1}^{N_{bnd}} \left[u\left(r_j^{bnd}, \theta_j^{bnd}\right) - u_{ex}\left(r_j^{bnd}, \theta_j^{bnd}\right)\right]^2.$$

In the initial grid we choose equally spaced points with a distance of 0.05 in the *x* and *y* directions. For the points on the boundary, we choose more densely spaced points, with a distance of 0.025 in Cartesian coordinates and we set a penalty factor of $\gamma = 500$ to ensure that the boundary conditions are respected. As before, we evaluate the model on grids with more points and append the points where the residual value is large to the training set in the next step.

The results obtained by the adaptive collocation scheme using a network with 3 hidden layers and 30 neurons each are shown in Fig. 5. In general, the residual values in a narrow region around the singularity are much larger than in the rest of the domain and they are selected in the subsequent training step. Also, larger residuals are observed along the line $y = 0$, with $x < 0$ as the neural network with a coarser training grid has difficulties in capturing correctly the end of the internal boundary. However, as can be seen from the plots, the error diminishes as the number of training points increases. The accuracy can be further improved by choosing larger networks although the number of training points needs to be increased as well.

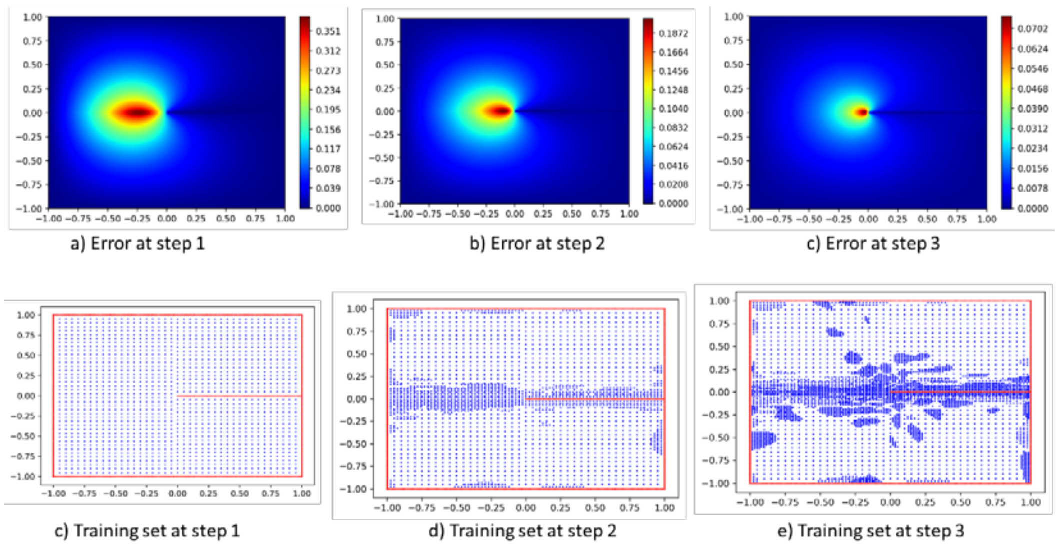


**Figure 5:** Error between the exact solution and computed solution for the Poisson equation with a singularity at origin and the training sets at each refinement step for a network with 4 hidden layers and 30 neurons per layer

### *4.4 Acoustic duct problem*

To investigate the applicability of neural networks to approximate the oscillatory solutions such as those obtained by solving the Helmholtz equation in acoustics. The benchmark problem under consideration has complex-valued governing equations of the form:

$$\Delta u(x,y) + k^2 u(x,y) = 0 \text{ for } (x,y) \in \Omega := (0,2) \times (0,1)$$

$$\frac{\partial u}{\partial n} = \cos(m\pi x)\,, \text{for } x = 0; \frac{\partial u}{\partial n} = -iku, \text{for } \ x = 2$$

$$\frac{\partial u}{\partial n} = 0, \text{for } y = 0 \text{ and } y = 1$$

Here we select $k = 12$ as the wave number and $m = 2$ as the mode number. This problem admits an analytical solution which can be written as:

$$u^{ex}(x,y) = \cos(m\pi y)(A_1 \exp(-ik_x x) + A_2 \exp(ik_x x)),$$

where $k_x = \sqrt{k^2 - (\pi m)^2}$ , and $A_1$ and $A_2$ are the solution of the $2 \times 2$ linear system:

$$\begin{bmatrix} ik_x & -ik_x \\ (k-k_x)\exp(-2ik_x) & (k+k_x)\exp(2ik_x) \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

In the following, we compute only the real part of the solution $u(x,y)$ as the imaginary part can be computed by a similar procedure.

As before, we define a loss functions which minimizes the residual of the governing equation at interior and boundary points:

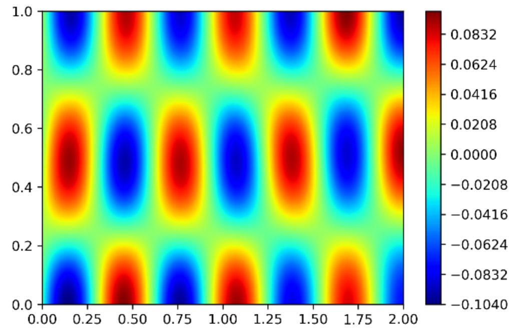
$$\mathbb{C}(u) := \frac{1}{N_{int}} \sum_{i=1}^{N_{int}} [\Delta u(x_i^*, y_i^*) + k^2 u(x_i^*, y_i^*)]^2 + \frac{\gamma}{N_{bnd}} \sum_{j=1}^{N_{bnd}} \left( \frac{\partial u^{ex}}{\partial n}\left(x_j^{bnd}, y_j^{bnd}\right) - \frac{\partial u}{\partial n}\left(x_j^{bnd}, y_j^{bnd}\right) \right)^2 .$$

The results of the adaptive collocation method are shown in Fig. 6. We have used a neural network with 3 hidden layers of 30 neurons each and a grid of $99 \times 49$ uniformly spaced points in the interior of the domain in the initial step. For the boundary, we have used $N_{bnd} = 400$ uniformly spaced points and a penalty parameter of $\gamma = 100$. As before, the size of the training set is increased based on the residual value on a finer grid (with double the points in each direction) in subsequent steps. Due to the oscillatory nature of the solution, the additional training points are also generally evenly distributed in the domain with higher concentration in the areas where the residual value was initially larger than average.

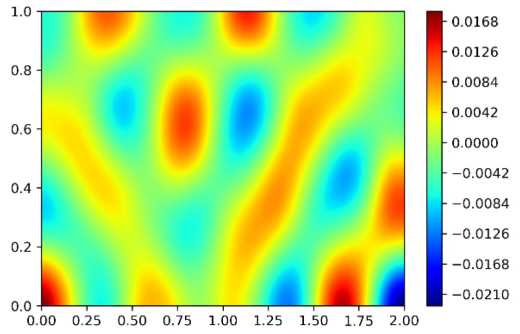
### *4.5 Inverse acoustic problem*

Here we consider the same governing equation and boundary conditions as in the previous example, but we seek instead to solve for the wave number $k$ while the value of the solution (for the correct $k$) is given. In this case we build a loss function that minimizes the difference between the given solution and the residual of the governing equations simultaneously:

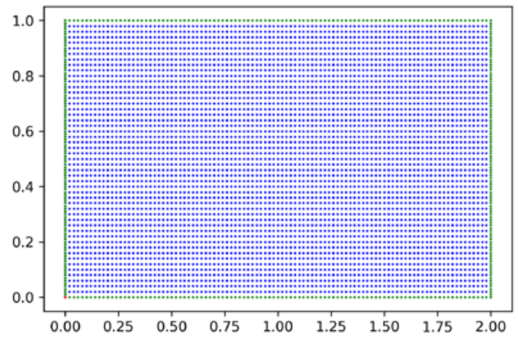
$$\mathbb{C}(u,k) := \frac{1}{N_{int}} \sum_{i=1}^{N_{int}} [\Delta u(x_i^*, y_i^*) + k^2 u(x_i^*, y_i^*)]^2 + \frac{1}{N_{int}} \sum_{i=1}^{N_{int}} [u^{ex}(x_i^*, y_i^*) - u(x_i^*, y_i^*)]^2 + \frac{\gamma}{N_{bnd}} \sum_{j=1}^{N_{bnd}} \left( \frac{\partial u^{ex}}{\partial n}\left(x_j^{bnd}, y_j^{bnd}\right) - \frac{\partial u}{\partial n}\left(x_j^{bnd}, y_j^{bnd}\right) \right)^2 .$$



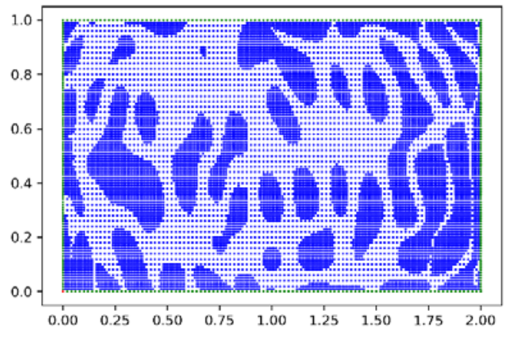
a) Computed solution $u_{comp}$



b) Error $u_{ex} - u_{comp}$



c) Initial training set

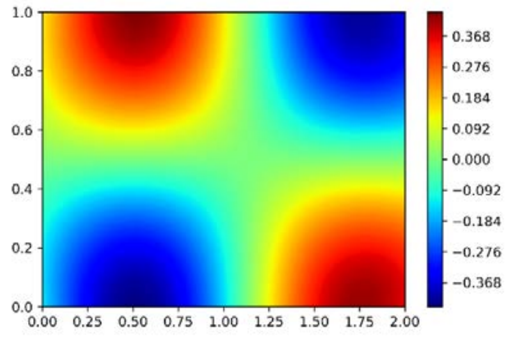


d) Final training set

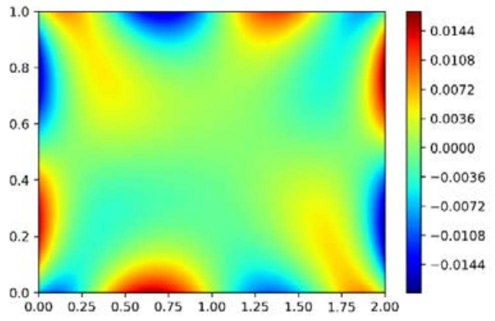
**Figure 6:** Computed solution, the error and the sets of training points for the acoustic duct benchmark problem with $\boldsymbol{k = 12}$ and $\boldsymbol{m = 2}$

Here $u_{ex}$ has the same form as in the previous section but with $k = 4$ and $m = 1$. We start with $k = 1$ as an initial guess and seek to minimize the loss function with $k$ as a free parameter. For this problem, we choose a grid of $149 \times 29$ equally spaced points in the interior of the domain and $N_{bnd} = 800$ boundary collocation points and $\gamma = 100$.

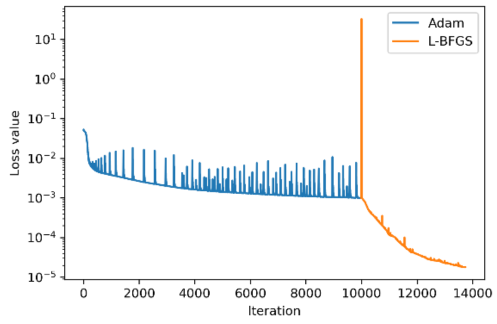
The results for this example are presented in Fig. 7. We can observe that the solution has been represented with reasonable accuracy both in terms of $u(x, y)$ as well $k$. The relative $L^2$ error for $u(x, y)$ in this example is 0.084, while the computed $k$ is 3.882 as compared to 4 in the reference solution. As in the other examples, we have used the Adam optimizer followed by a quasi-Newton method (L-BFGS). It can be noted that the latter converges significantly faster, however in many cases performing a stochastic gradient-descent like Adam helps the solver to avoid being trapped in early local minima.



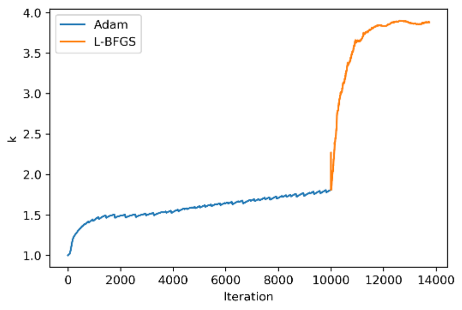
a) Computed solution $u_{comp}$



b) Error $u_{ex} - u_{comp}$



c) Initial training set



d) Final training set

**Figure 7:** Computed solution, the error and the convergence of the loss function and wave number $\boldsymbol{k}$ where the reference solution has $\boldsymbol{k} = \mathbf{4}$ for the inverse acoustic problem

## 5 Conclusions

We have presented a collocation method for solving boundary values problem using artificial neural networks. The method is completely mesh-free as only scattered sets of points are used in the training and evaluation sets. Although uniform grids of training points have been used in the initial training step, the method could be easily adapted to scattered data obtained e.g. by Latin hypercube sampling methods. The method was shown to produce results with good accuracy for the parameters chosen, although as common in deep learning methods, parameter selection may require some manual tuning.

A more detailed study of the convergence and approximation properties of neural networks, as well as selecting robust minimization procedures remain open as possible research topics. Moreover, the applicability of these methods to energy minimization formulations, for the differential equations which allow it, can be investigated in future work.

**Acknowledgements:** N. Alajlan and T. Rabczuk acknowledge the Distinguished Scientist Fellowship Program (DSFP) at King Saud University for supporting this work.

## References

**Berg, J.; Nyström K.** (2018): A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, vol. 317, pp. 28-41.

**Cybenko, G.** (1989): Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303-314.

**Weinan, E.; Yu, B.** (2018): The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, vol. 6, no. 1, pp. 1-12.

**Han, J.; Jentzen, A.; Weinan, E.** (2018): Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, vol. 115, no. 34, pp. 8505-8510.

**Hornik, K.; Stinchcombe, M.; White, H.** (1989): Multilayer feedforward networks are universal approximators. *Neural Networks*, vol. 2, no. 5, pp. 359-366.

**Kumar, M.; Yadav, N.** (2011) Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: a survey. *Computers & Mathematics with Applications*, vol. 62, no. 10, pp. 3796-3811.

**Lagaris, I. E.; Likas, A. C.; Papageorgiou, D. G.** (2000): Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1041-1049.

**Lagaris, I. E.; Likas, A.; Fotiadis, D. I.** (1998): Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987-1000.

**Lu, Z.; Pu, H.; Wang, F.; Hu, Z.; Wang, L.** (2017): The expressive power of neural networks: a view from the width. *Advances in Neural Information Processing Systems*, vol. 30, pp. 6231-6239.

**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*, vol. 20, no. 8, pp. 1221-1233.

**Raissi, M.; Perdikaris, P.; Karniadakis, G. E.** (2019): Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, vol. 378, pp. 686-707.

**Sirignano, J.; Spiliopoulos, K.** (2018): DGM: a deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, vol. 375, pp. 1339-1364.

**van Milligen, B. Ph.; Tribaldos, V.; Jiménez, J. A.** (1995): Neural network differential equation and plasma equilibrium solver. *Physical Review Letters*, vol. 75, no. 20, pp. 3594-3597.

**Wang, Z.; Zhang, Z.** (2019): A mesh-free method for interface problems using the deep learning approach. arXiv:1901.00618.

**Yadav, N.** (2015). *An Introduction to Neural Network Methods for Differential Equations*. Springer, Netherlands.