symmetry

MDPI

Article

An Artificial Neural Network Approach for Solving Space Fractional Differential Equations

Pingfei Dai [1] and Xiangyu Yu [2,*]

[1] School of Mathematics, Hangzhou Normal University, Hangzhou 311121, China; pfdai@hznu.edu.cn
[2] UniDT Technology (Shanghai) Co., Ltd., Shanghai 200040, China
* Correspondence: xiangyu.yu@unidt.com

Abstract: The linear algebraic system generated by the discretization of fractional differential equations has asymmetry, and the numerical solution of this kind of problems is more complex than that of symmetric problems due to the nonlocality of fractional order operators. In this paper, we propose the artificial neural network (ANN) algorithm to approximate the solutions of the fractional differential equations (FDEs). First, we apply truncated series expansion terms to replace unknown function in equations, then we use the neural network to get series coefficients, and the obtained series solution can make the norm value of loss function reach a satisfactory error. In the part of numerical experiments, the results verify that the proposed ANN algorithm can make the numerical results achieve high accuracy and good stability.

Keywords: fractional differential equations; power series expansion; learning algorithm; approximate solutions

Citation: Dai, P.; Yu, X. An Artificial Neural Network Approach for Solving Space Fractional Differential Equations. *Symmetry* **2022**, *14*, 535. https://doi.org/10.3390/sym14030535

Academic Editor: Juan Luis García Guirao

Received: 13 February 2022
Accepted: 3 March 2022
Published: 6 March 2022

1. Introduction

Fractional calculus is a field to study the applications of arbitrary order integral and differential and the mathematical properties. Fractional calculus operator is very suitable for describing materials with genetic properties and memory. Since the beginning of the 21st century, it has been widely applied in many regions, such as high energy physics, anomalous diffusion, complex viscoelastic materials, geophysics, biomedical engineering and others (see [1–6]). Subsequently, the research of fractional differential equations has attracted extensive attention, and gradually become a new and active research field.

Due to the nonlocality of fractional operators, the numerical solutions of this kind of problems are more complex than symmetric problems. Although it is hard to theoretical analyze the analytical solutions of fractional differential equations, researchers are more and more interested in its approximate methods and numerical solution. In 2006, Meerschaert Tadjern [7] proposed a finite difference approximation method to discretize the two-dimensional fractional dispersion equation with variable coefficients; in Ref. [8], Sun et al. gave a detailed introduction of the finite difference method for variable-order time fractional diffusion equations in a finite domain; in the literature [9], Deng proposed a kind of finite element method to solve the time space fractional Fokker–Planck equation; In Ref. [10], Li et al. studied three typical Caputo type partial differential equations by using the finite difference method/local discontinuous Galerkin finite element method. In Ref. [11], Wang et al. developed an indirect finite element method for solving Dirichlet boundary value problems of Caputo fractional differential equations. In addition, many researchers have achieved fruitful results in discrete fractional differential equations [12–18]. Due to the nonlocality of fractional differential operators, the numerical methods for solving spatial fractional differential equations usually produce full stiffness matrix, which is traditionally solved by Gauss elimination method. This requires $\mathcal{O}(N^3)$ of operations and $\mathcal{O}(N^2)$ of memory for per iteration, and the computational cost and memory load

are significantly increased compared with the numerical method of high order diffusion equation [19].

Compared with the traditional numerical methods, the approximate computation of ANN seems not very sensitive to the spatial dimension. ANN provides an adaptive mesh, but it does not need to explicitly deal with the mesh, only needs to solve the optimization problem. Based on these facts and advantages, network method for PDE problems was early expanded by Lagaris et al. [20,21], and has been applied to a large number of approximate problems of partial differential equations [22–25]. At present, there are some works that utilize neural networks approach to effectively solve fractional partial differential equations. In Ref. [26], Raissia et al. proposed the physics-informed neural networks (PINNs) to solve forward and inverse problems involving nonlinear partial differential equations and studied their convergence. Then in Ref. [27], Pang et al. solve space-time fractional advection-diffusion equations by fractional PINNs. The work focus on studying the parameter identification problem of fractional partial differential equations. The fPINNs can obtain superior accuracy results and can deal with the problem of high-dimensional irregular-domain . Furthermore, a wavelet neural network was proposed to get the fractional differential equations solution in Ref. [28]. In Ref. [29], Gao et al. obtained numerical methods by using a triangle neural network to solve fractional differential equations. In recent years, there are many other scholars and experts who have many works on solving fractional differential equations with neural networks [30–35].

Recently, ANNs are becoming more and more important because of their many advantages, such as they can completely approximate any complex nonlinear relationship and has strong fault tolerance and robustness for all quantitative or qualitative information stored in each neuron in the network with equipotential distribution. They can adopt parallel distributed processing methods, which makes it possible to carry out large-scale operations quickly, and they can also learn and adapt to unknown or uncertain systems. More, the power series method can be used to deal with complex mathematical problems, can provide a solution function of series polynomial, and the coefficient can be determined by certain methods. In this paper, our main research combines artificial neural networks (ANNs) and truncated power series method as an iterative minimization algorithm to obtain the approximate solution of fractional diffusion equation. Here, we use fractional differential definitions proposed by Caputo. In Section 2, we introduce the definition and notation of fractional calculus, and gives the framework and basic introduction of problem. In Section 3, we present the implementation of the ANN algorithm. The solution is expressed as an appropriate truncated series expansion. By using the minimum mean square error function, then the problem is transformed into a minimization problem. In Section 4, three numerical examples are given to verify the effectiveness of the proposed method.

2. Preliminaries and Notation

In this section, we first present some basic definitions and notations about the fractional calculus.

Definition 1 ([36]). *Let $f(x)$ be the continuously differentiable function of the finite interval $[a, b]$ on the real axis $\mathbb{R}$. The fractional integral is defined as*

$$ {}_0I_x^{\lambda} f(x) = \frac{1}{\Gamma(\lambda)} \int_0^x \frac{f(\tau)}{(x-\tau)^{1-\lambda}} \mathrm{d}\tau, \lambda > 0, \tag{1} $$

where $\Gamma(\cdot)$ denotes the Gamma function.

Definition 2 ([37]). *Let $f(x)$ be the continuously differentiable function of the finite interval $[a,b]$ on the real axis $\mathbb{R}$. The left-sided and the right-sided Caputo fractional derivative of order $\alpha > 0$ are defined by*

$$\begin{aligned} {}_a^C D_x^\alpha f(x) &= \frac{1}{\Gamma(n-\alpha)} \int_a^x \frac{f^n(t)}{(x-t)^{\alpha-n+1}} \mathrm{d}t, \\ {}_x^C D_b^\alpha f(x) &= \frac{(-1)^n}{\Gamma(n-\alpha)} \int_x^b \frac{f^n(t)}{(t-x)^{\alpha-n+1}} \mathrm{d}t, \end{aligned} \tag{2}$$

respectively, where $n-1 \leqslant \alpha < n$ and $\Gamma(\cdot)$ denotes the Gamma function.

Next, we present the result of Caputo fractional derivative of power function

$${}_a^C D_x^{\alpha_k}\left[x^i\right] = \begin{cases} 0, & i \in Z^+, i < \lceil \alpha_k \rceil \\ \frac{\Gamma(i+1)}{\Gamma(i+1-\alpha_k)} x^{i-\alpha_k}, & i \in Z^+, i \geq \lceil \alpha_k \rceil \end{cases} \tag{3}$$

and

$${}_a^C D_x^{\alpha_k} C = 0,$$

here, we use $\lceil \alpha_k \rceil$ to denote the smallest integer greater than or equal to α_k and C is a constant. In the following work, we mark ${}_a^C D_x^{\alpha_k}$ as ${}_a D_x^{\alpha_k}$ for the sake of simplicity.

Problem Description

In this paper, we consider the following fractional differential equation

$$\sum_{k=0}^{m} P_k(x) {}_a D_x^{\alpha_k}[u(x)] = g(x), x \in [a,b] \tag{4}$$

subject to boundary conditions $u(a) = 1$. Here $1 < \alpha_k$ are the fractional orders, P_k and $g(x)$ are given real-value analytical functions.

The traditional power series method is used to solve ordinary differential equations and partial differential equations. Furthermore, since it is still very difficult to find the analytical solutions of fractional differential equations, this paper mainly studies the power series solution of fractional differential equations. In this method, the truncated power series expansion is considered to replace the unknown functions in the equations. Once the unknown functions in the problem are replaced, a series of equations based on discrete points are obtained with unknown coefficients. Then the artificial neural network method is used to solve such a set of algebraic equations.

Here, we consider the following power series expansion

$$u(x) = u(0) + \sum_{i=0}^{\infty} a_i x^i, \tag{5}$$

where a_i are the unknown coefficients. When (5) is substituted into (4), we have

$$\sum_{k=0}^{m} P_k(x) {}_a D_x^{\alpha_k}[u(0) + \sum_{i=0}^{\infty} a_i x^i] = g(x), \tag{6}$$

here, we define $h_1 := (b-a)/(N_1+1)$ the grid size in x-direction with $x_i := a + ih_1$ for $i = 0, 1, \ldots, N_1 + 1$ (6). Then we obtain the following discretization scheme

$$\sum_{k=0}^{m} P_k(x) {}_a D_x^{\alpha_k}[u(0) + \sum_{i=0}^{n} a_i x^i] \approx g(x), \tag{7}$$

where n is the order of power-series polynomial.

3. Implement ANNs

According to the universal approximation theorem [38], for a feedforward neural network composed of a linear output layer and at least one hidden layer using an "extrusion" activation function, as long as the number of hidden layer neurons is sufficient, it can approximate any bounded closed set function defined in real number space with any accuracy. The neural network can be seen as a "universal" function and can be used to solve approximately complex FDEs. In this work, the ANN is shown in Figure 1. The architecture consists of multi-layers of neural networks with input layer, one or more hidden layers and the output layer. In Figure 1, each neuron in hidden layers consists of a function, which deals with the linear combination of weight matrix (the model parameters w_i are optimized by learning algorithm) and inputs of the neuron. The output of each neuron is the output of ANN (when the neuron is located in the output layer) or the input of another neuron of ANN. In this paper, the ANN employed to solve FDEs (4) can be mathematically represented by Formula (7).

In the neural structure, the relationship of each unit can be mathematically induced as below:

$$c_l = \sigma(\sum_{j=1}^{s} w_j x_j + b_l) \tag{8}$$

where σ is the activation functions, w_j and b_l denote the corresponding connection weights and bias term, respectively, c_l is both the input and output in the l_{th} and $l-1_{th}$ hidden layer. Here, we discrete x on the interval $[a, b]$ to obtain several points x_j where $j = 1, \ldots, s$. For example, we take $s = 10$, $s = 15$ and $s = 20$ respetively in the numerical experiments. These points and the corresponding $g(x_j)$ are the sample points we give.

When the neural network is employed to solve the numerical solution of problem (4), the loss function is defined as follow

$$E_j = \frac{1}{2}\left(\sum_{k=0}^{m} P_k(x_j)\zeta_{k,j} - g(x_j)\right)^2 + \lambda\|\omega\|_2^2, \quad j = 0, \ldots, s, \tag{9}$$

the value of λ is given artificially, and L2 regularization helps drive outlier weights closer to 0 but not quite to 0. Tor simplicity, the above mathematical symbols $\zeta_{k,j}$ can be expressed as follows

$$\zeta_{k,j} = \sum_{i=0}^{r} \frac{\Gamma(i+1)}{\Gamma(i+1-\alpha_k)} a_i x_j^{i-\alpha_k}.$$

The left parameter a_i is trained through the neural network, and the $g(x_j)$ value can be obtained by bringing it into the discrete point x_j. Then the left term in Equation (7) can be compared with the real value of the right term $g(x_j)$. The total error is then gained by summing the error functions at all collocation points, as shown below

$$E = \sum_{j=0}^{s} E_j,$$

then, the following optimization problem is achieved

$$\arg\min_{a_i} E = \sum_{j=0}^{s}\left(\frac{1}{2}(\sum_{k=0}^{m} P_k(x_j)\zeta_{k,j} - g(x_j))^2 + \lambda\omega_2^2\right) \tag{10}$$

In Figure 1, Equation (8) is a neuron in the hidden layer of Figure 1. The forward propagation of Figure 1 finally obtains a_i. Then substitute a_i into the left-hand term of Equation (7), discrete point x_j substitute the right-hand term to obtain the discrete value $g(x_j)$, and then execute Equation (9). When updating w_j and b_l, Equation (9) is used as a loss function. In this neural network, we use the Rectified Linear Unit (ReLU) function as activation function σ because ReLU has the following advantages. Firstly, it can make the

network training faster because its derivative is easier to obtain than sigmoid and tanh functions. Secondly, it increases the nonlinearity of the network, because it is a nonlinear function itself. When added to the neural network, it can be a grid fitting nonlinear mapping. Thirdly, it can also prevent the gradient from disappearing. When the value is too large or too small, the derivatives of sigmoid and tanh functions are close to 0, while relu is an unsaturated activation function and this phenomenon does not exist. Finally, it can make the grid sparse, because the part less than zero is zero, and the part greater than zero has value, so over fitting can be reduced.

In order to minimize the loss function, we choose to use Adaptive moment estimation (Adam) algorithm to optimize the loss function, because Adam algorithm is effective in practical application. It owns higher convergence speed and more valid learning effect when compared with other adaptive learning rate methods. It can also correct the issues existing in other optimization techniques, such as the disappearance of learning rate, too slow convergence speed or large fluctuation of loss function caused by high variance parameter update.

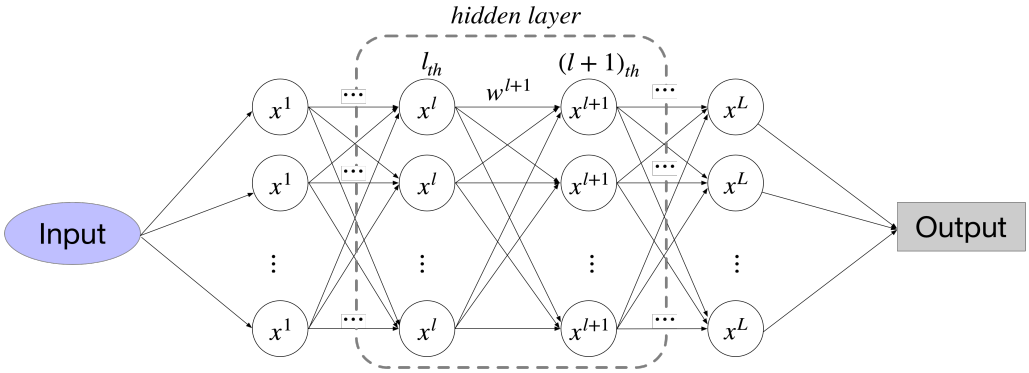

Figure 1. Artificial neural network proposed for solving FDEs.

4. Numerical Simulations

In order to verify the validity and efficiency of the proposed ANN algorithm, we give three examples in this section. All of the numerical results in the following examples were implemented using Python version 3.9.1. In the three examples, each hidden layer is set to contain 30 hidden layer neurons. In example 1, the proposed ANN algorithm sets up three layers of neural network, including one input layer, one output layer and one hidden layer; in examples 2 and 3, we set up five layers of neural networks including three hidden layers. In addition, when the right term of the equation is simple and the order of power-series polynomial is relatively small, we adopt 10^{-3} as the learning rate. When the order is set to be 9, we usually choose 5×10^{-4} as the learning rate, and the learning rate is adjusted between 10^{-3} and 5×10^{-4}. The initial output layer connection weight w_i (for $i = 1, \ldots, n$) were quantized with small random values to start the procedure, which is selected from the interval $(0, 1)$. In numerical experiments, in order to facilitate comparison, we compute the following mean square error

$$E_{mse} = \frac{1}{m+1} \sum_{i=0}^{m} (f(x_i) - y_i)^2,$$

where m denotes the number of discrete samples in domain [0,1].

4.1. Example 1

We consider solving the problem of Bagley–Torvik equation with initial condition as [39]:

$$ {}_aD_x^2 u(x) + {}_aD_x^{1.5}[u(x)] + u(x) = x + 1, 0 \le x \le 1,$$

with $u(0) = 1$.

We can find from Table 1 that with the increase of the number of iterations, our error accuracy can best reach 10^{-6}. Compared with the data in Table 2, we find that the error accuracy of ANN method in Table 2 can only reach 10^{-5} at most. For example, when $n = 9$ and $s = 20$, the accuracy of the ANN method obtained in this paper can reach 10^{-6}, while the accuracy of the method in Table 2 can only reach 10^{-4}. This shows that the convergence effect of the ANN method in this paper is better in the iteration. From Figure 2, we can observe that when the number of iteration steps reaches 25, the value of the loss function has a sharp decrease trend and can reach a relatively good accuracy. In Figure 3, the curve of the exact solution can almost coincide with the curve of the approximate solutions, which verifies the effectiveness of the proposed ANN algorithm, and Figure 4 evidences this conclusion. In addition, the total errors over different order of power-series polynomial were plotted for the iteration step number equals to 2000 in the Figure 5.

Table 1. Mean square error results for Example 1.

Iter	$n = 3$			$n = 5$		
	$s = 10$	$s = 15$	$s = 20$	$s = 10$	$s = 15$	$s = 20$
500	2.6523×10^{-4}	2.9910×10^{-4}	6.0083×10^{-4}	3.3160×10^{-5}	2.8045×10^{-4}	2.3854×10^{-3}
1000	1.6914×10^{-4}	1.8374×10^{-4}	3.5842×10^{-4}	5.4319×10^{-6}	9.5441×10^{-5}	9.3812×10^{-4}
1500	8.6155×10^{-5}	8.7060×10^{-5}	1.6233×10^{-4}	1.3877×10^{-6}	2.1126×10^{-5}	2.3634×10^{-4}
2000	3.3664×10^{-5}	2.9976×10^{-5}	5.1971×10^{-5}	1.2209×10^{-6}	5.3879×10^{-6}	3.4650×10^{-5}
Iter	$n = 7$			$n = 9$		
	$s = 10$	$s = 15$	$s = 20$	$s = 10$	$s = 15$	$s = 20$
500	1.0702×10^{-3}	1.2441×10^{-3}	1.6539×10^{-4}	1.1291×10^{-3}	4.0604×10^{-4}	8.8801×10^{-5}
1000	3.4174×10^{-5}	9.9941×10^{-4}	1.1063×10^{-4}	5.1723×10^{-4}	1.2310×10^{-4}	3.6049×10^{-5}
1500	2.4528×10^{-5}	7.0533×10^{-4}	9.2353×10^{-5}	1.8850×10^{-4}	4.4209×10^{-5}	5.0940×10^{-5}
2000	1.7723×10^{-5}	4.4926×10^{-4}	6.5423×10^{-5}	8.3826×10^{-5}	1.7819×10^{-5}	7.4516×10^{-6}

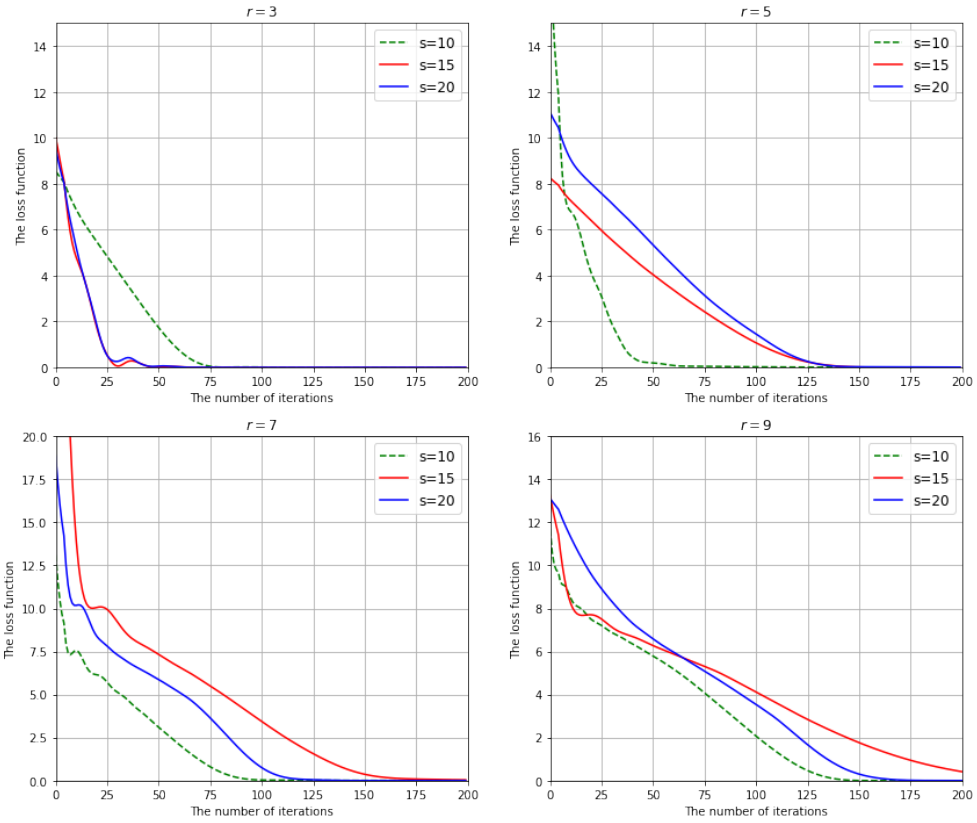

Figure 2. The loss function for different number of collocation point for Example 1.

Table 2. Mean square error results of ANN [30] for Example 1.

Iter	$n = 3$			$n = 5$		
	$s = 10$	$s = 15$	$s = 20$	$s = 10$	$s = 15$	$s = 20$
500	1.0996×10^{-3}	8.6408×10^{-4}	9.0756×10^{-4}	6.3823×10^{-4}	5.1589×10^{-4}	1.0696×10^{-3}
1000	9.0466×10^{-4}	7.0190×10^{-4}	7.9873×10^{-4}	4.4332×10^{-4}	4.0132×10^{-4}	5.3003×10^{-4}
1500	6.9259×10^{-4}	5.3601×10^{-4}	6.6735×10^{-4}	3.7657×10^{-4}	2.6475×10^{-4}	2.8372×10^{-4}
2000	5.0009×10^{-4}	3.9275×10^{-4}	5.3449×10^{-4}	3.1037×10^{-4}	9.7093×10^{-5}	2.3107×10^{-4}
Iter	$n = 7$			$n = 9$		
	$s = 10$	$s = 15$	$s = 20$	$s = 10$	$s = 15$	$s = 20$
500	6.6813×10^{-3}	8.7033×10^{-4}	1.6382×10^{-2}	1.6684×10^{-3}	1.4755×10^{-2}	5.0869×10^{-3}
1000	1.9484×10^{-3}	4.5613×10^{-4}	3.4877×10^{-3}	5.8310×10^{-4}	2.3891×10^{-3}	1.8448×10^{-3}
1500	8.2505×10^{-4}	2.5685×10^{-4}	1.4557×10^{-3}	4.1485×10^{-4}	9.6104×10^{-4}	7.5542×10^{-4}
2000	3.6007×10^{-4}	1.4610×10^{-4}	6.3923×10^{-4}	3.6057×10^{-4}	4.4318×10^{-4}	5.2341×10^{-4}

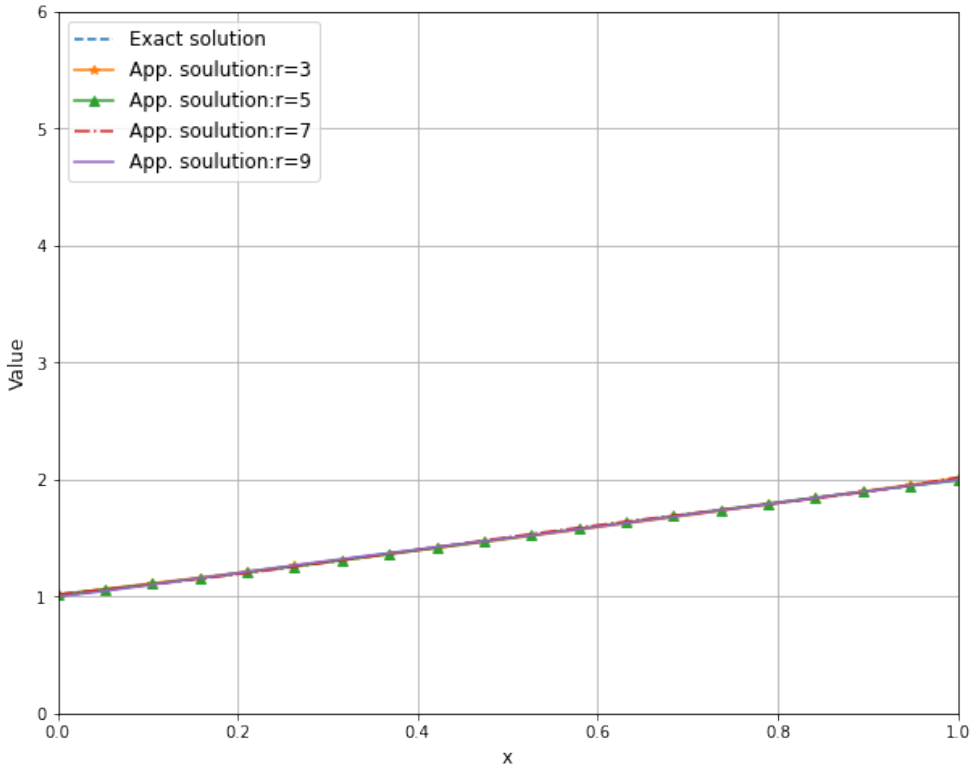

Figure 3. Exact and approximate solutions for iter = 2000.

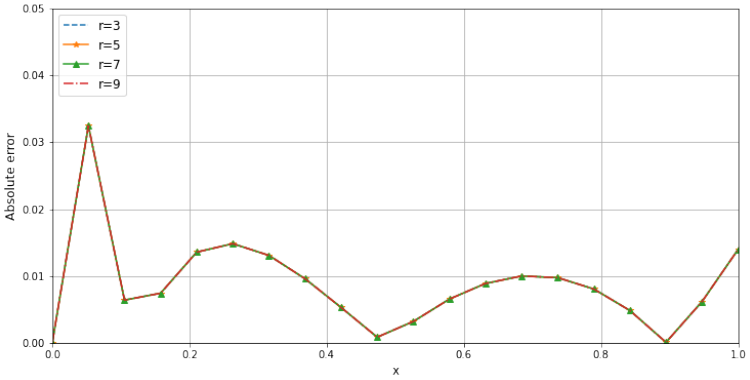

Figure 4. $f(x_i) - y_i$ for iter = 2000.

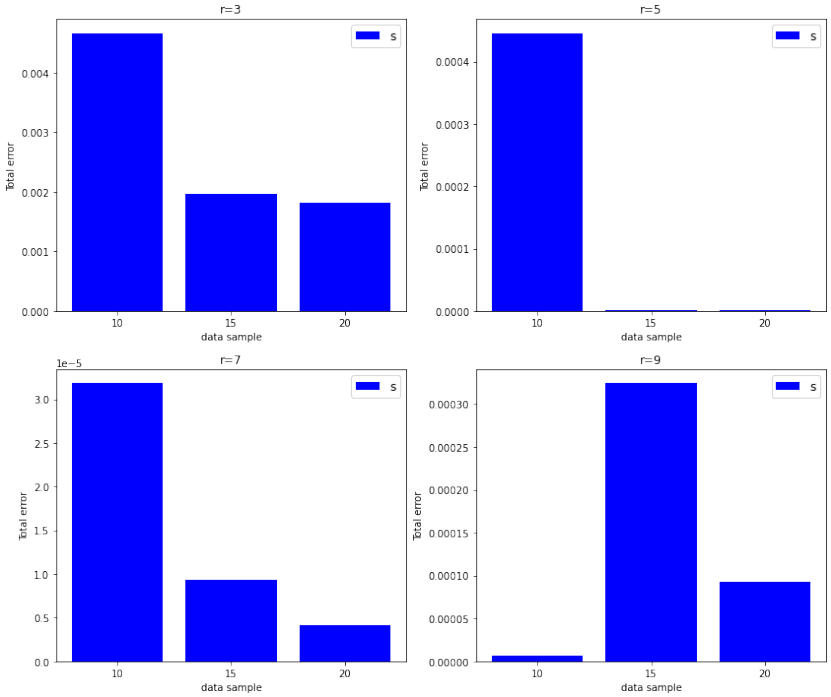

Figure 5. The total error over different order of power-series polynomial for t = 2000.

4.2. Example 2

Consider the following Bagley–Torvik equation with initial condition as:

$$ {}_aD_x^2 u(x) + {}_aD_x^{1.5}[u(x)] + u(x) = 2x^3 + x^2 + 1, 0 \le x \le 1, $$

with $u(0) = 1$.

The test Example 2 showed quite similar trends as of Example 1. Table 3 exhibited that mean square error tends to decrease with the increase of the order n of the power-series polynomial. For example, when n = 3 and s = 20, the accuracy of the error can only reach about 10^{-3}, and when n = 7 with s = 20, the accuracy of the error can reach 10^{-6}. The total errors over different order of power-series polynomial were plotted for the iteration step number equals to 2000 in the Figure 6. In Figure 7, we also found that the number of iterative steps is about 125, and the value of the loss function basically drops to the lowest point, indicating that the convergence speed of our method is quite considerable. As shown in Figure 8, our approximate solution can be very close to the exact solution, also and Figure 9 evidences this conclusion.

Table 3. Mean square error results for Example 2.

Iter	n **= 3**			n **= 5**		
	s **= 10**	s **= 15**	s **= 20**	s **= 10**	s **= 15**	s **= 20**
500	8.2966×10^{-2}	5.2626×10^{-2}	1.0117×10^{-2}	5.1567×10^{-2}	1.4999×10^{-6}	9.1941×10^{-4}
1000	4.4255×10^{-2}	4.6605×10^{-2}	1.8228×10^{-3}	3.4654×10^{-2}	1.3074×10^{-6}	1.3050×10^{-4}
1500	2.8333×10^{-2}	1.5791×10^{-2}	1.8212×10^{-3}	4.6556×10^{-4}	1.2690×10^{-6}	2.4591×10^{-6}
2000	4.6642×10^{-3}	1.9708×10^{-3}	1.8203×10^{-3}	4.4605×10^{-4}	1.2715×10^{-6}	2.2832×10^{-6}
Iter	n **= 7**			n **= 9**		
	s **= 10**	s **= 15**	s **= 20**	s **= 10**	s **= 15**	s **= 20**
500	1.0564×10^{-4}	5.2482×10^{-5}	1.3604×10^{-3}	9.5831×10^{-5}	1.2413×10^{-2}	1.4771×10^{-3}
1000	4.6845×10^{-5}	1.9055×10^{-5}	6.8806×10^{-6}	2.4573×10^{-4}	4.7626×10^{-3}	5.2961×10^{-4}
1500	3.6713×10^{-5}	1.3278×10^{-5}	5.2282×10^{-6}	8.5461×10^{-6}	1.2382×10^{-3}	1.9865×10^{-4}
2000	3.1830×10^{-5}	9.2861×10^{-6}	4.1084×10^{-6}	6.6666×10^{-6}	3.2416×10^{-4}	9.2339×10^{-5}

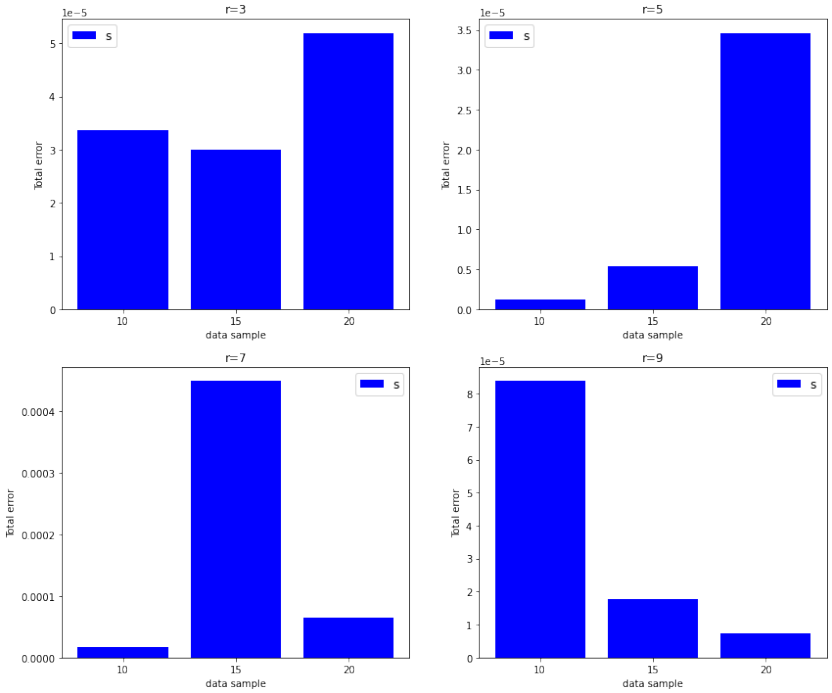

Figure 6. The total error over different order of power-series polynomial for t = 2000.

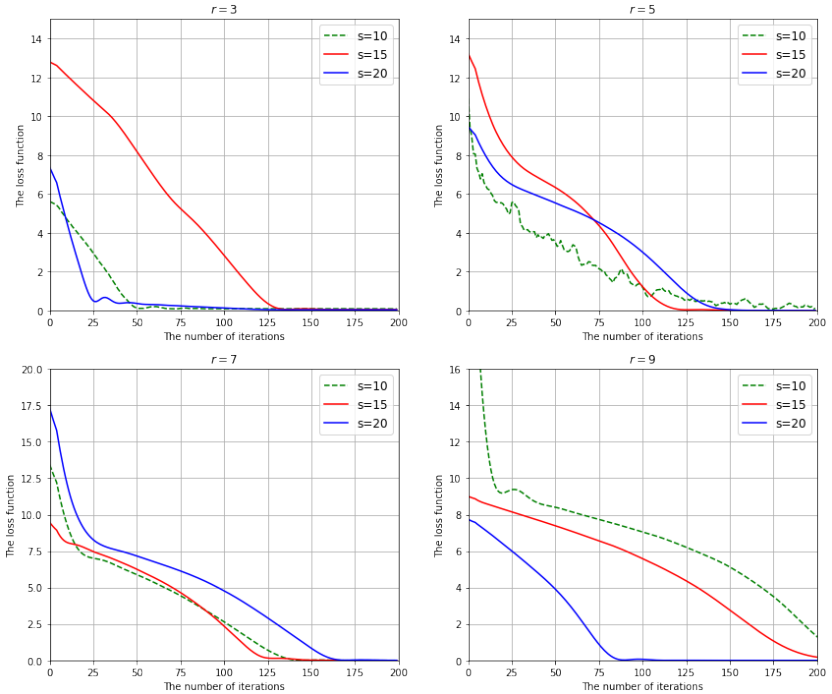

Figure 7. The loss function for different number of collocation point for Example 2.

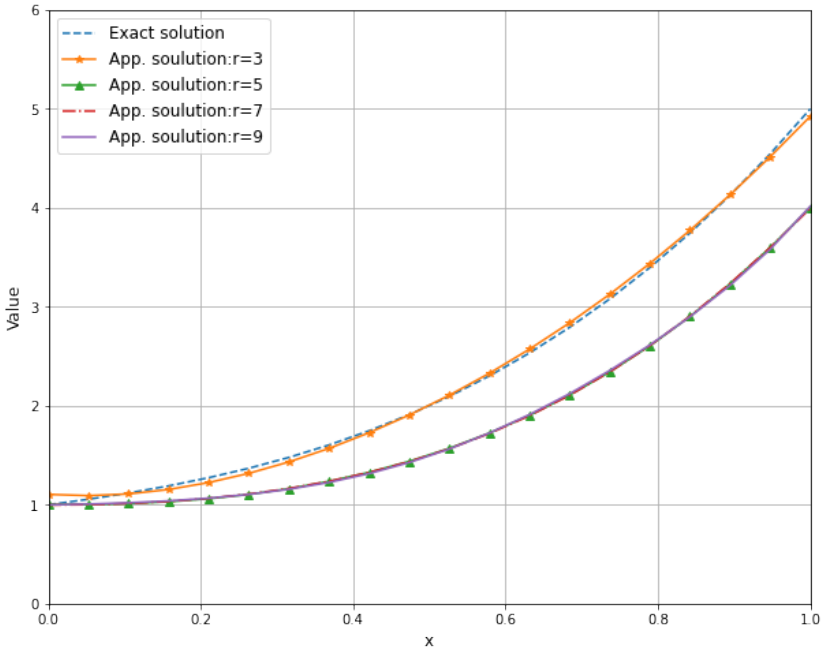

Figure 8. Exact and approximate solutions for iter = 2000.

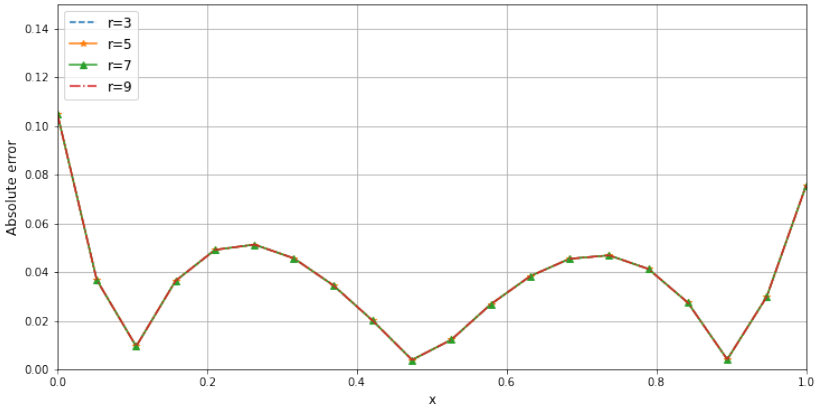

Figure 9. $f(x_i) - y_i$ for iter = 2000.

4.3. Example 3

Consider the following Bagley–Torvik equation with initial condition as:

$$ {}_aD_x^{3.2}u(x) + {}_aD_x^{1.8}[u(x)] + u(x) = 2x^3 + x^2 + x, 0 \leq x \leq 1, $$

with $u(0) = 1$.

As in the previous examples, Table 4 basically shows the same results. The values of the loss function for different number of collocation point were plotted in the Figure 10. Moreover, the relationship between exact solution and approximate solutions were presented in the Figures 11–13.

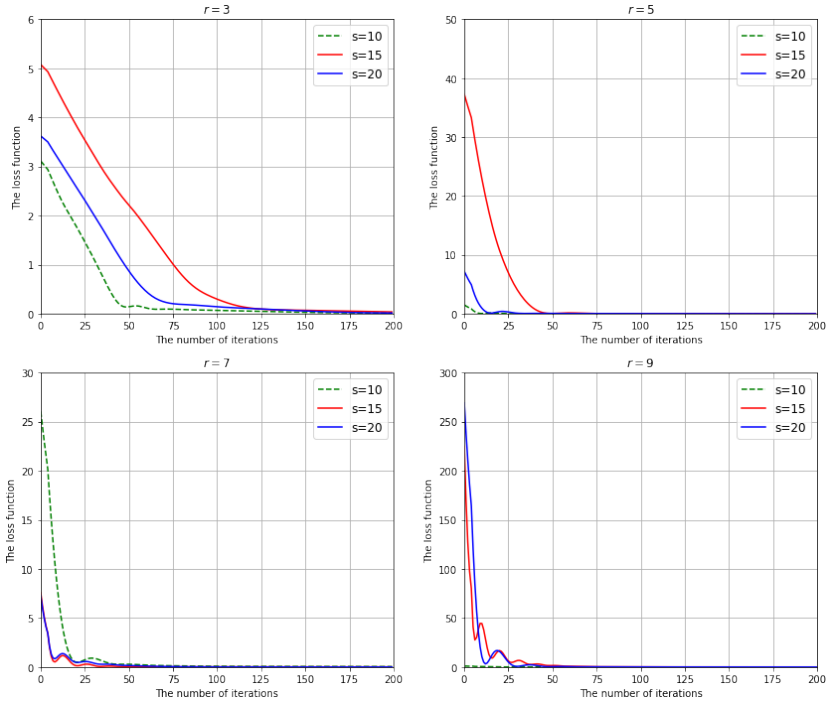

Figure 10. The loss function for different number of collocation point for Example 3.

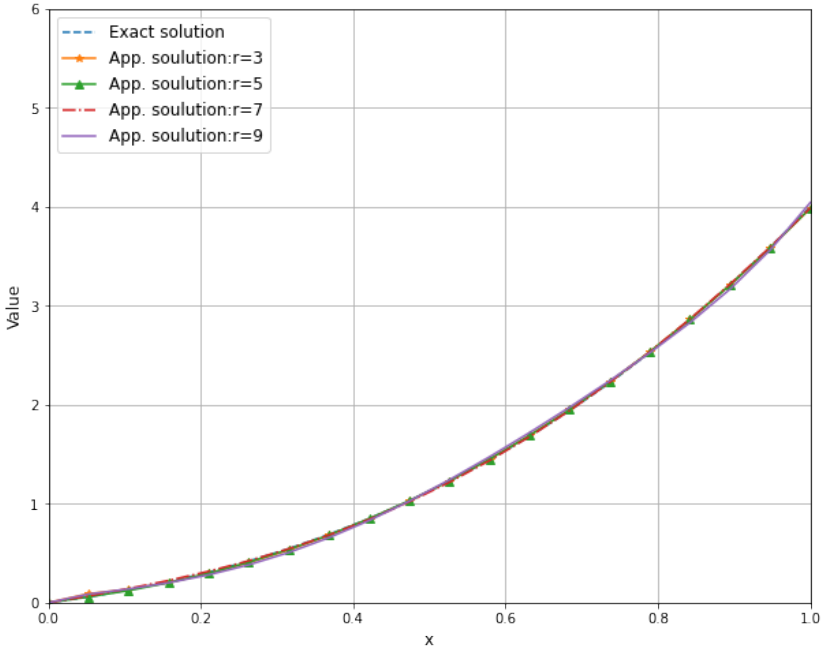

Figure 11. Exact and approximate solutions for iter = 2000.

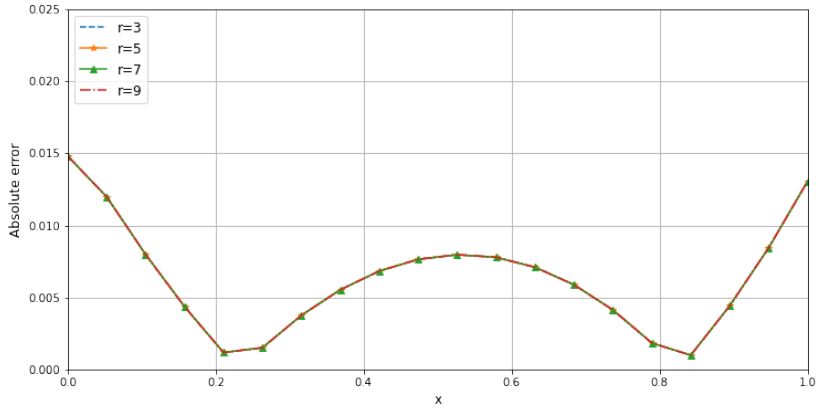

Figure 12. $f(x_i) - y_i$ for iter = 2000.

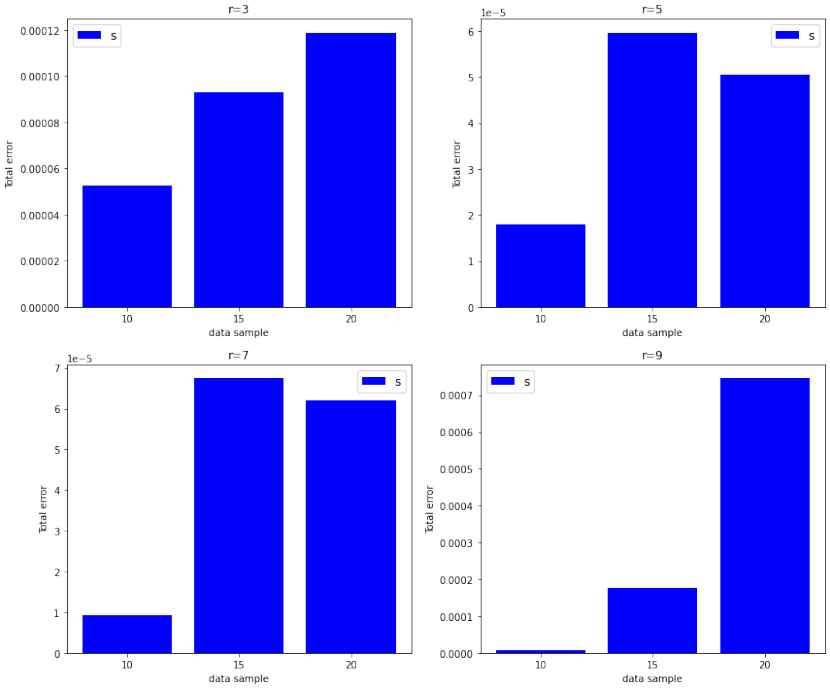

Figure 13. The total error over different order of power-series polynomial for t = 2000.

Table 4. Mean square error results for Example 3.

Iter	$n = 3$			$n = 5$		
	$s = 10$	$s = 15$	$s = 20$	$s = 10$	$s = 15$	$s = 20$
500	1.3920×10^{-2}	1.3119×10^{-2}	1.1351×10^{-2}	1.8116×10^{-5}	4.5366×10^{-3}	8.4585×10^{-5}
1000	8.8657×10^{-3}	9.2285×10^{-3}	4.4556×10^{-4}	1.8071×10^{-5}	1.2556×10^{-4}	7.2918×10^{-5}
1500	5.4655×10^{-5}	2.9104×10^{-4}	1.2161×10^{-4}	1.8011×10^{-5}	6.6137×10^{-5}	6.1525×10^{-5}
2000	5.2418×10^{-5}	9.2827×10^{-5}	1.1887×10^{-4}	1.7939×10^{-5}	5.9639×10^{-5}	5.0514×10^{-5}
Iter	$n = 7$			$n = 9$		
	$s = 10$	$s = 15$	$s = 20$	$s = 10$	$s = 15$	$s = 20$
500	4.4860×10^{-4}	5.8645×10^{-3}	4.1541×10^{-4}	1.5837×10^{-3}	1.4489×10^{-2}	1.6768×10^{-2}
1000	6.3946×10^{-5}	3.6488×10^{-4}	9.1928×10^{-5}	2.5737×10^{-4}	3.1810×10^{-4}	4.2928×10^{-3}
1500	2.6374×10^{-5}	9.1063×10^{-5}	7.7642×10^{-5}	5.1589×10^{-5}	1.9570×10^{-4}	1.0630×10^{-3}
2000	9.3822×10^{-6}	6.7459×10^{-5}	6.1965×10^{-5}	7.4717×10^{-6}	1.7564×10^{-4}	7.4550×10^{-4}

5. Conclusions

In this work, we present that it is possible to approximate the behavior of fractional differential equations by ANN algorithm when the derivative is fractional order. The advantages of the proposed ANN algorithm are reflected in the feasibility: the estimation made by the proposed neural network algorithm could get a satisfactory approximate solution with non-mesh discretization. Furthermore, the effectiveness and applicability of the algorithm are verified by solving the above multi-term FDEs.

Remark: The structure of neural network was built by the current authors using the special libraries of pytorch. We have uploaded the main source code on GitHub: https://github.com/hangzhounormal/sfdeNN (access on 12 February 2022). All computations were carried out on python version 3.9.1. The experiments are carried out on MacBook Air (M1, 2020) with the configuration: Apple M1 (8 Core) @3.20 GHz.

Author Contributions: Conceptualization, P.D. and X.Y.; methodology, P.D.; software, X.Y.; formal analysis, P.D.; writing—original draft preparation, P.D.; funding acquisition, P.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Zhejiang Natural Science Foundation, China under Grant number LQ22A010007; by the Start-up Foundation of Hangzhou Normal University under Grant number 4085C50220204094.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank the Xianliang Hu and Xinping Shao and Yu Xia for their many constructive comments and suggestions to improve the paper.

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

References

1. Baleanu, D.; Diethelm, K.; Scalas, E.; Trujillo, J.J. *Fractional Calculus: Models and Numerical Methods*; World Scientific: Singapore, 2012; Volume 3.
2. Baleanu, D.; Güvenç, Z.B.; Machado, J.T. *New Trends in Nanotechnology and Fractional Calculus Applications*; Springer: Berlin/Heidelberg, Germany, 2010.
3. Bazhlekova, E.; Bazhlekov, I. Viscoelastic flows with fractional derivative models: Computational approach by convolutional calculus of Dimovski. *Fract. Calc. Appl. Anal.* **2014**, *17*, 954–976. [CrossRef]
4. Su, X.; Xu, W.; Chen, W.; Yang, H. Fractional creep and relaxation models of viscoelastic materials via a non-Newtonian time-varying viscosity: Physical interpretation. *Mech. Mater.* **2020**, *140*, 103222. [CrossRef]
5. Arqub, O.A.; Al-Smadi, M. An adaptive numerical approach for the solutions of fractional advection–Diffusion and dispersion equations in singular case under Riesz's derivative operator. *Phys. A Stat. Mech. Appl.* **2020**, *540*, 123257. [CrossRef]
6. Stefański, T.P.; Gulgowski, J. Signal propagation in electromagnetic media described by fractional-order models. *Commun. Nonlinear Sci. Numer. Simul.* **2020**, *82*, 105029. [CrossRef]
7. Meerschaert, M.M.; Scheffler, H.P.; Tadjeran, C. Finite difference methods for two-dimensional fractional dispersion equation. *J. Comput. Phys.* **2006**, *211*, 249–261. [CrossRef]
8. Sun, H.; Chen, W.; Li, C.; Chen, Y. Finite difference schemes for variable-order time fractional diffusion equation. *Int. J. Bifurc. Chaos* **2012**, *22*, 1250085. [CrossRef]
9. Deng, W. Finite element method for the space and time fractional Fokker–Planck equation. *SIAM J. Numer. Anal.* **2009**, *47*, 204–226. [CrossRef]
10. Li, C.; Wang, Z. The local discontinuous Galerkin finite element methods for Caputo-type partial differential equations: Numerical analysis. *Appl. Numer. Math.* **2019**, *140*, 1–22. [CrossRef]
11. Wang, H.; Yang, D.; Zhu, S. Inhomogeneous Dirichlet boundary-value problems of space-fractional diffusion equations and their finite element approximations. *SIAM J. Numer. Anal.* **2014**, *52*, 1292–1310. [CrossRef]
12. Li, X.; Xu, C. Existence and uniqueness of the weak solution of the space-time fractional diffusion equation and a spectral method approximation. *Commun. Comput. Phys.* **2010**, *8*, 1016.
13. Pan, J.; Ng, M.; Wang, H. Fast preconditioned iterative methods for finite volume discretization of steady-state space-fractional diffusion equations. *Numer. Algorithms* **2017**, *74*, 153–173. [CrossRef]

14. Chen, M.; Deng, W. A second-order numerical method for two-dimensional two-sided space fractional convection diffusion equation. *Appl. Math. Model.* **2014**, *38*, 3244–3259. [CrossRef]
15. Zeng, F.; Li, C.; Liu, F.; Turner, I. Numerical algorithms for time-fractional subdiffusion equation with second-order accuracy. *SIAM J. Sci. Comput.* **2015**, *37*, A55–A78. [CrossRef]
16. Zhang, H.; Liu, F.; Zhuang, P.; Turner, I.; Anh, V. Numerical analysis of a new space–time variable fractional order advection–dispersion equation. *Appl. Math. Comput.* **2014**, *242*, 541–550. [CrossRef]
17. Gu, X.M.; Sun, H.W.; Zhao, Y.L.; Zheng, X. An implicit difference scheme for time-fractional diffusion equations with a time-invariant type variable order. *Appl. Math. Lett.* **2021**, *120*, 107270. [CrossRef]
18. Zhao, Y.L.; Gu, X.M.; Li, M.; Jian, H.Y. Preconditioners for all-at-once system from the fractional mobile/immobile advection–diffusion model. *J. Appl. Math. Comput.* **2021**, *65*, 669–691. [CrossRef]
19. Wang, H.; Wang, K.; Sircar, T. A direct O(N log2 N) finite difference method for fractional diffusion equations. *J. Comput. Phys.* **2010**, *229*, 8095–8104. [CrossRef]
20. Lagaris, I.E.; Likas, A.; Fotiadis, D.I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.* **1998**, *9*, 987–1000. [CrossRef] [PubMed]
21. Lagaris, I.E.; Likas, A.C.; Papageorgiou, D.G. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Trans. Neural Netw.* **2000**, *11*, 1041–1049. [CrossRef]
22. Li, Y.; Hu, X. Artificial neural network approximations of Cauchy inverse problem for linear PDEs. *Appl. Math. Comput.* **2022**, *414*, 126678. [CrossRef]
23. Khoo, Y.; Ying, L. SwitchNet: A neural network model for forward and inverse scattering problems. *SIAM J. Sci. Comput.* **2019**, *41*, A3182–A3201. [CrossRef]
24. Qin, T.; Wu, K.; Xiu, D. Data driven governing equations approximation using deep neural networks. *J. Comput. Phys.* **2019**, *395*, 620–635. [CrossRef]
25. Long, Z.; Lu, Y.; Dong, B. PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network. *J. Comput. Phys.* **2019**, *399*, 108925. [CrossRef]
26. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [CrossRef]
27. Pang, G.; Lu, L.; Karniadakis, G.E. fPINNs: Fractional physics-informed neural networks. *SIAM J. Sci. Comput.* **2019**, *41*, A2603–A2626. [CrossRef]
28. Wu, M.; Zhang, J.; Huang, Z.; Li, X.; Dong, Y. Numerical solutions of wavelet neural networks for fractional differential equations. *Math. Methods Appl. Sci.* **2021**. [CrossRef]
29. Gao, F.; Dong, Y.; Chi, C. Solving Fractional Differential Equations by Using Triangle Neural Network. *J. Funct. Spaces* **2021**, *2021*, 5589905. [CrossRef]
30. Rostami, F.; Jafarian, A. A new artificial neural network structure for solving high-order linear fractional differential equations. *Int. J. Comput. Math.* **2018**, *95*, 528–539. [CrossRef]
31. Raja, M.A.Z.; Khan, J.A.; Qureshi, I.M. A new stochastic approach for solution of Riccati differential equation of fractional order. *Ann. Math. Artif. Intell.* **2010**, *60*, 229–250. [CrossRef]
32. Qu, H.; Liu, X.; She, Z. Neural network method for fractional-order partial differential equations. *Neurocomputing* **2020**, *414*, 225–237. [CrossRef]
33. Qu, H.; Liu, X. A numerical method for solving fractional differential equations by using neural network. *Adv. Math. Phys.* **2015**, *2015*, 439526. [CrossRef]
34. Pakdaman, M.; Ahmadian, A.; Effati, S.; Salahshour, S.; Baleanu, D. Solving differential equations of fractional order using an optimization technique based on training artificial neural network. *Appl. Math. Comput.* **2017**, *293*, 81–95. [CrossRef]
35. Raja, M.A.Z.; Manzar, M.A.; Samar, R. An efficient computational intelligence approach for solving fractional order Riccati equations using ANN and SQP. *Appl. Math. Model.* **2015**, *39*, 3075–3093. [CrossRef]
36. Samko, S.G. Fractional integrals and derivatives, theory and applications. *Minsk Nauka I Tekhnika* **1987**. Available online: https://asset-pdf.scinapse.io/prod/1530054495/1530054495.pdf (accessed on 11 February 2022)
37. Liu, F.; Anh, V.; Turner, I. Numerical solution of the space fractional Fokker–Planck equation. *J. Comput. Appl. Math.* **2004**, *166*, 209–219. [CrossRef]
38. Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Netw.* **1991**, *4*, 251–257. [CrossRef]
39. Momani, S.; Odibat, Z. Numerical comparison of methods for solving linear differential equations of fractional order. *Chaos Solitons Fractals* **2007**, *31*, 1248–1255. [CrossRef]