International Journal of Scientific World, 11 (2) (2025) 1-7

International Journal of Scientific World

Website: www.sciencepubco.com/index.php/IJSW
https://doi.org/10.14419/j7yksy12
Review paper

Neural network for solving differential equations

Khalil M. *, Said M. , Ibrahim A. , Elserwi H. , Mostafa B.

[1] *Faculty of Engineering, October University for Modern Sciences and Arts (MSA), Egypt*
**Corresponding author E-mail: mkibrahim@msa.edu.eg*

Received: April 24, 2025, Accepted: June 2, 2025, Published: June 7, 2025

Abstract

Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs) are fundamental to modelling a wide range of scientific and engineering phenomena, including fluid dynamics, heat conduction, and biological systems. Classical numerical methods, such as the finite difference method (FDM) and finite element method (FEM), often suffer from high computational costs, difficulty in handling complex boundary conditions, and limited scalability. With the advancement of artificial intelligence, artificial neural networks (ANNs) and con-convolutional neural networks (CNNs) have emerged as powerful tools for solving differential equations. This paper explores the application of neural networks for solving ODEs and PDEs, analyzing their effectiveness, advantages, and limitations as neural networks are predicted to transform computational mathematics, offering more accurate results.

Keywords: *ANN; CNN; DNN; ODE; PDE.*

1. Introduction

Differential equations are used to model problems in physics, chemistry, biology, and economics. However, traditional analytical methods are sometimes insufficient for deriving closed-form solutions. To address this challenge, several numerical techniques have been developed, including the Finite Difference Method (FDM), Finite Element Method (FEM), Finite Volume Method (FVM), and Boundary Element Method (BEM). It is difficult to discretize domains into finite elements (FEs) using these approaches. [4]. To solve differential equations, neural networks, which simulate the structure and function of biological neural systems found in the human brain, can be employed. Inspired by the brain, they are intricately linked networks of neurons that can be trained to learn about the system. The challenges of pattern recognition, image processing, forecasting, and optimization have all been effectively solved by neural networks. [1], [2]

2. History of artificial neural networks

Alexander Bain introduced the concept of neural networks in 1873, suggesting that the human brain is composed of interconnected neurons and each action is linked to specific neurons. He also introduced the "rule of associations," which states that repeated occurrences stimulate connections between neurons. In 1943, Warren McCulloch and Walter Pitts presented the first artificial neuronal network, based on the biological neuron model. This structure can simulate binary logic operations and has evolved through similar structures. Donald Hebb formalized the learning rule for changes in synaptic connections in 1949. Marvin Minsky's 1951 creation of the Snark, the first neurocomputer, was typical of this age.

Frank Rosenblatt introduced the Perceptron class of artificial neural networks (ANN) in the 1960s, which could memorize input and output vector pairs using a novel learning rule. However, Marvin Minsky and Seymour Papert demonstrated that convergence to a solution depends on the values the net is trained with. Bernard Widrow and Marcian Hoff developed the Widrow-Hoff learning rule for Adaline, which is a different type of neural network processing element that generalizes to accept vectors other than those used for training. By the middle of the 1960s, researchers had exhausted their viable options. In the last episode of this period, Marvin Minsky and Seymour Papert launched a campaign to denigrate neural network research and redirect funds for neural network research to the field of "artificial intelligence. Even after Minsky and Papert showed the limits of perceptrons, neural network research went on. Neural networks were extensively studied in the fields of biological modeling, pattern recognition, and adaptive signal processing. In actuality, the 1970s saw the publication of work by many of the present leaders in the area. They were the ones who established the field of neural networks on solid ground and set the stage for its revival, along with those who joined during the following 13 years.

The backpropagation (BP) algorithm, discovered in 1986, is a widely used and successful method for multilayer networks. Unlike other biologically inspired learning algorithms, it is based on mathematical deductions. Other ANN architectures and learning rules are adaptations of neurological processes or phenomena. The IEEE International Conference on Neural Networks, the first open conference on neural networks in contemporary times, took place in San Diego in 1987, and the International Neural Network Society (INNS). The INNS journal Neural Networks was established in 1988, followed by Neural Computation in 1989 and the IEEE Transactions on Neural Networks in 1990. [2], [1]

The evolution of artificial neural networks (ANNs) is divided into three generations. The first generation, derived from the McCulloch and Pitts model, includes ANN Perceptron, Hopfield networks, and Boltzmann machines. In the second generation, continuous values are allowed within a certain range, highlighting networks trained with the BP algorithm. The transition from binary to continuous values was achieved through learning algorithms or new transfer functions. The third generation, SNN (Spiking Neural Networks) or PNN (Pulsed Neural Networks), resembles biological neural networks. The bifurcation of research in ANNs is due to the biological implausibility of second-generation models, which have limited practical applicability. ANNs are expanding neural computing approaches, with Deep Feed-forward Neural Networks (DFFNN) first appearing in the 1990s. DFFNNs have hidden layers and have a small error transfer. Conventional feedforward ANNs increase training times exponentially. Early 1990s approaches allowed rapid training of DFFNNs, forming the foundation of sophisticated machine learning systems. The well-known neural network types:

- Multi-Layer Perceptron Neural Network (MLP NNs)
- Radial Basis Function Neural Networks (RBF NNs)
- Hopfield Neural Network (HN)
- Hamming Neural Network (HNN)
- Kohonen Self-Organized Map Neural Network (KSOM NN)
- Time Delay Neural Network (TDNN)
- Deep Feed Forward Neural Networks (DFF NNs)
- Recurrent Neural Networks (RNNs)
- Long-Short Term Memory Neural Networks (LSTM NNs)
- Auto Encoders Neural Network (AE NN)
- Markov Chain Networks (MC Ns)

In addition, the architectures listed above serve as the theoretical underpinnings. It should be mentioned that the designs that have been given have been effectively used in a variety of fields, including marketing, economics, and health, to solve issues that call for pattern recognition, regression, and classification. [1], [2]

3. Artificial neural networks: an overview

Artificial neural networks (ANNs) are complex computer models created to mimic the biological neural networks' information processing skills present in the human brain. These networks are set up to learn from data and carry out intricate tasks like regression, classification, and other machine learning operations. An ANN's basic design is made up of several linked layers of neurons, each of which helps the network represent and resolve different issues. Three primary layer types are commonly seen in an ANN's architecture: input, hidden, and output layers [1], [4]:

1) Input Layer: This network's initial layer oversees receiving raw data. Each neuron in the input layer, which corresponds to a feature of the input data, provides information to the network.
2) Hidden Layers: After the input data is sent from the input layer, it is received by one or more hidden layers. In order to process inputs, neurons in each hidden layer apply a weighted summation. Following that, an activation function is used to execute this total. The activation function adds non-linearity to the network, allowing it to represent and comprehend complex input-output relationships. The input data is transformed by the hidden layers in ever more abstract ways as it moves through the network.
3) Output Layer: The output layer is the last layer in the network. It generates the ultimate outcome or prediction for the network after receiving the processed data from the last hidden layer. The job determines the shape and activation function of the output layer. For instance, in regression tasks, a linear activation function may be used to predict continuous values, while in classification tasks, the output layer may employ a softmax function to offer probabilities for various classes.

In an artificial neural network (ANN), each artificial neuron is linked to other neurons by weights, synapses, connections, or parameters. To represent the transmitting and receiving neurons, each neuron has a variable w_{ij}. All neurons receive the same value at the same moment, even though these connections provide distinct values. A neuron uses input data for two purposes: it calculates the relationship between connections and incoming values and applies a function to the outcome. The meanings of these procedures must be clarified because their names and aggregates are inconsistent. It is known as the combination function for the first operation, the activation function for the second, and the transfer function for the aggregate. [4] The loss function (or cost function) measures the error between the predicted output and the actual output, helping the model understand how far it is from the correct answer. Common examples include Mean Squared Error (MSE) and Cross-Entropy Loss. Backpropagation is an algorithm used to adjust the model's weights by propagating the error backward from the output layer to the input layer. It happens in all layers of the network, with the goal of minimizing the loss by updating the weights based on the gradient descent method. Optimizers are algorithms that help minimize the loss function by adjusting the weights during training. They determine how the model should update its weights based on the gradients. Common optimizers include Gradient Descent (GD), Stochastic Gradient Descent (SGD), and Adam, with Adam often providing better performance by adapting the learning rate for each parameter. [1], [2], [4]

4. Neural networks (NNs) for solving ODEs and PDEs

Recent advances in deep learning have opened new avenues for solving differential equations using neural networks [5]. These data-driven methods rely on the universal approximation capabilities of neural networks to approximate the solutions to both ordinary differential equations (ODEs) and partial differential equations (PDEs).
Consider a general Differential equation in the form:

For ODEs: $D[u(x)] = f(x), x \in \Omega \subset \mathbb{R}$

For PDEs: $D[u(x,t)] = f(x,t), (x,t) \in \Omega \subset \mathbb{R}^n \times \mathbb{R}$

with appropriate initial and/or boundary conditions.

The main idea is to approximate the unknown solutions $u(x)$ or $u(x,t)$ using a neural network $u_\theta(x)$, parameterized by weights θ. The trial solution $u_\theta(x)$ that satisfies the boundary/initial conditions. The neural network is trained to minimize a loss function constructed from the differential operator:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} |D[u_\theta(x_i)] - f(x_i)|^2 + (\text{penalty terms})$$

Where $\{x_i\}$ are collocation points in the domain Ω.

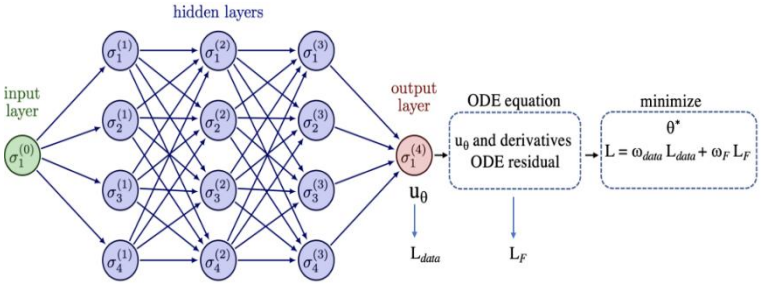

Fig. 1: Schematic Representation of the Structure for A Vanilla-Pinn Modelling an Ode Solution [6].

The loss is minimized using gradient-based optimization methods, such as stochastic gradient descent (SGD) or Adaptive Moment Estimation (Adam). Gradients of the neural network with respect to its inputs are computed using automatic differentiation, which ensures accurate and efficient calculations of derivatives needed in the residual terms.
Below is a comparison of key performance metrics (accuracy, runtime) between neural network methods and classical numerical approaches (e.g., finite difference, finite element) for solving differential equations (see Table 1) while in table 2, the strengths points and applications of the neural network in solving differential equations are presented [7]:

Table 1: A Comparison of Key Performance Metrics (Accuracy, Runtime) Between Neural Network Methods and Classical Numerical Approaches

Metric	Classical Methods	Neural Network Methods
Accuracy	High precision (e.g., 10^{-6} for simple ODEs/PDEs	Competitive (e.g.,$10^{-3} - 10^{-5}$ error)
Runtime	Fast for low-dimensional problems	Slow training, but fast evaluation
Scalability	Poor in high dimensions (curse of dimensionality)	Strong (e.g., 100D+ PDEs)
Data Needs	Minimal (equations discretized)	Minimal (physics-informed NNs use equations + Boundary conditions/Initial conditions)

Table 2: Strengths and Applications of Neural Network in Solving Differential Equations.

Neural Network Type	Strengths	Applications
ANN	Universal function approximator	Solving ODEs/PDEs, surrogate modeling
CNN	Captures spatial features efficiently	Grid-based PDEs, image-based simulations
RNN	Models temporal dependencies	Time-dependent ODEs/PDEs, dynamical systems
PINN	Enforces physical laws via loss constraints	Data-scarce physics problems, forward/inverse PDEs
DeepONet	Learns mappings between function spaces	Parametric PDEs, operator learning
DeepONet CNN	Combines CNN with DeepONet for high-dimensional data	Complex input-output PDE mappings, fast evaluation

Several neural network architectures that can be emerged for the numerical and analytical approximation of solutions to differential equations are presented in this section with an emphasis on their theoretical formulation, structural design, and applicability to various classes of differential equations.

4.1. Artificial neural network (ANNs)

Artificial neural networks (ANNs) offer a powerful approach to solving differential equations by leveraging their ability to approximate complex functions. The core idea is to represent the unknown solution of a differential equation using a neural network and then train this network so that its output satisfies the given differential equation and any associated initial or boundary conditions. An Approximation of Universal The theorem argues that a neural network may estimate any function to any given accuracy at a single hidden layer with a single input and output layer. A single-variable equation using functions is known as an ordinary differential equation (ODE). An ordinary differential equation often appears as [24]

$$f(x, g(x), g'(x), g''(x), \dots, g^{(n)}(x)) = 0 \quad (1)$$

Where $g(x)$ is the function to find, and $g^{(n)}(x)$ is the n-th derivative of $g(x)$.
The trial solution of (1) involves a function that ensures the output is zero when evaluated at the given conditions. Let the trial solution $g_t(x)$ be $g_t(x) = h_1(x) + h_2(x, N(x, P))$ [23, 24]
where $h_1(x)$is a function that makes $g_t(x)$satisfy a given set of conditions, $N(x, P)$a neural network with weights and biases described by P and $h_2(x, N(x, P))$some expression involving the neural network. The role of the function $h_2(x, N(x, P))$ is to ensure that the output from $N(x, P)$is zero when $g_t(x)$ is evaluated at the values of x where the given conditions must be satisfied. The function $h_1(x)$ should alone make $g_t(x)$ satisfy the conditions.
As described previously, an optimization method could be used to minimize the parameters of a neural network, that being its weights and biases, through backward propagation.
For the minimization to be defined, we need to have a cost function at hand to minimize.

Since we are looking at one input, the cost function is just f squared. The cost function $c(x, P)$ can therefore be expressed as

$$C(x, P) = (f(x, g(x), g'(x), g''(x), \dots, g^{(n)}(x)))^2 \quad (2)$$

If N inputs are given as a vector x with elements x_i for $i = 1, \dots, N$, the cost function becomes

$$C(x, P) = \frac{1}{N} \sum_{i=1}^{N} (f(x_i, g(x_i), g'(x_i), g''(x_i), \dots, g^{(n)}(x_i)))^2$$

The neural net should then find the parameters P that minimizes the cost function for a set of N training samples x_i.
The function approximation capabilities of feedforward neural networks enable neural network approaches to solve both ordinary and partial differential equations, producing a solution expressed in a closed analytic form. As a fundamental approximation element, this type uses a feedforward neural network whose parameters are changed to minimize a suitable error function. The gradient of the error with respect to the network parameters must be calculated to train the neural network using any optimization approach. This approach uses a sum of two components to represent a trial solution of the differential equation. There are no modifiable parameters in the first section, which satisfies the starting or boundary requirements. The second half uses a feed forward neural network with changeable parameters and is built to not alter the starting or boundary conditions. As a result, the network is trained to meet the differential equation, and the starting or boundary conditions are satisfied by building the trial solution. [24]

4.2. Convolutional neural networks (CNNs)

Neural networks utilized for image identification and classification tasks are called convolutional neural networks, or CNNs. By using learnable filters, they can extract important information and patterns from input photos. This information is extracted by the convolutional layer, which produces a feature map. Convolutional layers make up the first few layers, and then pooling layers are added to minimize size and prevent overfitting. For final classification, the feature maps are sent through completely linked layers. CNNs can recognize complex patterns and structures in incoming data by learning hierarchical representations of the data. They have achieved success in natural language processing, speech recognition, picture segmentation, object detection, and image identification [1], [3], [23], [24].
The Dirichlet and Neumann formulas can be used to decompose the differential equations into smaller parts. Neural networks were used to solve non-linear equations up to the seventh decimal place. Given that Lagaris et al. [8] is the first study to employ CNNs to solve PDEs, it merits a quick explanation. Examining a generic differential equation that requires a solution and is provided by

$$F\big(x, y(x,t), \nabla y(x,t), \nabla^2 y(x,t)\big) = 0, \bar{x} \in B \quad (3)$$

Here $x = (x_1, x_{2,} \dots, x_n) \in \mathbb{R}^n$ for certain boundary conditions as per an arbitrary boundary S, $B \subset \mathbb{R}^n$ is the defining domain and $y(x,t)$ is the solution needed. It should be noted that
For obtaining a solution to (3), first the domain B has to be discretized into a set of points
$\hat{B}$. Also, the arbitrary boundary S (given here) of the general equation has to discretized into a set of points $\hat{S}$. Then, the DE may be expressed as a system which hasconstraints of the generally defined boundary conditions as per:

$$F\big(x_i, y(x_i,t), \nabla y(x_i,t), \nabla^2 y(x_i,t)\big) = 0 \; \forall \, x_i \in \hat{B}$$

Where $y(x, t)$ is the solution. It can be obtained from two components given in:

$$y(x,t) = A(x) + f(x, N(x,p))$$

Here $A(x)$ has fixed parameters, p is the parameter set, and $N(x, p)$ is the neural network for minimization. Although the initial CNNs used for solving PDEs gave promising results, they had certain issues. These included a lack of interpretability, weak adaptation of their structure to problems, and average performance.

4.3. Physics-informed neural networks (PINNs)

The term Physics-informed neural networks (PINNs) refers to a new genre of CNNs that were introduced after being inspired by the works of Lagaris et al. (1998) and Hornik (1991) [8,17] where CNNs were used for universal approximation. In this new genre, physical constraints in the form of PDEs were added to the loss function. More precisely, the method applied the physics principles represented by PDEs as CNN loss functions. Consequently, it may be possible to develop solutions by optimizing the loss functions [3]. Discrete domains are not required for PINNs. They are also quite useful because they do not require a lot of calculations. PINNs use minimization strategies to non-linear parametric PDEs with the structure provided by

$$y(x,t) + N[y, \lambda] = 0, x \in \Omega, t \in [0, T] \quad (4)$$

Where $y(x, t)$ is the solution that is hidden $N[y, \lambda]$ is an operator of λ. Ω belongs to $\mathbb{R}^D$ where D is the number of dimensions. Despite having a PDE-focused CNN model that improved performance, PINN has the same drawbacks as other CNNs, including extensive training durations, a requirement for vast amounts of data, and a lack of interpretability. Despite this, it has some success as an expert system for PDE solutions. Its framework specialized in PDEs was the cause of this. [9-16].

4.4. DeepONet CNN

CNN deployment and training got easier because of increased computer power and the accessibility of high-performance computing (HPC) equipment. Following it, several methods that had previously been challenging to use were used. To support the application of neural networks for operator learning, Lu et al. looked to Chen et al.'s non-linear operators (Chen and Chen 1995a, b) as a theoretical foundation.

They developed Deep Operator Network (DeepONet), a powerful CNN. The authors employed several CNNs in the DeepONet model to diversify and target different net assemblies since CNNs have better impressibility and numerous unique benefits, such as in computer vision and sequential analysis, respectively. Additionally, in some CNNs, a split parallel structure can be used [3].

4.5. Recurrent neural network (RNN)

A Recurrent Neural Network (RNN) is an artificial neural network designed to process sequential data by utilizing internal memory to capture temporal dependencies. Unlike traditional feedforward neural networks, RNNs have recurrent connections, allowing information to persist and influence subsequent inputs, making them particularly effective for tasks where context and order are crucial. Applications of RNNs include Natural Language Processing (NLP) for tasks like language modelling, machine translation, and text generation, Speech Recognition for converting speech to text, and Time-Series Forecasting for predicting future values in sequences like stock prices and weather data. RNNs are particularly useful for solving time-dependent differential equations, where past values influence future outcomes, such as dynamic systems, chaotic systems, and biological simulations. [14-17] The advantages of RNNs include effective processing of sequential data, parameter sharing across time steps to enhance training efficiency, and the ability to model temporal dependencies and the principal advantage of RNN over ANN is that RNN can model a collection of records (i.e. time collection) so that each pattern can be assumed to be dependent on previous ones, recurrent neural networks are even used with convolutional layers to extend the powerful pixel neighborhood. However, they also have disadvantages, such as gradient exploding and vanishing problems. Training an RNN is a tough task. It cannot systematize very lengthy sequences if the usage of tanh or ReLu as an activation feature, vanishing and exploding gradient problems when dealing with long sequences, high computational demand due to sequential processing, and challenges in retaining long-term dependencies [8-17].The fundamental structure of an RNN consists of an input layer, a hidden layer, and an output layer, with recurrent connections that cycle information within the network. At each time step, t, the RNN takes an input vector, x_t , and updates its hidden state, h_t ,using the following equation: $h_t = \sigma_h(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$ where(W_{xh}) is the weight matrix between the input and hidden layer, (W_{hh}) is the weight matrix for the recurrent connection, (b_h) is the bias vector, and (σ_h) is the activation function, typically the hyperbolic tangent function (tanh) or the rectified linear unit. The output at each time step, t, is given by the following: $y_t = \sigma_y(W_{hy}h_t + b_y)$. Unlike traditional neural networks, RNNs have loops that enable them to "remember" past data, making them ideal for sequential data processing. For solving differential equations, consider a first-order ordinary differential equation (ODE) of the form: $\frac{dy(t)}{dt} = f(y(t), t)$ To approximate the solution $y(t)$, an RNN can be structured such that its hidden state $h(t)$ corresponds to the value of $y(t)$ at discrete time steps. The update rule for the hidden state in the RNN can be expressed as: $h_{t+1} = h_t + \Delta t \cdot f(h_t, t)$ Here, Δt represents a small time increment. This formulation mirrors the Euler method for numerical integration, where the next state is determined by the current state plus the product of the time step and the derivative at that state. In the context of RNNs, this update can be implemented as:
$h_{t+1} = h_t + \Delta t \cdot \sigma(W_h h_t + W_x x_t + b)$ where

- W_h and W_x are, weight matrices.
- x_t is the input at the time step t.
- b is a biased term.
- σ is an activation function.

These properties make RNNs a powerful tool for modelling sequential dependencies, especially in dynamic and time-dependent applications. [10-17]

4.6. Long short-term memory (LSTM)

Long Short-Term Memory (LSTM) networks are a specialized type of Recurrent Neural Network (RNN) designed to effectively capture long-term dependencies in sequential data. Introduced by Hochreiter and Schmidhuber in 1997, LSTMs address the limitations of traditional RNNs, particularly the vanishing gradient problem, which hampers the learning of long-range dependencies. LSTMs introduce a memory cell that controls what information is stored, updated, and forgotten. This is done through three gates: Forget Gate (ft) → Decides what information to forget Gate (ft) → Decides what new information to add,and Output Gate (ot) → Decides what part of the memory to output, LSTM is Better than standard RNN because Solves the vanishing gradient problem → Can remember long sequences, better at long-term dependencies → Works well for speech recognition, text generation, and time series forecasting, and More complex but more powerful → Handles sequences better than basic RNNs [8-17].

5. Improving classical numerical methods with CNNs and ANNs

The integration of convolutional neural networks (CNNs) and artificial neural networks (ANNs) with classical numerical methods has enhanced computational efficiency and optimization. Traditional numerical methods, such as Runge-Kutta, Newton-Raphson, and Euler, have been widely used to solve ordinary differential equations (ODEs) but often face limitations in convergence speed and accuracy. Recent research has shown that deep learning techniques can improve these methods by providing better function approximations and optimization strategies.

For instance, the Runge-Kutta optimizer (RUN) has been enhanced using neural networks to improve solution exploration [17]. Studies also demonstrate that applying Runge-Kutta methods to gradient-based optimizers like SGD and Adam leads to more stable training processes. In neural ODEs, approaches such as high-order implicit ODE integration using Newton's method and meta-solver optimization have improved accuracy and robustness [9], [18], [19].

These techniques have practical applications in fields like medical diagnosis, where the GORUN optimizer has improved disease classification models, and network security, where Runge-Kutta-based feature selection has achieved 99.8% accuracy in intrusion detection [19], [21] Moreover, RKCNNs (Runge-Kutta CNNs) have enhanced prediction and efficiency in deep learning models [22-24] Overall, integrating classical numerical methods with AI improves optimization, accelerates training, and expands neural networks' applications in medicine, security, and computational modeling.

6. Limitations of neural networks for solving differential equations

Neural networks have emerged as powerful tools for solving differential equations [25], [26]. However, despite their flexibility and potential, they come with several limitations that impact their reliability and efficiency in scientific computing tasks as follows:

- Scalability Issues

Neural networks used to solve differential equations, especially Physics-Informed Neural Networks (PINNs) struggle with scalability. For high-dimensional PDEs (e.g., Navier-Stokes or Schrödinger equations in 3D), the computational cost grows rapidly. Training becomes slow and unstable due to vanishing gradients, particularly in stiff systems or when using deep architectures.
Example: Solving a 2D diffusion equation using PINNs may take hours or days, while traditional numerical solvers (like finite element methods) complete in minutes with better accuracy.

- Data and Problem Dependency

Although PINNs are data-efficient in theory, their performance heavily depends on:

1) Boundary/initial condition quality (Errors here can dominate the solution).
2) The collocation points choice (Poor sampling can miss critical regions, leading to inaccurate predictions).
3) Discontinuity or sharp gradients in the solution (e.g., PINNs often fail when there are shock waves in fluid dynamics).

Example: For Burgers' equation with a steep front, PINNs may yield high relative errors unless architecture or loss weighting is carefully adjusted.

- Interpretability and Debugging

Unlike traditional solvers, neural networks provide no direct physical interpretation. It's often unclear why the model fails or where the error concentrates. Interpreting loss landscapes or network weights does not reveal specific equation behavior. Lack of transparency makes it difficult to enforce physical constraints beyond soft penalty terms in the loss function. Metrics like MSE or residual loss don't always indicate the physical validity of the solution.

7. Future directions

Promising research areas include integrating quantum computing to accelerate the training of neural networks for differential equations and developing hybrid solvers that combine PINNs with traditional numerical methods for improved accuracy and stability. Recent studies, such as Farea et al. [27], explore Quantum Classical Physics-Informed Neural Networks QCPINN for solving PDEs, while others like Xiao et al. [28] propose Physics-informed quantum neural networks for solving forward and inverse problems of partial differential equations. These approaches open the door to a growing trend toward addressing complex differential equations models in several real-world applications. Additionally, future work may focus on optimizing the integration of quantum hardware with PINNs to reduce training time and computational cost, especially for high-dimensional and nonlinear differential equations.

8. Conclusion

In this work, we explored the evolution and capabilities of neural networks in solving differential equations, from their historical development to modern architectures like PINNs, DeepONet, and LSTMs. Neural networks provide a powerful alternative to classical numerical methods, which can improve the accuracy of the differential solutions. By bridging the gap between deep learning and numerical analysis, the combination of CNNs and ANNs has further enhanced conventional methods. Neural networks are expected to revolutionize computational mathematics, providing faster and more scalable solutions. While neural networks offer flexibility for solving differential equations, they face critical challenges in scalability, data dependency, and interpretability. Careful architecture design, adaptive sampling, and hybrid approaches are often needed to mitigate these limitations. Future advancements will likely focus on optimizing architectures and improving generalization for complex real-world applications.

References

[1] Yadav, N., Yadav, A., & Kumar, M. (2015). *An introduction to neural network methods for differential equations* (Vol. 1, p. 7). Berlin: Springer. https://link.springer.com/content/pdf/10.1007/978-94-017-9816-7.pdf. https://doi.org/10.1007/978-94-017-9816-7_4.

[2] Pires, P. B., Santos, J. D., & Pereira, I. V. (2025). Artificial neural networks: history and state of the art. *Encyclopedia of Information Science and Technology, Sixth Edition*, 1-25. https://www.igi-global.com/pdf.aspx?tid=320788&ptid=307143&ctid=4&oa=true&isxn=9781668473665. https://doi.org/10.4018/978-1-6684-7366-5.ch037.

[3] Hafiz, A.M., Faiq, I. & Hassaballah, M. Solving partial differential equations using large-data models: a literature review. *Artif Intell Rev* 57, 152 (2024). https://doi.org/10.1007/s10462-024-10784-5.

[4] Dubey, S. K., Islahi, H., & Singh, R. (2024). Deep Neural Networks for Solving Ordinary Differential Equations: A Comprehensive Review. *International Journal of Intelligent Communication and Computer Science*, 1(2), 54-68. https://www.researchgate.net/publication/378715647.

[5] Raissi, M., Perdikaris, P. and Karniadakis, G.E., 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, *378*, pp.686-707. https://doi.org/10.1016/j.jcp.2018.10.045.

[6] Baty, H., 2023. Solving stiff ordinary differential equations using physics informed neural networks (PINNs): simple recipes to improve training of vanilla-PINNs. arXiv preprint arXiv:2304.08289.

[7] Hu, Z., Shukla, K., Karniadakis, G.E. and Kawaguchi, K., 2024. Tackling the curse of dimensionality with physics-informed neural networks. *Neural Networks*, *176*, p.106369. https://doi.org/10.1016/j.neunet.2024.106369.

[8] Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. IEEE Transactions on Neural Networks, 9(5), 987–1000.https://www.cs.uoi.gr/~lagaris/papers/TNN-LLF.pdf. https://doi.org/10.1109/72.712178.

[9] Crews, D. W. (2022). *Arbitrarily high order implicit ODE integration by correcting a neural network approximation with Newton's method*. https://arxiv.org/pdf/2203.00147.

[10] Gao, H., Sun, L., & Wang, J.-X. (2020). *PhyGeoNet: Physics-Informed Geometry-Adaptive Convolutional Neural Networks for Solving Parameterized Steady-State PDEs on Irregular Domain*. arXiv preprint arXiv:2004.13145. Retrieved from https://arxiv.org/abs/2004.13145. https://doi.org/10.1016/j.jcp.2020.110079.

[11] Tondak, A. (2024, December 6). *Recurrent neural networks: An introduction*. K21 Academy. https://k21academy.com/datascience-blog/machine-learning/recurrent-neural-networks/.

[12] Livieris, I. E., Pintelas, E., & Pintelas, P. (2020). *A CNN–LSTM model for gold price time-series forecasting*. Neural Computing and Applications, 32(23), 17351-17360. https://doi.org/10.1007/s00521-020-04867-x.

[13] Mienye, I. D., Swart, T. G., & Obaido, G. (2024). Recurrent Neural Networks: A Comprehensive Review of Architectures, Variants, and Applications. *Information*, 15(9), 517. https://doi.org/10.3390/info15090517.

[14] Quradaa, M., Hamouda, A., & Hamouda, S. (2023). "A Systematic Literature Review on the Applications of Recurrent Neural Networks in Code Clone Detection." *Neural Computing and Applications*. https://doi.org/10.1371/journal.pone.0296858.

[15] Ren, Z. (2024). Advancements of Exploiting Convolutional Neural Networks for Solving Differential Equations. *Proceedings of CONF-MLA 2024 Workshop*, 190-195. https://doi.org/10.54254/2755-2721/94/2024MELB0070.

[16] Zhang, R., Liu, Y., & Sun, H. (2020). *Physics-guided Convolutional Neural Network (PhyCNN) for Data-driven Seismic Response Modeling*. *Engineering Structures*, 215, 110704. https://doi.org/10.1016/j.engstruct.2020.110704.

[17] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. Neural Networks, 4(2), 251–257.https://web.njit.edu/~usman/courses/cs677/hornik-nn-1991.pdf. https://doi.org/10.1016/0893-6080(91)90009-T.

[18] Su, D., Jiang, Q., Liu, E., & Liu, M. (2024). Improving Optimizers by Runge-Kutta Method: A case study of SGD and Adam. In Proceedings of the 2024 IEEE International Conference on Intelligent Computing and Intelligent Systems (ICICIP). Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/ICICIP60808.2024.10477795.

[19] Su, D., Jiang, Q., Zhu, E., & Liu, M. (2023). An enhanced Runge Kutta boosted machine learning framework for medical diagnosis. Computers in Biology and Medicine, 166, 106949. https://doi.org/10.1016/j.compbiomed.2023.106949.

[20] Gusak, J., Katrutsa, A., Daulbaev, T., Cichocki, A., & Oseledets, I. (2021). Meta-solver for neural ordinary differential equations. arXiv preprint arXiv:2103.08561.https://arxiv.org/pdf/2103.08561.

[21] Yuan, L., Tian, X., Yuan, J., Zhang, J., Dai, X., Heidari, A. A., Chen, H., & Yu, S. (2024). Enhancing network security with information-guided-enhanced Runge Kutta feature selection for intrusion detection. Cluster Computing, 27(3), 2231–2245 https://doi.org/10.1007/s10586-024-04544-x.

[22] **Error! Reference source not found.Error! Reference source not found.Error! Reference source not found.**Zhu, M., Chang, B., & Fu, C. (2022). Convolutional Neural Networks combined with Runge-Kutta Methods. arXiv preprint arXiv:1802.08831v7.https://arxiv.org/pdf/1802.08831.

[23] Chiaramonte, M. M., & Kiener, M. (2013). Solving differential equations using neural networks. arXiv. https://arxiv.org/abs/1306.0451.

[24] Tveito, A., & Winther, R. (2005). Introduction to partial differential equations. Springer. https://www.sgo.fi/~j/baylie/Introduction%20to%20Partial%20Differential%20Equations%20-%20A%20Computational%20Approach%20-%20A.%20Tveito,%20R.%20Winther.pdf.

[25] Wang, S., Teng, Y. and Perdikaris, P., 2021. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM Journal on Scientific Computing, 43(5), pp.A3055-A3081. https://doi.org/10.1137/20M1318043.

[26] Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R. and Mahoney, M.W., 2021. Characterizing possible failure modes in physics-informed neural networks. Advances in neural information processing systems, 34, pp.26548-26560.

[27] Farea, A., Khan, S. and Celebi, M.S., 2025. QCPINN: Quantum-Classical Physics-Informed Neural Networks for Solving PDEs. arXiv preprint arXiv:2503.16678.

[28] Xiao, Y., Yang, L.M., Shu, C., Chew, S.C., Khoo, B.C., Cui, Y.D. and Liu, Y.Y., 2024. Physics-informed quantum neural network for solving forward and inverse problems of partial differential equations. Physics of Fluids, 36(9). https://doi.org/10.1063/5.0226232.