Review

ANNs-BASED METHOD FOR SOLVING PARTIAL DIFFERENTIAL EQUATIONS : A SURVEY

Danang A. Pratama [1,‡], Maharani A. Bakar [1,‡,*], Mustafa Man [1,‡,*], and Mashuri M. [2]

[1] Faculty of Ocean Engineering Technology and Informatics, Universiti Malaysia Terengganu, Malaysia; p4433@pps.umt.edu.my
[2] Department of Mathematics , University of Jenderal Soedirman, Jl.Dr.Soeparno No.61, Purwokerto, Jawa Tengah, Indonesia; mashuri@unsoed.ac.id
* Correspondence: maharani@umt.edu.my; Tel.: Tel.: +60-9668-3989
‡ These authors contributed equally to this work.

Abstract: Conventionally, partial differential equations (PDE) problems are solved numerically through discretization process by using finite difference approximations. The algebraic systems generated by this process are then finalized by using an iterative method. Recently, scientists invented a short cut approach, without discretization process, to solve the PDE problems, namely by using machine learning (ML). This is potential to make scientific machine learning as a new sub-field of research. Thus, given the interest in developing ML for solving PDEs, it makes an abundance of an easy-to-use methods that allows researchers to quickly set up and solve problems. In this review paper, we discussed at least three methods for solving high dimensional of PDEs, namely PyDEns, NeuroDiffEq, and Nangs, which are all based on artificial neural networks (ANNs). ANN is one of the methods under ML which proven to be a universal estimator function. Comparison of numerical results presented in solving the classical PDEs such as heat, wave, and Poisson equations, to look at the accuracy and efficiency of the methods. The results showed that the NeuroDiffEq and Nangs algorithms performed better to solve higher dimensional of PDEs than the PyDEns.

Keywords: artificial neural networks, discretization, machine learning, partial differential equations.

1. Introduction

Partial differential equation (PDE) consists of an equation specifying a relation between the derivatives of a function of one or more derivatives and the function itself [1]. The recent advances in modern science have been based on discovering the underlying PDE for the process in question [2]. Hence, the ability to solve PDEs fast and accurately is an active field of research and industrial interest [3]. Conventionally, PDE problems are solved numerically through discretization process by using finite difference approximations [4]. A number of methods for this approach include Runge-Kutta, finite difference, etc. are available for approximate the solution accurately and efficiently [5–7]. However, while these methods are efficient and well-studied, these traditional methods are require much memory space and time. Thus made the approximation computational process costly [7]. As alternative approach, we can replace traditional numerical discretization method with Artificial Neural Networks (ANNs) to approximates the PDE solution [8].

ANNs, as a core of machine learning, are ideal for solving a large scale of of ML tasks [9,10]. The simplest form of neural networks, called multilayer perceptrons, be the popular estimators function [11]. Furthermore, ANNs have been investigated since early 1990s to solve the PDE problems. Lee, & Kang[12], used a parallel architecture to solve the first order differential equations (ODEs) by implementing the Hopfield neural network models. Meade and Fernandez[13], on the other hand, solved the linear and non-linear ODEs by using the feed-forward neural networks. Lastly, Lagaris, Likas & Fotiadis[14], used artificial neural network for solving ODEs and PDEs by considering the initial and boundary conditions.

After took a long pause, the development of ANNs in PDE problem took more and more attempt to done in early 21st century. Malek et. al.[15] are used the hybrid neural

network and Nelder-Mead simplex method to find numerical solutions of high-order PDE. While, Hussian et. al.[16] modified the neural network to solve PDE. After that, the DeepGalerkin-Method which uses deep neural networks for solving tasks at high dimensions also introduced [17,18]. Then, other method namely Physics Informed Neural Network (PINN) are itroduced by Lu et. al. [8] for solving PDE, and improved by Guo et. al. [19]. This improved PINN takes the physical information in PDE as a regularization term, which improves the performance of neural networks. However, This is potential to make Scientific Machine Learning as a new sub-field of research.

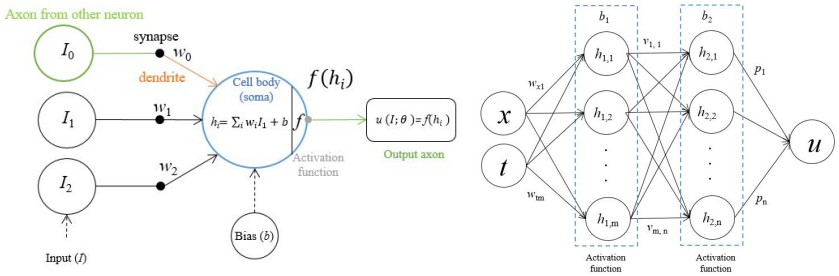
This possibility of using ANNs to solve PDE show nice advantages including continuous and differentiable solutions, good interpolation properties, and less memory-intensive [20]. Compared to the traditional methods, neural network approximation could taking advantage of the automatic differentiation [21], and could break the curse of dimensionality [22]. Given the interest in developing neural networks for solving PDE, makes a lots of an easier methods that allows researchers to solve problems [20]. We discovered that there are several methods that used to solve ordinary and partial differential equations (ODEs and PDEs) with neural networks: Nangs [23], PyDEns [24] and NeuroDiffEq [20].

In this paper, the comparisons between those three methods are discussed, regarding capability and the effectiveness in solving PDE. In section 2, we will overview general methodology of ANNs for solving PDE. In section 3, the approximation theory and error analysis which underlies each methods are also discussed. In section 4, the algorithms of each method for solving PDE are derived, and we demonstrate the results of each method for solving different types of PDE in section 5. Finally, we conclude the paper in section 6.

2. Artificial Neural Networks

Artificial Neural Networks (ANNs) are built quite a while, since first introduced back in 1943 by Warrent McCulloch and Walter Pitts. The inventors was inspired of how biological neurons in animal brain might work together to perform complex computations [9]. The early success of ANN was faded away at the early beginning, since left out behind other machine learning techniques. However, it turned around in the 1990s following the tremendous increased of computational power and huge quantities data available to train ANNs. This technology has been successfully applied to a wide variety of real-world applications [25], including in the differential equation problems [12] as discussed in this paper.

Perceptron (see Fig. 1a) is one of the simplest ANNs architectures which is heavily used today. Perceptron consists of more than one hidden layer called Multilayer Perceptron (See Fig. 1b).



(a) Perceptron Model **(b)** Multilayer Perceptron

Here we will also overview the general methodology of neural networks to solve PDE. Consider the PDE of the form:

$$\frac{\partial u}{\partial t}(x,y) + \mathcal{L}u(x,t) = f(x,t), \quad (x,t) \in [0,T] \times \Omega,\ \Omega \subset \mathcal{R}^d, \tag{1}$$

with initial condition

$$u(x, t=0) = u_0(x), \tag{2}$$

and boundary conditions

$$u(x,t) = g(x,t),\ x \in \sigma\Omega, \tag{3}$$

$\mathcal{L}$ is a spatial differential operator and f is known as a forcing function.

Let us assume a four-layered perceptron (Fig. 1b) with the inputs x, and t, which are hidden layers consisting with m and n hidden units (neurons) respectively. Its aim is to obtain the trained multilayer perceptron that approximates $u(x,t)$, by using the approximate function $u_{net}(t,x;\theta)$, for x and t are the inputs, and θ contains the adjustable parameter weights and biases. For every unit input x and t, the process begins from input layer to the first hidden layer described as follows

$$h_1 = \sum_{i=1}^{m}(w_{xi}x + w_{ti}t) + b_1, \tag{4}$$

where w_{xi} and w_{ti} are the weights of the inputs x and t to the first hidden layer respectively, and b_1 are the first hidden layer biases. Hence, it is activated by the hyperbolic tangent function ($tanh$) as follows [10]

$$f(h_1) = \frac{e^{h_1} - e^{-h_1}}{e^{h_1} + e^{-h_1}}. \tag{5}$$

The next step is the process of the feeding of the second hidden layer by the first hidden layer which is based on the formula as follows

$$h_2 = \sum_{j=1}^{n}\sum_{i=1}^{m} v_{ij}f(h_1) + b_2, \tag{6}$$

where v_{ij} are the weights of the first to the second hidden layers, and b_2 are the biases. When 6 became to output layer, it turned into the form

$$u_{net}(x,t;\theta) = \sum_{j=1}^{n} p_j f(h_2), \tag{7}$$

where p_j are the weights of the second hidden layers to the output layers. This feeding process also called Feed-Forward Neural Networks(FFNNs). Then, it is also easy to express the k-th derivatives of $u_{net}(x,t;\theta)$ with automatic differentiation (AD) in terms

$$\frac{\partial^k u_{net}}{\partial t^k}(x,t;\theta) = \sum_{j=1}^{n} \frac{\partial^k p_j f(h_2)}{\partial x^k}, \tag{8}$$

$$\frac{\partial^k u_{net}}{\partial t^k}(x,t;\theta) = \sum_{j=1}^{n} \frac{\partial^k p_j f(h_2)}{\partial t^k}, \tag{9}$$

for $k = 1, 2, \ldots, n$.

The performance of the solution $u_{net}(x,t;\theta)$ is measured through the computational of the loss function. Therefore, the goal of all methods solved PDE is to minimize the loss function as well. Each method mentioned earlier has different techniques to build the loss function. If the loss function reaches a near-zero value, we can assume that our ANNs is indeed a solution to the PDEs [20,23]. To ensure the loss function is minimum, we have to adjust the parameter weights and biases by updating them being optimum. Thus it is possible to use any optimization techniques. The three methods discussed in this study used Stochastic Gradient Descent (SGD) optimizer to speed up their convergence.

3. Approximation theories and error analysis of the methods

Three methods for solving PDEs discussed include PyDEns, NeuroDiffEq, and Nangs which all are based on ANN. Those methods differ by the way generating the data points, setting up the boundary conditions, as well as the loss functions.

3.1. PyDEns

PyDEns was built under of DeepGalerkin-Method as it was introduced in [17,18,24]. To approximate the solution $u(t,x)$ in 1 using neural network $u_{net}(t,x;\theta)$, the loss function which is associated with the training problem consists of [18,24]:

1. Generate m points inside the batches of b_1, b_2, b_3 from the domain $[0;T] \times \Omega$, $[0;T] \times \sigma\Omega$, and $\{x\} \times \sigma\Omega$ respectively. These are the uniform distribution v_1, v_2, v_3. Then, for each point (x,t), do the following operation.
2. Compute the accuracy of the approximation solutions which satisfy the differential operator:

$$\mathcal{J}(\theta)_{DE} = \left[\frac{\partial u_{net}}{\partial t}(x,t;\theta) + \mathcal{L}u_{net}(x,t;\theta) - f(x,t)\right]^2_{(x,t)\in b_1\ v_1}. \tag{10}$$

3. Compute on how well the approximation solutions satisfy the boundary conditions:

$$\mathcal{J}(\theta)_{BC} = [u_{net}(x,t;\theta) - g(x,t)]^2 \quad (x,t) \in b_2,\ v_2. \tag{11}$$

4. Measure on how well the approximation solutions satisfy the initial condition:

$$\mathcal{J}(\theta)_{IC} = [u_{net}(x,0;\theta) - u_0(x)]^2 \quad (x,t) \in b_3,\ v_3. \tag{12}$$

Combining the three terms 10, 11, and 12 above gives us the loss function associated with training the neural network:

$$\mathcal{J}(\theta) = \mathcal{J}(\theta)_{DE} + \mathcal{J}(\theta)_{BC} + \mathcal{J}(\theta)_{IC}. \tag{13}$$

In other words, the $u_{net}(t,x;\theta)$ is fitted to validate all components 11, 12, and 13. The next step is to minimize the loss function by using the stochastic gradient descent (SGD)[18].

3.2. NeuroDiffEq

The key idea of solving PDEs using NeuroDiffEq is by casting the trial approximate solution (TAS), $u_T(x,t;\theta)$ [20]. The algorithms of NeuroDiffEq are defined as follows:

1. Generate a set of input in $m \times n$ grid points $P = (x,t)$, where $x = (x_1, \ldots, x_m)$, $t = (t_1, \ldots, t_n)$ inside the domain $[0;T] \times \Omega$, which is fed through the multilayer perceptrons. The generated points are drawn from whatever distribution chosen, such as the uniform distribution, equally-space, etc.
2. Develop the TAS which is a known function consists of the input P and the outputs $u_{net}(x,t;\theta)$. Furthermore, the TAS can be defined as the form of [26]

$$u_T(x,t;\theta) = A(x) + F(x,t;u_{net}), \tag{14}$$

where $A(x)$ is chosen to satisfy the boundary conditions and $F(x,t;u_{net})$ is chosen to be zero for any $x,\ t$ on the boundary. This produces a TAS which automatically satisfies the boundary conditions regardless the ANN outputs. This approach is similar to the trial function approach [14], but with a different form of the trial function. Modifying the TAS for initial conditions can also be done. In general, the transformed solutions have the form of [20]

$$u_T(x,0;\theta) = u_0(x,0) + (1 - e^{-T})u_{net}(x,t;\theta). \tag{15}$$

3. The TAS is developed in order to minimize the error function [26]

$$\mathcal{J}(\theta) = \mathcal{J}(\theta)_{DE} + \eta\mathcal{J}(\theta)_{BC}. \tag{16}$$

It is noted that the first term of

$$\mathcal{J}(\theta)_{DE} = \left[\frac{\partial u_T}{\partial t}(x,t;\theta) + \mathcal{L}u_T(x,t;\theta) - f(x,t)\right]^2_{(x,t)\in P}, \tag{17}$$

is the approximate solutions of the PDE itself, where P is the composed of a finite set of points within the PDE domain, while the second term of

$$\mathcal{J}(\theta)_{BC} = [u_T(x,t;\theta) - g_D(x,t)]^2_{(x,t)\in P_D} + [\hat{n}(x,t)\cdot\mathcal{L}u_T(x,t;\theta) - g_M(x,t)]^2_{(x,t)\in P_M}, \tag{18}$$

satisfies the Dirichlet (D) and Neuman (M) boundary/initial conditions, where P_D and P_M are the set of points where the boundary values g_D and g_M are specified respectively, and $\hat{n}(x,t)$ is the inwardly directed unit normal to the boundary at (x,t). The weighting factor η determines the relative importance of the two error components, [26].

3.3. Nangs

The scenario of Nangs method for solving PDE problems is quite similar to the PyDEns. It is not necessarily, for instance, to build any trial solution in order to minimize the loss functions. The only difference is that to define a set of points inside the domain, it is generated without using any distributions, instead, we build a mesh points as done in the traditional methods. For more detail, the algorithms of solving PDE using Nangs are described as follows:

1. Define a set of mesh points of $P = (x_m, t_n) \in [0,T]\times\Omega$ inside the domain. These mesh points are the combination of the internal, boundary and initial points (see Fig. 2). In addition, they are used as the input values of the ANNs feed.

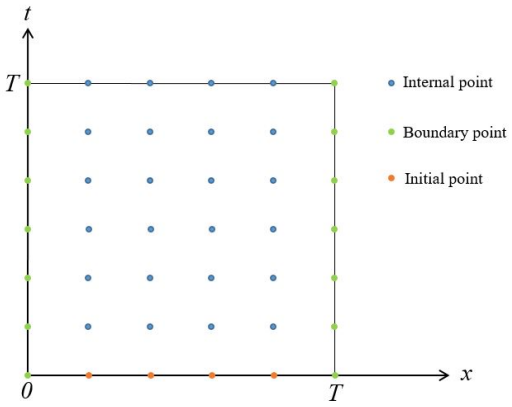


Figure 2. Example of internal, boundary and initial points of $m = 6 \times n = 7$

2. The internal points of ANN are compared with the right hand side of the PDE itself by using the loss function

$$\mathcal{J}(\theta)_{DE} = \left[\frac{\partial u_{net}}{\partial t}(x,t;\theta) + \mathcal{L}u_n(x,t;\theta) - f(x,t)\right]^2_{(x,t)\in P_{internal}}, \tag{19}$$

3. The initial and boundary points of the PDE are compared with the outputs of the initial and boundary conditions of ANN 2 and 3 respectively by using the loss function:

$$\mathcal{J}(\theta)_{BC} = [u_{net}(x,t;\theta) - g(x,t)]^2 \quad (x,t)\in P_{boundary} \tag{20}$$

$$\mathcal{J}(\theta)_{IC} = [u_{net}(x,0;\theta) - u_0(x)]^2. \quad x \in P_{initial} \tag{21}$$

4. PyDEns, NeuroDiffEq, and Nangs in solving heat equation

We discuss on how the three methods are vary to solve the PDE problems. As an illustration, we took some problems of PDE elliptic from [24], PDE hyperbolic, and PDE parabolic from [27]. Here we first solve the heat equation, which is an elliptic equation, by using the three methods. Given

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0, \quad 0 \leq x \leq 1, \; t \geq 0, \tag{22}$$

with the initial condition

$$u(x, 0) = sin(\pi x), \tag{23}$$

and boundary conditions

$$u(0, t) = u(1, t) = 0. \tag{24}$$

In order to compare the effectiveness of each method, we used the same ANNs architecture which contains 3 hidden layers with 32 neurons each. The accuracy of this PDE solutions by using ANNs algorithm is evaluated up to 200×200 dimensions for the unit input (x, t). Here, all methods uses the same optimization technique, which is stochastic gradient descent (SGD). The number of iterations used in solving the heat equation problems vary for each method, the reasons would be explained in the next section. For better results, we can do several things, for instance, change the ANN architecture, turn on the hyperparameter tuning which is appropriate to the ANNs architecture, add the number of iterations, or change the optimization method. Except for PyDEns, as far as we know, PyDEns can only use SGD.

4.1. Solving heat equation using PyDEns

As mentioned earlier, solving the heat equation 22 using PyDEns is based on Deep-Galerkin method. Firstly, we define a certain number of points in each batch, because the equations 22, 23, 24 contained the b_1, b_2, b_3 batches, which refer to PDE boundary and initial conditions respectively. Furthermore, to evaluate up to 200×200 dimension points, each batch would be filled with $x = (x_1, \ldots, x_{13.333})$ and $t = (t_1, \ldots, t_{13.333})$. Hence we compute the loss function inside of the domain $(x, t) \in [0, 1]$. The algorithm is completely described as in algorithm 1.

Algorithm 1 PyDEns algorithm for solving heat equation

1: Generate up to 200×200 points inside of the batches b_1, b_2, b_3 uniformly and inside of the $(x, t) \in [0, 1]$. Each batch consists of $x = (x_1, \ldots, x_{13.333})$ and $t = (t_1, \ldots, t_{13.333})$ points.
2: **while** $n = 0 \leq$ itteration **do**
3: **for** $(x, t) \in [0, 1]$ in b_1, b_2, b_3 **do**
4: Compute the output

$$u_{net}(x, t; \theta) = \sum_{i=1}^{32} p_i f(h_3)$$

$$\frac{\partial u_{net}}{\partial t}(x, t; \theta) = \sum_{i=1}^{32} \frac{\partial p_i f(h_3)}{\partial t}$$

$$\frac{\partial^2 u_{net}}{\partial x^2}(x, t; \theta) = \sum_{i=1}^{32} \frac{\partial^2 p_i f(h_3)}{\partial x^2}$$

5: Compute the loss function by comparing the output with the original problems as in equations 22, 23, and 24, as follows

$$\mathcal{J}(\theta) = \left[\frac{\partial u_n}{\partial t}(x_m, t_m; \theta) - \frac{\partial u_n^2}{\partial x^2}(x_m, t_m; \theta) \right]^2 + [u_n(x_m, t_m; \theta) - 0]^2,$$

$$+[u_n(x_m, 0; \theta) - sin(\pi x_m)]^2.$$

6: **end for**
7: optimize the weight and biases by applying the SGD optimizer.
8: **end while**

4.2. Solving heat equation using NeuroDiffEq

Solving PDE with NeuroDiffEq start with generating up to 200×200 data points inside the domain like other method did. However, NeuroDiffEq didn't separate into a batches. After generated the data points, we have to build TAS in order to satisfied the boundary/initial conditions 23 and 24. To know detail about TAS, you can refer to [26]. The detail of solving heat equation 22 are describe with algorithm 2.

Algorithm 2 NeuroDiffEq algorithm for solving heat equation

1: Generate uniformly 200×200 points $P = \{x_m, t_m\}_{m=1}^{200}$ inside of the domain $[0, 1]$, then divide them into the training and the validating data.
2: **while** $n = 0 \leq$ iteration **do**
3: **for** each $(x, t) \in P$ **do**
4: Compute

$$u_{net}(x, t; \theta) = \sum_{i=1}^{32} p_i f(h_3)$$

$$\frac{\partial u_{net}}{\partial t}(x, t; \theta) = \sum_{i=1}^{32} \frac{\partial p_i f(h_3)}{\partial t}$$

$$\frac{\partial^2 u_{net}}{\partial x^2}(x, t; \theta) = \sum_{i=1}^{32} \frac{\partial^2 p_i f(h_3)}{\partial x^2}$$

5: Build TAS which satisfied the initial condition based on 24 as follows

$$u_T(x, t; \theta) = xt(1 - x)(1 - t)u_n(x, t; \theta)$$

or it can also be expressed as

$$u_T(x, 0; \theta) = sin(\pi x) + (1 - e^1)u_n(x, 0; \theta).$$

6: Compute the loss function as in 13 by comparing the output in with equations 22, 23, and 24 as follows

$$\mathcal{J}(\theta) = \mathcal{J}_{DE} + \eta \mathcal{J}_{BC},$$

with

$$\mathcal{J}_{DE} = \left[\frac{\partial u_{net}}{\partial t}(x_m, t_m; \theta) - \frac{\partial u_{net}^2}{\partial x^2}(x_m, t_m; \theta)\right]^2,$$

and

$$\mathcal{J}_{BC} = [u_T(x_m, t_m; \theta) - 0]^2 + [u_T(x_m, 0; \theta) - sin(\pi x_m)]^2.$$

7: **end for**
8: Minimize the loss function by updating the weight and the biases by using SGD optimizer
9: **end while**

4.3. Solving heat equation using Nangs

To solve the heat equation 18 based on the Nangs method, firstly we define a set of points used as the training data. These points are the collected from the domain (x, t) which is built uniformly as a mesh points in the entire domain. Furthermore, the grid points are split into internal and and initial and boundary points. Note here, the initial

and boundary points are the points which are exactly on the edge of the domain, while the internal points are inside of the domain (see Fig. 2). These points would have different associated loss function. The loss function computed is inside of the domain $(x,t) \in [0,1]$. The algorithm goes in algorithm 3.

Algorithm 3 Nangs algorithm for solving heat equation

1: Set up 200×200 mesh points inside $(x,t) \in [0,1]$. Distinguish them into internal and boundary points.
2: **while** $n = 0 \leq$ iteration **do**
3: **for** each mesh point $(x,t) \in [0,1]$ **do**
4: Compute the outpus as follows

$$u_{net}(x,t;\theta) = \sum_{i=1}^{32} p_i f(h_3)$$

$$\frac{\partial u_{net}}{\partial t}(x,t;\theta) = \sum_{i=1}^{32} \frac{\partial p_i f(h_3)}{\partial t}$$

$$\frac{\partial^2 u_{net}}{\partial x^2}(x,t;\theta) = \sum_{i=1}^{32} \frac{\partial^2 p_i f(h_3)}{\partial x^2}$$

5: Compute each internal points, and compare with the original PDE of equations 22 as follows

$$\mathcal{J}(\subseteq)_{DE} = \left[\frac{\partial u_{net}}{\partial t}(x,t;\theta) - \frac{\partial u_{net}^2}{\partial x^2}(x,t;\theta)\right]^2.$$

6: Compute each initial and boundary points and compare with the original initial and boundary condition of the PDE as in 23 and 24 as follows

$$\mathcal{J}(\theta)_{Left} = [u_{net}(0,t;\theta) - 0]^2$$

$$\mathcal{J}(\theta)_{Right} = [u_{net}(1,t;\theta) - 0]^2$$

$$\mathcal{J}(\theta)_{Initial} = [u_{net}(x,0;\theta) - sin(\pi x)]^2$$

7: **end for**
8: Optimize the weight and the biases by using the SGD optimizer.
9: **end while**
10: The entire progress was repeated until the final iteration reached.

5. Comparison Results and Discussion

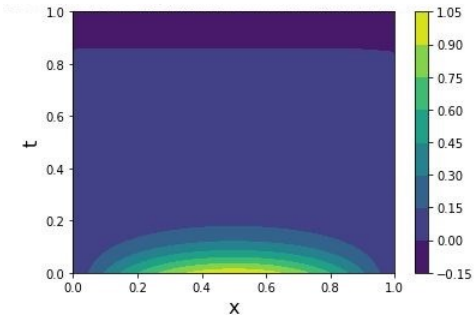
In this section, the three methods PyDEns, NeuroDiffEq, and Nangs to solve the PDEs heat, wave, and Poisson are discussed. The detail of the three algorithms to solve the heat equation has been discussed on the previous section, similarly, we used the three methods to solve the PDE wave and PDE Poisson. For the better results, we did several things such as changed the ANN architecture, performed the hyperparameter tuning, and added the number of iterations. We compared the performances of the three methods with the analytical solutions. All results is presented in several tables and figures.

5.1. Comparison between PyDEns, NeuroDiffEq, and Nangs methods in solving heat equation

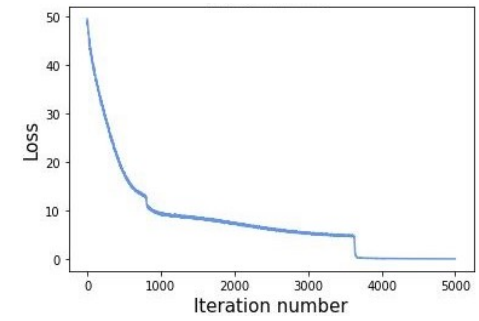
Table 1 showed the comparisons methods for solving heat equations as in 22, 23, and 24. The visualization of the results is displayed in figures 3a, 3b, 3c, 3d, 3e, and 3f to see clearly the performance of each method.

Table 1. Comparison result for solving heat equation

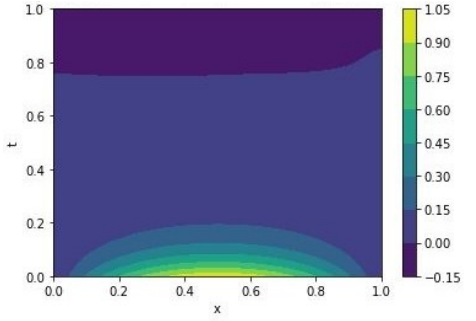
Dim	PyDEns			NeuroDiffEq			Nangs		
	Iter	Time	MSE	Iter	Time	MSE	Iter	Time	MSE
50	10000	03:57	3.9×10^{-4}	50	03:03	0.0153	100	05:29	0.0471
75	10000	06:54	2.2×10^{-4}		08:37	0.0031		05:55	0.0081
100	5000	06:53	0.0056		21:52	0.0017		06:22	0.0045
150	5000	14:25	0.0055		48:34	4.5×10^{-4}		09:25	5×10^{-4}
200	5000	27:35	0.0635		01:30:55	1.5×10^{-5}		14:13	1.9×10^{-5}



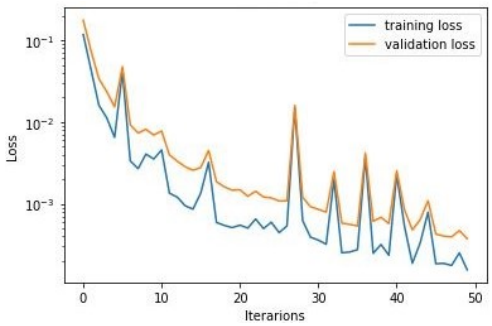
(a) PyDEns algorithm



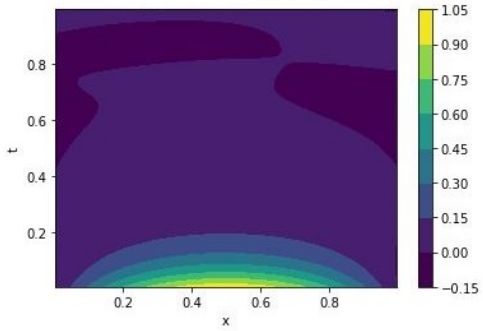
(b) MSE values of PyDEns algorithm



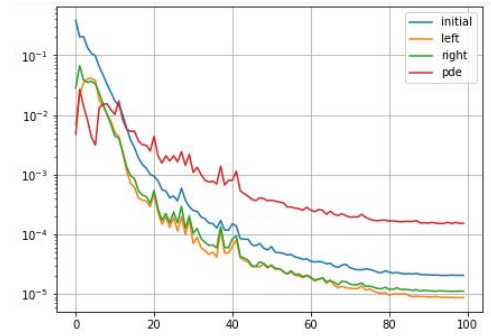
(c) NeuroDiffEq algorithm



(d) MSE values of NeuroDiffEq algorithm



(e) Nangs algorithm



(f) MSE values of Nangs algorithm

Figure 3. The three methods performances in solving heat equations, left : the approximate solutions visualization, right : the behaviour of the MSE

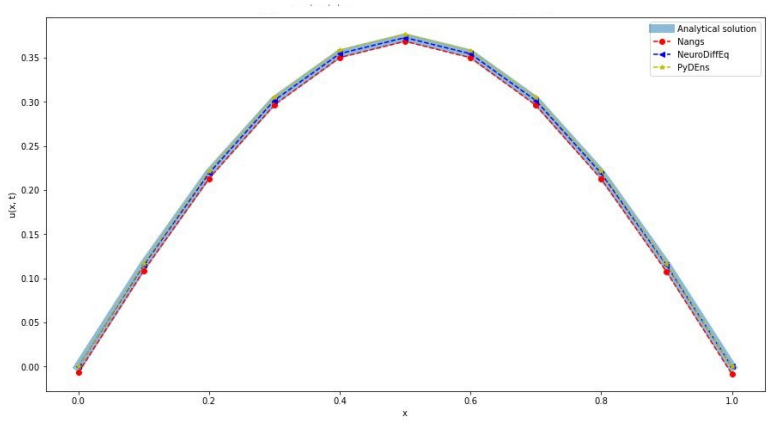


Figure 4. The approximate solutions of three methods vs analytical solution on solving 200 dimensions of heat equation when $t = 0.1$

As can be seen on the Table 1, PyDEns solved lower dimensions of the problem more accurately compared with the NeuroDiffEq and Nangs methods, where the MSE values of the three methods for solving 75 dimensions of PDE are respectively 2.2×10^{-4}, 0.0031, and 0.0081. In contrast, we have to set lower iteration on PyDEns since it gave NaN values on 10.000 iteration. As you can see, it will affect to the MSE result. Meanwhile, the other two methods can consistently solved higher dimensions, i.e. 200 dimensions, than the PyDEns method, with the MSE values of PyDEns, NeuroDiffEq, and Nangs methods are respectively 0.0635, 1.5×10^{-5}, and 1.9×10^{-5}. However, in terms of computational times, Nangs method is the best one since it consumed only 14.13 minutes to solve 200 dimensional problems, compared to 27.35 minutes and 1.30 hours for PyDEns and NeuroDiffEq respectively solving the same dimensions.

The performances of the three methods for solving heat equations were clearly seen in figures 3a - 3f. The accuracy of each method compared with the analytical solution was described as in Figure 4. The NeuroDiffEq curve gets closer to the analytical solution curve when solving 200 dimensional problems.

5.2. Comparison between PyDEns, NeuroDiffEq, and Nangs methods in solving wave equation

The PDE wave solved by using PyDEns, NeuroDiffEq, and Nangs methods is as follows [27]:

$$\frac{\partial^2 u}{\partial t^2} - 4\frac{\partial^2 u}{\partial x^2} = 0, \quad 0 \leq x \leq 1, \ t \geq 0, \tag{25}$$

with initial conditions

$$u(x, 0) = sin(\pi x), \tag{26}$$

$$\frac{\partial u}{\partial t}(x, 0) = 0, \tag{27}$$

and boundary conditions

$$u(0, t) = u(1, t) = 0. \tag{28}$$

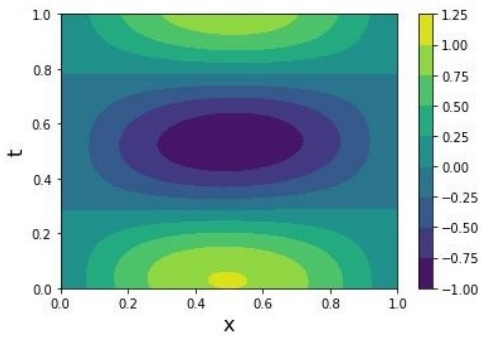
The analytical solution is given as follows

$$u(x, t) = sin(\pi x)cos(2\pi t). \tag{29}$$

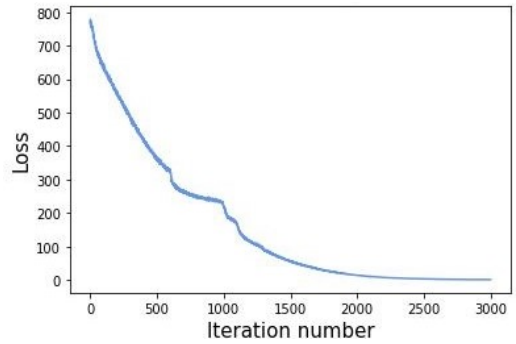
We presented all results of PDE wave solutions in Table 2 and Figures 5a, 5b, 5c, 5d, 5e, and 5f.

Table 2. Comparison result for solving wave equation

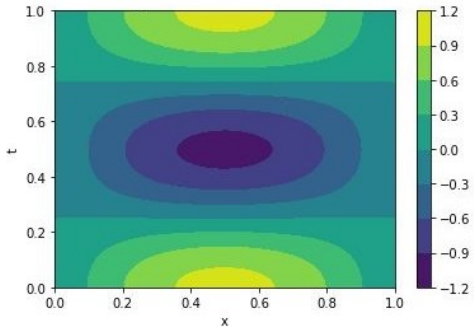
Dim	PyDEns			NeuroDiffEq			Nangs		
	Iter	Time	MSE	Iter	Time	MSE	Iter	Time	MSE
50	10000	04:58	0.0210	100	11:50	0.1051	100	05:00	0.1418
75	10000	10:39	0.0111		18:40	0.0149		05:51	0.0565
100	5000	10:20	0.0749		31:05	0.0244		07:25	0.0272
150	5000	20:17	0.0545		01:12:13	0.0056		11:41	0.0039
200	3000	23:15	0.6903		01:41:18	0.0076		15:51	5.2×10^{-4}



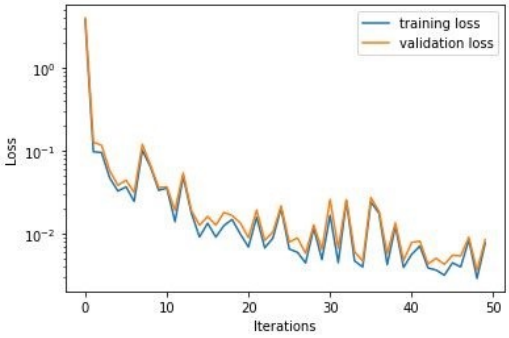
(a) PyDEns algorithm



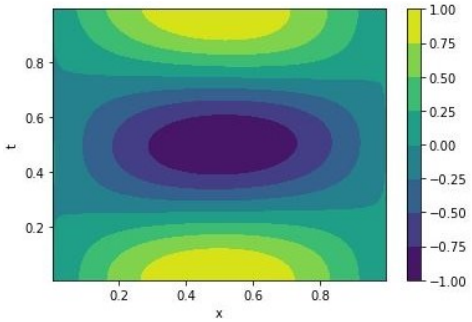
(b) MSE values of PyDEns algorithm



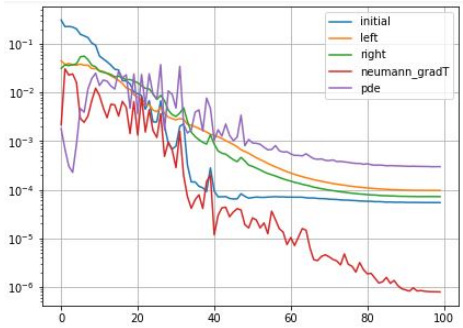
(c) NeuroDiffEq algorithm



(d) MSE values of NeuroDiffEq algorithm



(e) Nangs algorithm



(f) MSE values of Nangs algorithm

Figure 5. The three methods performances in solving wave equations, left : the approximate solutions visualization, right : the behaviour of the MSE

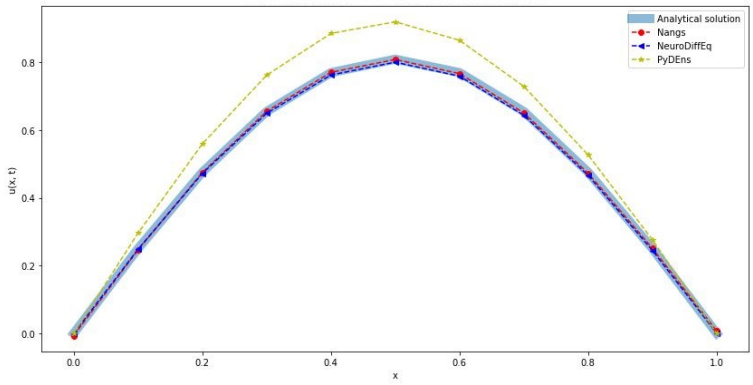


Figure 6. The approximate solutions of three methods vs analytical solution on solving 200 dimensions of wave equation when $t = 0.1$

Similar to the previous results, PyDEns only performed well when solving lower dimensions such as 50 and 75 dimensions. It was less accurate in solving the higher dimensions because we have to reduce the iterations to avoid NaN values. The other two methods, NeuroDiffEq and Nangs, consistently performed well when solving higher dimenional problems such a 150 and 200 dimensions. As can be seen in Table 2, the MSE values of PyDEns, NeuroDiffEq, and Nangs for solving 50 dimensional problems were respectively 0.0210, 0.1051, and 0.1418, whereas when solvig 200 dimensional problems, the MSE values were respectively 0.6903, 0.0076, and 5.2×10^{-4}. These results were supported by the figures 6, where the curves of PyDEns get fartest to the analytical solution from 200 dimensions, whereas NeuroDiffEq and Nangs are closer to the analytical solution when solving the higher dimensional problems. In terms of the efficiency of the three methods, however, Nangs method is the most efficient method for solving the high dimensional PDe wave problems, where it took 15.51 minutes, compared with 23.15 minutes and 1.42 hours for PyDEns and NeuroDiffEq respectively took the computational times.

5.3. Comparison between PyDEns, NeuroDiffEq, and Nangs methods in solving Poisson equation

The Poisson equations used in this study has the form of [24]:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad 0 \leq (x, y) \leq 1, \tag{30}$$

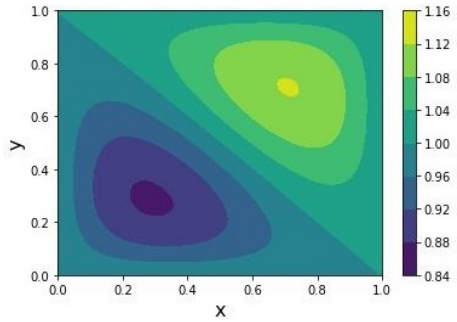
with boundary conditions

$$u(0, y) = u(1, y) = u(y, 0) = u(x, 1) = 1. \tag{31}$$

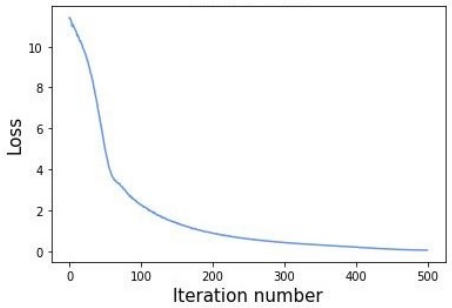
We presented all results of PDE Poisson solutions in Table 3 and Figures 7a, 7b, 7c, 7d, 7e, and 7f.

Table 3. Comparison result for solving Poisson equation

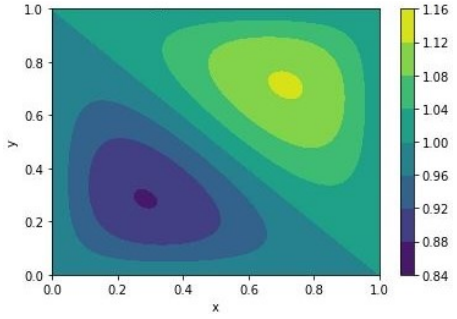
Dim	PyDEns			NeuroDiffEq			Nangs		
	Iter	Time	MSE	Iter	Time	MSE	Iter	Time	MSE
50	2000	01:37	0.0056	30	05:03	0.0087	100	05:48	0.0873
75	2000	03:09	0.0024		12:22	0.0069		07:40	0.0684
100	2000	05:43	0.0021		19:38	0.0022		09:49	0.0418
150	1000	06:22	0.0083		47:52	0.0021		14:23	0.0147
200	500	06:37	0.0453		01:29:15	0.0014		22:59	0.0018



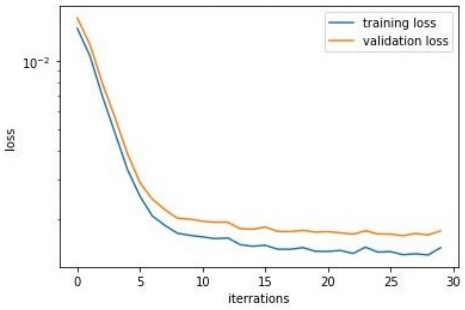
(a) PyDEns algorithm



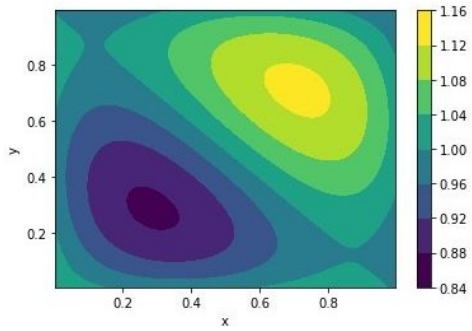
(b) MSE values of PyDEns algorithm



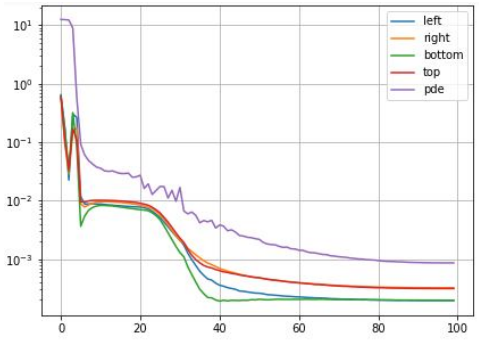
(c) NeuroDiffEq algorithm



(d) MSE values of NeuroDiffEq algorithm



(e) Nangs algorithm



(f) MSE values of Nangs algorithm

Figure 7. The three methods performances in solving Poisson equations, left : the approximate solutions visualization, right : the behaviour of the MSE

We can also solving PDE equation when there are no analytical solutions like solving Poisson equations. In table 3, PyDEns still showing us better performance when solving lower dimensions up to 100 dimensions. It was less accurate in solving the higher dimensions. NeuroDiffEq and Nangs, consistently performed well when solving higher dimenional problems such a 150 and 200 dimensions. As can be seen in Table 3, the MSE values of PyDEns, NeuroDiffEq, and Nangs for solving 50 dimensional problems were respectively 0.0056, 0.0087, and 0.0873, whereas when solving 200 dimensional problems, the MSE values were respectively 0.0453, 0.0014, and 0.0018. In the terms of computational times, differently, PyDEns consumed the least computational times with 6.37 minutes only for solving 200 dimensions, compared to NeuroDiffEq and Nangs which took 1.29 hours and 22.59 minutes for solving the sam problems.

6. Advantages and disadvantages of PyDEns, NeuroDiffEq, and Nangs

Based on the results explained in the previous section, we can see that in general, each method has its own advantages and disadvantages in the different situations. In terms of accuracy performance, Nangs method consistently produced the lowest MSE compared with PyDEns and NeuroDiffEq. Similar trend to the computational times, Nangs method is the fastest one compared with PyDEns and NeuroDiffEq, although in the case of Poisson equation, PyDEns is the fastest one. NeuroDiffEq method potentially produced a small MSE for solving high dimensional problems, however, it was very costly. PyDEns on the other hand, was only well performed when solving low dimensional problems.

There are also some weaknesses of the three methods, PyDEns gave a NaN result if we enforced it to solve high-dimensional PDE problems. Meanwhile, NeuroDiffEq as mentioned before, is very ineffective method for solving any dimensional PDEs. Lastly, Nangs method is very recommended to be modified to perform better. What we thought regarding Nangs method was by changing the optimizer, and adding the number of iterations, Nangs method would be the best method for solving high dimensional PDE problems.

7. Conclusion

We have discussed PyDEns, NeuroDiffEq, and Nangs methods for solving the variations of PDE problems, include heat equation, wave equation, and Poisson equation, with ranging dimensions which are from 50 to 200. These methods allows us to train ANNs to approximately solve the PDEs. We compared them in terms of the accuracy and the efficiency. As a results, both NeuroDiffEq and Nangs showed a well performance for solving the higher dimensional PDE problems, whereas PyDEns showed the least performance. In fact, PyDEns potentially produced a NaN of MSE when solving more than 100 dimensions. To overcome this issue, we should cut down the number of iterations, however it would affect in obtaining the large of MSE. Thus, we highly recommended to use PyDEns if we only solved lower dimensions, compared to use NeuroDiffEq which is more costly for the same problems. We also recommend to change the ANNs architecture to maximize the model performance, for instance to change the number of layers, neurons, learning rate, method for picking a sample inside domains, and even change the optimization method.

Author Contributions: Conceptualization, D.A.P and M.A.B.; methodology, D.A.P.; software, D.A.P.; validation, M.A.B, M.M.1 and M.M.2; formal analysis, M.A.B; investigation, D.A.P; resources, M.M.1; writing—original draft preparation, D.A.P.; writing—review and editing, M.A.B and M.M.2; visualization, D.A.P.; supervision, M.M.1; project administration, M.M.1 and M.M.2; funding acquisition, M.M.1. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Ministry of Higher Education Malaysia, grant number FRGS/1/2018/ICT04/UMT/02/1

Acknowledgments: This review paper is supported financially by the Universiti Malaysia Terengganu

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MDPI	Multidisciplinary Digital Publishing Institute
DOAJ	Directory of open access journals
TLA	Three letter acronym
LD	Linear dichroism

References

1. Gockenbach, M.S. *Partial differential equations: analytical and numerical methods*; Vol. 122, Siam, 2005.
2. Evans, G.; Blackledge, J.; Yardley, P. *Numerical methods for partial differential equations*; Springer Science & Business Media, 2012.
3. LeVeque, R.J.; Leveque, R.J. *Numerical methods for conservation laws*; Vol. 3, Springer, 1992.
4. Bakar, M.A.; Juliansyah, A.; Thalib, R.; Pratama, D.A. High Performances of Stabilized Lanczos-Types for Solving High Dimension Problems: A Survey. *Computer Science* **2021**, *16*, 837–854.
5. Press, W.H.; Press, W.H.; Flannery, B.P.; Teukolsky, S.A.; Vetterling, W.T.; Flannery, B.P.; Vetterling, W.T. *Numerical recipes in Pascal: the art of scientific computing*; Vol. 1, Cambridge university press, 1989.
6. Kunz, K.S.; Luebbers, R.J. *The finite difference time domain method for electromagnetics*; CRC press, 1993.
7. Hayati, M.; Karami, B. Feedforward neural network for solving partial differential equations. *Journal of Applied Sciences* **2007**, *7*, 2812–2817.
8. Lu, L.; Meng, X.; Mao, Z.; Karniadakis, G.E. DeepXDE: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502* **2019**.
9. Géron, A. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*; O'Reilly Media, 2019.
10. Purnama, B. *Pengantar Machine Learning: Konsep dan Praktikum dengan Contoh Latihan Berbasis R dan Python*; Penerbit Informatika, 2019.
11. Hornik, K.; Stinchcombe, M.; White, H.; others. Multilayer feedforward networks are universal approximators. *Neural networks* **1989**, *2*, 359–366.
12. Lee, H.; Kang, I.S. Neural algorithm for solving differential equations. *Journal of Computational Physics* **1990**, *91*, 110–131.
13. Meade Jr, A.J.; Fernandez, A.A. Solution of nonlinear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling* **1994**, *20*, 19–44.
14. Lagaris, I.E.; Likas, A.; Fotiadis, D.I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks* **1998**, *9*, 987–1000.
15. Malek, A.; Beidokhti, R.S. Numerical solution for high order differential equations using a hybrid neural network-optimization method. *Applied Mathematics and Computation* **2006**, *183*, 260–271.
16. Hussian, E.A.; Suhhiem, M.H. Numerical solution of partial differential equations by using modified artificial neural network. *Network and Complex Systems* **2015**, *5*, 11–21.
17. Sirignano, J.; Spiliopoulos, K. DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics* **2018**, *375*, 1339–1364.
18. Al-Aradi, A.; Correia, A.; Naiff, D.; Jardim, G.; Saporito, Y. Solving nonlinear and high-dimensional partial differential equations via deep learning. *arXiv preprint arXiv:1811.08782* **2018**.
19. Guo, Y.; Cao, X.; Liu, B.; Gao, M. Solving partial differential equations using deep learning and physical constraints. *Applied Sciences* **2020**, *10*, 5917.
20. Chen, F.; Sondak, D.; Protopapas, P.; Mattheakis, M.; Liu, S.; Agarwal, D.; Di Giovanni, M. NeuroDiffEq: A Python package for solving differential equations with neural networks. *Journal of Open Source Software* **2020**, *5*, 1931.
21. Rahaman, N.; Baratin, A.; Arpit, D.; Draxler, F.; Lin, M.; Hamprecht, F.; Bengio, Y.; Courville, A. On the spectral bias of neural networks. International Conference on Machine Learning. PMLR, 2019, pp. 5301–5310.
22. Poggio, T.; Mhaskar, H.; Rosasco, L.; Miranda, B.; Liao, Q. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing* **2017**, *14*, 503–519.
23. Pedro, J.B.; Maroñas, J.; Paredes, R. Solving Partial Differential Equations with Neural Networks. *arXiv preprint arXiv:1912.04737* **2019**.
24. Koryagin, A.; Khudorozkov, R.; Tsimfer, S. PyDEns: A python framework for solving differential equations with neural networks. *arXiv preprint arXiv:1909.11544* **2019**.
25. Mabbutt, S.; Picton, P.; Shaw, P.; Black, S. Review of Artificial Neural Networks (ANN) applied to corrosion monitoring. Journal of Physics: Conference Series. IOP Publishing, 2012, Vol. 364, p. 012114.
26. McFall, K.S. An artificial neural network method for solving boundary value problems with arbitrary irregular boundaries. PhD thesis, Georgia Institute of Technology, 2006.
27. Burden, R.; Faires, J. Numerical Analysis, 9th International Edition. *Brooks/Cole, Cencag Learning* **2011**.