International Journal of Intelligent Communication and Computer Science
Vol. 1, No. 2, 2023, Pages 54-68.

Deep Neural Networks for Solving Ordinary Differential Equations: A Comprehensive Review

Sourabh Kumar Dubey[1], Hibah Islahi[2] and Raghvendra Singh[3]

Review Paper

[1]Research Scholar, Manglayatan University, Aligarh, U.P., INDIA
[2] Institute of Applied Sciences, Manglayatan University, Aligarh, U.P., INDIA
[3] School of Sciences (Mathematics), Uttar Pradesh Rajarshi Tandon Open University, Prayagraj, U.P., INDIA

Email: sourabh.dubey27@gmail.com
Received: 28 Oct 2023, Revised: 31 Dec 2023 Accepted: 14 Jan 2024

Abstract:

The integration of Deep Neural Networks (DNNs) in the numerical solution of Ordinary Differential Equations (ODEs) has emerged as a powerful and versatile paradigm, revolutionizing the landscape of scientific computing. This paper presents an in-depth review of the current state-of-the-art methodologies and breakthroughs that harness the capabilities of DNNs for tackling the challenges posed by ODEs. We delve into diverse aspects, covering a spectrum of architectures, ranging from traditional feedforward networks to more sophisticated recurrent and convolutional architectures. The review explores key training strategies employed to enhance the efficacy of DNNs in approximating solutions to differential equations. Furthermore, we highlight the wide array of applications where these DNN-based approaches have demonstrated considerable success, ranging from physics simulations to dynamic systems modeling. The paper also addresses the interpretability and generalization challenges that come with integrating DNNs in the context of ODEs. In addition to summarizing the existing literature, this review endeavors to provide a critical analysis of the current state of research, identifying gaps and potential areas for improvement. The discussion extends to the challenges faced by researchers in optimizing DNNs for solving ODEs and proposes avenues for future investigations. By presenting a comprehensive overview of the field, this paper serves as an invaluable resource for researchers and practitioners seeking to leverage the potential of DNNs in the numerical solution of differential equations.

Keywords: ODE, DNN, ANN, CNN

1. Introduction

Differential equations represent a foundational and indispensable concept in the realm of mathematics, serving as a fundamental bridge between functions and their derivatives. These equations stand as a powerful mathematical tool for expressing and understanding the intricate relationships that underpin numerous phenomena encountered in the realms of science and engineering. The versatility of differential equations has made them a cornerstone in a multitude of disciplines, including physics, biology, economics, and engineering, offering valuable insights and predictive capabilities [1].
At its core, a differential equation can be defined as an equation that involves not only an unknown function but also one or more of its derivatives. This inclusion of derivatives in the equation transforms it into a dynamic and expressive tool for capturing how a function changes in response to various factors or conditions. The primary objective when dealing with a differential equation is to seek a function, often referred to as the "solution" or "satisfying function," that precisely adheres to the equation's specifications.

This pursuit involves solving for the unknown function in terms of its derivatives, thereby unraveling the underlying behavior and relationships that the function exhibits.

Due to their extraordinary capacities to solve a wide range of engineering problems, DNNs have attracted a lot of attention from the engineering community. Particularly, there is tremendous promise for using DNNs to solve systems of ODEs, which are used to describe a variety of physical processes. While it is possible to solve such systems of ODEs using time-tested traditional numerical techniques, each approach has its own set of positive and negative aspects, including considerations for precision, stability, convergence, and computational efficiency.

The fourth-order Runge-Kutta method (RK4) stands out as a popular option among these well-established techniques. In situations when the stiffness of the ODE system is not the primary issue, RK4 is particularly well-suited because it is known for its effectiveness in managing non-stiff problems. This technique belongs to the class of numerical methods known as finite difference techniques, which are recognized for their adaptability in approximating solutions to ODEs.

From 1940s, an alternative computational methodology known as artificial neural networks (ANNs) has been used. Because of this advancement in technology, ANNs are now powerful tools in a variety of fields. An artificial neural network is essentially a computational system created to demonstrate specific performance characteristics similar to biological neural networks. These networks aim to imitate how the human brain functions. The input layer, hidden layers, and output layer are the three basic layers that make up an ANN's architectural blueprint. Interconnected neurons or units make up each of these layers, and each neuron processes and transmits information. When an ANN has more than one hidden layer, it is regarded as a DNN. A DNN's ability to handle hard tasks and greater complexity are both facilitated by the presence of numerous hidden layers, which makes it an essential advancement in the fields of artificial intelligence and machine learning.

2. Introduction to Differential Equations

Differential equations can be classified into several types based on their characteristics, including:

Ordinary Differential Equations (ODEs): These equations involve a single independent variable and one or more derivatives of an unknown function with respect to that variable. They are often used to model processes that vary with a single independent variable, such as time. The general form of an ODE is [2]:

$$F(x, y, y', y'', ..., y(n)) = 0 \tag{1}$$

where y is the unknown function, y′ is its first derivative, y′′ is its second derivative, and so on, up to the nth derivative.

Partial Differential Equations (PDEs): These equations involve multiple independent variables and derivatives of an unknown function with respect to those variables. PDEs are commonly used to describe physical phenomena that depend on more than one variable, such as heat conduction, fluid flow, and wave propagation. The general form of a PDE is [3]:

$$F(x_1, x_2, ..., x_n, y, \frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, ..., \frac{\partial^2 y}{\partial x_1 \partial x_2}, ...) = 0 \tag{2}$$

where y is the unknown function, and $\frac{\partial y}{\partial x_i}$ represents partial derivatives with respect to each independent variable x_i.

Differential equations are essential for modeling real-world phenomena because they capture how rates of change in a system are related to the current state of the system. Solving differential equations can help us predict future behavior, optimize processes, and understand the underlying dynamics of various systems.

The solutions to differential equations can take various forms, depending on the equation's complexity and characteristics. These solutions can be expressed as explicit functions, parametric curves, or as numerical approximations using computational methods when exact solutions are not readily available.

2.1 Classification of Differential Equations

A differential equation is an equation that includes differential coefficients or differentials as one of its two main variables [3]. An equation is considered to be ordinary differential when all of the differential coefficients are defined in terms of a single independent variable. An equation is said to be partial differential if it contains two or more independent variables and partial differential coefficients with respect to any one of those variables.

Formulation of a differential equation:
In order to remove a particular arbitrary constant from a relation including variables and constants, an ordinary differential equation is formulated as a mathematical tool. Either the elimination of arbitrary constants or arbitrary functions can result in the formation of a partial differential equation.

Solution of differential equation:
In the realm of differential equations, the fundamental objective is to establish a relationship between variables that satisfies the specific conditions laid out by a given differential equation. When such a relationship is identified, it is acknowledged as a solution to the differential equation. These solutions are the keys to unveiling the intricate behaviors and patterns embedded within various phenomena across science and engineering.
During the process of solving a differential equation, there are distinct categories of solutions to consider. The general solution, often referred to as the complete solution, stands out as one of the essential concepts. This solution embodies a form where the number of arbitrary constants matches the order of the differential equation. These arbitrary constants introduce flexibility into the solution, representing the yet-to-be-determined parameters that can adapt to different scenarios. On the other hand, specific solutions emerge when these arbitrary constants are assigned precise, concrete values. By doing so, the general solution transforms into a particular solution tailored to a specific situation or set of conditions. This process of assigning values to the constants yields solutions that directly apply to real-world problems, making them practical and useful. However, the world of differential equations presents intriguing complexities. In certain scenarios, a differential equation may harbor an additional solution that defies derivation from the general solution by simply assigning specific values to the arbitrary constants. This enigmatic solution remains elusive, standing apart from the general and specific solutions. This unique solution, one that cannot be accessed through conventional means, adds depth and mystery to the realm of mathematical modeling. It is referred to as a unique solution due to its distinctiveness and inaccessibility through standard parameter assignment.

2.2 Traditional Method for the Solution of Differential Equations

The traditional methods for solving differential equations vary depending on the type of differential equation (ordinary or partial), its order, linearity, and other characteristics. Here are some of the traditional methods commonly used for solving differential equations [3]:

2.2.1 Methods For the Solution of ODEs:

Separation of Variables: This method is used for first-order ODEs that can be written in the form $\frac{dy}{dx} = g(x)h(y)$. By rearranging the terms and integrating both sides, you can solve for y in terms of x.

Exact Differential Equations: If an ODE can be written in the form $M(x,y)dx + N(x,y)dy = 0$ where $\frac{\partial M}{\partial y} = \frac{\partial N}{\partial x}$ it's called an exact differential equation. You can find a potential function (also known as a primitive or integrating factor) and use it to find the solution.

Homogeneous Differential Equations: In these equations, you can make a substitution to reduce the equation to a separable form. For example, if $\frac{dy}{dx} = F\left(\frac{y}{x}\right)$ you can substitute $u = \frac{y}{x}$ and reduce it to a separable form.

Method of Integrating Factors: For linear first-order ODEs, you can multiply both sides by an integrating factor that depends only on x or y to make the equation exact and then solve it.

Laplace Transforms: Laplace transforms are useful for solving linear ODEs with constant coefficients. By taking the Laplace transform of the equation, solving for the transformed variable, and then taking the inverse Laplace transform, you can find the solution.

2.2.2 For Partial Differential Equations (PDEs):

Method of Separation of Variables: This technique is used for solving linear PDEs with boundary or initial conditions. It involves assuming that the solution can be expressed as a product of functions, each of which depends on only one independent variable. This method is particularly effective for problems with certain geometries and boundary conditions.

Method of Characteristics: This method is used for first-order PDEs. It involves finding characteristic curves along which the PDE simplifies to an ODE. Solving the ODE along the characteristic curves can lead to a solution for the original PDE.

Method of Eigenfunction Expansion: This method is commonly used for linear, homogeneous PDEs. It involves expanding the solution in terms of a set of eigenfunctions of the differential operator in the PDE and determining the coefficients of the expansion using boundary or initial conditions.

Finite Difference Methods: These are numerical methods used to approximate solutions to PDEs by discretizing the domain into a grid and approximating derivatives using finite differences. Finite difference methods are particularly useful for solving PDEs on a computer.

Finite Element Methods: These are numerical techniques used to solve PDEs by dividing the domain into smaller subdomains (elements) and approximating the solution within each element using piecewise functions. Finite element methods are widely used in structural mechanics, heat transfer, and fluid dynamics.

2.3 Numerical Method for the Solution of Differential Equations

Numerical methods are widely used for solving differential equations when analytical solutions are difficult or impossible to obtain. These methods involve approximating the solution at discrete points in the domain of interest [4]. Here are some common numerical methods for solving differential equations:

Euler's Method: Euler's method is a simple and straightforward numerical method for solving ODEs [5]. It approximates the derivative by a finite difference and uses it to update the solution at each step. Euler's method is easy to implement but may require small step sizes for accurate results.

Runge-Kutta Methods: Runge-Kutta methods are a family of numerical techniques for solving ODEs. The most commonly used is the fourth-order Runge-Kutta method (RK4), which provides better accuracy than Euler's method [6]. It involves four intermediate steps at each time interval, leading to a more accurate approximation of the solution.

Finite Difference Methods: Finite difference methods are used to discretize both ODEs and PDEs [7]. These methods involve dividing the domain into a grid of points and approximating derivatives using finite differences. Explicit, implicit, and Crank-Nicolson schemes are commonly used finite difference methods for time-dependent problems.

Finite Element Methods (FEM): FEM is a numerical technique commonly used for solving PDEs [8]. It involves dividing the domain into smaller elements and approximating the solution within each element using piecewise functions. FEM is versatile and can handle complex geometries and boundary conditions.

Boundary Element Methods (BEM): BEM is a numerical method often used for solving boundary value problems in potential theory, such as Laplace's equation [9]. It focuses on approximating the solution on the boundary of the domain and is particularly useful for problems with open boundaries.

2.4 Limitations of the traditional methods

Traditional methods for solving ODEs have been widely used for many years, but they come with several limitations:

Limited Applicability: Traditional ODE solvers are often designed for specific types of ODEs, such as linear or moderately nonlinear equations. They may struggle to handle highly nonlinear, stiff, or chaotic ODEs effectively.

Stiffness Handling: Stiff ODEs, where certain components change much more rapidly than others, can be challenging for traditional methods. These methods may require very small time step sizes to maintain stability, leading to slow convergence and increased computational cost.

Discretization Errors: Many traditional methods rely on discretizing the ODE problem, which involves breaking the time domain into discrete time steps. This discretization can introduce errors, especially when dealing with irregular or rapidly changing solutions.

Fixed Time Steps: Most traditional ODE solvers use fixed time steps, which can be inefficient for problems where the solution changes slowly in some regions and rapidly in others. Adapting the time step dynamically based on the solution's behavior can be more efficient but is not a feature of most traditional methods.

Boundary and Initial Value Constraints: Traditional methods might struggle to handle complex boundary conditions or initial value constraints, especially when these conditions are discontinuous or irregular.

High-Dimensional Problems: Traditional methods face computational challenges when applied to high-dimensional ODE systems. The curse of dimensionality can lead to a rapid increase in computational cost as the number of dimensions grows.

Numerical Stability Issues: Some traditional methods may suffer from numerical stability issues, especially when dealing with very small or very large numbers, which can lead to inaccurate results.
While traditional ODE solvers have their place and are suitable for many problems, these limitations have led to the development of alternative approaches, such as numerical methods based on neural networks or machine learning, which can sometimes overcome these challenges and offer more flexibility and accuracy in solving complex ODEs.

3. Artificial Neural Network (ANN) Structure and Solution of ODE

An ANN has one hidden layers stacked between the input and output layers. The schematic diagram of ANN structure is shown in Figure 1.

Input Layer: The input layer is the initial layer of an ANN. It receives the raw input data, which can be features extracted from real-world data, such as images, text, or numerical values. Each neuron in this layer represents one feature.

Hidden Layers: Between the input and output layers, ANNs can have one or more hidden layers. These layers are composed of interconnected neurons. The term "hidden" refers to the fact that they are not directly connected to the external world (input or output).

Output Layer: The output layer is the final layer of the network, responsible for producing the network's predictions or classifications. The number of neurons in this layer depends on the specific problem; for example, in a binary classification task, there might be one neuron for each class.

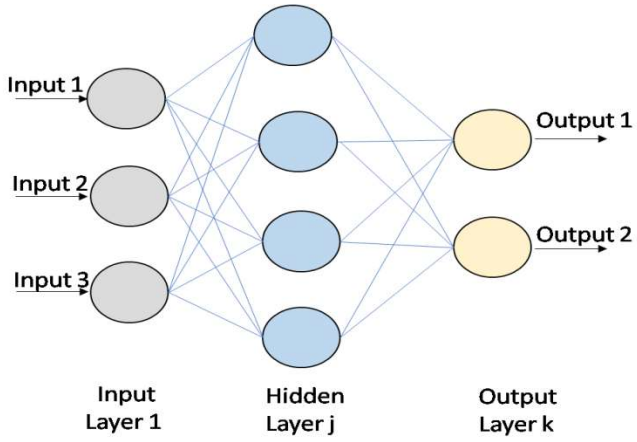

Figure 1: Schematic of ANN architecture

Connections (Edges): Each connection (edge) between neurons has an associated weight. These weights are learned during the training process and determine the strength of the connection between neurons. The weighted sum of inputs is computed for each neuron in the hidden and output layers.

Activation Functions: In the intricate landscape of neural networks, each neuron plays a pivotal role in processing information. One of the fundamental operations that a neuron performs is the application of an activation function to the weighted sum of its inputs. This seemingly simple step is, in fact, a crucial aspect of what makes neural networks powerful and adaptable to a wide array of tasks. The activation function serves as the neuron's decision-maker, determining whether it should "fire" or activate based on the information it receives. The weighted sum of inputs represents the neuron's level of excitation, akin to assessing how strongly it responds to the incoming data. There are several common activation functions in the neural network toolbox, each with its unique characteristics and advantages. One such function is the sigmoid function, which maps the neuron's excitation to an output between 0 and 1. This function is particularly useful in scenarios where the neuron's output needs to represent probabilities or when the problem exhibits a logistic or S-shaped behavior. Another widely used activation function is the hyperbolic tangent (tanh) function. It shares similarities with the sigmoid function but maps the output to a range between -1 and 1. This can be valuable in situations where data has both positive and negative values, allowing the neuron to capture more nuanced relationships. In recent years, the rectified linear unit (ReLU) function has gained immense popularity. This activation function simply outputs the input if it's positive and zero otherwise. ReLU brings a level of simplicity and efficiency to neural networks and has been shown to be particularly effective in deep learning architectures. What's remarkable about activation functions is that they introduce non-linearity into the neural network model. Without non-linearity, a neural network would be limited to approximating linear functions, severely restricting its capacity to model complex, real-world relationships in data. Activation functions enable neural networks to transcend linearity, allowing them to approximate highly intricate and non-linear functions, making them adaptable to a vast range of tasks, from image recognition to natural language processing. In essence, activation functions are the heart of what transforms neural networks into powerful function approximators. They imbue these models with the capacity to capture and represent the intricate patterns, nuances, and complexities of the data they are designed to process, making them a cornerstone of modern machine learning and artificial intelligence.

Biases: Neurons also have associated bias terms, which are constants added to the weighted sum before applying the activation function. Biases allow the network to shift and adjust its output.

3.1 Fitting Function by ANN

The universal approximation theorem [10], a fundamental result in the field of artificial neural networks (ANNs), asserts that it is possible to approximate any continuous function with a feed-forward neural network having just a single hidden layer. This theorem underscores the remarkable approximation capabilities of ANNs and their potential to model complex relationships in data. Mathematically, this neural network with a single hidden layer can be expressed in a matrix multiplication form, which is a concise way to represent the computations taking place within the network. This matrix multiplication form typically involves three sets of parameters:

$$F(x,w) = w_2\sigma(w_1x + b_1) + b_2 \tag{3}$$

where w_1 and w_2 are weights and b_1 and b_2 are bias. The optimum values of weights are obtained as

$$L(w) = \int_a^b \left[y(x) - F(x,w)\right]^2 dx \tag{4}$$

In discrete form

$$L(w) = \sum_i \left[y(x_i) - F(x_i,w)\right]^2 \tag{5}$$

where x_i belongs to [a,b].

If the error is sufficiently low, the ANN can be regarded as an effective approximation of the original function within the [a, b] domain.

$$y(x) \approx F(x,w) \tag{6}$$

3.2 Solution a single ordinary differential equation (ODE) using artificial neural networks

Next, we explore the creation of an ANN capable of approximating the solution for a first-order ODE.

$$y'(t) = F_1(y(t),t) \qquad y(t_0) = y_0 \tag{7}$$

The ANN solution can be written as

$$F(x,w) = w_2\sigma(w_1x + b_1) + b_2 \tag{8}$$

However, the above equation will not satisfy the initial condition $F(t_0,w) \neq y_0$.

We can incorporate initial condition as

$$\hat{y}(t,w) = y_0 + (t - t_0)F(t,w) \tag{9}$$

There will be some 'w' for which $\hat{y}(t_0,w) = y_0$.

We further have

$$\hat{y}'(t,w) \approx F_1(\hat{y}(t,w),t) \tag{10}$$

Defining the derivative as

$$\hat{y}'(t,w) = \frac{\partial\left[y_0 + (t-t_0)F(t,w)\right]}{\partial t} = (t-t_0)\frac{\partial F(t,w)}{\partial t} + F(t,w)\frac{\partial(t-t_0)}{\partial t} \tag{11}$$

The optimum values of weights are obtained as

$$L(w) = \int_{t_0}^{t_1} \left[\hat{y}'(t,w) - F_1(\hat{y}(t,w),t)\right]^2 dt \tag{12}$$

In discrete form

$$L(w) = \sum_i \left[\hat{y}'(t_i,w) - F_1(\hat{y}(t_i,w),t_i)\right]^2 \tag{13}$$

If the error is sufficiently low, the ANN can be regarded as an effective approximation of the original function within the [t_0, t_1] domain.

$$\hat{y}'(t,w) \approx y(t) \tag{14}$$

4. Deep Neural Network (DNN) Structure and Solution of ODE

A DNN is a specific type of ANN with a deep architecture, meaning it has multiple hidden layers stacked between the input and output layers. The key distinction of a DNN is its depth. The schematic diagram of DNN structure is shown in Figure 2. Here's an overview of a DNN's structure:

Input Layer: Similar to ANNs, the input layer receives raw data and consists of neurons corresponding to input features.

Multiple Hidden Layers: DNNs have two or more hidden layers. The number of layers can vary significantly, and this depth enables them to capture hierarchical features and representations in the data.

Output Layer: Like ANNs, DNNs have an output layer responsible for producing predictions or classifications. The number of neurons in the output layer depends on the specific task.

Connections (Edges), Activation Functions, and Biases: These elements operate similarly to ANNs. Each connection has a weight, and neurons apply activation functions to their inputs, incorporating biases to adjust the output.

Feature Hierarchy: The depth of DNNs allows them to automatically learn and represent complex features and patterns in data. Lower layers typically learn basic features (e.g., edges in an image), while higher layers combine these to form more abstract features (e.g., shapes or objects).

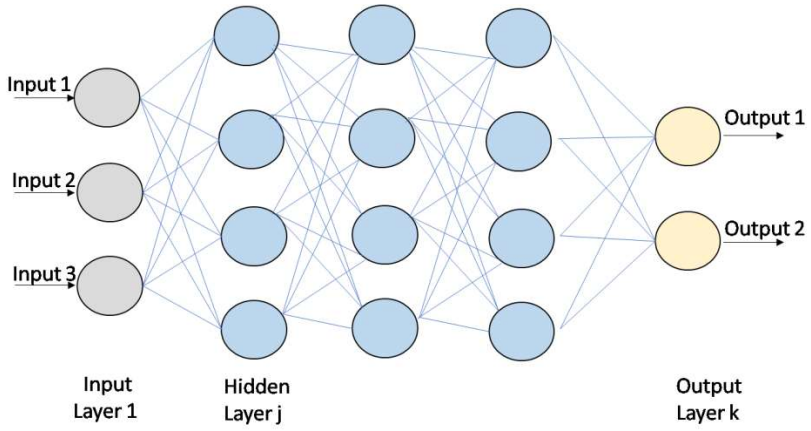

Figure 2: Schematic of DNN architecture

In summary, both ANNs and DNNs consist of layers of interconnected neurons, but DNNs distinguish themselves by having a more extensive hierarchy of hidden layers, which makes them capable of learning intricate patterns and representations in data, leading to their effectiveness in solving complex tasks in machine learning and artificial intelligence.

4.1 Deep Neural Method for the Solution of Differential Equations

Solving ODEs using DNNs involves training a neural network to approximate the solution of the ODE. This approach is often referred to as "Neural ODEs." Here's a basic outline of how you can solve ODEs using DNNs:

Formulate the ODE Problem: Define the ODE problem you want to solve. This includes specifying the ODE itself, initial conditions, and any boundary conditions if applicable. The general form of a first-order ODE is:

$$\frac{dy}{dt} = F(t, y) \tag{15}$$

Data Preparation: Generate a dataset of input-output pairs that can be used to train the neural network. For ODEs, this typically involves sampling values of the independent variable (e.g., time t) and calculating corresponding values of the dependent variable (e.g., y) using the known ODE. You'll need both the initial conditions and a set of time points for which you want to approximate y(t).

Design the Neural Network: Create an DNN architecture that can approximate the solution of the ODE. For solving ODEs, neural networks called "ODE solvers" are often used. The most common architecture is based on the idea of a continuous neural network that evolves along with the ODE. The network has a continuous depth that corresponds to the time variable tt. This is achieved using techniques like residual networks.

Define Loss Function: Define a loss function that measures the error between the predicted solution of the ODE by the neural network and the true solution obtained from the ODE itself. A common loss function is the mean squared error (MSE).

Training the Neural Network: Train the neural network using your dataset. The training process involves adjusting the network's parameters (weights and biases) to minimize the loss function. Techniques like gradient descent or its variants are typically used for optimization.

Prediction: After training, you can use the trained DNN to predict the solution of the ODE at any point in the domain of interest. You can evaluate the network for a range of time values to approximate the solution y(t).

4.2 Deep neural network for system of ODEs

We contemplate a densely connected network comprising multiple layers. Within this network, there exists a single neuron in the input layer representing the independent variable of the system of ODEs. Meanwhile, the output layer consists of several neurons (n), each corresponding to one of the unknown variables.
To train this Deep Neural Network (DNN), we select M sample points from the domain and assemble them into a matrix, denoted as $X = \left[t^{(1)}, \ldots . t^{(m)} \right] \in \mathbb{R}^{1 \times m}$. Each row of this matrix represents a sample point or training example. Additionally, we represent the output of the network with a matrix denoted as Y. For a specific example, $N_k \left(t^{(i)}, P_k \right)$ signifies the output of the j[th] unknown corresponding to the i[th] sample point, where '*j*' represents the associated parameters, weights, and bias.

$$\hat{y}_j \left(t, P_j \right) = a_j + \left(t - a \right) N_j \left(t, P_j \right), \qquad j = 1, \ldots . . n \, . \tag{16}$$

where P_k stands for the corresponding parameters

We establish a trial solution, denoted as " $\hat{y}$ " with equation (13). This trial solution in Equation (3) adheres to the initial conditions of the problem. We proceed to train the network with the objective of minimizing the total cost function defined in Equation (14). In this context, the cost function seeks to converge to zero, converges to zero. Here,

$$J = \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\frac{d\hat{y}_j}{dt} - f_j \right)^2 \tag{17}$$

where,

$$f_j = f_j \left(t^{(i)}, \hat{y}_j \left(t^{(i)}, P_j \right) \right) .$$

5. Literature Review

Danang et.al [11] the numerical resolution of an ordinary differential equation (ODE) fundamentally characterizes any particular physical phenomenon. Historically, ODEs have been tackled through a discretization procedure utilizing the classical finite difference approximation [11]. In more recent times, there has been a shift towards employing DNNs for approximating ODE solutions directly, albeit with the caveat that DNNs cannot operate independently without optimization. In our research endeavors, we embark on a quest for innovation by introducing a novel and synergistic approach that marries the power of DNNs with the optimization prowess of a Genetic Algorithm (GA). This fusion of two distinct computational paradigms forms the foundation of our research, offering a fresh perspective on solving complex problems. The Genetic Algorithm, a bio-inspired optimization technique, takes center stage in our methodology. It serves as a vehicle for initializing a population of trainable parameters within the DNN framework. This ingenious step streamlines the training process and enhances its efficiency, all within a single generation of the Genetic Algorithm. This integration enables us to harness the intrinsic capabilities of DNNs to model and adapt to intricate data patterns, while the Genetic Algorithm orchestrates the optimization of these parameters. Our primary research focus revolves around harnessing the remarkable potential of this DNN-GA hybrid solution to tackle a broad spectrum of problems, encompassing both linear and nonlinear ODEs. These equations encapsulate critical phenomena across numerous domains, including physics, engineering, and biology. By effectively addressing ODEs, our approach opens doors to breakthroughs and advancements in these fields.

Dufera, Tamirat Temesgen [12] the primary focus of this paper is to utilize deep artificial neural networks for the purpose of solving systems of ordinary differential equations. We have devised a vectorized algorithm and translated it into Python code for implementation [12]. Our study encompasses a series of experiments aimed at determining the optimal neural architecture. To train the neural network effectively, we have harnessed the adaptive moment minimization method. To gauge the performance of our approach, we have conducted a comparative analysis with a traditional numerical method, specifically the fourth-order Runge-Kutta method. Our findings demonstrate that the artificial neural network exhibits superior accuracy, particularly when dealing with smaller grid point configurations.

Shubham and Vishal [13] conducted research to explore the resolution of second-order differential equations through the application of various approximation techniques, including the finite difference method [13]. Furthermore, we conducted a comparative analysis between the solutions obtained through these approaches and the exact solution of the second-order differential equation. The findings reveal that the Finite Difference Method emerges as the approach yielding the most accurate solution for the differential equation. It can be deduced that the solutions derived from the Galerkin Method and the Rayleigh-Ritz Method exhibit similarities, indicating a high degree of concurrence between the Galerkin approach and the Rayleigh-Ritz method.

G. O. Akinlabi et al [14] conducted a study on Numerical approximation of second-order boundary value problems via hybrid boundary value method [14]. In this study they have applied the Hybrid Boundary Value Method (HyBVM) to solve two second order BVPs with boundary conditions, comparing the results to other BVMs in the literature, as well as the maximum error and efficiency. The Numerov method was used to build these methods and data from both on-step and off-step points were used.

Bakirova, E. A., et al [15] investigates the convergence, stability, and accuracy of a suggested numerical approximation approach for solving the boundary value problem of a parameterized integro-differential equation. A parameterized loaded differential equation is used to approximate the integro-differential equation [15]. Parameterized loaded differential equation receives a new general solution, and its characteristics are discussed. It is shown that the general solution to the loaded differential equation with a parameter may be written as a set of linear algebraic equations with respect to arbitrary vectors, and that these equations are solvable. Solution of the Cauchy problems for ODE yields the system's coefficients and right-hand sides. The boundary value problem for the parameterized loaded differential equation is considered, and several algorithms are presented for its solution. A connection is made between the qualitative features of the original problem and the approximation, and differences between the two are estimated.

Jajarmi, and Dumitru [16] conducted a study on New Iterative Method for the Numerical Solution of High-Order Non-linear Fractional Boundary Value Problems [16]. They devise a productive numerical method for solving a class of nonlinear fractional boundary value problems (BVPs). The approach that has been suggested does not include any perturbation, discretization, linearization, or other restrictive assumptions. Instead, it generates an exact solution in the form of a series that converges in a consistent manner. In addition, the exact solution can be found by simply solving a series of linear BVPs of fractional order. As a result, the approach that was presented is both effective and simple to put into practice from a pragmatic point of view. They offer an iterative technique that is both efficient in terms of computation and capable of producing an approximation of the solution that is sufficiently accurate.

Anitescu et. al. ([17] introduced an approach to address PDE by employing artificial neural networks alongside an adaptable collocation technique. In this process, we initiate training with a sparser grid of points and gradually incorporate additional points in subsequent stages, guided by the residual's magnitude at a broader array of evaluation points [17]. This methodology enhances the neural network's robustness in approximating solutions and can yield substantial computational efficiencies, especially in cases where the solution exhibits non-smooth behavior. We provide numerical outcomes for standard problems involving scalar-valued PDEs, including the Poisson and Helmholtz equations, as well as an inverse acoustics problem.

Amin, and Meidani [18] developed efficient numerical methods to solve high-dimensional random PDEs has proven to be a formidable challenge, primarily due to the widely recognized curse of dimensionality [18]. In response to this challenge, we introduce a novel solution framework that leverages deep learning techniques. More specifically, our approach involves approximating the random PDE by employing a feed-forward fully-connected deep residual network. This network can enforce initial and boundary constraints, either strongly or weakly, as required. What sets this framework apart is its mesh-free nature, enabling it to handle irregular computational domains effectively. The parameters of the approximating deep neural network are determined through an iterative process, utilizing various adaptations of the Stochastic Gradient Descent (SGD) algorithm. To validate the effectiveness of our proposed frameworks, we conducted numerical experiments on diffusion and heat conduction problems. These experiments showcased the remarkable accuracy of our approach, comparing favorably to the results obtained through Monte Carlo-based finite element methods that have converged.

Biala T. A. et. al [19] studied on the effectiveness of Boundary Value Methods (BVMs) on second-order PDEs [19]. With the help of the Lanczos-Chebyshev reduction technique, the PDEs are changed into a set of ODEs of the second order. We talk about the requirements that must be met for the BVMs to converge, as well as the computational difficulties of the methods. There are several numerical examples provided to demonstrate how straightforward and accurate the methodology is. In addition, the Boundary Value Methods have been applied in order to make approximations of second-order PDEs. In order to accomplish this goal, the Lanczos-Chebyshev reduction method was applied to the (PDEs, which resulted in the formation of an analogous second-order system. It can be shown that this method is both accurate and straightforward in its execution.

Omar Zurni [20] examined that by utilizing block approaches, it is possible to directly answer the second order boundary value issue [20]. This approach may be done either with or without the usage of initial values, depending on your preference. It has not, however, been looked into whether or not a comparison of the impacts of beginning values on the outcomes obtained from solving a second order boundary value issue is possible. As a result, a two-step block technique for the numerical solution of a two-point second order boundary value issue is provided within the scope of this study. The solution to boundary value issues may be found by deriving a technique from Taylor series expansions. This leads to a family of block matrix equations, which can be utilized to solve the problems. The approach is applied using starting and non-starting data, and the results are compared with other methods that have previously been developed. When applied using initial values, the findings demonstrate that the method's accuracy improves significantly.

Pandey Pramod Kumar [21] presented an effective numerical approach to the resolution of a system of two-point boundary value problems with Dirichlet boundary conditions And go through the steps involved in constructing a method of the Numerov type by supposing an additional continuity constraint on the answer [21]. The order of the methodology that is being proposed is quadratic. This strategy that is being proposed is

useful for tackling difficulties involving obstacles. Also considered an obstacle issue, which was addressed using the proposed approach in order to demonstrate both the effectiveness and the precision. The results are analyzed and compared to the other approaches.

Chedjou et.al [22] introduced and validates a comprehensive and universally applicable computational approach for addressing nonlinear differential equations (NDEs). This approach is grounded in neurocomputing principles and leverages cellular neural networks (CNNs) [22]. The CNN processor architecture is employed to guarantee high precision, stability, convergence, and minimal memory usage. Notably, this paper tackles a significant challenge by ensuring these computational characteristics across all system states, whether regular or chaotic, and under all possible bifurcation conditions that NDEs may exhibit. A key essence of this paper is to formulate and demonstrate a solving methodology that establishes CNN processors, whether implemented in hardware or software, as universal solvers for NDE models.

Jahanshahi Mohammad et al [23] investigated the solutions to boundary value issues are based on turning a differential equation with boundary conditions into a mixed Voletrra and Fredholm integral equation [23]. After that, we will utilize the specific case of the successive approximations approach to solve the equation that we have got. Convergence analysis and error estimate are both covered in this article. Additionally, a numerical example is provided to illustrate how accurately the suggested method works.

Mukhtar, Nur Zahidah, et al. [24] presented a four point direct block one-step approach for solving directly the general second order non stiff initial value problems (IVPs) of ODE is shown here [24]. The mathematical issues that emerge in the real world may be written down in the form of differential equations. These problems can be found in the domains of science and engineering, such as fluid dynamics, electric circuits, the motion of rockets and satellites, and other areas of application. The approach that has been presented would estimate the approximation solutions at four different sites at the same time by employing varying step sizes. The effectiveness of the suggested strategy is demonstrated through the use of numerical results.

Xuemei Zhang et.al [25] investigates the existence of solutions for a family of nonlinear impulsive integro-differential equations in Banach spaces that have second-order boundary-value problems with integral boundary conditions [25]. The fixed point theorem for stringent set contraction operators serves as the foundation for the reasoning. An example is being developed in the meantime to illustrate the key findings.

5.2 Limitations of State of the art Methods

While DNNs show promise for solving ODEs, they also come with certain limitations and challenges. Here are some key limitations of DNN-based ODE solutions:

Data-Intensive Training: DNNs, especially deep architectures, often require large volumes of data for training. In the context of ODEs, obtaining labeled data with known solutions for a wide range of ODE types and conditions can be challenging and may limit the applicability of DNNs.

Generalization to Unseen ODEs: DNNs may struggle to generalize effectively to ODEs with different characteristics, such as nonlinearities, stiff behavior, or high dimensionality, especially if the training data doesn't cover a diverse set of scenarios. Generalization may require substantial amounts of data and sophisticated network architectures.
Numerical Stability: Ensuring numerical stability, especially when dealing with stiff ODEs or rapid changes in solutions, can be challenging. DNNs may not always maintain stability during the solution process, leading to unreliable results.

Choice of Hyperparameters: DNNs require careful tuning of hyperparameters such as the network architecture, learning rate, batch size, and regularization techniques. Selecting appropriate hyperparameters can be challenging and can significantly affect the performance of the DNN-based solver.

Overfitting and Data Noise: DNNs are susceptible to overfitting when the training dataset is noisy or contains outliers. Robust techniques are required to mitigate the impact of noisy data on the accuracy of DNN-based ODE solutions.

Boundary and Initial Conditions: Incorporating boundary and initial conditions into DNN-based ODE solvers can be non-trivial, and ensuring that these conditions are satisfied accurately is a challenge, particularly for PDEs.

To mitigate these limitations, ongoing research is exploring methods for enhancing the robustness, interpretability, and generalization capabilities of DNN-based ODE solvers. Additionally, hybrid approaches that combine the strengths of DNNs with traditional numerical methods are being investigated to address specific challenges in ODE solving.

6. Future Directions

Hybrid Approaches: Investigate the integration of DNNs with traditional numerical methods to develop hybrid approaches that capitalize on the strengths of both. This may enhance the stability, efficiency, and interpretability of the solutions obtained for ODEs.

Uncertainty Quantification: Explore methodologies for incorporating uncertainty estimates into DNN-based solutions, providing confidence intervals and uncertainty quantification for the predictions. This is crucial for applications in scientific simulations where understanding the reliability of the results is imperative.

Adaptive Learning Rates: Develop adaptive learning rate schemes that dynamically adjust the learning rates during training to improve convergence and efficiency, especially for ODEs with complex dynamics or rapidly changing solution behaviors.

Incorporating Prior Knowledge: Investigate ways to incorporate prior knowledge, domain-specific information, or physical constraints into the DNN architectures to enhance the interpretability and accuracy of the solutions, particularly in scientific and engineering applications.

Parallelization Strategies: Explore parallelization strategies tailored for DNN-based ODE solvers to leverage the capabilities of modern parallel computing architectures, enhancing scalability and reducing computational time for large-scale simulations.

Transfer Learning: Explore the potential of transfer learning techniques to train DNNs on a diverse set of ODEs, allowing for better generalization across different types of problems and domains.

Benchmarking and Standardization: Establish benchmarks and standardized datasets for evaluating the performance of DNN-based ODE solvers. This would facilitate fair comparisons between different methodologies and encourage the adoption of best practices in the field.

Explainability and Interpretability: Develop techniques to enhance the explainability and interpretability of DNN-based solutions for ODEs, ensuring that the models provide insights into the underlying dynamics and contributing factors.

7. Conclusion

In conclusion, the integration of DNNs for solving ODEs represents a dynamic and promising area of research with vast potential for transformative impacts on scientific computing and applied mathematics. This comprehensive review has examined the current landscape, methodologies, and challenges associated with leveraging DNNs for the numerical solution of ODEs. The survey reveals the remarkable progress made in harnessing the expressive power of DNNs to approximate solutions for a diverse range of ODEs, from simple to complex systems. The flexibility of DNN architectures, coupled with advancements in training strategies, has enabled the successful application of these models across various domains, including physics simulations,

engineering problems, and dynamic systems modeling. However, challenges such as interpretability, uncertainty quantification, and the need for hybrid approaches persist. Addressing these challenges is crucial to further enhance the reliability and applicability of DNN-based ODE solvers. Future directions should focus on developing hybrid methodologies, incorporating uncertainty estimates, and enhancing the interpretability of solutions. Additionally, adaptive learning rates, transfer learning, and the incorporation of prior knowledge hold promise for improving the efficiency and generalization capabilities of DNN-based approaches. As we move forward, collaborative efforts between researchers in machine learning, numerical analysis, and domain-specific sciences will be pivotal in advancing the field. By fostering interdisciplinary collaborations, we can unlock new insights and synergies that propel the development of robust and practical solutions for a wide range of ODE applications.

In summary, while DNNs have showcased their potential in revolutionizing the numerical solution of ODEs, there remains a rich landscape for exploration and refinement. This review aims to inspire and guide future research endeavors, encouraging the scientific community to push the boundaries of innovation and contribute to the ongoing evolution of this exciting and transformative field.

References

1. Hsu, Sze-Bi, and Kuo-Chang Chen. Ordinary differential equations with applications. Vol. 23. World scientific, 2022.
2. Schiesser, W. E. Time Delay ODE/PDE Models: Applications in Biomedical Science and Engineering. CRC Press, 2019.
3. Borzì, Alfio. Modelling with ordinary differential equations: a comprehensive approach. CRC Press, 2020.
4. Li, Jichun, and Yi-Tung Chen. Computational partial differential equations using MATLAB®. Crc Press, 2019.
5. Okeke, Anthony Anya, Buba MT Hambagda, and Pius Tumba. "Accuracy Study on Numerical Solutions of Initial Value Problems (IVP) in Ordinary Differential Equations (ODE)."
6. Butcher, John Charles. B-series: algebraic analysis of numerical methods. Vol. 55. Berlin: Springer, 2021.
7. Suchde, Pratik, and Joerg Kuhnert. "A meshfree generalized finite difference method for surface PDEs." Computers & Mathematics with Applications 78, no. 8 (2019): 2789-2805.
8. Baccouch, Mahboub, ed. Finite Element Methods and Their Applications. BoD–Books on Demand, 2021.
9. Lei, Jun, Qin Wang, Xia Liu, Yan Gu, and Chia-Ming Fan. "A novel space-time generalized FDM for transient heat conduction problems." Engineering Analysis with Boundary Elements 119 (2020): 1-12.
10. Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." Neural networks 2.5 (1989): 359-366.
11. Danang, Pratama A., and Maharani A. Bakar. "Adaptive deep neural network using genetics algorithms for solving linear and non-linear ordinary differential equations." In AIP Conference Proceedings, vol. 2472, no. 1. AIP Publishing, 2022.
12. Dufera, Tamirat Temesgen. "Deep neural network for system of ordinary differential equations: Vectorized algorithm and simulation." Machine Learning with Applications 5 (2021): 100058.
13. Shubham V. Deshmukh , Vishal V. Shukla, 2021, Comparing Numerical Solution to a Second Order Boundary Value Problem by Variational Techniques, international journal of engineering research & technology (ijert).
14. Akinlabi, G.O., Busari, A.A., Abatan, O.G. and Odunlami, O.A., 2021. Numerical approximation of second-order boundary value problems via hybrid boundary value method. In Journal of Physics: Conference Series (Vol. 1734, No. 1, p. 012022). IOP Publishing.
15. Bakirova, E.A., Assanova, A.T. and Kadirbayeva, Z.M., 2021. A problem with parameter for the integro-differential equations. Mathematical Modelling and Analysis, 26(1), pp.34-54.
16. Jajarmi, A. and Baleanu, D., 2020. A new iterative method for the numerical solution of high-order non-linear fractional boundary value problems. Frontiers in Physics, 8, p.220.

17. Anitescu, Cosmin, Elena Atroshchenko, Naif Alajlan, and Timon Rabczuk. "Artificial Neural Network Methods for the Solution of Second Order Boundary Value Problems." Computers, Materials & Continua 59, no. 1 (2019).
18. Nabian, Mohammad Amin, and Hadi Meidani. "A deep neural network surrogate for high-dimensional random partial differential equations." arXiv preprint arXiv:1806.02957 (2018).
19. Biala, T.A., Jator, S.N. and Adeniyi, R.B., 2017. Boundary Value Methods for Second-Order PDEs via the Lanczos-Chebyshev Reduction Technique. Mathematical Problems in Engineering, 2017.
20. Omar, Z. and Adeyeye, O., 2016. Solving two-point second order boundary value problems using two-step block method with starting and non-starting values. International Journal of Applied Engineering Research, 11(4), pp.2407-2410.
21. Pandey, P.K., 2015. Solving system of second order boundary value problems by Numerov type method. Journal of Science and Arts, 15(2), p.143.
22. Chedjou, Jean Chamberlain, and Kyandoghere Kyamakya. "A universal concept based on cellular neural networks for ultrafast and flexible solving of differential equations." IEEE Transactions on Neural Networks and Learning Systems 26, no. 4 (2014): 749-762.
23. Jahanshahi, M., Nazari, D. and Aliev, N., 2013. A special successive approximations method for solving boundary value problems including ordinary differential equations. Mathematical Sciences, 7, pp.1-4.
24. Mukhtar, N.Z., Majid, Z.A., Ismail, F. and Suleiman, M., 2012. Numerical solution for solving second order ordinary differential equations using block method. In International Journal of Modern Physics: Conference Series (Vol. 9, pp. 560-565). World Scientific Publishing Company.
25. Xuemei Zhang; Meiqiang Feng; Weigao Ge (2010). Zhang, X., Feng, M. and Ge, W., 2010. Existence of solutions of boundary value problems with integral boundary conditions for second-order impulsive integro-differential equations in Banach spaces. Journal of Computational and Applied Mathematics, 233(8), pp.1915-1926.