AIMS Mathematics, 11(3): 7610–7632.
DOI: 10.3934/math.2026313
Received: 24 January 2026
Revised: 14 March 2026
Accepted: 18 March 2026
Published: 23 March 2026

https://www.aimspress.com/journal/Math

Research article

Numerical solutions of fractional order integro-differential equations by using artificial neural networks and power series method

Mohd Noor[1], **Musim Malik**[1] **and Mohammad Sajid**[2,*]

[1] School of Mathematical & Statistical Sciences, Indian Institute of Technology, Mandi, 175005, Himachal Pradesh, India

[2] Department of Mechanical Engineering, College of Engineering, Qassim University, Saudi Arabia

* **Correspondence:** Email: msajd@qu.edu.sa.

Abstract: This study presents a neural network-based framework for finding approximate series solutions to Abel's integral equation (AIE) and fractional order Volterra-type integro-differential equations (FOVIDEs). The suggested method changes the problem by writing the solution as a power series that has been truncated. This makes the original differential equation a problem of estimating parameters. Then, a special neural network is trained to find the coefficients of the series with good accuracy. Numerical tests show that the proposed method works, maintains accuracy, and converges. This shows that it can be used as a strong computational tool for solving fractional order systems.

Keywords: Abel's integral equation; artificial neural networks; fractional order Volterra integro-differential equations; power series
Mathematics Subject Classification: 68T07, 45D05, 45E10

1. Introduction

Fractional calculus (FC) is a new area of mathematical analysis that expands on the traditional integer order framework to include derivatives and integrals of any order, not just integers [1, 2]. The rapid expansion of FC is attributable to its applicability across various domains [3]. Scientists in physics, chemistry, and engineering are very interested in FC because fractional order operators can often explain complex things better. This is due to their emphasis on long range (global) dependencies in a system's behavior, as opposed to merely local variations, similar to integer derivatives [4–7]. In certain experiments, fractional order models demonstrate superior data fitting compared to their integer order equivalents [8].

Integro-differential equations (IDEs) are essential for modeling a variety of physical processes. They have an unknown function, like $u(t)$, and differential and integral operators that work on t. In

physics, astrophysics, population dynamics, biology, ecology, electromagnetism, reactor kinetics, fluid mechanics, chemical reactions, and one-dimensional viscoelasticity, these are useful [9–11]. Fractional integro-differential equations (FIDEs), formulated with fractional operators, can represent complex systems in scientific research and industrial applications [12–14]. Because it is hard to find closed-form solutions for many FIDEs, different numerical methods have been created to get an idea of how they work [15–17].

One of the most well-known FIDEs is Abel's integral equation (AIE), which is useful in many mathematical ways. It is utilized in various disciplines, including classical mechanics, electrical engineering, astrophysics, and viscoelasticity [18–20]. AIE is important for helping us learn more about science and technology because it is used in both theory and practice.

Artificial neural networks (ANNs) have been very good at solving differential equations in the last few years. They are often more accurate and faster than traditional numerical methods. Scientists from many fields are very interested in them because they can model complex solution landscapes. Lagaris introduced ANN-based methods for solving both ordinary differential equations (ODEs) and partial differential equations (PDEs) in the late 20th century [21]. This was a big turning point that led to extensive follow-up research on how ANNs could be used in this field.

Using ANNs to find approximate solutions has many advantages. One of the best things about this method is that the solution is given in a closed analytical form, which makes it easier to understand and use in different situations. Also, ANNs are great at generalizing, which means that once they are trained, they can handle many different situations that they were not trained on. Another big benefit is that ANNs do not need to discretize derivatives, which is a common step in traditional numerical methods that can cause mistakes and make calculations more difficult.

Consequently, the literature has witnessed an increasing number of articles addressing the application of ANNs for approximating solutions to diverse categories of differential equations. Some of these are ODEs [21–24], PDEs [25–28], and fractional order differential equations [29–31]. More recently, advanced machine learning and neural network-based techniques have also been developed for solving fractional models [32–35].

Hybrid approaches combining neural networks with truncated power series (TPS) or similar polynomial representations have been explored for integer-order differential equations, but their application to FIDEs remains relatively limited due to the nonlocal nature of fractional operators. The goal of this study is to create a new iterative method for solving certain types of FIDEs by combining the TPS approach with ANNs. The proposed method starts by rewriting the equation as an optimization problem. This problem is then solved using a single-layer neural network. If the FIDE has an analytical solution over a certain interval, the problem is solved by replacing it with a TPS and finding the unknown coefficients. Figuring out these coefficients is the hardest part of the process.

The FIDE becomes an optimization problem with a least mean square error function after discretizing the interval and making the necessary changes. The generalized delta rule, a common way to learn perceptrons, is used to lower the cost across the coefficient space. The method puts input samples into the network, figures out the error, and then uses gradient descent optimization to change the weight parameters (i.e., the series coefficients). The iterative process starts with an initial guess and then makes small changes to the TPS coefficients over and over again. After the coefficients are found, the solution function is estimated as a TPS that converges quickly. The novelty of this work is evident in the following aspects:

- We integrate ANNs with the TPS method to effectively solve FIDEs.
- Eliminates the need for grid-based numerical schemes, reducing computational cost and avoiding numerical instability.
- Gradient descent with momentum ensures fast convergence and stable learning, yielding results with very low absolute error.
- Requires minimal memory and executes rapidly (seconds on GPU), outperforming conventional numerical solvers.
- The method is easily extendable to other fractional, nonlinear, and delay-type integro-differential systems.

The proposed TPS-based formulation evaluates fractional derivatives of polynomial basis functions analytically using Gamma-function identities, thereby avoiding the repeated numerical approximation of fractional operators that is typically required in physics-informed neural network (PINN) frameworks. Unlike PINNs, where computing fractional derivatives is mathematically complex and computationally expensive, the proposed method naturally incorporates fractional terms within the power-series formulation.

This article is organized in the following way. Section 2 starts with a review of the most important definitions and basic ideas in the theory of FC. We go into more detail about the ANN approach in Section 3. We talk about the fractional order Volterra integro-differential equations (FOVIDEs) and AIE in Section 4. In Section 5, we explain the proposed method. In Section 6, we give examples that show how well the proposed method works for finding solutions to the types of FIDEs we are interested in. Section 7 talks about the results. Section 8 talks about the conclusions and suggestions.

2. Preliminaries

First, we review essential concepts and core definitions of fractional order derivatives that form the foundation for subsequent developments. As is well known, advances in FC have led to the development of various definitions by researchers. In this work, we focus exclusively on the widely used fractional derivative definition introduced by Caputo, as detailed in [36, 37].

Definition 1. Let $u(x)$ be a continuously differentiable function on a finite interval $[a, b]$ up to order k. The Caputo derivative ${}_aD_x^\alpha$ and fractional integral operator ${}_aI_x^\alpha$ of order $\alpha > 0$ are defined as follows:

$$ {}_aD_x^\alpha [u(x)] = \begin{cases} \frac{d^k u(x)}{dx^k}, & \alpha = k \in \mathbb{N}, \\ \frac{1}{\Gamma(k-\alpha)} \int_a^x \frac{u^{(k)}(\tau)}{(x-\tau)^{k-\alpha}} \, d\tau, & x > a, \\ & 0 \le k-1 < \alpha < k, \end{cases} \tag{2.1} $$

$$ {}_aI_x^\alpha [u(x)] = \frac{1}{\Gamma(\alpha)} \int_a^x \frac{u(\tau)}{(x-\tau)^{1-\alpha}} \, d\tau, \tag{2.2} $$

respectively.

Extensive research has explored the behavior and utility of the Caputo fractional operator. In this section, we highlight its principal properties and practical applications. In particular, the fractional derivative of a constant function is zero for any order, and the operator additionally satisfies the

following properties:

$$ {}_aD_x^{\alpha}\left[x^k\right] = \begin{cases} 0, & k \in \mathbb{N},\ k < \lceil \alpha \rceil, \\ \frac{\Gamma(k+1)}{\Gamma(k+1-\alpha)} x^{k-\alpha}, & x > a, \quad k \in \mathbb{N},\ k \geq \lceil \alpha \rceil, \end{cases} \tag{2.3} $$

$$ {}_0I_x^{\alpha}\left[x^k\right] = \frac{\Gamma(k+1)}{\Gamma(k+1+\alpha)} x^{k+\alpha}, \quad k \in \mathbb{N}. \tag{2.4} $$

In these formulas, $\lceil \alpha \rceil$ denotes the least integer that is not less than α. For further information and detailed mathematical properties of FC, readers are encouraged to consult [38].

3. Artifcial neural network

First, we talk about ANNs in this section. Then we talk about how they fit into our problem. Many complicated problems in the real world can be thought of as optimization tasks that need strong ways to solve them. Out of all the options, ANNs have become a flexible and powerful way to model math and solve problems related to natural events. Neural networks have already shown that they work well in many different scientific fields. It is important to understand how the proposed network architecture works before judging how well it works. The goal of this section is to give readers an idea of why ANNs are becoming more popular in modern applications.

ANNs are advanced models for processing information that are based on how the biological nervous system works. ANNs are made up of units that are linked to each other. These units take signals from many different sources, change them (usually in a non-linear way), and then send them out. This ability to get useful information from unclear data is what makes ANNs so useful for solving tough problems that regular programming methods can not handle.

In short, ANNs are made to solve very complicated problems in the real world that regular computers have a hard time with. In traditional computing, algorithms require a problem to be fully defined and then broken down into smaller formulas in order to create a high-level language program. ANNs, on the other hand, provide a more flexible and intuitive way to solve problems. For a more in-depth look at ANNs and their different meanings, refer to [39, 40].

To create a strong iterative method for finding the approximate solution to the FOVIDEs and AIE, we present a systematic approach for the suggested ANN architecture. Figure 1 shows a simple three-layer ANN with one external input, n hidden neurons, and one output. In this architecture, w and v are the $1 \times n$ weight vectors, and b is the bias term.

Here, the input signal, denoted by t, is first multiplied by the connection weight w_i, resulting in a weighted input. This weighted input is then passed through the activation functions σ_i, producing an intermediate result. The network output, $N(t, \theta)$, is calculated by summing the product of the hidden layer outputs and the weight parameters v_i, along with the bias term b.

For every neuron in the proposed architecture, the mapping between its input and output is defined as follows:

- **Input unit**

$$ \alpha^1 = t. \tag{3.1} $$

- **Hidden units**

$$\alpha_i^2 = \sigma_i(t \cdot w_i), \qquad i = 1, \ldots, n. \tag{3.2}$$

- **Output unit**

$$N(t, \theta) = \sum_{i=1}^{n} \left(\alpha_i^2 \cdot v_i\right) + b. \tag{3.3}$$

Compared to traditional numerical techniques, neural network-based approaches provide several benefits:

- Once trained, neural networks can predict the solution of a differential equation at any point within the defined domain.
- Owing to their expressive capability, neural network solutions are inherently analytical, meaning they are represented as continuously differentiable functions.
- These methods exhibit high accuracy in solutions while also demonstrating strong generalization abilities.
- The computational burden of neural network techniques does not grow significantly as the number of training points increases.

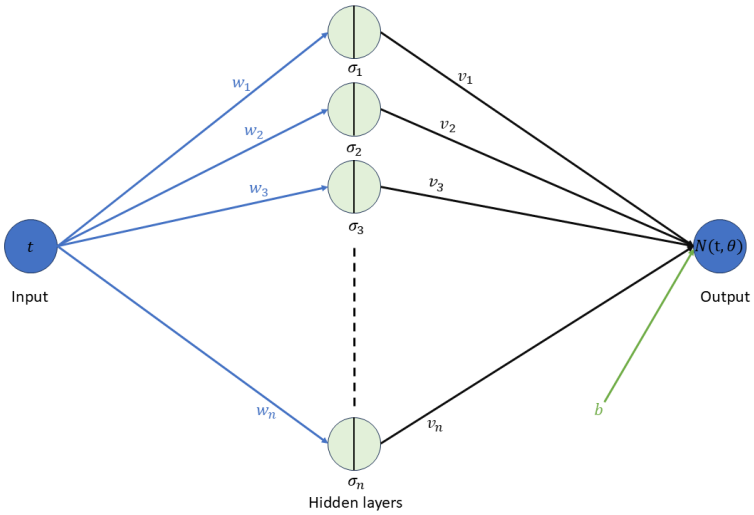


Figure 1. Well-labeled structure of single-layered ANN.

4. Problem statement

In this section, we will discuss AIE and FOVIDEs. We will first state FOVIDEs, and then we will talk about AIE. FOVIDEs are very important in many areas of science and engineering, especially when it comes to modeling systems that have memory effects and hereditary properties. These equations naturally emerge in the fields of physics, biology, finance, and engineering, where historical conditions affect current dynamics.

FIDEs, which include non-integer derivatives, are better at modeling real-world events than classical integer-order equations. The Caputo fractional derivative in these equations makes it easier to model strange diffusion, viscoelastic materials, the spread of diseases, and signal processing. In physics, these

kinds of equations show how heat moves through fractal media. In biomedical applications, they show how drugs spread and tumors grow. Fractional-order models have also been more effective at capturing system dynamics with long-range dependencies in control systems.

In this study, we consider an ordinary linear FOVIDE given by

$$P(t){}_0D_t^{\alpha_1}[u(t)] + Q(t){}_0I_t^{\alpha_2}[u(t)] = H(t), \tag{4.1}$$

$$1 < \alpha_1, \alpha_2 \leq 2,\ a \leq t \leq b,$$

subject to the initial condition

$$u(0) = \beta_1, \quad \text{and} \quad u'(0) = \beta_2,$$

where P, Q, and H are specified real-valued analytic functions on the continuous region (a, b).

We now discuss AIE. Imagine a particle sliding without friction under gravity along a smooth path in a vertical plane. If it takes a time $\Psi(t)$ to descend from height t down to a fixed endpoint (denoted 0), then finding the curve's equation is equivalent to solving an integral equation. In this setup, Abel's formulation can be expressed as:

$$\Psi(t) = \int_0^t \frac{\mathcal{F}(x)}{\sqrt{t-x}}\, dx. \tag{4.2}$$

Here, $\Psi(t)$ denotes the prescribed time function, while $\mathcal{F}(t)$ represents the unknown curve function that must be determined.

Even though these equations can be used in many different situations, it is still very hard to find analytical or numerical solutions because they have both differential and integral parts, and fractional derivatives are not local. The interaction of differentiation and integration in fractional order systems creates problems that standard numerical methods don't handle well.

This research aims to develop a novel computational approach for solving FOVIDEs. The primary objectives are to:

- Construct an accurate and computationally efficient scheme for approximating the solution.
- Demonstrate the effectiveness of the approach through numerical simulations and real-world applications.

5. Methodology

The main aim of this section is to utilize an ANN-based strategy to approximate solutions for a particular class of FIDEs. Accordingly, we first introduce an initial formulation of the ordinary linear FOVIDEs involving the Caputo derivative in the form:

$$P(t){}_0D_t^{\alpha_1}[u(t)] + Q(t){}_0I_t^{\alpha_2}[u(t)] = H(t), \tag{5.1}$$

$$1 < \alpha_1, \alpha_2 \leq 2,\ a \leq t \leq b,$$

under the influence of initial condition

$$u(0) = \beta_1, \quad \text{and} \quad u'(0) = \beta_2.$$

In this context, P, Q, and H denote real-valued analytic functions that are defined and continuous on the interval (a, b).

The proposed ANN–TPS framework combines the analytical representation of the TPS with the optimization capability of neural networks. The TPS provides a systematic approximation of the unknown solution, while the neural network is used to determine the unknown series coefficients by minimizing the residual of the governing equation at selected collocation points. This hybrid approach preserves the analytical structure of the solution and enables efficient computation for FIDEs. One of the most powerful tools for tackling functional equations is the power series technique, in which the unknown function $u(t)$ is expanded as a sum of terms that converge rapidly. Each term represents a component of the overall solution. In the setting of fractional differential equations, this expansion takes the form of a generalized power series:

$$u(t) = \sum_{i=0}^{\infty} a_i t^i. \tag{5.2}$$

Suppose we have initial condition $u(0) = \beta_1$ and $u'(0) = \beta_2$. Thus, Eq (5.2) can be written as:

$$u(t) = \beta_1 + \beta_2 t + \sum_{i=2}^{\infty} a_i t^i. \tag{5.3}$$

Substituting (5.3) into (5.1), we have the formula:

$$P(t){}_0D_t^{\alpha_1}\left[\beta_1 + \beta_2 t + \sum_{i=2}^{\infty} a_i t^i\right] +$$

$$Q(t){}_0I_t^{\alpha_2}\left[\beta_1 + \beta_2 t + \sum_{i=2}^{\infty} a_i t^i\right] = H(t). \tag{5.4}$$

Now, simplifying Eq (5.4), we get

$$P(t)\left[{}_0D_t^{\alpha_1}(\beta_1 + \beta_2 t) + \sum_{i=2}^{\infty} a_i({}_0D_t^{\alpha_1})(t^i)\right] +$$

$$Q(t)\left[{}_0I_t^{\alpha_2}(\beta_1 + \beta_2 t) + \sum_{i=2}^{\infty} a_i({}_0I_t^{\alpha_2})(t^i)\right] = H(t), \tag{5.5}$$

where

$${}_0D_t^{\alpha}[t^i] = \begin{cases} 0, & i \in \mathbb{Z}^+,\ i < \lceil\alpha\rceil, \\ \frac{\Gamma(i+1)}{\Gamma(i+1-\alpha)} t^{i-\alpha}, & t > c,\ i \in \mathbb{Z}^+,\ i \geq \lceil\alpha\rceil \end{cases}$$

and

$${}_0I_t^{\alpha}\left[t^k\right] = \frac{\Gamma(k+1)}{\Gamma(k+1+\alpha)} t^{k+\alpha}, \quad k \in \mathbb{Z}^+.$$

We can calculate the fractional derivative and integral term by term for Eq (5.3). Here, a_i (for $i \in \mathbb{N}$) are the unknown coefficients of the power series that must be determined. For $m \in \mathbb{Z}^+$, let

$\Omega = \{\lambda_0, \ldots, \lambda_{m-1}\}$ ($\lambda_j = [t_j, t_{j+1}]$) be a discretization of interval $[0, T]$ with the points $t_j = \frac{jT}{m}$, (for $j = 0, \ldots, m$). Now, let us substitute the collocation point t_j into the Eq (5.5), as below:

$$P(t_j)(\gamma_1(t_j) + \gamma_2(t_j)) + Q(t_j)(\eta_1(t_j) + \eta_2(t_j)) = H(t_j), \tag{5.6}$$

where,

$$\gamma_1(t_j) = {}_0D_t^{\alpha_1}(\beta_1 + \beta_2 t), \quad \gamma_2(t_j) = \sum_{i=2}^{\infty} a_i({}_0D_t^{\alpha_1})(t^i),$$

$$\eta_1(t_j) = {}_0I_t^{\alpha_2}(\beta_1 + \beta_2 t), \quad \text{and} \quad \eta_2(t_j) = \sum_{i=2}^{\infty} a_i({}_0I_t^{\alpha_2})(t^i)$$

$$\text{at} \quad t = t_j.$$

Next, we present the method for calculating the unknown coefficients in the power series expansion. For a specific objective, only the first $(n + 1)$ terms defined in Eq (5.3) are employed. Consequently, by expressing the solution as a TPS, we approximate it by determining the unknown coefficients a_i for $i = 1, 2, \ldots, n$. Ultimately, the solution $u(t)$ is approximated using this TPS.

$$u_n(t) = \beta_1 + \beta_2 t + \sum_{i=2}^{n} a_i t^i.$$

Below, we use the notations $w_i = t^{i-1}$, $v_i = a_i$, each $\sigma_i(t) = t$ (to mimic the polynomial basis of the TPS), and $b = \beta_1 + \beta_2 t$. With these assumptions, it can easily be argued that the output of the designed ANN is equivalent to the mentioned TPS (i.e., $N(t) = \beta_1 + \beta_2 t + \sum_{i=2}^{n} a_i t^i$).

Now, $\gamma_2(t_j)$ and $\eta_2(t_j)$ can be written as,

$$\gamma_2(t_j) = \sum_{i=2}^{n} a_i({}_0D_t^{\alpha_1})(t^i) \quad \text{and} \quad \eta_2(t_j) = \sum_{i=2}^{n} a_i({}_0I_t^{\alpha_2})(t^i) \tag{5.7}$$

$$\text{at} \quad t = t_j.$$

To train the network for optimization, it is essential to fine-tune the connection weights a_i to minimize the overall network error. The error is typically measured using the mean squared cost function. Therefore, for Eq (5.6), the network's instantaneous error can be expressed as:

$$E_j = \frac{1}{2}(P(t_j)(\gamma_1(t_j) + \gamma_2(t_j)) + Q(t_j)(\eta_1(t_j) + \eta_2(t_j)) - H(t_j))^2, \qquad j = 0, \ldots, m. \tag{5.8}$$

Finding ways to lower the prescribed error function across a group of node points is a very interesting problem. To do this, you need to use an effective error correction method that slowly brings the total network error down to zero. For more information on this topic, see [41].

In ANNs, learning means looking for the best weight values that will lower the overall error in a specific area. For this purpose, an incremental learning strategy is presented as a self-adaptive method for lowering a given cost function. This cost function is analytic, so the gradient descent method naturally comes to mind as a great way to come up with a good learning rule. An example of this method is coming up with an unsupervised backpropagation algorithm that updates the parameters

over and over again to lower the target function across the weight space. Here is a summary of how well the algorithm works:

Initially, the weight parameters a_i (with $i = 1, \ldots, n$) are randomly initialized. Successive iterations then leverage the training samples to adjust the network's connection weights through incremental updates aimed at minimizing the objective function. The standard algorithm proceeds as follows:

$$a_i(r+1) = a_i(r) + \Delta a_i(r), \quad i = 1, \ldots, n, \tag{5.9}$$

$$\Delta a_i(r) = -\eta \frac{\partial E_j}{\partial a_i} + \gamma \Delta a_i(r-1), \tag{5.10}$$

where

$$\frac{\partial E_j}{\partial a_i} = (P(t_j)(\gamma_1(t_j) + \gamma_2(t_j)) + Q(t_j)(\eta_1(t_j) + \eta_2(t_j)) - H(t_j))$$
$$((P(t_j)(\frac{\partial \gamma_1(t_j)}{\partial a_i} + \frac{\partial \gamma_2(t_j)}{\partial a_i}) + (Q(t_j)(\frac{\partial \eta_1(t_j)}{\partial a_i} + \frac{\partial \eta_2(t_j)}{\partial a_i})),$$

where

$$\frac{\partial \gamma_1(t_j)}{\partial a_i} = 0, \quad \frac{\partial \eta_1(t_j)}{\partial a_i} = 0, \quad \frac{\partial \gamma_2(t_j)}{\partial a_i} = \left({}_0D_t^{\alpha_1})(t^i\right)\Big|_{t=t_j},$$
$$\text{and} \quad \frac{\partial \eta_2(t_j)}{\partial a_i} = ({}_0I_t^{\alpha_2})\left(t^i\right)\Big|_{t=t_j}.$$

In this case, η is a small, constant learning rate, and γ is the momentum coefficient. Backpropagation depends on gradient descent, so its performance is very sensitive to the values of these parameters. If you choose the wrong values at the start, learning will take longer. In this case, the index r in $a_i(r)$ stands for the iteration number, and the subscript j in x_j stands for the training point.

We use the chain rule to find the partial derivatives, which gives us the learning rule we want for the weights in the hidden layer. If a solution is guaranteed to exist, one can use the derived relationships to get close to it in a limited number of steps. The word "cycling" usually refers to the learning process in which the network goes through all of the training node points once. In practice, you need to go through the set of training points several times to get an accurate solution vector.

Algorithm:

- **Initialization**
  - Define the number of terms in the TPS approximation: n.
  - Select $m + 1$ training points in the given interval: $\{t_0, t_1, ..., t_m\}$.
  - Initialize weights $\{a_1, a_2, ..., a_n\}$ randomly.
  - Set learning rate η and momentum constant γ.

- **Construct the TPS approximation and the approximate solution**
  - Define the approximate solution:

$$u_n(t) = \beta_1 + \beta_2 t + \sum_{i=2}^{n} a_i t^i.$$

– Define the objective function:

$$E_j = \frac{1}{2}(P(t_j)(\gamma_1(t_j) + \gamma_2(t_j)) + Q(t_j)(\eta_1(t_j) + \eta_2(t_j)) - H(t_j))^2,$$

$$j = 0, \ldots, m.$$

- **Train the neural network**
  - Repeat for a fixed number of iterations or until convergence:
    - Compute the gradient of the objective function.
    - Update weights using gradient descent with momentum.
- **Output**
  - Trained weights $a_1, a_2, ..., a_n$.
  - Approximate solution $u_n(t)$ for the given problem.

End Algorithm.

6. Numerical examples

There are six numerical examples that show the theoretical development is correct. The results of the proposed method are compared to the exact analytical solutions. We ran all of the examples below on Google Colab with a T4 GPU as the hardware accelerator and Python 3 as the runtime type. The following specifications were used in these simulations:

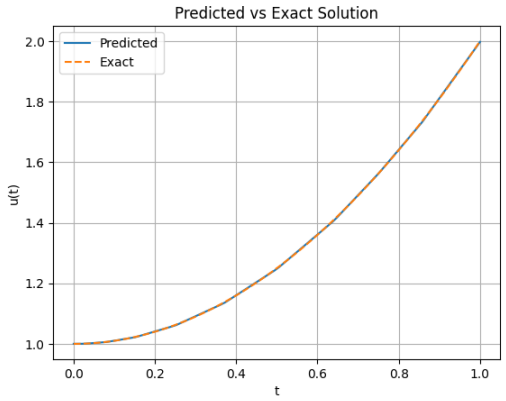
- Number of neurons: $n = 20$.
- Number of training points: $m = 200$.
- Learning rate: $\eta = 0.1$.
- Momentum constant: $\gamma = 0.05$.
- Iterations=10,000.

Example 6.1. *Consider the following higher-order linear FOVIDE:*

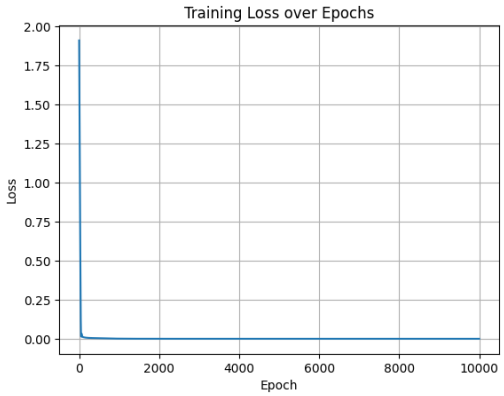
$$D_t^{1.5}[u(t)] + I_{0,t}^2[u(t)] = \frac{\Gamma(3)}{\Gamma\left(\frac{15}{10}\right)} t^{\frac{1}{2}} + \frac{\Gamma(3)}{\Gamma(5)} t^4, \quad 0 \le t \le 1, \tag{6.1}$$

$$u(0) = 1, \quad \textit{and} \quad u'(0) = 0. \tag{6.2}$$

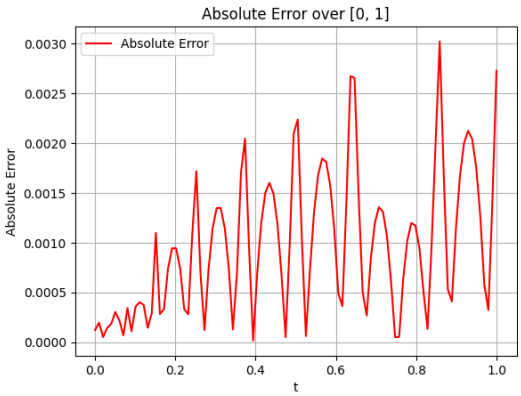
The exact solution is $u(x) = t^2 + 1$. The proposed method's performance is shown in Figure 2(a),(c). Figure 2(a) shows the exact and predicted solutions, showing that they are very close to each other. Figure 2(c) shows the absolute error, which further proves that the approximation is correct. Figure 2(b) shows how the training loss gets closer together over time, which shows that the model is being trained well. Table 1 shows the exact and predicted values for different times t, as well as the absolute errors for each. Table 4 shows the method's computational efficiency by showing how long it takes to run in seconds.



(a) Exact vs predicted for Example 6.1.



(b) Training loss for Example 6.1.



(c) Absolute error for Example 6.1.

Figure 2. Exact versus predicted solution, training loss convergence, and absolute error for Example 6.1.

Table 1. Exact and predicted values with absolute error for Examples 6.1–6.3.

	Example 6.1			Example 6.2			Example 6.3		
t	Exact	Predicted	Absolute Error	Exact	Predicted	Absolute Error	Exact	Predicted	Absolute Error
0.0	1.000	1.000	1.20e-04	0.000	0.000	1.68e-04	0.000	0.000	4.30e-04
0.1	1.010	1.010	3.41e-04	0.010	0.010	1.06e-04	0.010	0.010	2.44e-04
0.2	1.040	1.041	9.60e-04	0.040	0.040	2.87e-04	0.040	0.040	2.61e-04
0.3	1.090	1.091	1.31e-03	0.090	0.090	1.39e-04	0.090	0.090	1.45e-04
0.4	1.160	1.160	4.36e-04	0.160	0.160	3.01e-04	0.160	0.160	4.76e-04
0.5	1.250	1.247	2.74e-03	0.250	0.250	4.91e-04	0.250	0.250	1.39e-04
0.6	1.360	1.361	9.00e-04	0.360	0.360	4.26e-04	0.360	0.359	1.25e-03
0.7	1.490	1.491	1.27e-03	0.490	0.490	4.40e-04	0.490	0.488	1.90e-03
0.8	1.640	1.641	1.14e-03	0.640	0.640	4.01e-04	0.640	0.638	1.82e-03
0.9	1.810	1.811	1.21e-03	0.810	0.810	1.09e-04	0.810	0.809	9.14e-04
1.0	2.000	1.997	2.73e-03	1.000	1.000	2.93e-04	1.000	0.998	1.75e-03

Example 6.2. *Consider the following fractional initial value problem:*

$$D_t^2 [u(t)] + D_t^{\frac{1}{2}} [u(t)] + u(t) - I_{0,t}^1 [(t-s)u(s)] = R(t), \tag{6.3}$$

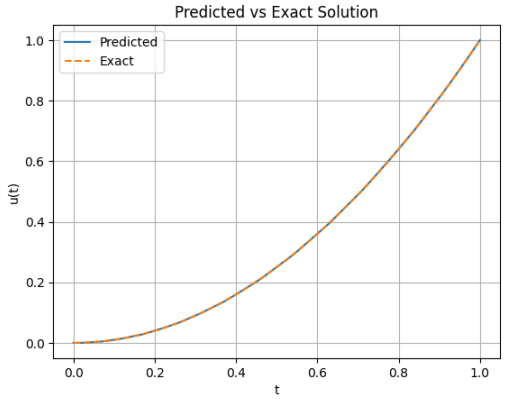
$$0 \leq t \leq 1,$$

where

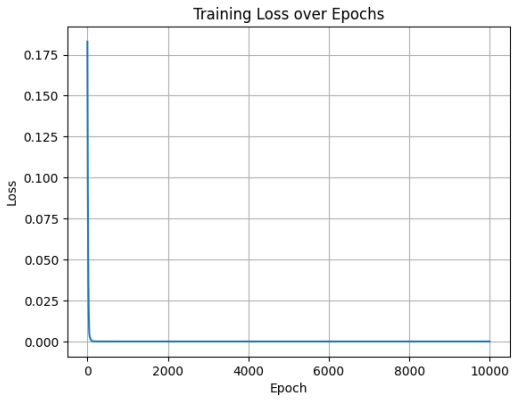
$$R(t) = -\frac{1}{12}t^4 + t^2 + \frac{2}{\Gamma\left(\frac{5}{2}\right)}t^{\frac{3}{2}} + 2,$$

$$u(0) = 0, \quad \textit{and} \quad u'(0) = 0. \tag{6.4}$$

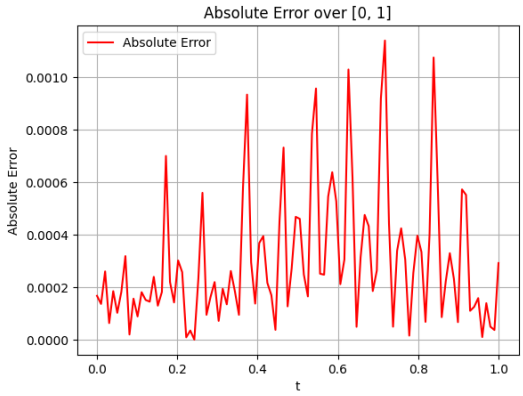
The exact solution is $u(x) = t^2$. The proposed method's performance is shown in Figure 3(a),(c). In particular, Figure 3(a) compares the exact and predicted solutions and shows that they are very close to each other. Figure 3(c) shows the absolute error, which further shows that the approximation is correct. Figure 3(b) shows the convergence of the training loss over iterations, which shows that the model is being trained well. Table 1 shows the absolute errors for different time points t by comparing the exact and predicted values. Table 4 shows the method's computational efficiency by showing how long it takes to run in seconds.



(a) Exact vs predicted for Example 6.2.



(b) Training loss for Example 6.2.



(c) Absolute error for Example 6.2.

Figure 3. Exact versus predicted solution, training loss convergence, and absolute error for Example 6.2.

Example 6.3. *Consider the FIDE of the form*

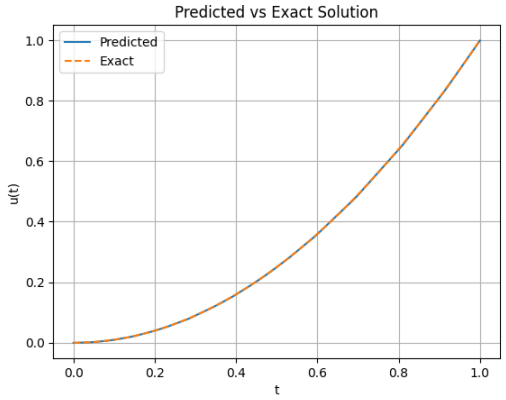
$$ {}^{c}D_{t}^{\alpha}u(t) = \frac{\Gamma(3)}{\Gamma\left(2-\frac{1}{3}+1\right)}t^{2-\frac{1}{3}} - \frac{t^5}{4} + \int_0^t tr\, u(r)\, dr \tag{6.5} $$

subject to

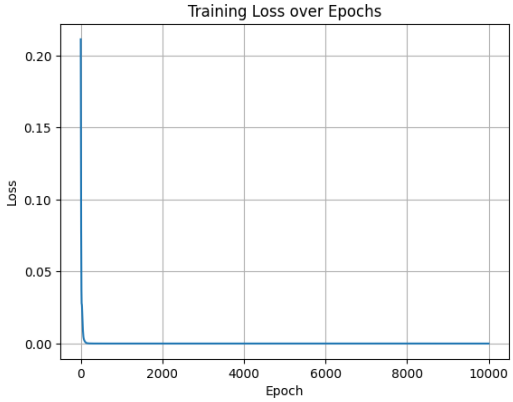
$$ u(0) = 0, \quad and \quad u'(0) = 0, \tag{6.6} $$

$$ u(t) = t^2 \quad when \quad \alpha = \frac{1}{3}. $$

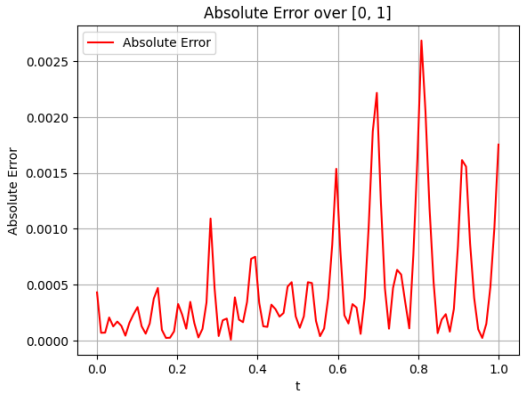
Figure 4(a),(c) shows how well the suggested method works. Figure 4(a) compares the exact and predicted solutions, showing that they are very close to each other. Figure 4(c) shows the absolute error, which further proves that the approximation is very accurate. Figure 4(b) shows that the training loss converges over iterations, which means that the model is being trained well. Table 1 shows a comparison of exact and predicted values for different time points t, as well as the absolute errors that go with them. The computational efficiency of the method is summarized in Table 4, which reports the execution time in seconds.



(a) Exact vs predicted for Example 6.3.



(b) Training loss for Example 6.3.



(c) Absolute error for Example 6.3.

Figure 4. Exact versus predicted solution, training loss convergence, and absolute error for Example 6.3.

Example 6.4. *Consider the FIDE of the form*

$$ {}^{c}D_{t}^{\alpha}u(t) = 1 - \frac{(2\alpha+3)}{\Gamma(\alpha+3)}t^{\alpha+2} + \int_{0}^{t}(t+r)\,u(r)\,dr \tag{6.7} $$

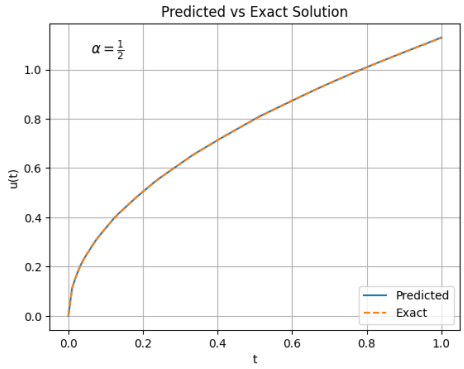
subject to

$$ u(0) = 0, \tag{6.8} $$
$$ u(t) = \frac{t^{\alpha}}{\Gamma(\alpha+1)}. $$

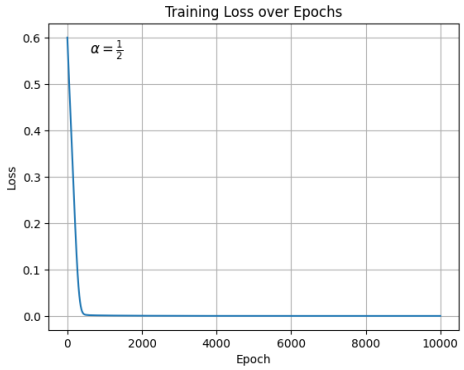
Figures 5(a),(c),(e) and 6(a)–(c), for different values of α, illustrate the performance of the proposed method. Figure 5(a),(c),(e) shows the exact and predicted solutions, showing that they are very close to each other. Figure 6(a)–(c) shows the absolute error, which further proves that the approximation is correct. Figure 5(f) shows how the training loss converges over time, which means that the model is being trained well. Table 2 shows the absolute errors for different time points t as well as a comparison of exact and predicted values. Table 4 shows how fast the method works by reporting the time it takes to run in seconds.

Table 2. Exact and predicted values with absolute error for Example 6.4 with different α's.

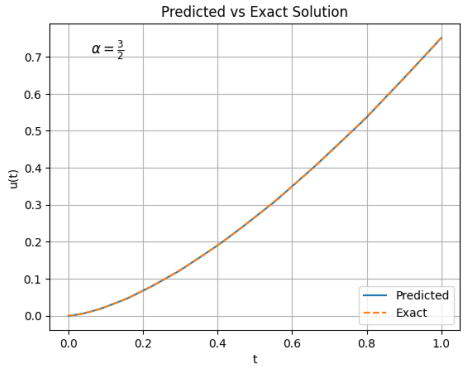
	For $\alpha = \frac{1}{2}$			For $\alpha = \frac{3}{2}$			For $\alpha = \frac{5}{2}$		
t	Exact	Predicted	Absolute Error	Exact	Predicted	Absolute Error	Exact	Predicted	Absolute Error
0.0	0.000	0.001	7.78e-04	0.000	0.00	7.25e-04	0.000	0.000	5.15e-06
0.1	0.357	0.356	1.01e-03	0.024	0.024	6.05e-05	0.001	0.001	6.78e-06
0.2	0.505	0.504	6.81e-04	0.067	0.068	2.32e-04	0.005	0.005	1.10e-04
0.3	0.618	0.618	4.32e-04	0.124	0.123	8.78e-04	0.015	0.015	2.17e-04
0.4	0.714	0.713	7.04e-04	0.190	0.190	2.92e-04	0.030	0.031	1.36e-04
0.5	0.798	0.799	1.29e-03	0.266	0.266	3.13e-04	0.053	0.053	4.88e-04
0.6	0.874	0.873	6.85e-04	0.350	0.350	2.74e-04	0.084	0.084	1.48e-04
0.7	0.944	0.944	1.67e-04	0.441	0.441	5.61e-04	0.123	0.123	3.14e-05
0.8	1.009	1.009	5.82e-05	0.538	0.537	1.49e-03	0.172	0.172	4.28e-04
0.9	1.070	1.070	5.42e-04	0.642	0.643	5.38e-04	0.231	0.231	6.25e-04
1.0	1.128	1.130	1.40e-03	0.752	0.751	1.05e-03	0.301	0.300	6.64e-04



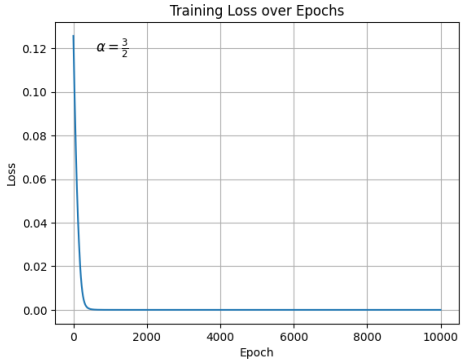
(a) Exact vs predicted for Example 6.4 when $\alpha = \frac{1}{2}$.



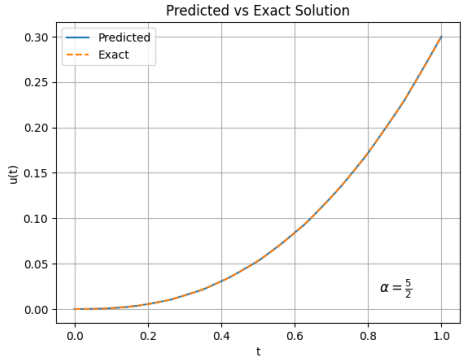
(b) Training loss for Example 6.4 when $\alpha = \frac{1}{2}$.



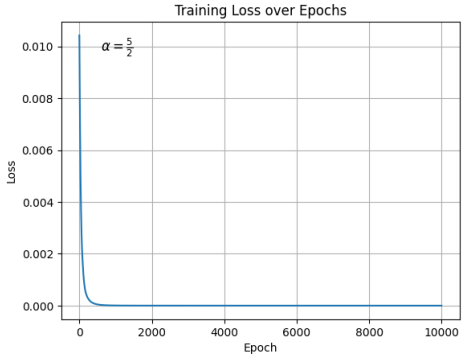
(c) Exact vs predicted for Example 6.4 when $\alpha = \frac{3}{2}$.



(d) Training loss for Example 6.4 when $\alpha = \frac{3}{2}$.

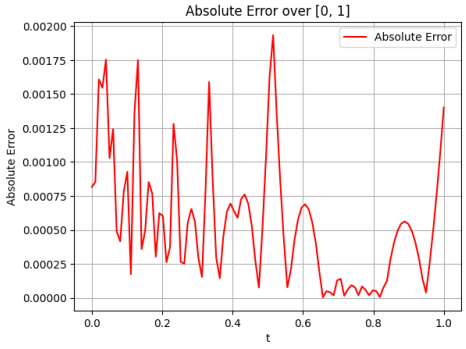


(e) Exact vs predicted for Example 6.4 when $\alpha = \frac{5}{2}$.

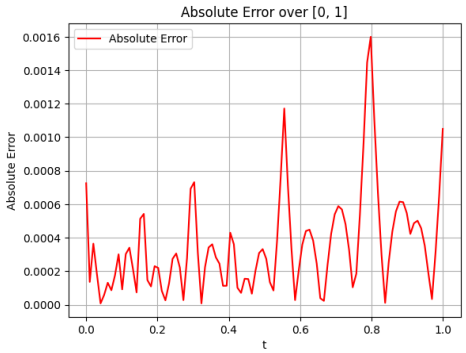


(f) Training loss for Example 6.4 when $\alpha = \frac{5}{2}$.

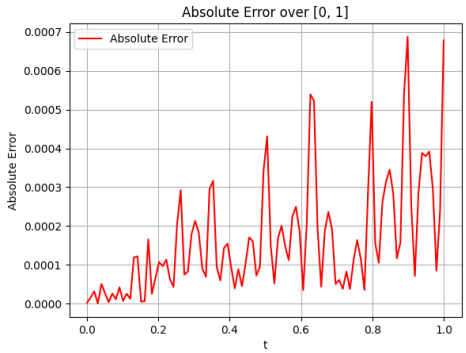
Figure 5. Exact versus predicted solution and training loss convergence for Example 6.4 with different α's



(a) Absolute error for Example 6.4 when $\alpha = \frac{1}{2}$.



(b) Absolute error for Example 6.4 when $\alpha = \frac{3}{2}$.



(c) Absolute error for Example 6.4 when $\alpha = \frac{5}{2}$.

Figure 6. Absolute errors for Example 6.4 with different α's.

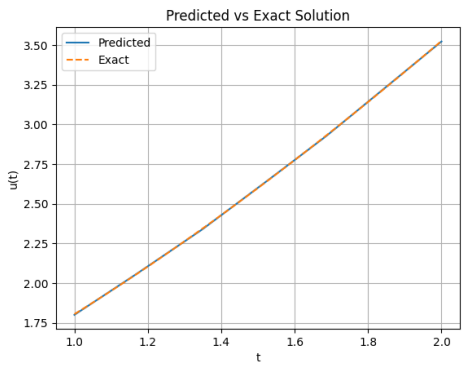
Example 6.5. *Consider the following AIE:*

$$\left[1 + t + t^2 = \int_0^t \frac{u(r)}{\sqrt{t-r}}\, dr\right]. \tag{6.9}$$

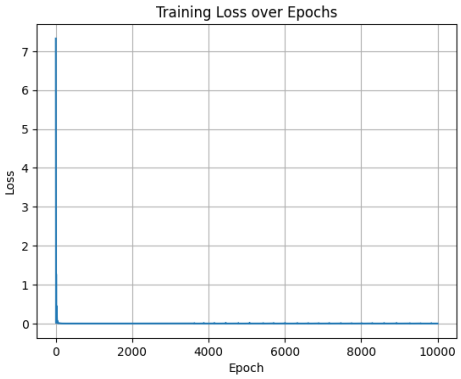
The exact solution is given by

$$u(t) = \frac{1}{\pi}\left[t^{-1/2} + 2t^{1/2} + \frac{8}{3}t^{3/2}\right].$$

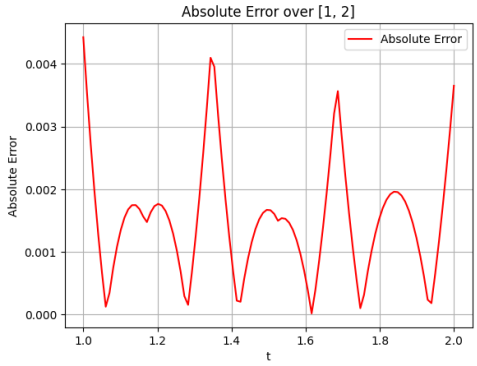
Figure 7(a),(c) shows how well the suggested method works. Figure 7(a) shows the exact and predicted solutions, showing that they are very close to each other. Figure 7(c) shows the absolute error, which further proves that the approximation is correct. Figure 7(b) shows how the training loss changes over time, which shows that the model is being trained well. Table 3 shows the differences between the exact and predicted values at different times t, as well as the absolute errors that go with them. Table 4 shows the method's computational efficiency by showing how long it takes to run in seconds.



(a) Exact vs predicted for Example 6.5.



(b) Training loss for Example 6.5.



(c) Absolute error for Example 6.5.

Figure 7. Exact versus predicted solution, training loss convergence, and absolute error for Example 6.5.

Table 3. Exact and predicted values with absolute error for Examples 6.5–6.7.

	Example 6.5			Example 6.6			Example 6.7		
t	Exact	Predicted	Absolute Error	Exact	Predicted	Absolute Error	Exact	Predicted	Absolute Error
1.0	1.804	1.799	4.42e-03	1.000	1.001	8.08e-04	0.637	0.638	9.55e-04
1.1	1.950	1.952	1.32e-03	1.032	1.032	3.03e-04	0.668	0.667	3.89e-04
1.2	2.104	2.106	1.76e-03	1.063	1.063	7.01e-05	0.697	0.697	3.49e-04
1.3	2.263	2.262	1.06e-03	1.091	1.091	1.50e-04	0.726	0.727	9.03e-04
1.4	2.428	2.427	9.13e-04	1.119	1.119	4.06e-04	0.753	0.753	8.43e-05
1.5	2.599	2.601	1.67e-03	1.145	1.144	3.88e-04	0.780	0.779	4.04e-04
1.6	2.775	2.775	5.78e-04	1.170	1.170	5.01e-05	0.805	0.805	1.44e-04
1.7	2.956	2.953	2.67e-03	1.193	1.194	4.59e-04	0.830	0.830	4.39e-05
1.8	3.141	3.143	1.56e-03	1.216	1.216	3.13e-04	0.854	0.854	3.62e-04
1.9	3.331	3.333	1.20e-03	1.239	1.238	2.49e-04	0.878	0.877	1.09e-04
2.0	3.526	3.523	3.65e-03	1.260	1.260	5.77e-04	0.900	0.901	7.53e-04

Table 4. Execution time of the proposed method on Google Colab using a T4 GPU.

Examples	Example 6.1	Example 6.2	Example 6.3	Example 6.4 ($\alpha = \frac{1}{2}$)	Example 6.4 ($\alpha = \frac{3}{2}$)	Example 6.4 ($\alpha = \frac{5}{2}$)	Example 6.5	Example 6.6	Example 6.7
Time	35 s	27 s	42 s	39 s	33 s	37 s	33 s	33 s	32 s

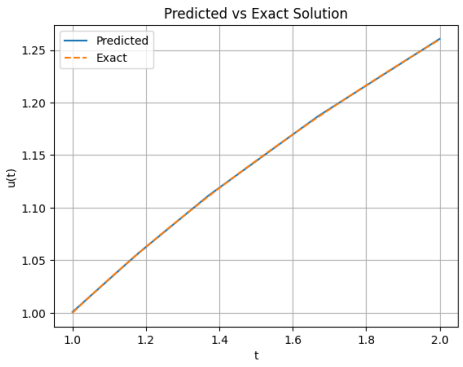
Example 6.6. *Consider the following fractional order AIE:*

$$\frac{2\pi}{3\sqrt{3}}t = \int_0^t \frac{u(r)}{(t-r)^{1/3}}\,dr.$$

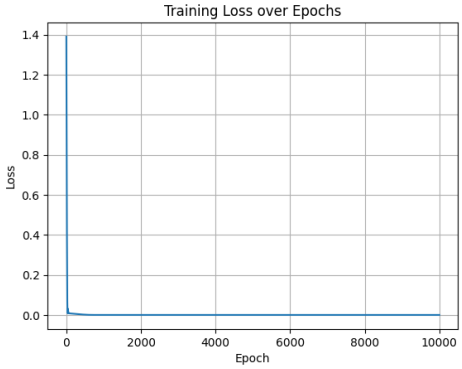
The exact solution is given by

$$u(t) = \frac{2\pi}{3\sqrt{3}\,\Gamma\left(\frac{4}{3}\right)\Gamma\left(\frac{2}{3}\right)}t^{1/3}.$$

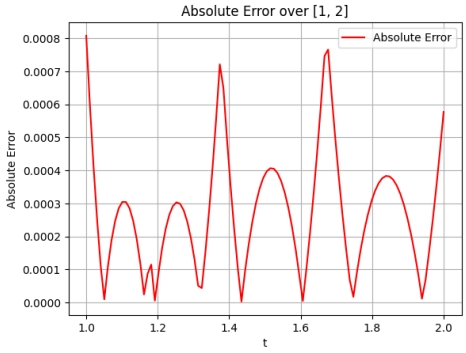
Figure 8(a),(c) shows how well the proposed method works. Figure 8(a) shows the exact and predicted solutions, showing that they are very close to each other. Figure 8(c) shows the absolute error, which further proves that the approximation is correct. Figure 8(b) shows how the training loss changes over time, which shows that the model is being trained well. Table 3 shows a comparison of exact and predicted values for different time points t, as well as the absolute errors that go with them. Table 4 shows the method's computational efficiency by showing how long it takes to run in seconds.



(a) Exact vs predicted for Example 6.6.



(b) Training loss for Example 6.6.



(c) Absolute error for Example 6.6.

Figure 8. Exact versus predicted solution, training loss convergence, and absolute error for Example 6.6.

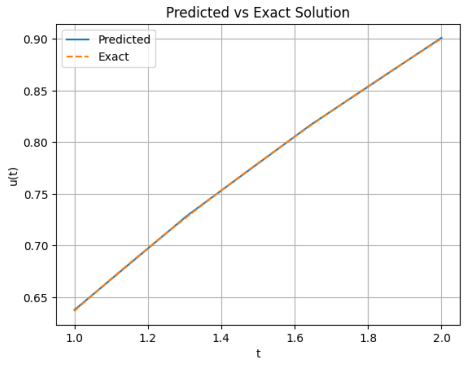
Example 6.7. *Consider the following AIE:*

$$t = \int_0^t \frac{u(r)}{(t-r)^{1/2}}\,dr.$$

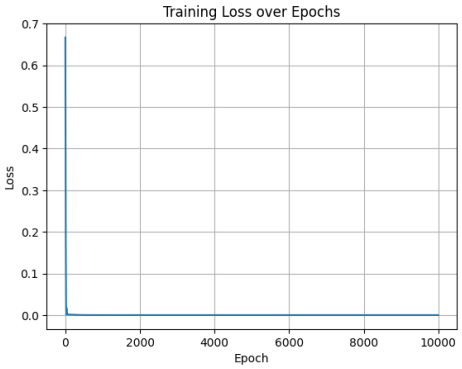
The exact solution is given by

$$u(t) = \frac{2}{\pi} t^{1/2}.$$

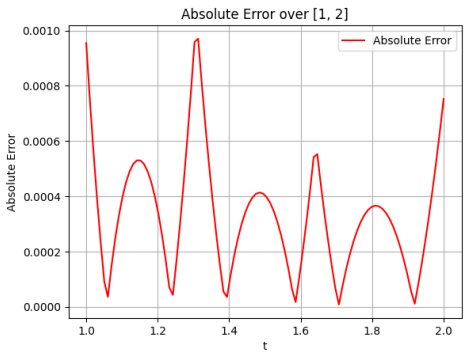
Figure 9(a),(c) illustrates the performance of the proposed method. Specifically, Figure 9(a) compares the exact and predicted solutions, demonstrating a high degree of accuracy, while Figure 9(c) presents the absolute error, further confirming the precision of the approximation. Additionally, the convergence of the training loss over iterations is depicted in Figure 9(b), indicating effective model training. Table 3 provides a comparison of exact and predicted values for various time points t, along with the corresponding absolute errors. The computational efficiency of the method is summarized in Table 4, which reports the execution time in seconds.



(a) Exact vs predicted for Example 6.7.



(b) Training loss for Example 6.7.



(c) Absolute error for Example 6.7.

Figure 9. Exact versus predicted solution, training loss convergence, and absolute error for Example 6.7.

7. Results and discussion

The proposed ANN–power series (ANN–PS) method was applied to several FOVIDEs and AIE. The obtained results show excellent agreement with the exact solutions, achieving absolute errors in the range of 10^{-04} – 10^{-06}, confirming the method's high accuracy and fast convergence. Compared to conventional numerical methods, the ANN–PS approach avoids symbolic computation and truncation errors by learning the series coefficients adaptively. It also outperforms PINNs, as fractional derivatives are difficult and computationally expensive to handle in PINN frameworks, while they are naturally

incorporated here. The method is computationally efficient, requiring fewer iterations and less memory, and provides an analytical, closed-form solution. The gradient descent-momentum algorithm ensures stable and smooth convergence without oscillations. Overall, the proposed ANN–PS framework offers a robust, accurate, and interpretable alternative to existing numerical and neural network-based methods for FIDEs.

8. Conclusions

The primary objective of this paper is to develop a numerical framework for addressing FOVIDEs and AIE. This was achieved by progressively integrating neural networks with the power series framework, resulting in an approximate solution articulated as a generalized TPS. Seven numerical examples, with both the predicted and actual values, show how useful it is. Additionally, the absolute errors for specific values of t are presented in Tables 1–3, highlighting the method's accuracy. The numerical experiments show that, with an appropriate selection of step parameters such as the learning rate and momentum constant, the proposed algorithm is an effective tool for determining the unknown series coefficients. Furthermore, the simulation results, when compared with exact solutions, underscore the method's reliability. Ultimately, the iterative framework presented here is adaptable to a variety of other sophisticated fractional order problems, thereby expanding its range of applicability. Although the proposed ANN–TPS framework demonstrates good accuracy and efficiency for the considered problems, the method assumes that the solution can be well approximated by a TPS. For problems with limited smoothness, higher truncation orders may be required, which could increase computational cost. Furthermore, the present study focuses on one-dimensional fractional Volterra integro-differential and Abel-type equations. Future work will explore extensions to more complex nonlinear systems, higher-dimensional fractional models, and systematic comparisons with other neural-network-based solvers.

Author contributions

M. Noor, M. Malik and M. Sajid: Conceptualization, methodology; M. Noor and M. Malik: Formal analysis; M. Noor: Investigation, writing-original draft preparation; M. Malik: Supervision; M. Malik and M. Sajid: Writing-review and editing; M. Sajid: Funding. All authors have read and approved the final version of the manuscript for publication.

Use of Generative-AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

The Researchers would like to thank the Deanship of Graduate Studies and Scientific Research at Qassim University for financial support (QU-APC-2026).

Conflict of interest

The authors declare that they have no known competing interests.

References

1. A. A. Kilbas, H. M. Srivastava, J. J. Trujillo, *Theory and applications of fractional differential equations*, Elsevier, 2006.
2. I. Podlubny, *Fractional differential equations, mathematics in science and engineering*, New York: Academic press, 1999.
3. F. Mainardi, *Fractional calculus and waves in linear viscoelasticity: An introduction to mathematical models*, World Scientific, 2022.
4. R. Herrmann, Folded potentials in cluster physics-a comparison of Yukawa and Coulomb potentials with Riesz fractional integrals, *J. Phys. A-Math. Theor.*, **46** (2013), 405203. https://doi.org/10.1088/1751-8113/46/40/405203
5. T. Odzijewicz, A. B. Malinowska, D. F. Torres, Fractional calculus of variations in terms of a generalized fractional integral with applications to physics, *Abst. Appl. Anal.*, 2012. https://doi.org/10.1155/2012/871912
6. S. Kumar, A. Kumar, B. Mohan, Evolutionary dynamics of solitary wave profiles and abundant analytical solutions to a (3+ 1)-dimensional burgers system in ocean physics and hydrodynamics, *J. Ocean Eng. Sci.*, **8** (2023), 1–14. https://doi.org/10.1016/j.joes.2021.11.002
7. S. Kumar, B. Mohan, R. Kumar, Lump, soliton, and interaction solutions to a generalized two-mode higher-order nonlinear evolution equation in plasma physics, *Nonlinear Dynam.*, **110** (2022), 693–704. https://doi.org/10.1007/s11071-022-07647-5
8. R. Hilfer, *Applications of fractional calculus in physics*, World scientific, 2000.
9. R. C. Koeller, Applications of fractional calculus to the theory of viscoelasticity, *J. Appl. Mech.*, **51** (1984), 299–307. https://doi.org/10.1115/1.3167616
10. J. G. Lu, G. Chen, A note on the fractional-order Chen system, *Chaos Soliton. Fract.*, **27** (2006), 685–688. https://doi.org/10.1016/j.chaos.2005.04.037
11. L. Debnath, Recent applications of fractional calculus to science and engineering, *Int. J. Math. Math. Sci.*, 2003, 3413–3442. https://doi.org/10.1155/S0161171203301486
12. S. Kumar, B. Mohan, A generalized nonlinear fifth-order KdV-type equation with multiple soliton solutions: Painlevé analysis and Hirota Bilinear technique, *Phys. Scr.*, **97** (2022), 125214. https://doi.org/10.1088/1402-4896/aca2fa
13. H. Jafari, V. Daftardar-Gejji, Solving a system of nonlinear fractional differential equations using Adomian decomposition, *J. Comput. Appl. Math.*, **196** (2006), 644–651. https://doi.org/10.1016/j.cam.2005.10.017
14. I. Hashim, O. Abdulaziz, S. Momani, Homotopy analysis method for fractional IVPs, *Commun. Nonlinear Sci. Numer. Simul.*, **14** (2009), 674–684. https://doi.org/10.1016/j.cnsns.2007.09.014

15. S. Kumar, I. Hamid, M. A. Abdou, Some specific optical wave solutions and combined other solitons to the advanced (3+ 1)-dimensional Schrödinger equation in nonlinear optical fibers, *Opt. Quant. Electron.*, **55** (2023), 728. https://doi.org/10.1007/s11082-023-04976-6

16. X. B. Yin, S. Kumar, D. Kumar, A modified homotopy analysis method for solution of fractional wave equations, *Adv. Mech. Eng.*, **7** (2015). https://doi.org/10.1177/1687814015620330

17. M. Hamarsheh, A. I. Ismail, Z. Odibat, An analytic solution for fractional order Riccati equations by using optimal homotopy asymptotic method, *Appl. Math. Sci.*, **10** (2016), 1131–1150. https://doi.org/10.12988/ams.2016.6118

18. R. Gorenflo, S. Vessella, *Abel integral equations*, Berlin: Springer, 1991.

19. M. Rahman, *Integral equations and their applications*, WIT press, 2007.

20. S. De, B. N. Mandal, A. Chakrabarti, Use of Abel integral equations in water wave scattering by two surface-piercing barriers, *Wave Motion*, **47** (2010), 279–288. https://doi.org/10.1016/j.wavemoti.2009.12.002

21. I. E. Lagaris, A. Likas, D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, *IEEE T. Neural Network.*, **9** (1998), 987–1000. https://doi.org/10.1109/72.712178

22. A. K. Sahoo, S. Chakraverty, A neural network approach for the solution of Van der Pol-Mathieu-Duffing oscillator model, *Evol. Intell.*, **17** (2024), 1425–1435. https://doi.org/10.1007/s12065-023-00835-1

23. R. Ahmed, M. Malik, Dynamics of an SIRD model with Crowley-Martin incidence rate by using the machine learning approach, *Int. J. Comput. Math.*, **102** (2025), 1506–1527. https://doi.org/10.1080/00207160.2025.2505745

24. M. Noor, M. Malik, Deep neural network approach to solve coupled differential equations with multiple delays, *Phys. Scr.*, **100** (2025), 075259. https://doi.org/10.1088/1402-4896/ade7c2

25. Z. Liu, Y. Yang, Q. Cai, Neural network as a function approximator and its application in solving differential equations, *Appl. Math. Mech.*, **40** (2019), 237–248. https://doi.org/10.1007/s10483-019-2429-8

26. L. P. Aarts, P. Van Der Veer, Neural network method for solving partial differential equations, *Neural Process. Lett.*, **14** (2001), 261–271. https://doi.org/10.1023/A:1012784129883

27. M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *J. Comput. Phys.*, **378** (2019), 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

28. Y. Bo, D. Tian, X. Liu, Y. Jin, Discrete maximum principle and energy stability of the compact difference scheme for two-dimensional Allen–Cahn equation, *J. Funct. Space.*, **2022** (2022), 8522231. https://doi.org/10.1155/2022/8522231

29. S. M. Sivalingam, P. Kumar, V. Govindaraj, A Chebyshev neural network-based numerical scheme to solve distributed-order fractional differential equations, *Comput. Math. Appl.*, **164** (2024), 150–165. https://doi.org/10.1016/j.camwa.2024.04.005

30. T. Allahviranloo, A. Jafarian, R. Saneifard, N. Ghalami, S. M. Nia, F. Kiani, et al., An application of artificial neural networks for solving fractional higher-order linear integro-differential equations, *Bound. Value Probl.*, **2023** (2023), 74. https://doi.org/10.1186/s13661-023-01762-x
31. A. Jafarian, M. Mokhtarpour, D. Baleanu, Artificial neural network approach for a class of fractional ordinary differential equation, *Neural Comput. Appl.*, **28** (2017), 765–773. https://doi.org/10.1007/s00521-015-2104-8
32. S. Mall, S. Chakraverty, Artificial neural network approach for solving fractional order initial value problems, *arXiv preprint*, 2018.
33. S. M. Sivalingam, P. Kumar, V. Govindaraj, A novel numerical scheme for fractional differential equations using extreme learning machine, *Physica A*, **622** (2023), 128887. https://doi.org/10.1016/j.physa.2023.128887
34. M. Pakdaman, A. Ahmadian, S. Effati, S. Salahshour, D. Baleanu, Solving differential equations of fractional order using an optimization technique based on training artificial neural network, *Appl. Math. Comput.*, **293** (2017), 81–95. https://doi.org/10.1016/j.amc.2016.07.021
35. I. O. Isah, N. Senu, A. Ahmadian, A high-performance neural network algorithm using a Legendre ensemble-based extreme learning machine for solving fractional partial differential equations, *J. Comput. Appl. Math.*, **477** (2025). https://doi.org/10.1016/j.cam.2025.117220
36. A. Yıldırım, H. Koçak, Homotopy perturbation method for solving the space-time fractional advection-dispersion equation, *Adv. Water Resour.*, **32** (2009), 1711–1716. https://doi.org/10.1016/j.advwatres.2009.09.003
37. D. Baleanu, K. Diethelm, E. Scalas, J. J. Trujillo, *Fractional calculus: Models and numerical methods*, World Scientific, 2012.
38. X. J. Yang, *Advanced local fractional calculus and its applications*, World Science, New York, NY, USA, 2012.
39. D. Graupe, *Principles of artificial neural networks*, World Scientific, 2013.
40. M. Hanss, Applied fuzzy arithmetic, Springer-Verlag, Berlin, Heidelberg, 2005.
41. M. H. Hassoun, *Fundamentals of artificial neural networks*, MIT Press, Cambridge, 1995.

AIMS Press