COBEM 2021 26th International Congress of Mechanical Engineering
ABCM Associação Brasileira de Engenharia e Ciências Mecânicas
November 22-26, 2021, Virtual Congress, Brazil

COB-2021-0094
ANN-based Mesh-free Method to Solve Partial Differential Equations

Filipi Teixeira Kunz
Ney Rafael Sêcco
Instituto Tecnológico de Aeronáutica, São José dos Campos - São Paulo, Brazil
filipi.kunz@ga.ita.br, ney@ita.br

Abstract. *There are well established methods to numerically solve partial differential equations (PDEs), such as Finite Difference, Finite Elements, and Finite Volumes approaches, which usually require the discretization of domain in meshes. However, the generation of high-quality meshes for complex domains is a time-consuming task that demands skilled specialists. In this manuscript we present a mesh-free approach to solve PDEs using feedforward artificial neural networks (ANNs). This methodology involves a training procedure that adjusts ANNs outputs and their derivatives to match PDEs and their associated boundary conditions at a given set of points. We test the procedure by solving canonical PDEs problems of linear advection, steady and unsteady heat transfer, Burgers' equation, and potential flow. We modify training and network parameters such as optimizer and number of neurons to analyze the outcomes in solution speed and accuracy. This method shows versatility, as the same numerical solver works with hyperbolic, elliptical, and parabolic PDEs. Even though the application of ANN-based solution is computationally more expensive than traditional mesh-based approaches, substantial time can be saved in terms of mesh generation.*

Keywords: *Artificial Neural Network, Partial Differential Equation, Mesh-Free Methods.*

1. INTRODUCTION

Partial Differential Equations (PDEs) are the basis of several models employed in engineering, what arouses the interest of several authors to develop effective applications to solve these problems. Analytical solutions to PDEs are seldomly available, especially in complex-shaped domains. Therefore, engineering applications rely on numerical methods to solve these equations.

There are traditional numerical methods for solving PDEs, such as finite elements, finite volumes, and finite differences. These methods require the discretization of the domain in meshes. However, the mesh generation process demands time and experience from the user (Katz, 2009), even when automatic mesh generation techniques are employed. In addition, more difficulties arise when the domain varies during the simulation, such as aerodynamic shape optimization (Secco and Martins, 2019) and fluid-structure interaction (Keye and Rudnik, 2015), since the mesh should be deformed according to changes in its boundaries while preventing the generation of negative-volume cells.

As an alternative, new methods propose the use of Artificial Neural Networks (ANNs) to solve PDEs without meshes to avoid issues found in classical numerical methods. The solution of PDEs with ANNs is addressed in the literature in different forms. Chronologically, the work of Dissanayake and Phan-Thien (1994) is highlighted, whose property of its study is the minimization of the classical Lagrangian, addressing separately the error of the PDE and the boundary condition (BC) for different network sizes. Subsequently, Lagaris *et al.* (1998) solve PDEs of different orders, satisfying the BCs by construction. With similar treatment at the BC proposed by Lagaris *et al.* (1998), Malek and Shekari Beidokhti (2006) make use of the same concept for solving ordinary differential equations (ODEs). Lastly, Shirvany *et al.* (2009) apply a neural network update procedure into their study to automatically increase the number of neurons at the hidden layer to reduce the PDE residual.

In this work we present a numerical procedure to train a feedforward ANN with a specific number of neurons to estimate a PDE solution. This method does not demand a mesh discretizing of the domain. A cloud of points is used in the training procedure instead, and connectivity information among these points is not necessary. The analytical definition of an ANN allows the computation of derivatives at each training point in the domain. Therefore, it is possible to adjust the ANN parameters until its derivatives match the PDE statement. The optimization problem associated with the ANN training in this work considers the minimization of the PDE residuals as objective function, while BCs are treated as constraints. An enhanced version of the backpropagation algorithm is also used, allowing the computation of the sensitivities of the ANN derivatives with respect to its weights and biases for gradient-based optimization. Finally, the methodology is applied to canonical test cases with analytical solutions to verify the performance of the method.

2. METHODOLOGY

Aarts and der Veer (2001) state that a feedforward ANN (Figure 1) with one hidden layer and a linear output layer can approximate an arbitrary continuous function and its derivatives to any desired accuracy. This suggests than an ANN may also be trained to approximate a function that satisfies a PDE and its BCs. Furthermore, Berg and Nyström (2018) show that an ANN with additional hidden layers can reduce the training time and achieve better accuracy at the function approximation.

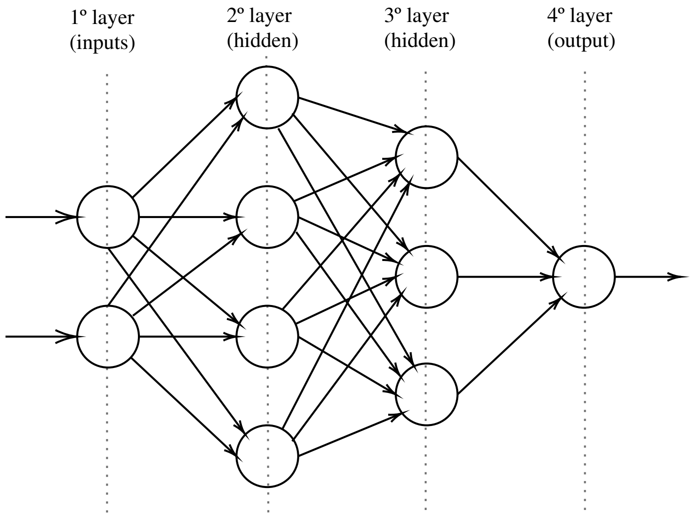


Figure 1: Feedforward neural network.

The ANN training is represented by an optimization problem where ANN weights ($\boldsymbol{\theta}$) are adjusted to minimize an objective function and satisfy a set of constraints. Therefore, we must carefully select objective and constraints that tailor the ANN to the desired PDE and BCs. Previous works mainly use two approaches to enforce BCs at the ANN training. The first approach explores the use of a trial function, which satisfies the BCs by construction (Lagaris *et al.*, 1998; Malek and Shekari Beidokhti, 2006; Baymani *et al.*, 2010; Berg and Nyström, 2018), and then adds the ANN contribution to model the solution inside the domain. The second one, applies penalty factors to deal with the BCs directly at the objective function (Lagaris *et al.*, 2000; Anitescu *et al.*, 2019), leading to an unconstrained optimization problem.

Here we cast the ANN training as a constrained optimization problem where the objective ($f(\boldsymbol{\theta})$) is the minimization of an averaged error of PDE residuals over a cloud of n_t points distributed at the domain, while equality constraints ($c_i(\boldsymbol{\theta}) = 0$) are associated to satisfaction of the BCs at a given number of boundary points. The Augmented Lagrangian Method (ALM) converts the constrained optimization problem in a sequence of unconstrained optimizations (Nocedal and Wright, 2006). The Augmented Lagrangian $\mathcal{L}_A$ to be minimized in each unconstrained optimization is defined as:

$$\mathcal{L}_A(\boldsymbol{\theta}, \boldsymbol{\lambda}, \mu) = f(\boldsymbol{\theta}) - \sum_{i=1}^{n_c} \lambda_i c_i + \frac{\mu}{2} \sum_{i=1}^{n_c} c_i^2 \tag{1}$$

where:

- $f(\boldsymbol{\theta})$ is the non-augmented objective function, which is an average of PDE residuals at the training points with an additional regularization factor, γ;
- c_i are equality constrains associated to the conditions at the i-th boundary point;
- λ_i and μ are the Lagrange multipliers and penalty parameter, respectively, which are iteratively updated until constraints are satisfied;
- $\boldsymbol{\theta}$ denotes the design variables (ANN weights and biases), which are adjusted during the optimization process.

The non-augmented objective function is defined as:

$$f(\boldsymbol{\theta}) = MSE + \gamma \cdot \frac{1}{2n_\theta} \boldsymbol{\theta}^T \cdot \boldsymbol{\theta} \tag{2}$$

where MSE represents the mean squared error of the PDE residuals at the n_t training points inside the domain (R_i):

$$MSE = \frac{1}{2n_t} \sum_{i=1}^{n_t} R_i^2 \tag{3}$$

The second term of Eq. (2) is a regularization term to prevent overfitting and numerical issues associated with high ANN weights. The γ parameter is selected by the user to control the regularization effect and n_θ is the total number of weights and biases of the ANN.

2.1 ANN training example: 1-D Linear advection

We will use the 1-D linear advection as an example to outline the ANN training procedure. This equation has the following definition and BCs:

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0 \qquad \text{with } u(x,0) = \alpha(x) \text{ and } u(-1,t) = \beta(t) \tag{4}$$

defined over the domain $t \in [0,1]$ and $x \in [-1,1]$. We initially generate an ANN to represent a function $u'(x,t,\boldsymbol{\theta})$ defined over the same $x - t$ domain. Evaluating the differential problem stated in Eq. (4) using the ANN at n_t training points (x_k, t_k) distributed over the domain, we can get the PDE residual R_k at the k-th training point:

$$\frac{\partial u'_k}{\partial t} + \frac{\partial u'_k}{\partial x} = R_k \;\; , for \;\; k = 1, ..., n_t \tag{5}$$

For the BC constraints, we distribute n_{low} points along the $t = 0$ boundary and n_{left} points along the $x = -1$ boundary. Then, it is possible to obtain boundary constraints c_i for the i-th boundary point as:

$$c_i = \begin{cases} u'(x_i, 0, \boldsymbol{\theta}) - \alpha(x_i) & \text{for } i = 1, ..., n_{\text{low}} \\ \\ u'(-1, t_i, \boldsymbol{\theta}) - \beta(t_i) & \text{for } i = n_{\text{low}} + 1, ..., n_{\text{low}} + n_{\text{left}} \end{cases} \tag{6}$$

Initial tests conducted by the authors indicate that the boundary points used in the boundary constraints (Eq. (6)) should also be included in the averaged PDE residual evaluation (Eqs. (3) and (5)) for better ANN generalization performance. Applying 2, 3, 5, and 6 into 1 gives the Augmented Lagrangian for this problem:

$$\mathcal{L}_A(\boldsymbol{\theta}, \boldsymbol{\lambda}, \mu) = \frac{1}{2n_t} \sum_{k=1}^{n_t} \left(\frac{\partial u'_k}{\partial t} + \frac{\partial u'_k}{\partial x} \right)^2 + \gamma \cdot \frac{1}{2n_\theta} \boldsymbol{\theta}^T \cdot \boldsymbol{\theta} - \sum_{i=1}^{n_{\text{low}}+n_{\text{left}}} \lambda_i c_i + \frac{\mu}{2} \sum_{i=1}^{n_{\text{low}}+n_{\text{left}}} c_i^2 \tag{7}$$

The minimization of the Augmented Lagrangian requires the use of gradient-based optimizers due to the large number of ANN parameters to be adjusted. We need to get the $\mathcal{L}_A$ partial derivatives with respect to each weight and bias θ_j of the ANN for gradient-based minimization:

$$\begin{aligned} \tfrac{\partial}{\partial \theta_j} \mathcal{L}_A(\boldsymbol{\theta}, \boldsymbol{\lambda}, \mu) = \;& \frac{1}{n_t} \sum_{k=1}^{n_t} \left[\left(\frac{\partial u'_k}{\partial t} + \frac{\partial u'_k}{\partial x} \right) \cdot \left(\frac{\partial}{\partial \theta_j} \left(\frac{\partial u'_k}{\partial t} \right) + \frac{\partial}{\partial \theta_j} \left(\frac{\partial u'_k}{\partial x} \right) \right) \right] + \frac{\gamma}{n_\theta} \boldsymbol{\theta} - \\ & - \sum_{i=1}^{n_{\text{low}}+n_{\text{left}}} \lambda_i \frac{\partial c_i}{\partial \theta_j} + \mu \sum_{i=1}^{n_{\text{low}}+n_{\text{left}}} c_i \frac{\partial c_i}{\partial \theta_j} \end{aligned} \tag{8}$$

where $\frac{\partial u'_k}{\partial t}$, $\frac{\partial u'_k}{\partial x}$ and its derivatives with respect to weights and biases are obtained with an enhanced version of ANN feedforward and backpropagation algorithms developed in this work.

The same approach can be applied to different PDE and BC definitions by modifying the Augmented Lagrangian and its partial derivatives.

2.2 Test Cases: Canonical PDE Problems

After introducing the ANN training procedure for the linear advection problem, we proceed with the presentation of the additional canonical problems. It also includes the linear advection stated before and 4 other equations. Table 1 shows the definitions of each canonical problem. Furthermore, Table 2 specifies the ANN settings and domain points distribution. Both random an structured distribution of points are tested. In this table, the number of PDE evaluation points includes the interior points and boundary points, as indicated in Section 2.1The selected problems cover the major aspects of PDEs, evaluating mainly the effect of propagation orientation and the PDE type. These problems also allow comparisons with analytical solutions.

Table 1: Definitions of the canonical test cases.

	Differential Equation	Evaluated Domain	Boundary Conditions
Hyperbolic PDE: 1-D Linear Advection	$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0$	$x \in [-1, 1]$ $t \in [0, 1]$	$u(x,0) = \begin{cases} 0 & \text{, for } -1 \leq x < -0.75 \\ 1 & \text{, for } -0.75 \leq x \leq -0.25 \\ 0 & \text{, for } -0.25 < x \leq 1 \end{cases}$ $u(-1,t) = 0$, for $0 \leq t \leq 1$
Hyperbolic PDE: 1-D Periodic Linear Advection	$\frac{\partial u}{\partial t} + 0.7\frac{\partial u}{\partial x} = 0$	$x \in [-1, 1]$ $t \in [0, 1]$	$u(x,0) = \sin(2\pi x)$, for $-1 \leq x \leq 1$
Elliptic PDE: 2-D Steady Heat Transfer	$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$	$x \in [0, 1]$ $y \in [0, 1]$	$u(x,0) = 1$, for $0 < x \leq 1$ $u(x,1) = 0$, for $0 \leq x \leq 1$ $u(0,y) = 0$, for $0 \leq y < 1$ $u(1,y) = 0$, for $0 \leq y < 1$
Parabolic PDE: 1-D Unsteady Heat Transfer	$\frac{\partial u}{\partial t} - 0.1\frac{\partial^2 u}{\partial x^2} = 0$	$x \in [0, 1]$ $t \in [0, 1]$	$u(x,0) = 0$, for $0 \leq x \leq 1$ $u(0,t) = 1$, for $0 < t \leq 1$ $u(1,t) = 0$, for $0 < t \leq 1$
Nonlinear PDE: 1-D Inviscid Burgers' equation	$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0$	$x \in [0, 1]$ $t \in [0, 1]$	$u(x,0) = \begin{cases} 0.1 & \text{, for } 0 \leq x < 0.3 \\ x - 0.2 & \text{, for } 0.3 \leq x < 0.4 \\ 0.2 & \text{, for } 0.4 \leq x \leq 0.6 \\ 0.8 - x & \text{, for } 0.6 < x \leq 0.7 \\ 0.1 & \text{, for } 0.7 < x \leq 1 \end{cases}$ $u(0,t) = 0.1$, for $0 \leq t \leq 1$

Table 2: Domain characteristics.

	Domain points distribution	Nº of BC evaluation points	Nº of PDE evaluation points	Nº of neurons on each layer	Nº weights to be adjusted
Hyperbolic PDE: 1-D Linear Advection	Random	61	145	[3, 2]	20
Hyperbolic PDE: 1-D Periodic Linear Advection	Random	31	145	[5, 5]	51
Elliptic PDE: 2-D Steady Heat Transfer	Regular	76	140	[8, 8]	105
Parabolic PDE: 1-D Unsteady Heat Transfer	Random	91	481	[5, 5]	51
Nonlinear PDE: 1-D Inviscid Burgers' equation	Random	81	1001	[10, 8]	127

2.3 Aerospace-related Application: Potential Flow Around a Cylinder

In this section, we explore the capabilities of the ANN to solve fluid mechanics problems like the potential flow around a cylinder. Let u be the velocity potential that should satisfy the Laplace's equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{9}$$

The domain for this problem is defined in terms of polar coordinates by $0.05 \leq r \leq 1.00$ and $0 \leq \theta \leq 2\pi$. The cylinder is represented by the inner radius boundary. In order to show the versatility of this ANN implementation, we will solve this problem in cartesian coordinates. The boundary conditions are:

Farfield:

$$u(x, y) = x, \text{ for } x \in [-1, 1] \text{ and } y = \pm\sqrt{1 - x^2} \tag{10}$$

Wall:

$$\frac{\partial u}{\partial x} \cdot \frac{x}{0.05} + \frac{\partial u}{\partial y} \cdot \frac{y}{0.05} = 0, \text{ for } x \in [-0.05, 0.05] \text{ and } y = \pm\sqrt{0.05 - x^2} \tag{11}$$

Two ANNs are defined to solve this problem. The first is trained with 1200 random points between $0.15 < r < 1.00$ and the second one uses another set of 1200 training points in the interval of $0.05 < r < 0.15$. Additional interface conditions are added at $r = 0.15$ to guarantee C^1 continuity between predictions of these networks. Therefore, one network is responsible by carrying out the free-stream information to the vicinity of the cylinder, while the second one focuses on learning the near-field effects. In addition to the 2400 training points cited previously, we distribute 40 points along each of the boundaries (farfield and wall) and also along the interface between ANNs at $r = 0.15$. Both networks have two hidden layers with 8 neurons each, and they are trained with 10 ALM iterations using the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm and a regularization factor, γ, of 0.001. Since this case is out of the scope of the optimizer selection described further in the next section, the BFGS optimization algorithm is adopted and the remaining optimizers will not be used to solve this problem.

2.4 Optimizer selection details

The Augmented Lagrangian and its gradients should be provided to gradient-based optimization algorithms for its minimization. One of such optimizers is the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm, which is successfully applied to ANN training at the studies presented by Lagaris *et al.* (1998); Lagaris *et al.* (2000); Berg and Nyström (2018); Anitescu *et al.* (2019); Raissi *et al.* (2019). In this work we use the default BFGS implementation from the *Scipy* Python module (Jones *et al.*, 2001).

However, as stated by Berg and Nyström (2018), the line search routine of the BFGS method may fail due to the Hessian matrix being ill-conditioned. Berg and Nyström (2018) avoid the possible line search failure from the BFGS optimizer by starting the ANN training with some iterations with a Stochastic Gradient Descent (SGD) algorithm and then switching to the BFGS method at later stages. Thus, it is also studied a gradient-based method called Scaled Conjugate Gradient (SCG) (Møller, 1993), which applies the traditional conjugate gradient with a scale factor. One drawback of the SCG algorithm is its slower convergence rate compared to BFGS.

Since each iteration of the ALM used in this work involves an unconstrained optimization, we evaluate different optimization strategies by switching the SCG and BFGS algorithms at each iteration:

- Method 1: BFGS at all ALM iterations;
- Method 2: SCG at all ALM iterations;
- Method 3: SCG at a fixed number of ALM iterations, followed by BFGS for the remaining ones (without the Lagrange multipliers reinitialization).

The optimization problem associated with ANN training is known to have local minima. Therefore, we train ANNs starting from 21 different parameter configurations, which are randomly initialized using the technique described by Nguyen and Widrow (1990). Then, it is possible to evaluate the average MSE and training time of each optimization method at each problem and select the method that more robustly finds low MSE values. For this purpose, we will evaluate the canonical problems presented at Table 1. The optimization settings applied to each canonical problem are presented at Table 3.

Table 3: Optimization settings for each canonical problem.

	1-D Linear Advection	**1-D Periodic Linear Advection**	**2-D Steady Heat Transfer**	**1-D Unsteady Heat Transfer**	**1-D Inviscid Burgers' equation**
Reg. factor (γ)	0.00	0.00	0.20	0.01	0.00
ALM Iterations	10	10	10	10	15
Forced SCG Iterations	1	1	2	2	1
Minor Iterations	300	300	2500	2000	1200

From Table 3, each problem has its own settings. The **Regularization factor** avoids ANN weights of large magnitudes, preventing possible divergences. The **ALM Iterations** values indicates the maximum number of unconstrained optimization to be performed within the ALM algorithm, and the **Forced Iterations** row sets how many of these ALM iterations use the SCG optimizer before switching to BFGS when applying Method 3. The **Minor Iterations** values represent the maximum number of SCG or BFGS iterations allowed within a single ALM iteration.

3. RESULTS

3.1 Optimizer selection

After evaluating all the assigned problems presented at the optimizer details procedure (Section 2.4), the Geometric Mean Squared Error (GMSE) and average time are extracted. At table 4, is compiled the residual and training time behind the applied ANN at each canonical problem. The best (green) and worst (orange) residual and training time for each optimization method is also highlighted.

Table 4: Average residue and training time. The green and orange cells represent the best and worst residual/training time, respectively.

	Average Residue (GMSE)				
	1-D Linear advection	1-D Periodic Linear Advection	2-D Steady Heat Transfer	1-D Unsteady Heat Transfer	1-D Inviscid Burguer's equation
Method 1 (SCG)	1.02E-02	3.77E-02	1.23E-03	3.21E-04	4.33E-05
Time, s	17	195	493	512	2003
Method 2 (BFGS)	3.94E-02	2.42E-01	1.06E-02	7.64E-04	3.65E-06
Time, s	5	98	263	164	1029
Method 3 (SCG+BFGS)	1.57E-02	2.24E-02	8.70E-03	4.33E-04	1.75E-06
Time, s	6	116	315	246	1138

The geometric average of the validation MSE, is denoted by the following equation:

$$GMSE = \left(\prod_{i=1}^{n} MSE_i \right)^{1/n} \tag{12}$$

where MSE_i denotes each seed MSE. Looking at the average training time to solve each problem, it depends directly on the number of layers, neurons, and especially the number of training points used to represent the boundary and the PDE itself. Also, the computer used to evaluated these problems has the following setup:

- Processor: I7-6700K 4,00 GHz;
- Operational System: Windows 10 (x64);
- 16 GB RAM;

The reason behind the use of GMSE is due to the poor representativity of the simple average to take into account outliers related to ANNs that converged to poor local minima, leading to training errors with several orders of magnitude above the rest.

When we look at the GMSE values, method 1 stands out as the best option for most canonical problems at the cost of having the highest training time. In contrast, the lowest training time is achieved by method 2 (pure BFGS), which has the highest MSE residual. The combination of SCG plus BFGS (method 3), has shown a good trade off between residual and training time. Therefore, the results of the ANNs trained with this method will be presented in the next sections.

3.2 ANN performance for canonical test cases

In this section we present how the ANNs approximated the PDE solutions over the domain of each canonical case. The best ANN in terms of MSE among the 21 networks trained with different starting weights for the linear advection problem gives the domain values shown in Figure 2. The ANN generalized the PDE solution even in regions distant from training points. However, the training technique employed in this work is still susceptible to local minima. For instance, if we take the ANN with the highest MSE after training (worst performance), we get results that do not follow the analytical PDE solution (Figure 3).

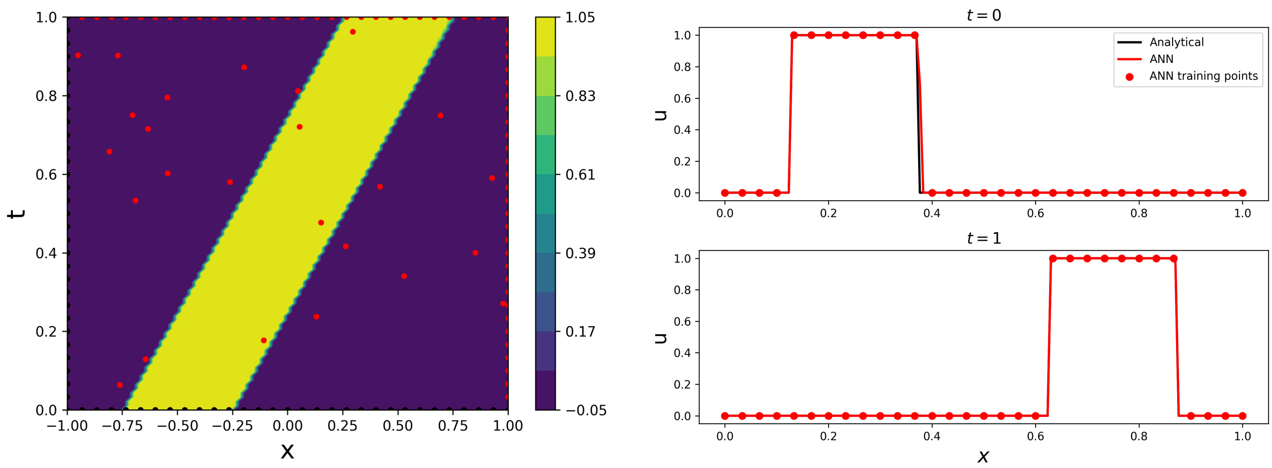
Figures 4 shows the domains values of best ANN obtained for the 1-D periodic linear advection case. The same ANN training procedure captures the PDE solution even with an opposite advection velocity. In contrast, other numerical methods may require techniques such as upwinding or artificial dissipation to reach stable solutions. The ANN training methodology also works for elliptic, parabolic, and nonlinear PDEs, as illustrated by results of the best-performing ANNs for the remaining canonical cases (Figures 5, 6, and 7, respectively).

In order to compare the performance of the ANN solutions against the finite difference method, we evaluate the steady heat transfer in a square plate with the same domain, PDE, and BCs stated at Table 2 using a finite difference approach with Gauss-Seidel method. In general, the interior points are updated iteratively in accordance to the respective relationship:

$$u_{i,j}^{(k+1)} = \frac{u_{i-1,j}^{(k+1)} + u_{i+1,j}^{(k)} + u_{i,j-1}^{(k+1)} + u_{i,j+1}^{(k)}}{4} \tag{13}$$

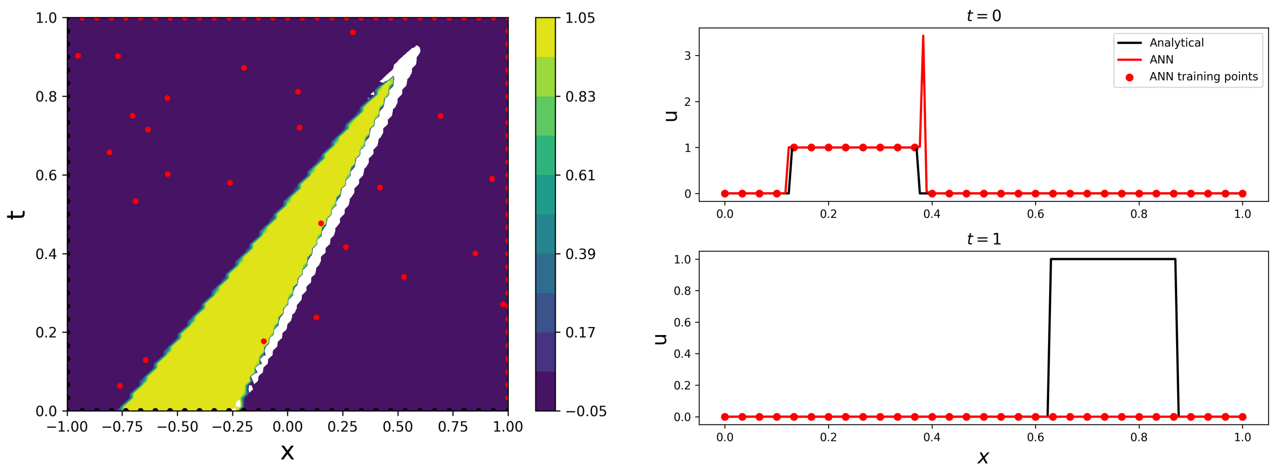
In order to reach the same MSE level as the ANN solution compared to the analytical result, the finite difference approach takes 26 seconds on a mesh of 100×100 points, against 322 seconds taken by the ANN. Even though the ANN-based approach is one order of magnitude slower, substantial time can be saved for more complex problems in terms of domain discretization.

Finally, Figure 8 shows the flowfield in the vicinity of the cylinder and the surface pressure distribution given by the ANN trained for the potential flow test case. The entire training procedure takes 4170 seconds, reaching an MSE of $8.1 \cdot 10^{-6}$. This results shows that the proposed methodology is capable of handling cases with different boundary condition types (Dirichlet BC at the farfield and Newmann BC at the wall). The versatility in terms of using unstructured meshes comes at the cost of a higher computational cost compared to traditional numerical methods such as Finite Differences.



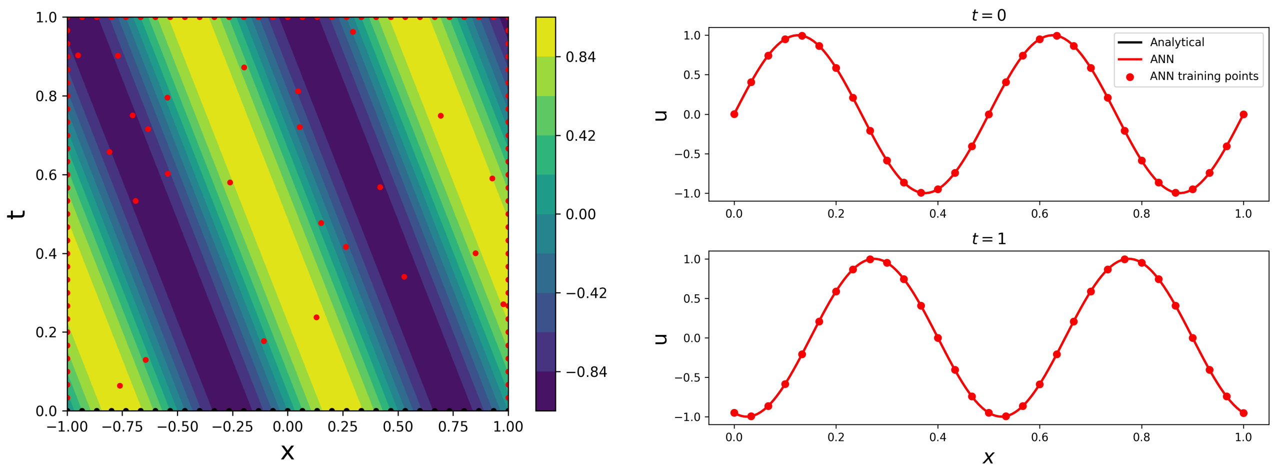
(a) 1-D Linear Advection domain. (b) 1-D Linear Advection slice.

Figure 2: 1-D Linear Advection solution for the best training case ($MSE = 1.8 \cdot 10^{-3}$).



(a) 1-D Linear Advection domain. (b) 1-D Linear Advection slice.

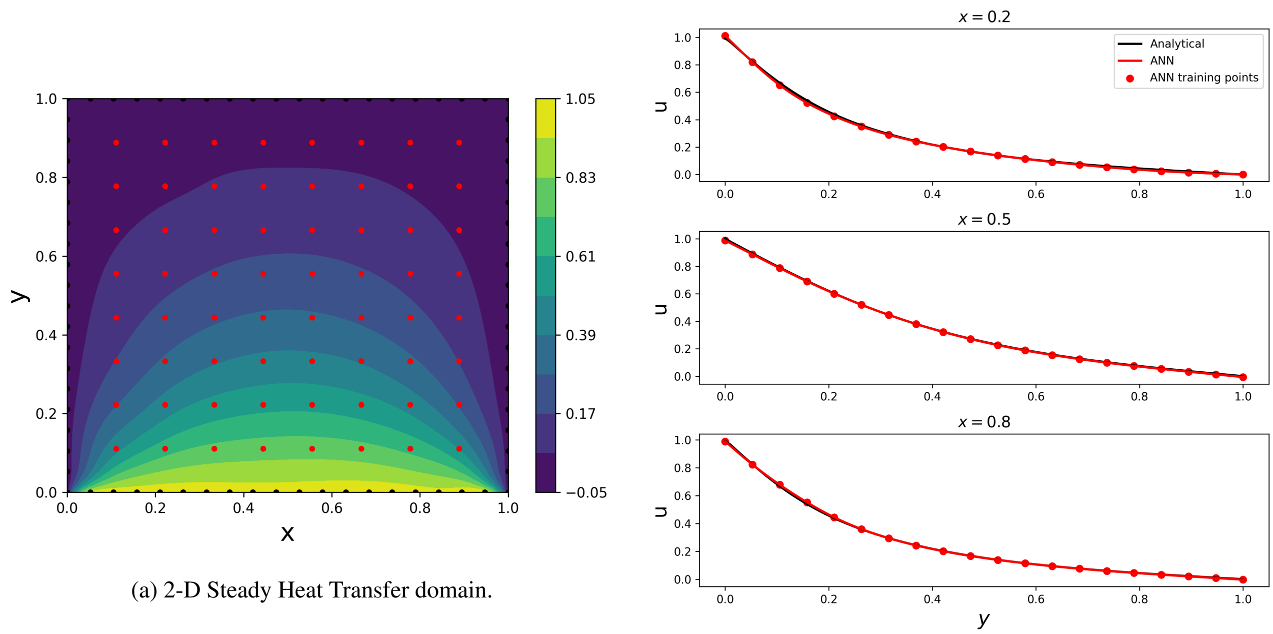
Figure 3: 1-D Linear Advection solution for the worst training case ($MSE = 2.2 \cdot 10^{-1}$).



(a) 1-D Periodic Linear Advection domain.

(b) 1-D Periodic Linear Advection slice.

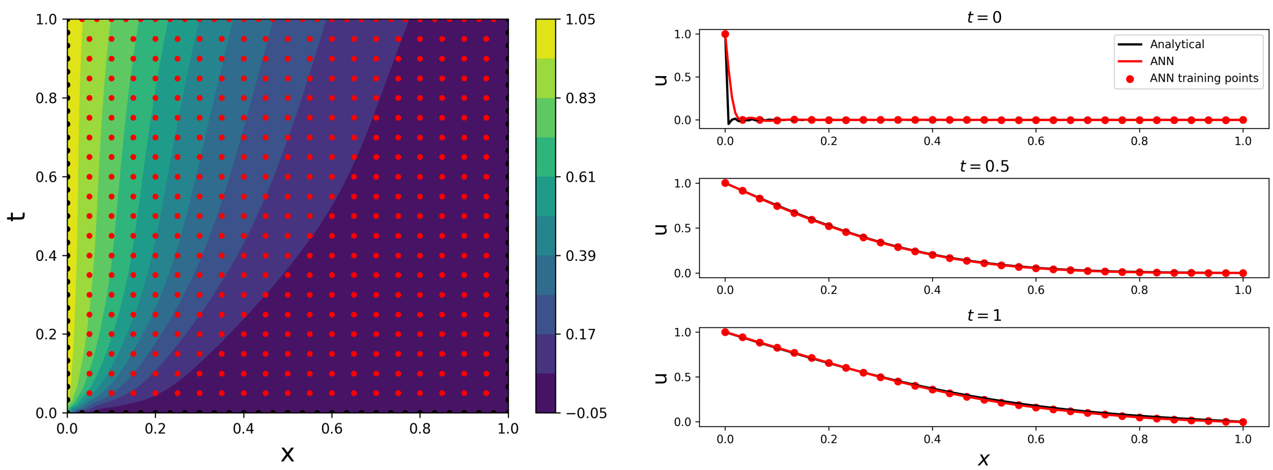
Figure 4: 1-D Periodic Linear Advection solution.



(a) 2-D Steady Heat Transfer domain.

(b) 2-D Steady Heat Transfer slice.

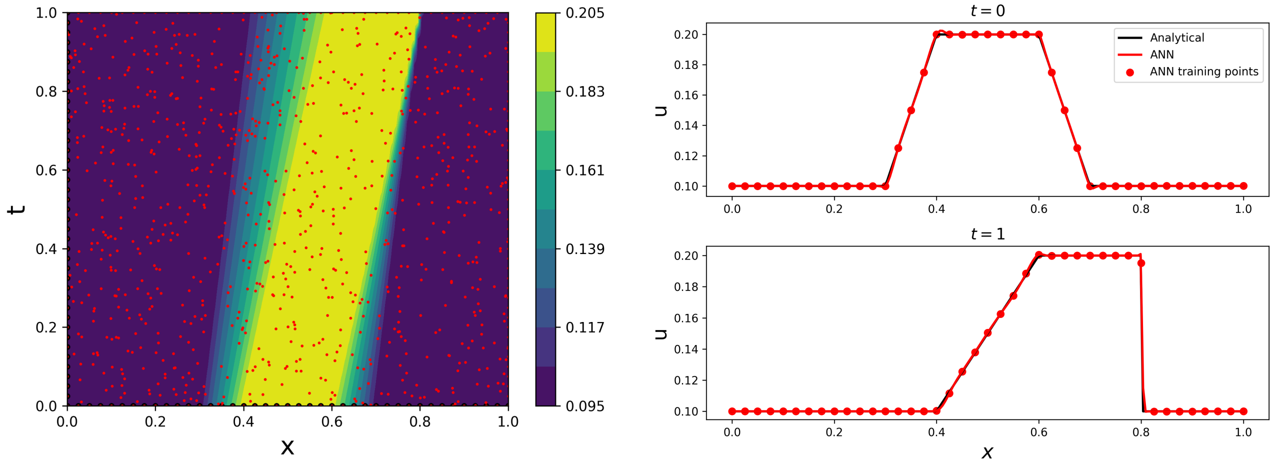
Figure 5: 2-D Steady Heat Transfer solution.



(a) 1-D Unsteady Heat Transfer domain.

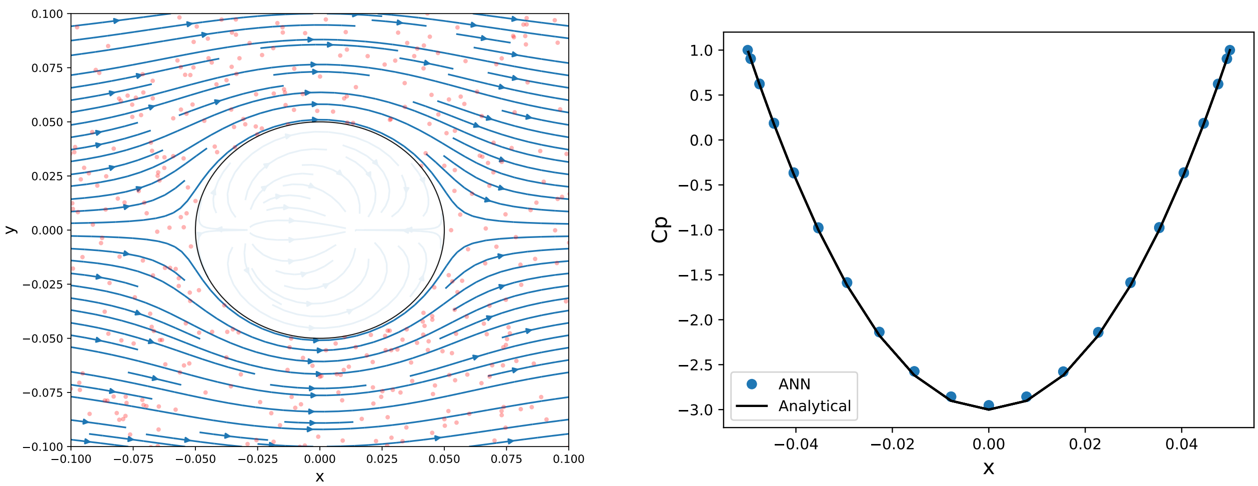
(b) 1-D Unsteady Heat Transfer slices.

Figure 6: 1-D Unsteady Heat Transfer solution.



(a) 1-D Inviscid Burgers' equation domain. (b) 1-D Inviscid Burgers' equation slices.

Figure 7: 1-D Inviscid Burgers' equation solution.



(a) Near-field streamlines and training points.

(b) Pressure coefficient distribution over the cylinder.

Figure 8: 2-D Potential Flow solution.

4. CONCLUSION

This work introduces a mesh-free method to obtain numerical solutions to PDEs using ANNs. The numerical method shows versatility, since it manages to approximate solutions for different types of PDEs and BCs on regular or random distribution of points over the domain.

The ANN training with a combination of SCG plus BFGS (Method 3) showed an intermediate compromise between training time and solution accuracy compared to the original optimization algorithms. However, the evaluation of canonical problems showed that the ANN-based solution is still sensitive with respect to the initialization of weights at the beginning of the optimization.

Further contributions are necessary to improve the robustness of the method with respect to the starting point, decrease the training time, and also to reduce the number of user-defined parameters such as number of neurons and regularization factors.

5. REFERENCES

Aarts, L.P. and der Veer, P.V., 2001. "Neural network method for solving partial differential equations". *Neural Process. Lett.*, Vol. 14, No. 3, pp. 261–271.

Anitescu, C., Atroshchenko, E., Alajlan, N. and Rabczuk, T., 2019. "Artificial neural network methods for the solution of second order boundary value problems". *Computers, Materials and Continua*, Vol. 59, No. 1, pp. 345–359.

Baymani, M., Kerayechian, A. and Effati, S., 2010. "Artificial neural networks approach for solving stokes problem".

Applied Mathematics, Vol. 1, No. 4, p. 288.

Berg, J. and Nyström, K., 2018. "A unified deep artificial neural network approach to partial differential equations in complex geometries". *Neurocomputing*, Vol. 317, pp. 28–41. ISSN 0925-2312.

Dissanayake, M.W.M.G. and Phan-Thien, N., 1994. "Neural-network-based approximations for solving partial differential equations". *Communications in Numerical Methods in Engineering*, Vol. 10, No. 3, pp. 195–201.

Jones, E., Oliphant, T., Peterson, P. *et al.*, 2001. "SciPy: Open source scientific tools for Python". `http://www.scipy.org`. Accessed: 2021-06-03.

Katz, A.J., 2009. *Meshless methods for computational fluid dynamics*. Stanford University Stanford, CA.

Keye, S. and Rudnik, R., 2015. "Validation of wing deformation simulations for the nasa crm model using fluid-structure interaction computations". In *53rd AIAA Aerospace Sciences Meeting*.

Lagaris, I.E., Likas, A. and 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.

Lagaris, I.E., Likas, A.C. and 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.

Malek, A. and Shekari Beidokhti, R., 2006. "Numerical solution for high order differential equations using a hybrid neural network—optimization method". *Applied Mathematics and Computation*, Vol. 183, No. 1, pp. 260–271. ISSN 0096-3003.

Møller, M.F., 1993. "A scaled conjugate gradient algorithm for fast supervised learning". *Neural Networks*, Vol. 6, No. 4, pp. 525–533. ISSN 0893-6080.

Nguyen, D. and Widrow, B., 1990. "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights". In *1990 IJCNN International Joint Conference on Neural Networks*. pp. 21–26 vol.3.

Nocedal, J. and Wright, S.J., 2006. *Numerical Optimization*. Springer, New York, NY, USA, 2nd edition.

Raissi, M., Perdikaris, P. and Karniadakis, G., 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. ISSN 0021-9991.

Secco, N.R. and Martins, J.R.R.A., 2019. "RANS-based aerodynamic shape optimization of a strut-braced wing with overset meshes". *Journal of Aircraft*, Vol. 56, No. 1, pp. 217–227.

Shirvany, Y., Hayati, M. and Moradian, R., 2009. "Multilayer perceptron neural networks with novel unsupervised training method for numerical solution of the partial differential equations". *Applied Soft Computing*, Vol. 9, No. 1, pp. 20–29. ISSN 1568-4946.

6. RESPONSIBILITY NOTICE

The authors are solely responsible for the printed material included in this paper.