Physica Scripta

PAPER

A deep learning framework for solving fractional partial differential equations

RECEIVED
16 January 2025

REVISED
28 February 2025

ACCEPTED FOR PUBLICATION
6 March 2025

PUBLISHED
19 March 2025

Amina Ali[1,2], **Norazak Senu**[1,3,*], **Ali Ahmadian**[4,5,*] **and Nadihah Wahi**[1,3]

[1] Department of Mathematics and Statistics, Universiti Putra Malaysia, 43400 UPM, Serdang, Malaysia
[2] Department of Mathematics, College of Education, University of Sulaimani, Sulaymaniyah, Iraq
[3] Institute for Mathematical Research, Universiti Putra Malaysia, 43400 UPM, Serdang, Malaysia
[4] Jadara University Research Center, Jadara University, Jordan
[5] Faculty of Engineering and Natural Sciences, Istanbul Okan University, Istanbul, Turkey
[*] Authors to whom any correspondence should be addressed.

E-mail: norazak@upm.edu.my **and** ahmadian.hosseini@gmail.com

Keywords: laplace transform method, fractional partial differential equations, artificial neural networks, gradient descent, deep neural network

Abstract

This research focuses on the study and solution of fractional partial differential equations (FPDEs), a critical area in mathematical analysis. FPDEs pose significant challenges due to their complexity, often requiring extensive computational resources to solve. Given the scarcity of exact solutions, numerical methods have been a primary approach for tackling FPDEs. However, these methods often yield substantial but limited results. The ongoing quest for more effective solutions has led researchers to explore new methodologies. Recent advancements in deep learning (DL), particularly in deep neural networks (DNNs), offer promising tools for solving FPDEs due to their exceptional function-approximation capabilities, demonstrated in diverse applications such as image classification and natural language processing. This research addresses the challenges of solving FPDEs by proposing a novel deep feedforward neural network (DFNN) framework. The method integrates the Laplace transform for memory-efficient Caputo derivative approximations and demonstrates superior accuracy across various examples. The results highlight the framework's versatility and computational efficiency, establishing it as a powerful tool for solving FPDEs.

1. Introduction

Fractional integrals and derivatives have a rich historical background, emerging nearly concurrently with integer-order calculus. A noteworthy characteristic of fractional derivatives is their inherent time memory or historical heredity [1]. This unique attribute makes fractional derivatives highly applicable across a wide range of fields [2]. When solving fractional differential equations, two primary categories of methods exist: analytical and numerical. Analytical methods include techniques such as the Fourier transform, Laplace transform [3], Mellin transform, and Green's function technique. Additionally, the Lie symmetry analysis method has emerged as a modern analytical approach for solving FPDEs [4, 5]. However, most fractional differential equations do not have analytical solutions. Even when solutions exist, they often involve intricate functions such as the Mittag-Leffler function, Wright function, and H-function, making numerical computations challenging. Consequently, it is essential to investigate and develop numerical techniques for solving fractional models.

Over the past decade, DL has undergone a significant transformation, particularly in the development of deep artificial neural networks (ANNs). Although ANNs have existed since the 1940s [6] and have been applied across various domains, recent advancements in deep learning, especially in the context of differential equations, have been particularly noteworthy. For an extensive historical overview, especially concerning differential equations, see Chapter 2 of [7]. The remarkable achievements of DL over the past ten years can be attributed to the seamless integration of enhanced theoretical foundations beginning with unsupervised pre-training and

deep belief networks alongside advancements in hardware, particularly general-purpose graphics processing units (GPUs), as highlighted in [8, 9]. Deep ANNs have demonstrated significant success across a wide range of applications, including image interpretation, pattern recognition, object localization, language comprehension, and emerging fields such as autonomous transportation and autonomous vehicles. While deep ANNs have led to major breakthroughs in key application areas, questions remain regarding the fundamental processes that underpin their effectiveness. In the domain of function approximation, it has been recognized since the 1990s that ANNs serve as universal approximators, capable of approximating any continuous function along with its derivatives. This insight is well-documented in the research of [10–12]. In the context of partial differential equations (PDEs), the conventional approach has been to employ single-hidden-layer ANNs for solving PDEs. This choice is based on the understanding that a single layer, when equipped with a sufficient number of neurons, can approximate any given function. This capability arises from the fact that all necessary gradients can be explicitly computed in analytical form, as noted in [13–15]. However, a still-limited but growing body of research has begun exploring deep ANNs for PDE solving [16, 17].

Overall, ANNs offer the advantage of being continuous, computable functions that can be evaluated at any point, both within and beyond the domain, eliminating the need for reconstitution. However, the precise mechanisms underlying their remarkable effectiveness remain an active area of research and exploration. DNN approaches offer several advantages over traditional numerical techniques, including [18]:

1. Traditional numerical approaches operate iteratively and typically require defining a discretization interval before computation. If a solution is needed between two grid points, the entire process must be restarted from the initial stage. In contrast, DNN methods overcome this limitation by enabling numerical solutions at any point within the domain without requiring repetitive iterations.
2. While solving inverse problems is often challenging or even impossible with most traditional numerical methods, DNNs offer the advantage of handling inverse problems with minimal modifications to code originally designed for forward problems. This adaptability is a significant strength.
3. Various traditional numerical methods, such as the finite difference and finite element approaches, rely on grid-based computation models, making high-dimensional problems challenging to handle. In contrast, DNN methods leverage automatic differentiation, are generally meshless, and can effectively overcome the challenges posed by the curse of dimensionality.

Here is a concise literature review on the proposed method: Weinan *et al* developed a DNN-based method for solving variational problems [19]. In [20], a Physics-Informed Neural Network (PINN) was introduced, specifically designed for supervised learning tasks related to solving nonlinear PDEs governing various physical laws. Berg *et al* explored the use of DFNNs for solving PDEs in complex geometries [21]. Jin *et al* employed PINNs to integrate governing equations directly into the DNN through automatic differentiation, effectively handling constraints in the simulation of incompressible laminar and turbulent flows [22]. Sheng *et al* proposed the Penalty-Free Neural Network (PFNN) method, which provides an efficient solution for a particular class of second-order boundary value problems arising in complex geometries [23]. Ye *et al* developed DNN-based methods for addressing both forward and inverse problems associated with time-fractional diffusion equations featuring conformable derivatives [24]. Wei *et al* designed a DNN for solving time-fractional Fokker-Planck equations of order α, where $0 < \alpha < 1$ [25]. Fang *et al* introduced a novel approach for solving a specific class of FPDEs of order α, where $0 < \alpha < 1$, using DNNs to address both the equations and their corresponding inverse problems [26]. Shi *et al* developed fractional physics-informed neural networks (fPINNs) method for solving the time-fractional Huxley equation [27]. Shi *et al* presented a fast L1-fractional fPINNs (FL1-fPINN)for solving time-fractional reaction-diffusion [28].

The key contributions of this study are as follows:

1. A widely used numerical scheme (L_1 and L_2) for approximating fractional derivatives, as outlined in the literature, requires historical data from all previous time steps to compute the term $\frac{\partial^{\gamma}\Psi(\zeta,\eta)}{\partial\eta^{\gamma}}$. This dependence on extensive past data leads to significant memory challenges in long-duration simulations, potentially causing computational bottlenecks. To address this issue, we propose a novel approximation of the Caputo-type fractional derivative using the Laplace transform. This approach effectively alleviates memory constraints, providing a more efficient and scalable solution for solving FPDEs.
2. A DFNN with multiple hidden layers is developed to solve FPDEs in the Caputo sense.
3. A novel trial solution is developed for a specific class of FPDEs, incorporating the appropriate initial and boundary conditions.

4. FPDEs with orders ranging from 0 to 1 and 1 to 2 are successfully solved, demonstrating the method's versatility across a range of problems.
5. Our DFNN method demonstrates superior accuracy in solving FPDEs when compared to existing methods, establishing its effectiveness as a reliable computational tool.

The following sections in this paper are structured as outlined below: section 2 provides a concise introduction to the pertinent definitions relevant to our study. Section 3 outlines the DFNN method, including the establishment of the DFNN architecture and the derivation of the cost function. Section 4 presents the solutions of several examples along with their corresponding results. Lastly, a synopsis of the paper's findings is provided in section 5.

2. Preliminary concepts

This section offers a definition of the fractional derivatives used in this study, as well as the Laplace transform for the Caputo fractional derivative.

Definition 2.1. Defined for an order $\alpha > 0$, the Caputo derivative is expressed as follows in [29, 30]:

$$ {}_a^C D_\eta^\alpha \phi(\zeta, \eta) = \begin{cases} \frac{1}{\Gamma(p-\alpha)} \int_a^\eta \frac{\partial^p \phi(\zeta, \eta)}{\partial l^p} (\eta - l)^{p-\alpha-1} dl, & for \quad p-1 < \alpha < p, \\ \frac{\partial^p}{\partial \eta^p} \phi(\zeta, \eta), & for \quad \alpha = p, \end{cases} $$

where p is a natural number.

Definition 2.2. Considering the function h defined for t in the non-negative domain, the Laplace transform of h, denoted as $\mathcal{L}\{h\}$, is established through the improper integral, as described in [31]:

$$ \mathcal{L}\{h(x)\} = H(s) = \int_0^\infty e^{-sx} h(x) dx. \tag{2.1} $$

Provided that the integral in (2.1) exists and is convergent. The inverse Laplace transform is defined as:

$$ \mathcal{L}^{-1}\{H(s)\} = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} e^{sx} H(s) ds. $$

Definition 2.3. The Laplace transformation of the Caputo fractional derivative having an order γ is outlined in [32]:

$$ \mathcal{L}\{{}_0^C D_t^\gamma h(t)\} = s^\gamma \mathcal{L}\{h(t)\} - \sum_{k=0}^{l-1} s^{\gamma-k-1} ({}_0 D_t^k h)(0). $$

3. Dfnn model construction

In this section, we offer a detailed introduction to our method.

3.1. Design of the DFNN architecture

The architecture employed in this study can aptly be referred to as a DFNN. The following notations are briefly introduced in this architecture: As shown in figure 1, consider an $l+1$ layer network, consisting of an input layer, l hidden layers, and an output layer. The number of neurons in each hidden layer is assumed to be equal and is denoted as n. The layer notation appears as a superscript on each node, indicating its position within a specific layer. This notation extends to the superscripts of the weights and biases, specifying the subsequent layer to which they contribute. The weights are denoted as w_{ij}^l, where the indices follow these ranges: l from 1 to $l+1$, i from 1 to n, and j from 1 to n. The subscripts i and j have specific meanings: they represent the weights from the neuron at position j in the $(l-1)$-th layer to the neuron at position i in the l-th layer. Regarding biases, they are represented as $b^{[l]}$, where l ranges from 1 to $l+1$. The network's input is denoted as $\mu = (\zeta, \eta)^T$, and its output, represented by $N(\zeta, \eta, p)$, is obtained from the node $h^{[l+1]}$, reflecting the count of unknown variables in the FPDEs. It is essential to acknowledge the pivotal role of the activation function, represented by ϕ, in transforming the input to produce the network's output. In this study, we utilize the sigmoid function, defined as $\phi(\zeta) = \frac{1}{1+e^{-\zeta}}$, for every node in each hidden layer.

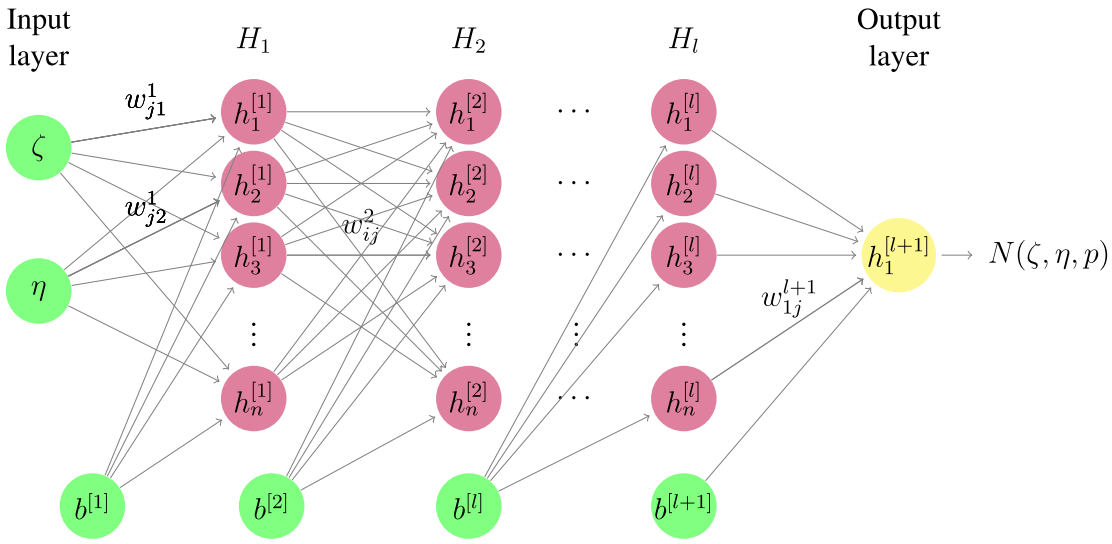


Figure 1. The designed DFNN structure.

3.2. FPDEs problem formulation

We aim to solve the following FPDEs with the initial and boundary conditions:

$$\left.\begin{aligned} &\frac{\partial^\gamma \Psi(\zeta,\eta)}{\partial \eta^\gamma} = G(\zeta, \eta, \Psi, \Psi_\eta, \Psi_\zeta, \Psi_{\zeta\zeta}, \Psi_{\eta\eta}), \quad 0 \leqslant \zeta \leqslant 1, 0 \leqslant \eta \leqslant 1, 0 < \gamma \leqslant 1,\\ &\Psi(\zeta, 0) = \phi_1(\zeta), \quad 0 \leqslant \zeta \leqslant 1,\\ &\Psi(0, \eta) = \phi_2(\eta), \Psi(1, \eta) = \phi_3(\eta), \quad 0 \leqslant \eta \leqslant 1. \end{aligned}\right\} \tag{3.1}$$

$$\left.\begin{aligned} &\frac{\partial^\gamma \Psi(\zeta,\eta)}{\partial \eta^\gamma} = G(\zeta, \eta, \Psi, \Psi_\eta, \Psi_\zeta, \Psi_{\zeta\zeta}, \Psi_{\eta\eta}), \quad 0 \leqslant \zeta \leqslant 1, 0 \leqslant \eta \leqslant 1, 1 < \gamma \leqslant 2,\\ &\Psi(\zeta, 0) = \phi_1(\zeta), \Psi_\eta(\zeta, 0) = \phi_2(\zeta), \quad 0 \leqslant \zeta \leqslant 1,\\ &\Psi(0, \eta) = \phi_3(\eta), \Psi(1, \eta) = \phi_4(\eta), \quad 0 \leqslant \eta \leqslant 1. \end{aligned}\right\} \tag{3.2}$$

3.3. Approximation of the caputo time fractional derivative

First, we use the Laplace transform approach to approximate the Caputo-type time-fractional derivative [33]:

1. when $0 < \gamma < 1$:

$$\begin{aligned} \mathcal{L}\left\{\frac{\partial^\gamma \Psi(\zeta,\eta)}{\partial \eta^\gamma}\right\} &= s^\gamma \bar{\Psi}(\zeta, s) - s^{\gamma-1}\Psi(\zeta, 0),\\ &= s^\gamma[\bar{\Psi}(\zeta, s) - s^{-1}\Psi(\zeta, 0)]. \end{aligned} \tag{3.3}$$

Here, $\bar{\Psi}(\zeta, s)$ denotes the Laplace transform of $\Psi(\zeta, \eta)$. Considering $0 < \gamma < 1$, we can linearize the term s^γ in the following manner:

$$s^\gamma \approx \gamma s + (1 - \gamma).$$

Subsequently, we substitute this linearized term into (3.3). This results in:

$$\begin{aligned} \mathcal{L}\left\{\frac{\partial^\gamma \Psi(\zeta,\eta)}{\partial \eta^\gamma}\right\} &\approx (\gamma s + (1 - \gamma))[\bar{\Psi}(\zeta, s) - s^{-1}\Psi(\zeta, 0)],\\ &= \gamma s[\bar{\Psi}(\zeta, s) - s^{-1}\Psi(\zeta, 0)] + (1 - \gamma)[\bar{\Psi}(\zeta, s) - s^{-1}\Psi(\zeta, 0)]. \end{aligned}$$

Therefore, the inverse Laplace transform yields,

$$\frac{\partial^\gamma \Psi(\zeta, \eta)}{\partial \eta^\gamma} \approx \gamma \frac{\partial \Psi(\zeta, \eta)}{\partial \eta} + (1 - \gamma)[\Psi(\zeta, \eta) - \Psi(\zeta, 0)].$$

2. when $1 < \gamma < 2$:

$$\mathcal{L}\left\{\frac{\partial^{\gamma}\Psi(\zeta,\eta)}{\partial\eta^{\gamma}}\right\} = s^{\gamma}\bar{\Psi}(\zeta,0) - s^{\gamma-1}\Psi(\zeta,0) - s^{\gamma-2}\Psi_{\eta}(\zeta,0),$$
$$= s^{\gamma}[\bar{\Psi}(\zeta,s) - s^{-1}\Psi(\zeta,0) - s^{-2}\Psi_{\eta}(\zeta,0)]. \quad (3.4)$$

For $1<\gamma<2$, applying linear interpolation to the s^{γ} power function at interpolation points 1 and 2 yields the following result:

$$s^{\gamma} \approx s^{2}(\gamma-1) + (2-\gamma)s.$$

Putting it into (3.4) yields,

$$\mathcal{L}\left\{\frac{\partial^{\gamma}\Psi(\zeta,\eta)}{\partial\eta^{\gamma}}\right\} \approx (s^{2}(\gamma-1) + (2-\gamma)s)[\bar{\Psi}(\zeta,s) - s^{-1}\Psi(\zeta,0) - s^{-2}\Psi_{\eta}(\zeta,0)],$$
$$= (\gamma-1)s^{2}[\bar{\Psi}(\zeta,s) - s^{-1}\Psi(\zeta,0) - s^{-2}\Psi_{\eta}(\zeta,0)]$$
$$+ (2-\gamma)s[\bar{\Psi}(\zeta,s) - s^{-1}\Psi(\zeta,0) - s^{-2}\Psi_{\eta}(\zeta,0)].$$

Inverse Laplace transform results in the following:

$$\frac{\partial^{\gamma}\Psi(\zeta,\eta)}{\partial\eta^{\gamma}} \approx (\gamma-1)\frac{\partial^{2}\Psi(\zeta,\eta)}{\partial\eta^{2}}$$
$$+ (2-\gamma)\frac{\partial\Psi(\zeta,\eta)}{\partial\eta} - (2-\gamma)\Psi_{\eta}(\zeta,0).$$

3.4. Proposed solution

Drawing inspiration from Lagaris's concept [13], we formulate a trial solution $\hat{\Psi}(\mu,p)$ that adheres to the initial boundary value conditions, aiming to solve (3.1) and (3.2),

$$\hat{\Psi}(\mu,p) = A(\zeta,\eta) + \zeta(1-\zeta)\eta N(\mu,p),$$
$$A(\zeta,\eta) = (1-\zeta)\phi_{2}(\eta) + \zeta\phi_{3}(\eta) + \phi_{1}(\zeta) - (1-\zeta)\phi_{1}(0) - \zeta\phi_{1}(1). \quad (3.5)$$

$$\hat{\Psi}(\mu,p) = A(\zeta,\eta) + \zeta(1-\zeta)\eta^{2}N(\mu,p),$$
$$A(\zeta,\eta) = (1-\zeta)\phi_{3}(\eta) + \zeta\phi_{4}(\eta) + (1-\eta^{2})(\phi_{1}(\zeta) - ((1-\zeta)\phi_{1}(0) + \zeta\phi_{1}(1)))$$
$$+ \eta((\phi_{2}(\zeta) - ((1-\zeta)\phi_{2}(0) + \zeta\phi_{2}(1))). \quad (3.6)$$

where $A(\zeta,\eta)$ satisfies the initial boundary conditions.

3.5. Learning algorithm

The cost function can be defined as follows:

$$C(\mu,p) = \frac{1}{2}\sum_{i=1}^{S}\left[\frac{\partial^{\gamma}\hat{\Psi}(\mu_{i},p)}{\partial\eta^{\gamma}} - G(\zeta_{i},\eta_{i},\hat{\Psi},\hat{\Psi}_{\eta},\hat{\Psi}_{\zeta},\hat{\Psi}_{\zeta\zeta},\hat{\Psi}_{\eta\eta})\right]^{2}, \quad (3.7)$$

S represents the total number of discretized points for ζ and η. Gradient descent has been used to minimize (3.7), so the following derivatives with respect to DFNN's parameters are needed to compute this minimization. Let's take one hidden layer as an example to compute these derivations:The output of DFNN is :

$$N(\mu,p) = \sum_{j=1}^{n} w_{1j}^{2}\phi(w_{j1}^{1}\zeta + w_{j2}^{1}\eta + b_{j}^{1}) + b_{1}^{2}.$$

The derivative of $N(\mu,p)$ concerning ζ,η is as follows:

$$\frac{\partial N(\mu,p)}{\partial\zeta} = \sum_{j=1}^{n} w_{1j}^{2}w_{j1}^{1}\phi^{(1)}(w_{j1}^{1}\zeta + w_{j2}^{1}\eta + b_{j}^{1}),$$

$$\frac{\partial^{2}N(\mu,p)}{\partial\zeta^{2}} = \sum_{j=1}^{n} w_{1j}^{2}(w_{j1}^{1})^{2}\phi^{(2)}(w_{j1}^{1}\zeta + w_{j2}^{1}\eta + b_{j}^{1}),$$

$$\frac{\partial N(\mu,p)}{\partial\eta} = \sum_{j=1}^{n} w_{1j}^{2}w_{j2}^{1}\phi^{(1)}(w_{j1}^{1}\zeta + w_{j2}^{1}\eta + b_{j}^{1}),$$

$$\frac{\partial^{2}N(\mu,p)}{\partial\eta^{2}} = \sum_{j=1}^{n} w_{1j}^{2}(w_{j2}^{1})^{2}\phi^{(2)}(w_{j1}^{1}\zeta + w_{j2}^{1}\eta + b_{j}^{1}).$$

Where $\phi^{(1)}$ and $\phi^{(2)}$ are the first and second derivatives of the activation function ϕ, respectively. The following are the derivative of $N(\mu,p)$, $N_{\zeta}(\mu,p)$, $N_{\zeta\zeta}(\mu,p)$, $N_{\eta}(\mu,p)$, and $N_{\eta\eta}(\mu,p)$ with respect to w_{j1}^{1} and w_{j2}^{1}, respectively.

$$\frac{\partial N(\mu,p)}{\partial w_{j1}^1} = \sum_{j=1}^{n} w_{1j}^2 \zeta \phi^{(1)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1),$$
$$\frac{\partial N_\zeta(\mu,p)}{\partial w_{j1}^1} = \sum_{j=1}^{n} w_{1j}^2 (w_{j1}^1 \zeta \phi^{(2)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1) + \phi^{(1)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1)),$$
$$\frac{\partial N_{\zeta\zeta}(\mu,p)}{\partial w_{j1}^1} = \sum_{j=1}^{n} w_{1j}^2 ((w_{j1}^1)^2 \zeta \phi^{(3)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1) + 2w_{j1}^1 \phi^{(2)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1)),$$

$$\frac{\partial N(\mu,p)}{\partial w_{j2}^1} = \sum_{j=1}^{n} w_{1j}^2 \eta \phi^{(1)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1),$$
$$\frac{\partial N_\eta(\mu,p)}{\partial w_{j2}^1} = \sum_{j=1}^{n} w_{1j}^2 (w_{j2}^1 \eta \phi^{(2)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1) + \phi^{(1)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1)),$$
$$\frac{\partial N_{\eta\eta}(\mu,p)}{\partial w_{j2}^1} = \sum_{j=1}^{n} w_{1j}^2 ((w_{j2}^1)^2 \eta \phi^{(3)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1) + 2w_{j2}^1 \phi^{(2)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1)).$$

The derivative of $N(\mu,p), N_\zeta(\mu,p), N_\eta(\mu,p), N_{\eta\eta}(\mu,p)$ and $N_{\zeta\zeta}(\mu,p)$ with respect to w_{1j}^2 is as follows:

$$\frac{\partial N(\mu,p)}{\partial w_{1j}^2} = \sum_{j=1}^{n} \phi(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1),$$
$$\frac{\partial N_\zeta(\mu,p)}{\partial w_{1j}^2} = \sum_{j=1}^{n} w_{j1}^1 \phi^{(1)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1),$$
$$\frac{\partial N_{\zeta\zeta}(\mu,p)}{\partial w_{1j}^2} = \sum_{j=1}^{n} (w_{j1}^1)^2 \phi^{(2)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1),$$

$$\frac{\partial N_\eta(\mu,p)}{\partial w_{1j}^2} = \sum_{j=1}^{n} w_{j1}^1 \phi^{(1)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1),$$
$$\frac{\partial N_{\eta\eta}(\mu,p)}{\partial w_{1j}^2} = \sum_{j=1}^{n} (w_{j1}^1)^2 \phi^{(2)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1).$$

The derivative of $N(\mu,p), N_\zeta(\mu,p), N_\eta(\mu,p), N_{\eta\eta}(\mu,p)$ and $N_{\zeta\zeta}(\mu,p)$ concerning b_1^2 is as follows:

$$\frac{N(\mu,p)}{\partial b_1^2} = 1,$$
$$\frac{N_\zeta(\mu,p)}{\partial b_1^2} = 0,$$
$$\frac{N_{\zeta\zeta}(\mu,p)}{\partial b_1^2} = 0,$$
$$\frac{N_\eta(\mu,p)}{\partial b_1^2} = 0,$$
$$\frac{N_{\eta\eta}(\mu,p)}{\partial b_1^2} = 0.$$

The derivative of $N(\mu,p), N_\zeta(\mu,p), N_\eta(\mu,p), N_{\eta\eta}(\mu,p)$ and $N_{\zeta\zeta}(\mu,p)$ concerning b_j^1 is as follows:

$$\frac{N(\mu,p)}{\partial b_j^1} = \sum_{j=1}^{n} w_{1j}^2 \phi^{(1)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1),$$
$$\frac{N_\zeta(\mu,p)}{\partial b_j^1} = \sum_{j=1}^{n} w_{1j}^2 w_{j1}^1 \phi^{(2)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1),$$
$$\frac{N_{\zeta\zeta}(\mu,p)}{\partial b_j^1} = \sum_{j=1}^{n} w_{1j}^2 (w_{j1}^1)^2 \phi^{(3)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1),$$
$$\frac{N_\eta(\mu,p)}{\partial b_j^1} = \sum_{j=1}^{n} w_{1j}^2 w_{j2}^1 \phi^{(2)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1),$$
$$\frac{N_{\eta\eta}(\mu,p)}{\partial b_j^1} = \sum_{j=1}^{n} w_{1j}^2 (w_{j2}^1)^2 \phi^{(3)}(w_{j1}^1\zeta + w_{j2}^1\eta + b_j^1).$$

Finally, the derivative of (3.7) can be taken with respect to DFNN's parameters as follows:

$$\frac{\partial C(\mu,p)}{\partial w_{ij}^2}=\sum_{i=1}^{S}\left(\frac{\partial^\gamma\hat{\Psi}(\mu_i,p)}{\partial\eta^\gamma}-G(\zeta_i,\eta_i,\hat{\Psi},\hat{\Psi}_\eta,\hat{\Psi}_\zeta,\hat{\Psi}_{\zeta\zeta},\hat{\Psi}_{\eta\eta})\right)\left(\frac{\partial}{\partial w_{ij}^2}\left(\frac{\partial^\gamma\hat{\Psi}(\mu_i,p)}{\partial\eta^\gamma}\right)-\frac{\partial G}{\partial w_{ij}^2}\right),$$

$$\frac{\partial C(\mu,p)}{\partial w_{j1}^1}=\sum_{i=1}^{S}\left(\frac{\partial^\gamma\hat{\Psi}(\mu_i,p)}{\partial\eta^\gamma}-G(\zeta_i,\eta_i,\hat{\Psi},\hat{\Psi}_\eta,\hat{\Psi}_\zeta,\hat{\Psi}_{\zeta\zeta},\hat{\Psi}_{\eta\eta})\right)\left(\frac{\partial}{\partial w_{j1}^1}\left(\frac{\partial^\gamma\hat{\Psi}(\mu_i,p)}{\partial\eta^\gamma}\right)-\frac{\partial G}{\partial w_{j1}^1}\right),$$

$$\frac{\partial C(\mu,p)}{\partial w_{j2}^1}=\sum_{i=1}^{S}\left(\frac{\partial^\gamma\hat{\Psi}(\mu_i,p)}{\partial\eta^\gamma}-G(\zeta_i,\eta_i,\hat{\Psi},\hat{\Psi}_\eta,\hat{\Psi}_\zeta,\hat{\Psi}_{\zeta\zeta},\hat{\Psi}_{\eta\eta})\right)\left(\frac{\partial}{\partial w_{j2}^1}\left(\frac{\partial^\gamma\hat{\Psi}(\mu_i,p)}{\partial\eta^\gamma}\right)-\frac{\partial G}{\partial w_{j2}^1}\right),$$

$$\frac{\partial C(\mu,p)}{\partial b_j^1}=\sum_{i=1}^{S}\left(\frac{\partial^\gamma\hat{\Psi}(\mu_i,p)}{\partial\eta^\gamma}-G(\zeta_i,\eta_i,\hat{\Psi},\hat{\Psi}_\eta,\hat{\Psi}_\zeta,\hat{\Psi}_{\zeta\zeta},\hat{\Psi}_{\eta\eta})\right)\left(\frac{\partial}{\partial b_j^1}\left(\frac{\partial^\gamma\hat{\Psi}(\mu_i,p)}{\partial\eta^\gamma}\right)-\frac{\partial G}{\partial b_j^1}\right),$$

$$\frac{\partial C(\mu,p)}{\partial b_1^2}=\sum_{i=1}^{S}\left(\frac{\partial^\gamma\hat{\Psi}(\mu_i,p)}{\partial\eta^\gamma}-G(\zeta_i,\eta_i,\hat{\Psi},\hat{\Psi}_\eta,\hat{\Psi}_\zeta,\hat{\Psi}_{\zeta\zeta},\hat{\Psi}_{\eta\eta})\right)\left(\frac{\partial}{\partial b_1^2}\left(\frac{\partial^\gamma\hat{\Psi}(\mu_i,p)}{\partial\eta^\gamma}\right)-\frac{\partial G}{\partial b_1^2}\right).$$

Then, the parameters are updated using the following equations:

$$w_{ij}^{n+1}=w_{ij}^n-\delta\frac{\partial C}{\partial w_{ij}^n},$$

$$b_j^{n+1}=b_j^n-\delta\frac{\partial C}{\partial b_j^n}.$$

Notably, these derivatives are just for one hidden layer; if we increase the number of hidden layers, it will be more complicated, so to avoid this complexity, we used automatic differentiation [34] in Python to minimize the cost function.

4. Numerical results and discussion

Within this section, we demonstrate the application of DFNNs in solving FPDEs by presenting multiple illustrative examples. The neural network is trained for a total of 20000 epochs using 121 mesh points over the interval $[0, 1] \times [0, 1]$. The CPU time, measured in seconds, is provided to assess the method's efficiency. Additionally, MAE refers to the Maximum Absolute Error. By thoroughly examining each scenario, our objective is to highlight the efficacy and versatility of the DFNN approach in handling the complexities inherent in FPDEs.

Example 4.1. Consider the following nonlinear fractional-order heat equation [35]:

$$\frac{\partial^\gamma\Psi(\zeta,\eta)}{\partial\eta^\gamma}-\Psi(\zeta,\eta)\frac{\partial\Psi(\zeta,\eta)}{\partial\zeta}-\frac{\partial^2\Psi(\zeta,\eta)}{\partial\zeta^2}=0,\quad 0\leqslant\zeta\leqslant 1,\ 0\leqslant\eta\leqslant 1,\ 0<\gamma\leqslant 1,$$

with $\Psi(\zeta,0)=2-\zeta$, $\Psi(0,\eta)=\frac{2}{1+\eta}$, and $\Psi(1,\eta)=\frac{1}{1+\eta}$. where for $\gamma=1$, the exact solution is $\Psi(\zeta,\eta)=\frac{2-\zeta}{1+\eta}$. As per (3.5), the designated DFNN is: $\hat{\Psi}(\mu,p)=(1-\zeta)\frac{2}{1+\eta}+\zeta\frac{1}{1+\eta}+2-\zeta-2(1-\zeta)-\zeta+\zeta(1-\zeta)\eta N(\mu,p)$.

Table 1 shows the numerical results for different values of γ. The table indicates that the DFNN model's accuracy improves as γ approaches 1. When $\gamma=0.99$, the approximation is very close to the exact solution, reflecting high accuracy. Moreover, the CPU time remains consistently low for all values of γ. Table 2 presents the absolute errors (AEs) for various values of (ζ,η) with $\gamma=1$. The errors remain minimal, ranging from 1.43×10^{-6} to 1.79×10^{-6} across all evaluated points. Table 3 compares the MAEs, Cost function, and CPU times across various configurations of the DFNN method, each characterized by different neuron distributions, hidden layer depths, and epoch counts. The model configurations are labeled as follows: a four-layer model with 20000 epochs (DFNN-4L20 K), a four-layer model with 25000 epochs (DFNN-4L25 K), a four-layer model with 30000 epochs (DFNN-4L30 K), a single-layer model with 20000 epochs (DFNN-1L20 K), a single-layer model with 5000 epochs (DFNN-1L5 K), and a four-layer model with 5000 epochs (DFNN-4L5 K). The results clearly demonstrate that the DFNN-4L20 K configuration with $N_n=(35,25,20,15)$ delivers the best performance, achieving the lowest MAE of 1.32×10^{-5} and a corresponding cost function value of 2.12×10^{-8}, with a CPU time of 94.39548 seconds. In contrast, the DFNN-4L20 K configuration with a reduced neuron distribution of $N_n=(10,10,10,10)$ exhibits significantly poorer performance, with an MAE of 5.01×10^{-4} and a cost function value of 2.54×10^{-5}. Increasing the number of training epochs to 25 K for the configuration $N_n=(35,25,20,15)$ results in a slightly higher MAE of 1.88×10^{-5} and a cost function value of 2.18×10^{-8}, accompanied by an increase in CPU time to 117.80738 seconds. However, extending training to 30 K epochs leads to a noticeable decline in performance, with the MAE increasing sharply to 1.18×10^{-4} and the cost

Table 1. Numerical results for Example 4.1 with different values of γ.

		DFNN			
(ζ, η)	Exact	$\gamma = 0.5$	$\gamma = 0.7$	$\gamma = 0.99$	$\gamma = 1$
(0,0)	2	2	2	2	2
(0.1, 0.1)	1.727272	1.727265	1.727255	1.727277	1.727257
(0.2, 0.2)	1.500000	1.499998	1.499953	1.500014	1.499955
(0.3, 0.3)	1.307692	1.307738	1.307628	1.307717	1.307621
(0.4, 0.4)	1.142857	1.143001	1.142799	1.142886	1.142772
(0.5, 0.5)	1.000000	1.000283	0.999976	1.000027	0.999920
(0.6, 0.6)	0.875000	0.875436	0.875544	0.875020	0.874943
(0.7, 0.7)	0.764706	0.765262	0.764799	0.764714	0.764683
(0.8, 0.8)	0.666667	0.667246	0.666091	0.666800	0.666677
(0.9, 0.9)	0.578947	0.579374	0.579064	0.578939	0.578973
(1,1)	0.5	0.5	0.5	0.5	0.5
CPU Time(s)		110.67262	127.20520	125.10181	117.03224

Table 2. Absolute errors for Example 4.1 with $\gamma = 1$.

(ζ, η)	AEs
(0,0)	0
(0.1, 0.1)	1.79×10^{-6}
(0.2, 0.2)	5.23×10^{-6}
(0.3, 0.3)	9.18×10^{-6}
(0.4, 0.4)	1.19×10^{-5}
(0.5, 0.5)	1.26×10^{-5}
(0.6, 0.6)	1.14×10^{-5}
(0.7, 0.7)	8.44×10^{-6}
(0.8, 0.8)	4.83×10^{-6}
(0.9, 0.9)	1.43×10^{-6}
(1,1)	0

Table 3. Comparison of MAEs, Cost function, and CPU time for DFNN-1L and DFNN-4L when $\alpha = 1$ in Problem 4.1.

N_n	Method	MAEs	Cost function	CPU Time(s)
35	DFNN-1L5K	1.18×10^{-3}	4.73×10^{-4}	15.61976
(35, 25, 20, 15)	DFNN-4L20K	1.32×10^{-5}	2.12×10^{-8}	94.39548
(10, 10, 10, 10)	DFNN-4L20K	5.01×10^{-4}	2.54×10^{-5}	97.67032
(35, 25, 20, 15)	DFNN-4L25K	1.88×10^{-5}	2.18×10^{-8}	117.80738
(35, 25, 20, 15)	DFNN-4L30K	1.18×10^{-4}	1.46×10^{-6}	145.52670
35	DFNN-1L20K	2.74×10^{-4}	1.35×10^{-5}	59.51180
(35, 25, 20, 15)	DFNN-4L5K	1.37×10^{-4}	4.32×10^{-6}	64.62576

function rising to 1.46×10^{-6}, along with a substantial increase in computational time to 145.52670 seconds. The single-layer configurations, DFNN-1L5 K and DFNN-1L20 K, both with $N_n = 35$, result in higher MAEs of 1.18×10^{-3} and 2.74×10^{-4}, respectively, along with elevated cost function values of 4.73×10^{-4} and 1.35×10^{-5}. Additionally, the DFNN-4L5 K configuration with $N_n = (35, 25, 20, 15)$ achieves an MAE of 1.37×10^{-4} and a cost function value of 4.32×10^{-6}, with a CPU time of 64.62576 seconds. While this configuration outperforms the single-layer models in accuracy, it remains inferior to deeper models trained with more epochs. Given that the accuracy at 25 K epochs does not surpass that of the 20 K epoch configuration and that performance at 30 K epochs deteriorates significantly compared to both 20 K and 25 K, the DFNN-4L20 K configuration with $N_n = (35, 25, 20, 15)$ is identified as the optimal model. This configuration offers the best balance between accuracy and computational efficiency. Figure 2 presents a comparison of the exact and DFNN solutions for $\gamma = 1$. The close similarity between them indicates that the DFNN model provides an accurate approximation of the exact solution. Figure 3 shows the absolute errors and the cost function behavior over

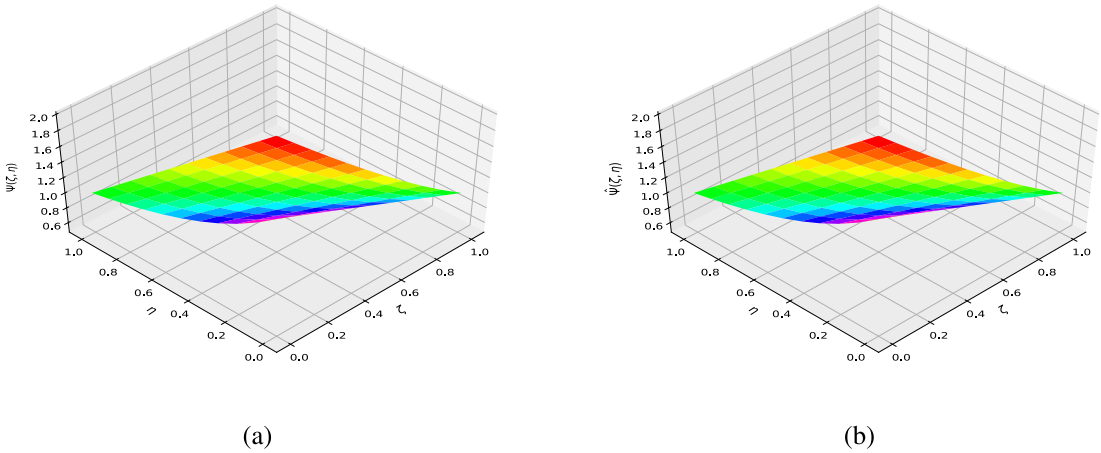


Figure 2. Comparison of the exact solution and DFNN for Example 4.1 with $\gamma = 1$. (a) represents the exact solution, while (b) presents the DFNN approximation.

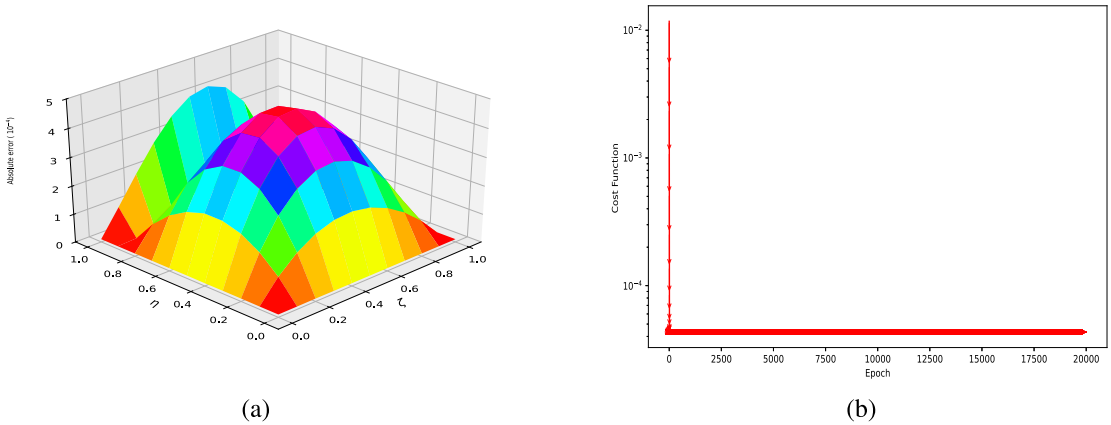


Figure 3. Absolute errors with $\gamma = 1$ and convergence of the cost function for Example 4.1. (a) represents the absolute errors, while (b) illustrates the convergence behavior of the cost function.

20000 epochs. The graph indicates that the cost function decreases rapidly and stabilizes near zero as the number of epochs increases, demonstrating the effective convergence of the model during training.

Example 4.2. Let consider the following FPDE of order $\gamma\,(0 < \gamma < 1)$ as [33]:

$$\begin{cases} \frac{\partial^{\gamma}\Psi(\zeta,\eta)}{\partial\eta^{\gamma}} = \frac{\partial^{2}\Psi(\zeta,\eta)}{\partial\zeta^{2}} + \left(\frac{6}{\Gamma(4-\gamma)}\eta^{3-\gamma} + \eta^{3}\right)\cos(\zeta) - e^{\zeta},\ 0 < \zeta < 1,\ \eta > 0, \\ \Psi(\zeta,\,0) = e^{\zeta},\ 0 < \zeta < 1, \\ \Psi(0,\,\eta) = \eta^{3} + 1,\ \Psi(1,\,\eta) = \eta^{3}\cos(1) + e,\ \eta > 0. \end{cases}$$

Where the exact solution is $\Psi(\zeta,\,\eta) = \eta^{3}\cos(\zeta) + e^{\zeta}$, according to (3.5), the assigned DFNN is $\hat{\Psi}(\mu,\,p) = e^{\zeta} + \eta^{3}(1-\zeta) + \zeta\eta^{3}cos(1) + \zeta(1-\zeta)\eta N(\mu,\,p)$.

Table 4 provides detailed numerical results across various γ values (0.1, 0.75, and 0.95), evaluated at different (ζ, η) points within the domain. The DFNN model demonstrates strong performance, closely approximating the exact solutions across the tested range of γ values.Furthermore, the CPU time decreases as γ increases. Table 5 presents the AEs for different γ values (0.1, 0.75, and 0.95). Figure 4 visually demonstrates the effectiveness of the DFNN model in approximating the exact solutions at $\gamma = 0.75$. Figure 5 presents the absolute errors and the reduction and subsequent stabilization of the cost function over 20000 epochs, emphasizing the efficiency of the training process. The sharp decline in the cost function early on signifies rapid initial learning.

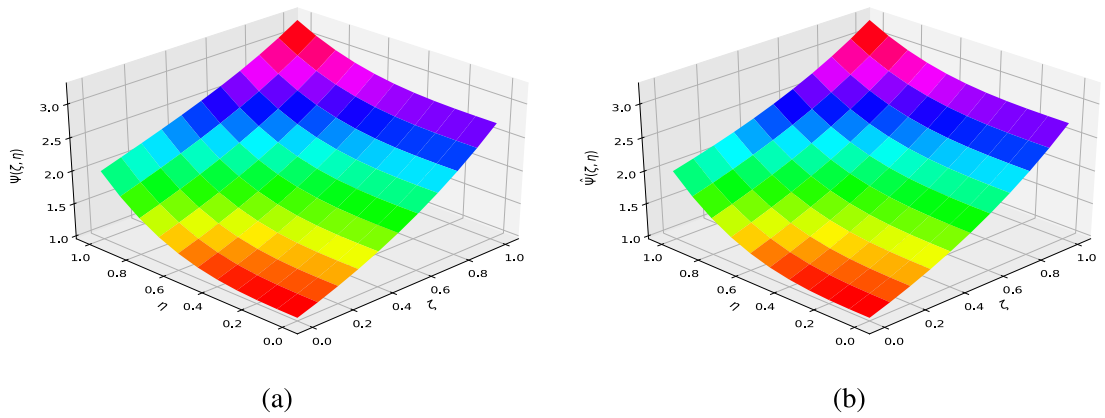


Figure 4. Comparison of the exact solution and DFNN for Example 4.2 with $\gamma = 0.75$. (a) Shows the exact solution, while (b) presents the DFNN approximation.

Table 4. Numerical results for Example 4.2 with different values of γ.

		DFNN		
(ζ, η)	Exact	$\gamma = 0.1$	$\gamma = 0.75$	$\gamma = 0.95$
(0,0)	1	1	1	1
(0.1, 0.1)	1.106166	1.107981	1.108270	1.108756
(0.2, 0.2)	1.229243	1.235304	1.236285	1.238005
(0.3, 0.3)	1.375653	1.386276	1.388114	1.391484
(0.4, 0.4)	1.550773	1.564191	1.566854	1.571964
(0.5, 0.5)	1.758419	1.771351	1.774639	1.781262
(0.6, 0.6)	2.000391	2.009074	2.012656	2.020252
(0.7, 0.7)	2.276093	2.277711	2.281163	2.288883
(0.8, 0.8)	2.582255	2.576671	2.579500	2.586195
(0.9, 0.9)	2.912757	2.904436	2.906116	2.910336
(1,1)	3.258584	3.258584	3.258584	3.258584
CPU Time(s)		120.22104	118.61125	99.85458

Table 5. Absolute errors for Example 4.2 with various γ values.

	AEs		
(ζ, η)	$\gamma = 0.1$	$\gamma = 0.75$	$\gamma = 0.95$
(0,0)	0	0	0
(0.1, 0.1)	1.81×10^{-3}	21.04×10^{-3}	2.59×10^{-3}
(0.2, 0.2)	6.06×10^{-3}	7.04×10^{-3}	8.76×10^{-3}
(0.3, 0.3)	1.06×10^{-2}	1.24×10^{-2}	1.58×10^{-2}
(0.4, 0.4)	1.34×10^{-2}	1.60×10^{-2}	2.11×10^{-2}
(0.5, 0.5)	1.29×10^{-2}	1.62×10^{-2}	2.28×10^{-2}
(0.6, 0.6)	8.68×10^{-3}	1.22×10^{-2}	1.98×10^{-2}
(0.7, 0.7)	1.61×10^{-3}	5.06×10^{-3}	1.27×10^{-2}
(0.8, 0.8)	5.58×10^{-3}	2.75×10^{-3}	3.94×10^{-3}
(0.9, 0.9)	8.32×10^{-3}	6.64×10^{-3}	2.42×10^{-3}
(1,1)	0	0	0

Example 4.3. Consider the following time- fractional telegraph equation of order γ ($1 < \gamma < 2$) [36]:

$$\begin{cases} \frac{\partial^{\gamma}\Psi(\zeta,\eta)}{\partial\eta^{\gamma}} + \frac{\partial\Psi(\zeta,\eta)}{\partial\eta} - \frac{\partial^{2}\Psi(\zeta,\eta)}{\partial\zeta^{2}} = h(\zeta,\eta),\ 0 \leqslant \zeta \leqslant 1,\ 0 < \eta \leqslant 1, \\ \Psi(\zeta,0) = 0,\ \Psi_{t}(\zeta,0) = 0, \\ \Psi(0,\eta) = 0, \quad \Psi(1,\eta) = 0. \end{cases}$$

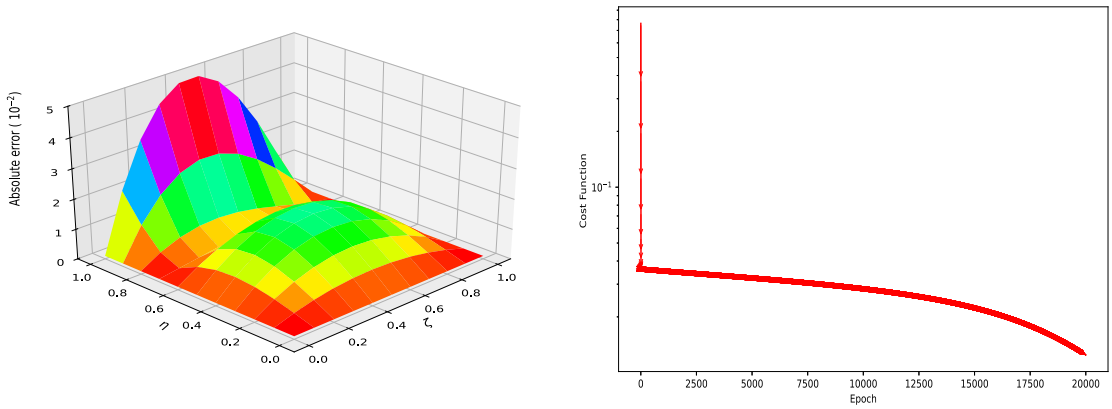


Figure 5. Absolute errors with $\gamma = 0.75$ and convergence of the cost function for Example 4.2. (a) Depicts the absolute errors, while (b) illustrates the convergence behavior of the cost function.

Table 6. Numerical results for Example 4.3 with different values of γ.

		DFNN		
(ζ, η)	Exact	$\gamma = 1.3$	$\gamma = 1.5$	$\gamma = 1.99$
(0.1, 0.1)	−0.000900	−0.000913	−0.000912	−0.000900
(0.2, 0.2)	−0.006400	−0.006491	−0.006490	−0.006398
(0.3, 0.3)	−0.018900	−0.019163	−0.019170	−0.018901
(0.4, 0.4)	−0.038400	−0.038925	−0.038958	−0.038411
(0.5, 0.5)	−0.062500	−0.063338	−0.063422	−0.062535
(0.6, 0.6)	−0.086400	−0.087537	−0.087693	−0.086471
(0.7, 0.7)	−0.102900	−0.104229	−0.104462	−0.103012
(0.8, 0.8)	−0.102400	−0.103697	−0.103976	−0.102538
(0.9, 0.9)	−0.072900	−0.073806	−0.074037	−0.073017
(1, 1)	0	0	0	0
CPU Time(s)		120.96981	138.05785	101.74970

Table 7. Absolute errors for Example 4.3 with various γ values.

	AEs		
(ζ, η)	$\gamma = 1.3$	$\gamma = 1.5$	$\gamma = 1.99$
(0.1, 0.1)	1.29×10^{-5}	1.24×10^{-5}	4.52×10^{-7}
(0.2, 0.2)	9.07×10^{-5}	9.001×10^{-5}	1.50×10^{-6}
(0.3, 0.3)	2.63×10^{-4}	2.70×10^{-4}	6.22×10^{-7}
(0.4, 0.4)	5.24×10^{-4}	5.57×10^{-4}	1.14×10^{-5}
(0.5, 0.5)	8.38×10^{-4}	9.21×10^{-4}	3.52×10^{-5}
(0.6, 0.6)	1.13×10^{-3}	1.29×10^{-3}	8.12×10^{-5}
(0.7, 0.7)	1.32×10^{-3}	1.56×10^{-3}	1.12×10^{-4}
(0.8, 0.8)	1.29×10^{-3}	1.57×10^{-3}	1.38×10^{-4}
(0.9, 0.9)	9.05×10^{-4}	1.13×10^{-3}	1.16×10^{-4}
(1, 1)	0	0	0

with $h(\zeta, \eta) = 2(\zeta^2 - \zeta)\eta\left(\frac{\Gamma(3-\gamma)+\eta^{1-\gamma}}{\Gamma(3-\gamma)}\right) - 2\eta^2$ and $\Psi(\zeta, \eta) = (\zeta^2 - \zeta)\eta^2$. The DFNN can be written as follows from (3.6): $\hat{\Psi}(\mu, p) = \zeta(1 - \zeta)\eta^2 N(\mu, p)$.

Table 6 illustrates the numerical results for Problem 4.3 across various γ values. Furthermore, CPU time remains consistently low for all γ settings. Table 7 presents the AEs across different values of γ. It is observed that as γ increases, the AEs generally decrease, indicating improved accuracy. Figure 6 provides a visual comparison between the exact

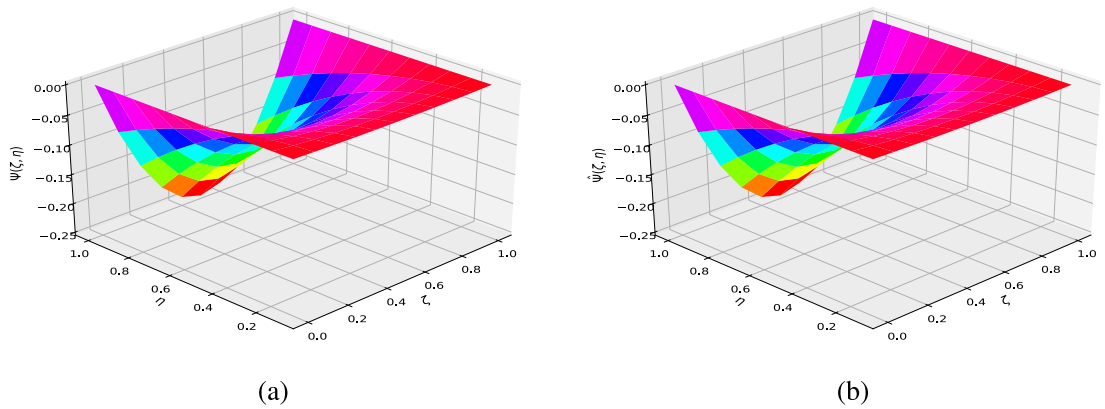


Figure 6. Comparison of the exact solution and DFNN for Example 4.3 with $\gamma = 1.99$. (a) Displays the exact solution, while (b) presents the DFNN approximation.

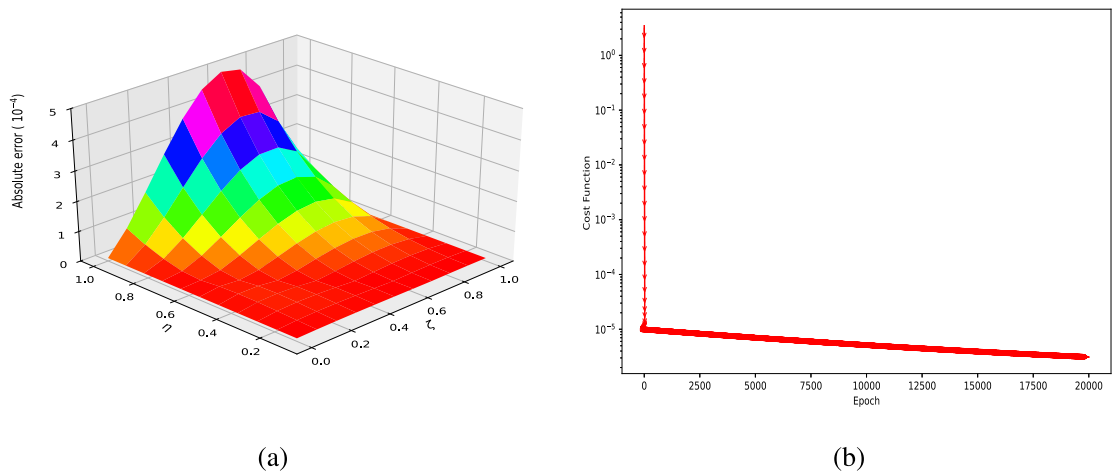


Figure 7. Absolute errors with $\gamma = 1.99$ and convergence of the cost function for Example 4.3. (a) Depicts the absolute errors, while (b) illustrates the convergence behavior of the cost function.

solutions and the DFNN approximations, demonstrating the model's capability to effectively capture the underlying dynamics of the problem. Figure 7 shows the absolute error surface, offering a visual representation of the discrepancies between the DFNN and exact solutions across the domain, along with the cost function's trajectory over training epochs. The sharp initial decrease in the cost function, followed by a plateau, suggests that the model quickly adapts to the problem structure, achieving a significant reduction in error early in the training process.

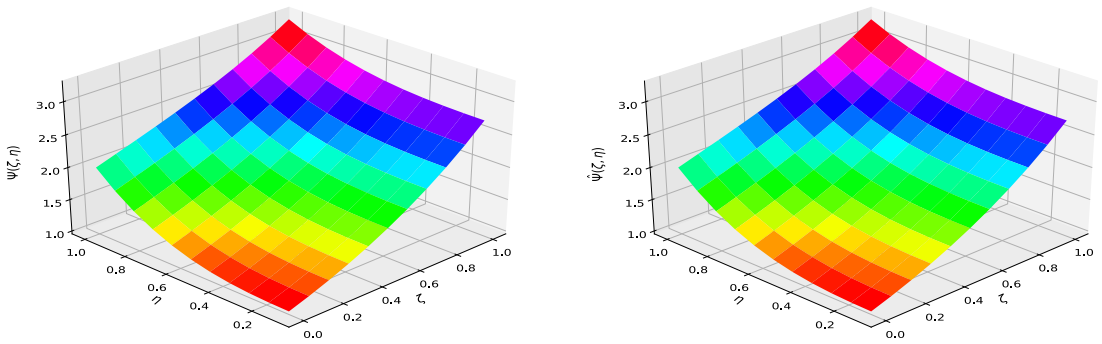
Example 4.4. Consider the following FPDE of order $\gamma(1 < \gamma < 2)$ [37]:

$$\begin{cases} \frac{\partial^{\gamma}\Psi(\zeta,\eta)}{\partial\eta^{\gamma}} = \frac{\partial^{2}\Psi(\zeta,\eta)}{\partial\zeta^{2}} + \left(\frac{6}{\Gamma(4-\gamma)}\eta^{3-\gamma} + \eta^{3}\right)\cos(\zeta) - e^{\zeta},\ 0 < \zeta < 1,\ \eta > 0, \\ \Psi(\zeta,\ 0) = e^{\zeta},\ \Psi_{t}(\zeta,\ 0) = 0,\ 0 < \zeta < 1, \\ \Psi(0,\ \eta) = \eta^{3} + 1,\ \Psi(1,\ \eta) = \eta^{3}\cos(1) + e,\ \eta > 0. \end{cases}$$

Where the exact solution is $\Psi(\zeta,\ \eta) = \eta^{3}\cos(\zeta) + e^{\zeta}$.

From (3.6), the DNN may be expressed as follows: $\hat{\Psi}(\mu,\ p) = (1 - \zeta)(\eta^{3} + 1) + \zeta(\eta^{3}cos(1) + e) + (1 - \eta^{2})(e^{-\zeta} - (1 - \zeta) - \zeta e) + \zeta(1 - \zeta)\eta^{2}N(\mu,\ p)$.

Table 8 presents numerical results for different values of γ. The values are evaluated at various points (ζ, η), highlighting the model's accuracy in approximating the exact solutions. Furthermore, for all settings of γ, the CPU time remains consistently low. Table 9 provides a comparison of absolute errors between the proposed DFNN method and the PRKM method [37] for various (ζ, η) values when $\gamma = 1.3$. The table clearly shows that the DFNN method consistently outperforms PRKM [37], yielding lower absolute errors across all parameter



(a) (b)

Figure 8. Comparison of the exact solution and DFNN with $\gamma = 1.95$ for Example 4.4. (a) Displays the exact solution, while (b) presents the DFNN solution.

Table 8. Numerical results for Example 4.4 with different values of γ.

		DFNN		
(ζ, η)	Exact	$\gamma = 1.3$	$\gamma = 1.7$	$\gamma = 1.95$
(0.1,0.1)	1.106166	1.106400	1.105654	1.106283
(0.2,0.2)	1.229243	1.230610	1.225875	1.229905
(0.3,0.3)	1.375653	1.378859	1.366573	1.377156
(0.4,0.4)	1.550773	1.555706	1.534195	1.553018
(0.5,0.5)	1.758419	1.764077	1.734667	1.760947
(0.6,0.6)	2.000391	2.005245	1.972277	2.002617
(0.7,0.7)	2.276093	2.278747	2.248544	2.277608
(0.8,0.8)	2.582255	2.582226	2.561062	2.583033
(0.9,0.9)	2.912757	2.911176	2.902329	2.913098
(1,1)	3.258584	3.258584	3.258584	3.258584
CPU Time(s)		127.03214	112.54682	103.44827

Table 9. Comparison of absolute errors for Example 4.4 with $\gamma = 1.3$.

(ζ, η)	DFNN	PRKM($h = 0.1$)[37]
$(0.1, 0.1)$	2.34007×10^{-4}	2.59785×10^{-4}
$(0.2, 0.2)$	1.36709×10^{-3}	5.06435×10^{-3}
$(0.3, 0.3)$	3.20589×10^{-3}	1.37166×10^{-2}
$(0.4, 0.4)$	4.93347×10^{-3}	2.48823×10^{-2}
$(0.5, 0.5)$	5.65838×10^{-3}	3.66127×10^{-2}
$(0.6, 0.6)$	4.85373×10^{-3}	4.63938×10^{-2}
$(0.7, 0.7)$	2.65384×10^{-3}	5.12172×10^{-2}
$(0.8, 0.8)$	2.86102×10^{-5}	4.76556×10^{-2}
$(0.9, 0.9)$	1.58071×10^{-3}	3.19344×10^{-2}

values, further demonstrating its superior accuracy. Figure 8 displays a comparison between the exact solution and the DFNN solution for $\gamma = 1.95$. Figure 9 illustrates the cost function and absolute error surface for $\gamma = 1.95$, underscoring the efficiency of model training and its accuracy in approximating the exact solution.

Example 4.5. The time fractional telegraph equation is expressed as follows [38]:

$$\begin{cases} \frac{\partial^{1.8}\Psi(\zeta,\eta)}{\partial\eta^{1.8}} + \frac{\partial^{0.8}\Psi(\zeta,\eta)}{\partial\eta^{0.8}} = \frac{\partial^2\Psi(\zeta,\eta)}{\partial\zeta^2} + f(\zeta,\eta),\ 0 \leqslant \zeta \leqslant 1,\ 0 \leqslant \eta \leqslant 1, \\ \Psi(\zeta,0) = \zeta(1-\zeta),\ \Psi_\eta(\zeta,0) = 0, \quad 0 < \zeta < 1,\ \Psi(0,\eta) = 0,\ \Psi(1,\eta) = 0. \end{cases}$$

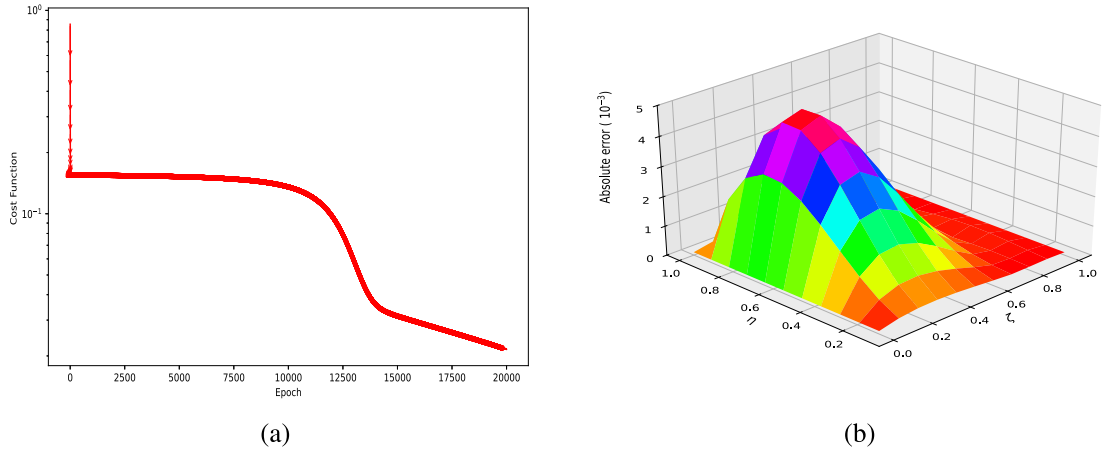


Figure 9. Convergence of the cost function and absolute errors for Example 4.4 with $\gamma = 1.95$. (a) Shows the convergence behavior of the cost function, while (b) presents the absolute errors.

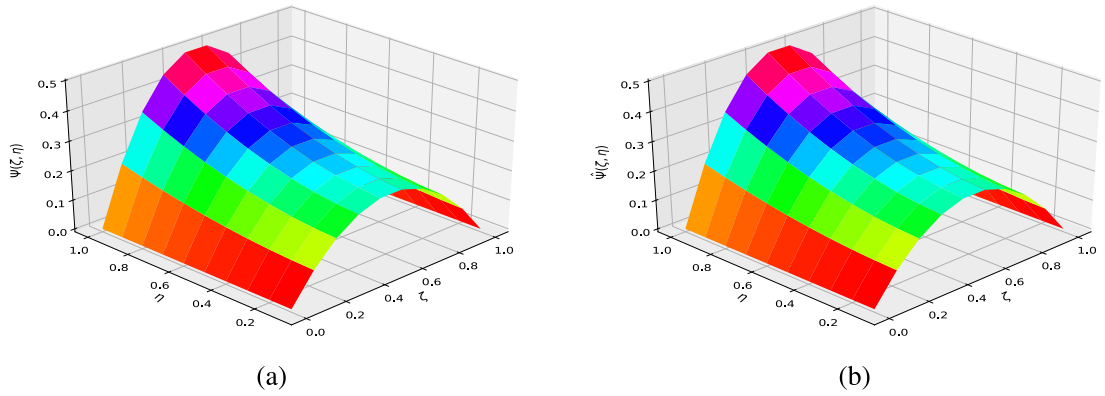


Figure 10. Comparison of the exact solution and DFNN for Example 4.5. (a) Shows the exact solution, while (b) presents the DFNN solution.

Table 10. Numerical results for Example 4.5.

(ζ, η)	Exact	DFNN	AEs
(0.1, 0.1)	0.090900	0.090907	7.29×10^{-6}
(0.2, 0.2)	0.166400	0.166453	5.30×10^{-5}
(0.3, 0.3)	0.228900	0.229060	1.60×10^{-4}
(0.4, 0.4)	0.278400	0.278732	3.32×10^{-4}
(0.5, 0.5)	0.312500	0.313052	5.52×10^{-4}
(0.6, 0.6)	0.326400	0.327179	7.78×10^{-4}
(0.7, 0.7)	0.312900	0.313845	9.45×10^{-4}
(0.8, 0.8)	0.262400	0.263359	9.58×10^{-4}
(0.9, 0.9)	0.162900	0.163595	6.95×10^{-4}
(1, 1)	0	0	0
CPU Time(s)		116.86251	

Where the exact solution is $\Psi(\zeta, \eta) = (1 + \eta^2)\zeta(1 - \zeta)$ and $f(\zeta, \eta) = \zeta(1 - \zeta)\left(\frac{2}{\Gamma(1.2)}\eta^{0.2} + \frac{2}{\Gamma(2.2)}\eta^{1.2}\right) + 2(1 + \eta^2)$.

Based on equation (3.6), DFNN can be written as: $\hat{\Psi}(\mu, p) = (1 - \zeta)\zeta(1 - \eta^2) + \zeta(1 - \zeta)\eta^2 N(\mu, p)$.

Table 10 presents numerical results, highlighting the model's accuracy across the domain, with minimal discrepancies noted between the exact and DFNN solutions, as evidenced by the small AEs. Figure 10 provides a

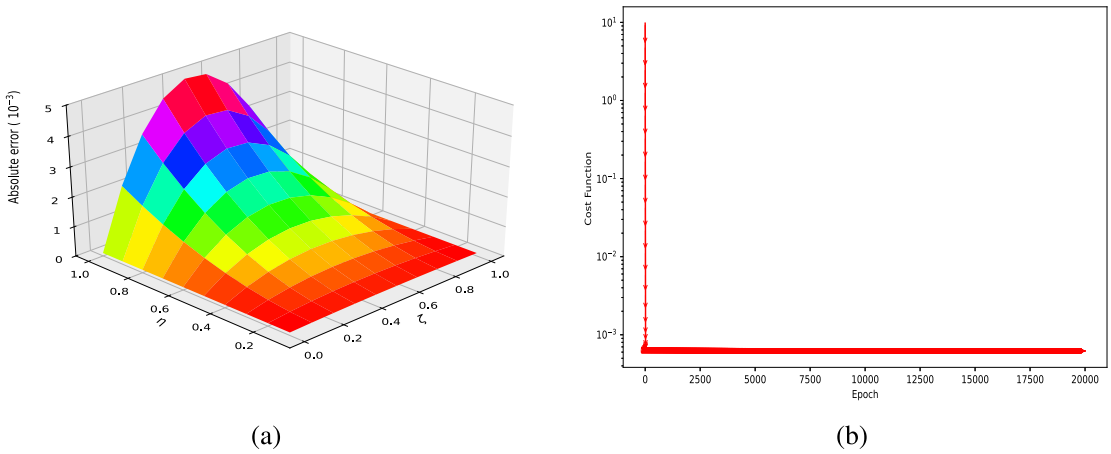


Figure 11. Absolute errors and convergence of the cost function for Example 4.5. (a) Depicts the absolute errors, while (b) illustrates the convergence behavior of the cost function.

visual comparison between the exact solutions and the DFNN approximations, demonstrating the model's capability to effectively capture the underlying dynamics of the problem. Figure 11 provides a visual representation of the absolute error surface, illustrating the discrepancies between the DFNN and exact solutions across the domain. Additionally, it depicts the trajectory of the cost function over training epochs. The steep initial decline in the cost function is followed by a stabilization phase.

5. Conclusion

This study introduces a novel algorithm based on DFNNs for solving FPDEs, demonstrating significant improvements over existing methods. The proposed approach leverages gradient descent optimization for efficient training of the network. The algorithm's structure is built around two essential components: the approximation of fractional derivatives and the design of the DFNN architecture. A key strength of this method is its flexibility in choosing activation functions, which enhances its adaptability and effectiveness in solving a wide range of FPDEs. Comparative results from five test cases show that the proposed scheme consistently outperforms existing methods, achieving superior accuracy and minimizing absolute errors. Additionally, the majority of network parameters were optimally tuned, with the cost function converging to near zero by the end of each training epoch. This highlights the robustness and accuracy of the proposed approach, underscoring its advantages over traditional techniques.

Acknowledgments

The authors are very thankful to Malaysia Ministry of Education for awarded Fundamental Research Grant Scheme (Ref. No. FRGS/1/2022/STG06/UPM/02/2) for supporting this work.

Data availability statement

The data cannot be made publicly available upon publication because they contain sensitive personal information. The data that support the findings of this study are available upon reasonable request from the authors.

ORCID iDs

Amina Ali https://orcid.org/0000-0001-5194-1300
Norazak Senu https://orcid.org/0000-0001-8614-8281
Ali Ahmadian https://orcid.org/0000-0002-0106-7050

References

[1] Li C and Cai M 2019 *Theory and Numerical Approximations of Fractional Integrals and Derivatives* (SIAM)
[2] Li C and Zeng F 2015 *Numerical Methods for Fractional Calculus* vol 24 (CRC Press)
[3] Salahshour S, Allahviranloo T and Abbasbandy S 2012 Solving fuzzy fractional differential equations by fuzzy laplace transforms *Commun. Nonlinear Sci. Numer. Simul.* **17** 1372–81
[4] Yu J and Feng Y 2024 Group classification of time fractional black-scholes equation with time-dependent coefficients *Fractional Calculus and AppliedAnalysis* **27** 2335–58
[5] Yu J and Feng Y 2024 On the generalized time fractional reaction-diffusion equation: Lie symmetries, exact solutions and conservation laws *Chaos,Solitons & Fractals* **182** 114855
[6] McCulloch W S and Pitts W 1943 A logical calculus of the ideas immanent in nervous activity *The Bulletin of Mathematical Biophysics* **5** 115–33
[7] Yadav N *et al* 2015 *An Introduction to Neural Networkmethods for Differential Equations* vol 1 (Springer)
[8] Hinton G E, Osindero S and Teh Y-W 2006 A fast learning algorithm for deep belief nets *Neural Comput.* **18** 1527–54
[9] Krizhevsky A, Sutskever I and Hinton G E 2017 Imagenet classification with deep convolutional neural networks *Commun. ACM* **60** 84–90
[10] Cotter N E 1990 The stone-weierstrass theorem and its application to neural networks *IEEEtransactions on Neural Networks* **1** 290–5
[11] Hornik K, Stinchcombe M and White H 1989 Multilayer feedforward networks are universal approximators *Neural Netw.* **2** 359–66
[12] Hornik K, Stinchcombe M and White H 1990 Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks *Neural Netw.* **3** 551–60
[13] Lagaris I E, Likas A and Fotiadis D I 1998 Artificial neural networks for solving ordinary and partial differential equations *IEEE Transactions Onneural Networks* **9** 987–1000
[14] Lagaris I E, Likas A C and Papageorgiou D G 2000 Neural-network methods for boundary value problems with irregular boundaries *IEEE Transactionson Neural Networks* **11** 1041–9
[15] McFall K S and Mahan J R 2009 Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions *IEEE Trans. Neural Netw.* **20** 1221–33
[16] Chaudhari P, Oberman A, Osher S, Soatto S and Carlier G 2017 Partial differential equations for training deep neural networks *51st Asilomar Conference on Signals, Systems, and Computers* vol 2017 (IEEE) pp 1627–31
[17] Han J *et al* 2017 Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations *Communications in Mathematics and Statistics* **5** 349–80
[18] Lu L, Meng X, Mao Z and Karniadakis G E 2021 Deepxde: a deep learning library for solving differential equations *SIAM Rev.* **63** 208–28
[19] Yu B *et al* 2018 The deep ritz method: a deep learning-based numerical algorithm for solving variational problems *Communications in Mathematics and Statistics* **6** 1–12
[20] Raissi M, Perdikaris P and Karniadakis G E 2019 Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations *J. Comput. Phys.* **378** 686–707
[21] Berg J and Nyström K 2018 A unified deep artificial neural network approach to partial differential equations in complex geometries *Neurocomputing* **317** 28–41
[22] Jin X, Cai S, Li H and Karniadakis G E 2021 Nsfnets (navier-stokes flow nets): physics-informed neural networks for the incompressible navier-stokes equations *J. Comput. Phys.* **426** 109951
[23] Sheng H and Yang C 2021 Pfnn: a penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries *J. Comput. Phys.* **428** 110085
[24] Ye Y, Fan H, Li Y, Liu X and Zhang H 2022 Deep neural network methods for solving forward and inverse problems of time fractional diffusion equations with conformable derivative *Neurocomputing* **509** 177–92
[25] Wei J-L, Wu G-C, Liu B-Q and Zhao Z 2022 New semi-analytical solutions of the time-fractional fokker-planck equation by the neural network method *Optik* **259** 168896
[26] Fang X, Qiao L, Zhang F and Sun F 2023 Explore deep network for a class of fractional partial differential equations *Chaos, Solitons Fractals* **172** 113528
[27] Shi J, Yang X and Liu X 2024 A novel fractional physics-informed neural networks method for solving the time-fractional huxley equation *Neural Computing and Applications* **36** 19097–119
[28] Shi J, Liu X and Yang X 2025 Data-driven solutions and parameter estimation of the high-dimensional time-fractional reaction-diffusion equations using an improved fpinn method *Nonlinear Dyn.* 1–28
[29] Podlubny I 2000 Matrix approach to discrete fractional calculus *Fractional Calculus and Applied Analysis* **3** 359–86
[30] Shen S, Liu F, Chen J, Turner I and Anh V 2012 Numerical techniques for the variable order time fractional diffusion equation *Appl. Math. Comput.* **218** 10861–70
[31] Fahad H M, Rehman M U and Fernandez A 2023 On laplace transforms with respect to functions and their applications to fractional differential equations *Math. Methods Appl. Sci.* **46** 8304–23
[32] Podlubny I 1999 Fractional differential equations *Mathematics in Science and Engineering*
[33] Ren J, Sun Z-Z and Dai W 2016 New approximations for solving the caputo-type fractional partial differential equations *Appl. Math. Modelling* **40** 2625–36
[34] Baydin A G, Pearlmutter B A, Radul A A and Siskind J M 2018 Automatic differentiation in machine learning: a survey *Journal of Marchine Learning Research* **18** 1–43
[35] Faheem M, Khan A and Raza A 2022 A high resolution hermite wavelet technique for solving space-time-fractional partial differential equations *Math. Comput. Simul.* **194** 588–609
[36] Mamadu E J, Njoseh I N and Ojarikre H I 2022 Space discretization of time-fractional telegraph equation with mamadu-njoseh basis functions *Applied Mathematics* **13** 760–73
[37] Wang Y-L, Jia L-na and Zhang H-lu 2019 Numerical solution for a class of space-time fractional equation by the piecewise reproducing kernel method *Int. J. Comput. Math.* **96** 2100–11
[38] Wu L and Yang X 2020 An efficient alternating segment parallel difference method for the time fractional telegraph equation *Advances inMathematical Physics* **2020** 1–11