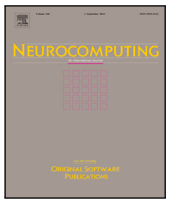
Neurocomputing 623 (2025) 129414

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom



fKAN: Fractional Kolmogorov–Arnold Networks with trainable Jacobi basis functions

Alireza Afzal Aghaei

Independent Researcher, Isfahan, Iran

ARTICLE INFO

Communicated by L. Tong

Keywords:
Activation functions
Jacobi polynomials
Kolmogorov–Arnold Networks
Physics-informed deep learning

ABSTRACT

Recent advancements in neural network design have given rise to the development of Kolmogorov–Arnold Networks (KANs), which enhance interpretability and precision of these systems. This paper presents the Fractional Kolmogorov–Arnold Network (fKAN), a novel neural network architecture that incorporates the distinctive attributes of KANs with a trainable adaptive fractional-orthogonal Jacobi function as its basis function. By leveraging the unique mathematical properties of fractional Jacobi functions, including simple derivative formulas, non-polynomial behavior, and activity for both positive and negative input values, this approach ensures efficient learning and enhanced accuracy. The proposed architecture is evaluated across a range of tasks in deep learning and physics-informed deep learning. Precision is tested on synthetic regression data, image classification, transfer learning, image denoising, and sentiment analysis. Additionally, the performance is measured on various differential equations, including ordinary, partial, and fractional delay differential equations. The results demonstrate that integrating fractional Jacobi functions into KANs significantly improves training speed and performance across diverse fields and applications.

1. Introduction

Neural networks, inspired by the structure and functionality of the human brain, have undergone significant evolution since the introduction of artificial neurons in the 1940s [1]. Over the years, innovations such as Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks [2], and Restricted Boltzmann Machines [1] have profoundly transformed various fields by addressing key challenges in data representation, sequential processing, and feature learning. Central to these architectures is the process of learning through a combination of linear transformations and nonlinear activation functions, enabling neural networks to approximate highly complex mappings [2].

Among the recent advancements in this field, Kolmogorov–Arnold Neural Networks (KANs) have emerged as a robust framework for function approximation [3]. Built upon the Kolmogorov–Arnold representation theorem, KANs decompose multivariate functions into sums of univariate functions, with each component parameterized by learnable elements [4]. A distinctive feature of KANs is the incorporation of learnable activation functions along network edges, which facilitates dynamic adaptation to data distributions and improves generalization capabilities. Traditional KAN architectures typically utilize spline-based activations, offering smooth and continuous transformations. Nevertheless, the computational demands of splines, stemming from their piecewise construction, and their limited adaptability to certain tasks highlight the need for further advancements [5].

This work introduces a novel extension of KANs, referred to as Fractional KAN (fKAN), which employs fractional-order Jacobi polynomials as replacement of splines. The fractional nature of these functions offers greater flexibility than the fixed polynomial basis used in splines, enabling improved accuracy across a majority of tasks. Furthermore, unlike splines, fractional Jacobi polynomials are not piecewise, leading to significantly more efficient implementation.

To validate the proposed fKAN, we conduct extensive experiments on diverse tasks, including regression, classification, transfer learning, image denoising, and sentiment analysis. Additionally, the method is evaluated in physics-informed deep learning contexts by approximating solutions to challenging differential equations such as the Lane–Emden equation, the one-dimensional Burgers equation, and a delay differential equation with the Caputo fractional derivative. To support these evaluations, a specialized neural network block is introduced, enabling the seamless integration of the proposed activation functions into existing architectures, including CNNs and LSTMs.

The remainder of this paper is structured as follows. Section 2 reviews related works on learnable activation networks and the KAN framework. Section 3 provides essential background on MLPs and KANs and introduces fractional order of Jacobi polynomials. Section 4 details the development of the fKAN architecture. Section 5 presents experimental validation across benchmark tasks. Finally, Section 6 concludes with a discussion of the findings and potential future directions.

E-mail address: alirezaafzalaghaei@gmail.com.

https://doi.org/10.1016/j.neucom.2025.129414
Received 14 June 2024; Received in revised form 3 January 2025; Accepted 8 January 2025
Available online 13 January 2025

2. Related work

Learnable activation functions have a rich history [6]. Early studies introduced trainable variants of classical activation functions, such as sigmoid [7], hyperbolic tangent [8], and adaptive polynomial functions [9]. Over time, adaptive activation functions expanded to include more modern innovations, such as adaptive versions of the ReLU. Notable examples include Parametric ReLU by He et al. [10], the Learnable Extended Activation Function (LEAF) by Bodyanskiy et al. [11], modReLU by Arjovsky et al. [12], and DELU by Pishchik [13]. These activation functions allow parameters to be learned during training, enabling them to adapt dynamically to the input data. This adaptability addresses issues such as the dead neuron problem in ReLU and improves the network's overall learning capacity by optimizing more components of the model.

Splines and B-splines form another class of tunable functions used in neural networks. For example, Scardapane et al. [14] utilized cubic splines to learn activation functions directly from data. Fakhoury et al. [15] proposed interpretable spline-based architectures, while Neumayer et al. [16] focused on approximating Lipschitz functions with spline-based networks. Bohra et al. [17] introduced an efficient B-spline-based training approach for deep networks, eliminating the need for traditional activation functions. Kolmogorov–Arnold Networks built upon these advancements by incorporating spline-based findings within a deep learning framework inspired by the Kolmogorov–Arnold representation theorem. This approach emphasized compact and efficient network architectures with desirable depth.

KANs have inspired multiple extensions to address their computational and application-specific challenges. Wavelet KANs [18], introduced by Bozorgasl et al. integrated orthogonal wavelets into the KAN framework, enhancing performance and interpretability. Similarly, Fourier KANs [19] leveraged Fourier basis functions, demonstrating significant improvements in graph-based recommendation systems within collaborative filtering tasks. Temporal KANs [20], proposed by Genet et al. extended KANs to sequential data by incorporating LSTM-inspired architectures, achieving strong results in time-series analysis and market notional trade. Samadi et al. [21] studied the smoothness properties of KANs, leading to smooth KANs with enhanced generalization capabilities. Abueidda et al. [22] extended KANs to solve complex operator equations in mechanical problems through deep operator networks. Additionally, Koenig et al. [23] applied KANs to solve ordinary differential equations, while Aghaei [24] focused on the optimal control of partial differential equations.

Among the various types of KANs, polynomial KANs [25] stand out as a computationally efficient alternative due to the simplicity, continuity, and global adaptability of polynomials. However, theoretical limitations, such as their inability to serve as universal function approximators [26,27], and practical issues like gradient explosion, numerical instability (e.g., Runge's phenomenon), and difficulties in backpropagation remain significant challenges. Theories of orthogonal polynomials [28–30] have addressed some of these concerns, leading to neural networks based on orthogonal polynomial families.

Orthogonal polynomial-based neural networks have demonstrated utility across various domains. Orthogonal polynomials like Chebyshev, Legendre, and Jacobi mitigate derivative computation issues in backpropagation while inherently bounding neuron activation ranges [31]. For instance, Chebyshev polynomials have been successfully employed in applications like image denoising [32] and hydraulic generator regulation [33]. Legendre polynomials have been employed for max-pooling layer modeling [34], and Laguerre polynomials have been used in wind power forecasting [35] and wireless sensor networks [36]. Hermite functions were utilized by Ebrahimi et al. [37] to classify ECG signals, while Bernstein polynomials were applied to graph neural networks by Guo et al. [38]. Building on these advancements, recent applications of orthogonal polynomials in KANs include Chebyshev KANs [39–41] and Jacobi KANs [42], highlighting their growing significance in function approximation tasks.

The generalized hypergeometric function serves as a unifying framework for many of these orthogonal basis functions. This function can be reduced to specific cases like Chebyshev, Jacobi, and Legendre polynomials [30,43,44]. Fig. 1 illustrates these hierarchical relationships, demonstrating how general mathematical frameworks give rise to specialized activation functions tailored for specific tasks. For instance, Chebyshev polynomials can be derived as parameterized forms of Gegenbauer or Jacobi functions.

Jacobi polynomials, positioned mid-hierarchy in Fig. 1, form the foundation of this work, where we propose a novel trainable function for KANs. These polynomials share desirable properties such as differentiability, boundedness, and orthogonality with lower-hierarchy counterparts, but they stand out due to their enhanced tunability and adaptability to a wider range of functions. Building on their established polynomial structures [45,46], this study extends them into fractional spaces to address the inherent limitations of traditional orthogonal polynomials. Fractional orthogonal functions, incorporating fractional order parameters [47], offer solutions to challenges in numerical analysis, differential equation simulation [48,49], and neural network design [50]. These functions have demonstrated improved prediction accuracy and trainability in various forms, including B-splines [51,52], Bernoulli functions [53], Jacobi functions [54], and Legendre polynomials [55]. This paper focuses on optimizing fractional parameters alongside Jacobi hyperparameters during training, bridging theoretical developments with practical implementations to extend KAN architectures into new application domains.

3. Background

In this section, we provide an overview of MLPs and KANs, focusing on the role of activation functions within these networks. We then explore the key characteristics that define an effective activation function. Following this, we discuss Jacobi polynomials and their relationship to other orthogonal functions, highlighting their suitability for neural networks.

3.1. Multi-Layer Perceptron

A Multi-Layer Perceptron, grounded in the universal approximation theory [65–67], aims to find a weighted summation of various basis functions. Specifically, an approximation to function $\chi(\cdot)$ can be done using:

$$\chi(\boldsymbol{\zeta}) \approx \hat{\chi}(\boldsymbol{\zeta}) = \sum_{q=1}^{Q} \dot{\theta}_q \sigma \left(b_q + \sum_{j=1}^{d} \theta_{qj} \boldsymbol{\zeta}_j \right). \tag{1}$$

The universal approximation theory states that a network with a single hidden layer containing a finite number of neurons, can approximate continuous functions on compact subsets of $\mathbb{R}^d$, under mild assumptions on the activation function $\sigma(\cdot)$. In a mathematical sense, it ensures that exists some $\theta, \dot{\theta}$ and b such that:

$$\sup_{\boldsymbol{\zeta} \in \mathbb{R}^d} \| \chi(\boldsymbol{\zeta}) - \hat{\chi}(\boldsymbol{\zeta}) \| < \epsilon. \tag{2}$$

In practice, the accuracy of this estimation improves as a result of the non-linear nested composition of these approximations [68–70]. Mathematically, an MLP can be represented as:

$$\begin{aligned} &\mathcal{H}_0 = \boldsymbol{\zeta}, && \boldsymbol{\zeta} \in \mathbb{R}^d, \\ &\mathcal{H}_i = \sigma_i(\theta^{(i)} \mathcal{H}_{i-1} + b^{(i)}), && i = 1, 2, \ldots, L-1, \\ &\mathcal{H}_L = \psi(\theta^{(L)} \mathcal{H}_{L-1} + b^{(L)}). \end{aligned} \tag{3}$$

In this definition, $\boldsymbol{\zeta}$ represents the input sample fed into the input layer $\mathcal{H}_0$. The output of the ith layer, $\mathcal{H}_i$, is computed based on the output of the previous layer, $\mathcal{H}_{i-1}$. This computation involves matrix multiplication between the learnable weights $\theta^{(i)}$ and the output of the previous layer, $\mathcal{H}_{i-1}$. This weight matrix can be either dense or possess

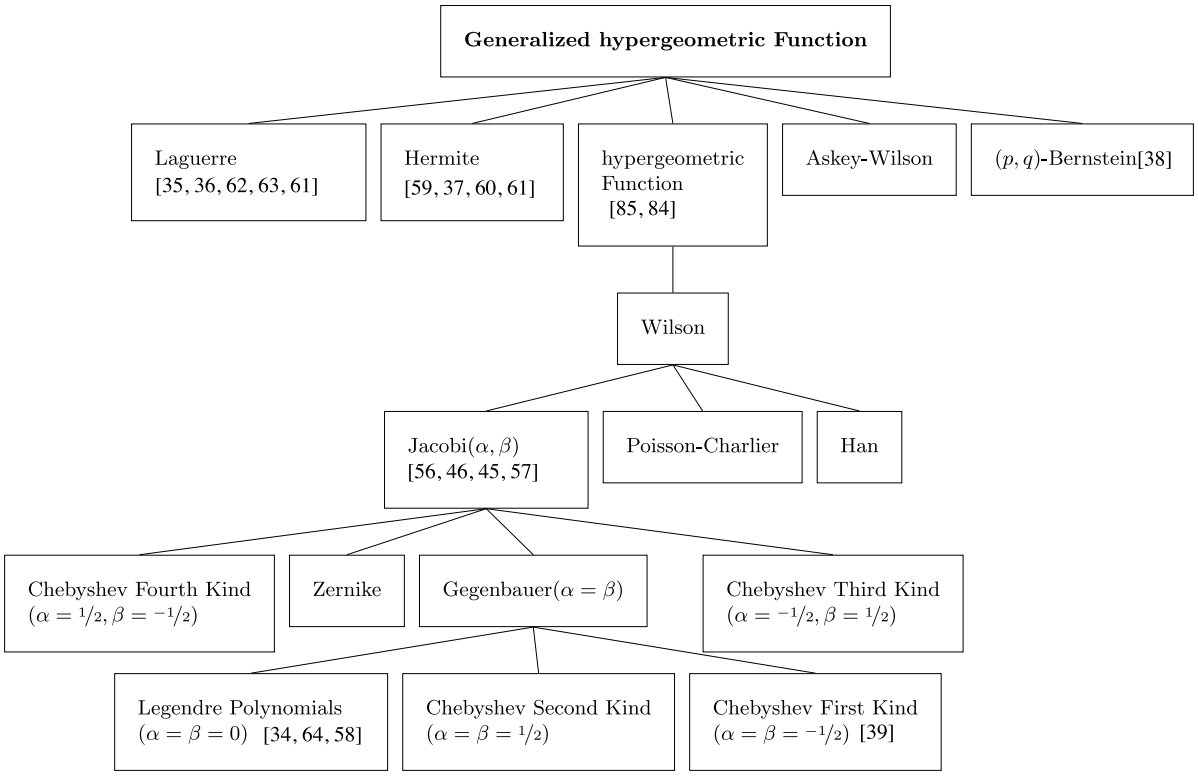


Fig. 1. Hierarchical classification of the generalized hypergeometric function. This figure illustrates that Chebyshev and Gegenbauer polynomials are specific instances within the broader category of Jacobi polynomials, which is itself a subclass of the hypergeometric function (see [34–37,39,45,46,56–64]).

a special structure, such as a Toeplitz matrix, which is commonly used in CNNs [71]. Additionally, a bias term $b^{(i)}$ is added to this result to shift the activation function and better fit the data. In this scenario, a non-linear activation function $\sigma_i(\cdot)$ is applied to the result to overcome the limitations of linear combinations and capture more complex patterns within the data. The output layer of the network, $\mathcal{H}_L$, is computed using a final activation function, which is typically a linear function for regression tasks and a sigmoid or softmax function for classification tasks.

Following the forward pass, the neural network's overall error on the training data is computed. During the backpropagation phase, the gradients of the error with respect to each weight are calculated and used to update the weights, thereby minimizing the error. In this phase, the derivative of the activation function plays a crucial role, as it is used to adjust the network weights in a manner that reduces the overall network error. Therefore, one of the most significant properties of an activation function is its differentiability. According to [72], an effective activation function should possess the following properties:

1. Differentiability: During the optimization process, the ability to compute the activation function's derivative is vital. Differentiability ensures the smooth updating of weights; without it, the process is disrupted.
2. Non-linearity: Activation functions generally fall into two categories — linear and non-linear. Networks with multiple layers of linear or identical activation functions behave similarly to single-layer networks with the same characteristics. Therefore, non-linear activation functions are particularly valuable, as they enable the network to extract complex information from data, enhancing the network's capacity for learning.
3. Finite Range/Boundedness: Since the activation function determines a neuron's final output, limiting the output values to a specific interval enhances network stability.
4. Vanishing and Exploding Gradient Issues: In the backpropagation process, the chain rule is employed to adjust weights. In essence, the gradients in the initial layers are derived from the product of gradients in the final layers. If these generated gradients become exceedingly small or excessively large, the challenges of gradient vanishing or exploding arise, hindering the correct weight updates in the initial layers.
5. Computational Efficiency: The activation function's formulation should be structured to ensure that both the forward and backward processes in the network do not involve intricate or resource-intensive computations.

3.2. Kolmogorov–Arnold Networks

In contrast to MLPs, KANs are based on the Kolmogorov–Arnold representation theorem, which states that a multivariate continuous function can be expressed as a finite composition of continuous functions of a single variable and the binary operation of addition [73,74]. Mathematically:

$$\chi(\boldsymbol{\zeta}) \approx \hat{\chi}(\boldsymbol{\zeta}) = \sum_{q=0}^{2d} \boldsymbol{\Phi}_q \left(\sum_{p=1}^{d} \phi_{q,p}(\boldsymbol{\zeta}_p) \right), \tag{4}$$

for some d with trainable functions $\boldsymbol{\Phi}_q : \mathbb{R} \to \mathbb{R}$, $\phi_{q,p} : [0,1] \to \mathbb{R}$ and known function $\chi(\boldsymbol{\zeta}) : [0,1]^d \to \mathbb{R}$. As proposed by the KAN [4], one may consider B-spline [44,75] curves combined with the SiLU activation function [76] for these functions:

$$\phi(\zeta) = \dot{\theta}\mathrm{SiLU}(\zeta) + \ddot{\theta} \sum_{k=0}^{K} \theta_k \mathcal{B}_k(\zeta). \tag{5}$$

Regardless of the choice for $\phi(\zeta)$, the KAN is formulated as a composition of Kolmogorov–Arnold approximation:

$$\hat{\chi}(\zeta) = \sum_{b_{L-1}=1}^{d_{L-1}} \phi_{L-1,b_L,b_{L-1}} \left(\sum_{b_{L-2}=1}^{d_{L-2}} \cdots \left(\sum_{b_2=1}^{d_2} \phi_{2,b_3,b_2} \left(\sum_{b_1=1}^{d_1} \phi_{1,b_2,b_1} \times \left(\sum_{b_0=1}^{d} \phi_{0,b_1,b_0}(\zeta_{b_0}) \right) \right) \right) \cdots \right), \tag{6}$$

which in matrix form, can be formulated as:

$$\hat{\chi}(\zeta) = \left(\mathbf{\Phi}_{n-1} \circ \cdots \circ \mathbf{\Phi}_1 \circ \mathbf{\Phi}_0 \right)(\zeta), \tag{7}$$

where $\mathbf{\Phi}_{p,q} = \phi_{p,q}(\cdot)$. The convergence of this approximation can be seen as:

$$\max_{|\alpha| \le m} \sup_{\zeta \in [0,1]^d} \| \mathfrak{D}^{\alpha} \{ \chi(\zeta) - \hat{\chi}(\zeta) \} \| < \epsilon, \tag{8}$$

where $\mathfrak{D}$ is the derivative operator, and ϵ is a constant based on the behavior of $\chi(\cdot)$ and trainable functions $\phi_{p,q}(\cdot)$. The complete proof of the convergence of this approximation can be found in [4,73,77,78].

As a consequence of this theorem and its utility, various extensions have been proposed to facilitate faster implementations. For example, Lorentz [79] proved that the functions $\mathbf{\Phi}_q(\cdot)$ can be reduced to a single function $\mathbf{\Phi}(\cdot)$:

$$\hat{\chi}(\zeta) = \sum_{q=0}^{2d} \mathbf{\Phi} \left(\sum_{p=1}^{d} \phi_{q,p}(\zeta_p) \right). \tag{9}$$

In another work, Sprecher [80] showed that the functions $\phi_{p,q}$ can be replaced by a single function $\phi(\cdot)$:

$$\hat{\chi}(\zeta) = \sum_{q=0}^{2d} \mathbf{\Phi} \left(q + \sum_{p=1}^{d} \theta_p \phi(\zeta_p + \nu q) \right). \tag{10}$$

Both of these variants aimed to provide an easier approach for implementation. However, they have not yet addressed the issues of the Kolmogorov–Arnold representation, which include the non-smoothness of the inner functions and the time complexity of KANs [81].

Recently, some research has proposed replacements for B-splines and alternative formulations of KANs to potentially achieve more accurate solutions with lower time complexity. For example, Fourier KANs [19], Wavelet KANs [18], and radial basis function (RBF) KANs [82]. Another approach is Chebyshev KANs, which utilize orthogonal polynomials as the univariate functions in KANs [39,83]. This architecture is defined as:

$$\hat{\chi}(\zeta) = \sum_{q=1}^{n} \sum_{p=1}^{d} \theta_{p,q} T_q(\tanh(\zeta_p)). \tag{11}$$

Here, n, a hyperparameter, represents the degree of the Chebyshev polynomials, $T_q(\cdot)$ denotes the Chebyshev polynomial of degree q, and $\theta_{p,q}$ are the trainable weights. This approach is a counterpart of KANs, named Learnable Activation Networks (LANs).

Although the Chebyshev KAN demonstrated acceptable accuracy while maintaining low computational complexity [39,83], it still inherits the limitations associated with polynomials as discussed earlier. In the following section, we introduce fractional Jacobi functions as a potential replacement for Chebyshev polynomials in KANs.

3.3. Jacobi polynomials

The Jacobi polynomials $\mathcal{J}_n^{(\alpha,\beta)}(\zeta)$ are a special case of hypergeometric functions. The generalized form of the hypergeometric function ${}_p\mathcal{F}_q : \mathbb{C}^{p+q+1} \to \mathbb{C}$ can be represented as:

$${}_p\mathcal{F}_q \left(\begin{matrix} a_1, a_2, \dots, a_p \\ b_1, b_2, \dots, b_q \end{matrix}; \zeta \right) = \sum_{n=0}^{\infty} \frac{(a_1)_n (a_2)_n \dots (a_p)_n}{(b_1)_n (b_2)_n \dots (b_q)_n} \frac{\zeta^n}{n!}, \tag{12}$$

where $(a)_n$ is the rising sequential product, known as the Pochhammer symbol, defined by:

$$(a_i)_n = \frac{\Gamma(a_i + n)}{\Gamma(a_i)} = a_i(a_i + 1)(a_i + 2) \dots (a_i + n - 1), \quad n > 0, \tag{13}$$

in which $\Gamma(\xi) = \int_0^{\infty} s^{\xi - 1} \exp(-s) \mathrm{d}s$ is the Gamma function [30]. In the case of $n = 1$, this empty product is defined as one [84]. Vieira [85] proved that the infinite hypergeometric series converges if any of the following conditions are met:

(a) $p \le q$,

(b) $q = p - 1 \wedge |\zeta| < 1$,

(c) $q = p - 1 \wedge \mathscr{R} \left(\sum_{i=1}^{p-1} (b_i - p) - \sum_{j=1}^{q} a_j \right) > 0 \wedge |\zeta| = 1$.

For $p = 2$ and $q = 1$, this analytic function simplifies to the Gaussian hypergeometric function:

$${}_2\mathcal{F}_1 \left(\begin{matrix} a_1, a_2 \\ b_1 \end{matrix}; \zeta \right) = \sum_{n=0}^{\infty} \frac{(a_1)_n (a_2)_n}{(b_1)_n} \frac{\zeta^n}{n!}. \tag{14}$$

The analytical function ${}_p\mathcal{F}_q(\cdot)$ with any non-positive values of a_i reduces to a polynomial of degree $-a_i$ in the argument ζ. For example, Askey–Wilson polynomials, Wilson polynomials, continuous Hahn polynomials, and Jacobi polynomials are some such cases. For the specific choice $p = 2, q = 1, a_1 = -n, a_2 = n + 1 + \alpha + \beta$, and $b_1 = 1 + \alpha$ where $\alpha, \beta > -1$, the Gaussian hypergeometric series reduces to the Jacobi polynomials:

$$\mathcal{J}_n^{(\alpha,\beta)}(\zeta) = \frac{(\alpha + 1)_n}{n!} {}_2\mathcal{F}_1 \left(\begin{matrix} -n, n + 1 + \alpha + \beta \\ 1 + \alpha \end{matrix}; \tfrac{1}{2}(1 - \zeta) \right). \tag{15}$$

Jacobi polynomials appear in various scientific fields such as signal processing, numerical analysis, machine learning, economics, and quantum mechanics [47]. Hence, they can be defined through several definitions. For example, they are the eigenfunctions of $\mathcal{L}(\mathcal{J}_n^{(\alpha,\beta)}) = \lambda_n \mathcal{J}_n^{(\alpha,\beta)}$ for the linear second-order Sturm–Liouville differential operator:

$$\begin{aligned} \mathcal{L}(\mathcal{J}_n^{(\alpha,\beta)}) &:= -(1 - \zeta)^{-\alpha}(1 + \zeta)^{-\beta} \frac{\mathrm{d}}{\mathrm{d}\zeta} \left((1 - \zeta)^{\alpha+1}(1 + \zeta)^{\beta+1} \frac{\mathrm{d}}{\mathrm{d}\zeta} \mathcal{J}_n^{(\alpha,\beta)}(\zeta) \right) \\ &= (\zeta^2 - 1) \frac{\mathrm{d}^2}{\mathrm{d}\zeta^2} \mathcal{J}_n^{(\alpha,\beta)}(\zeta) + (\alpha - \beta + (\alpha + \beta + 2)\zeta) \frac{\mathrm{d}}{\mathrm{d}\zeta} \mathcal{J}_n^{(\alpha,\beta)}(\zeta), \end{aligned} \tag{16}$$

with the eigenvalues $\lambda_n = n(n + 1 + \alpha + \beta)$. Alternatively, the following three-term recurrence formula can generate the Jacobi polynomials:

$$\begin{aligned} &\mathcal{J}_0^{(\alpha,\beta)}(\zeta) = 1, \\ &\mathcal{J}_1^{(\alpha,\beta)}(\zeta) = A_0 \zeta + B_0, \\ &\mathcal{J}_{n+1}^{(\alpha,\beta)}(\zeta) = (A_n \zeta + B_n) \mathcal{J}_n^{(\alpha,\beta)}(\zeta) - C_n \mathcal{J}_{n-1}^{(\alpha,\beta)}(\zeta), \quad n \ge 1, \end{aligned} \tag{17}$$

where

$$\begin{aligned} A_n &= \frac{(2n + \alpha + \beta + 1)(2n + \alpha + \beta + 2)}{2(n + 1)(n + \alpha + \beta + 1)}, \\ B_n &= \frac{(\alpha^2 - \beta^2)(2n + \alpha + \beta + 1)}{2(n + 1)(n + \alpha + \beta + 1)(2n + \alpha + \beta)}, \\ C_n &= \frac{(n + \alpha)(n + \beta)(2n + \alpha + \beta + 2)}{(n + 1)(n + \alpha + \beta + 1)(2n + \alpha + \beta)}. \end{aligned} \tag{18}$$

Choosing different values for α and β results in specific well-known orthogonal polynomials that appear in different engineering applications. Fig. 1 depicts some of the most used functions in this hierarchy. In the following, we focus on the Jacobi polynomials and their properties which are used in the following sections.

Theorem 1. *The Jacobi polynomials with $\alpha, \beta > -1$ are a set of orthogonal functions on the interval $[-1, 1]$:*

$$\langle \mathcal{J}_m^{(\alpha,\beta)}, \mathcal{J}_n^{(\alpha,\beta)} \rangle = \int_{-1}^{1} \mathcal{J}_m^{(\alpha,\beta)}(\zeta) \mathcal{J}_n^{(\alpha,\beta)}(\zeta)(1 - \zeta)^{\alpha}(1 + \zeta)^{\beta} \mathrm{d}\zeta = \| \mathcal{J}_n^{(\alpha,\beta)} \|_2 \delta_{m,n}, \tag{19}$$

where $\langle f, g \rangle = \int_{\Omega} f(\xi) g(\xi) w(\xi) \mathrm{d}\xi$ is the inner product operator over the domain $\Omega \subseteq \mathbb{R}$.

Proof. The proof is a direct consequence of Sturm–Liouville theory [30, 47].

Theorem 2. *The Jacobi polynomials have the following symmetry on the interval* $[-1, 1]$ *[30]:*

$$\mathcal{J}_n^{(\alpha,\beta)}(-\zeta) = (-1)^n \mathcal{J}_n^{(\beta,\alpha)}(\zeta). \tag{20}$$

Proof. The proof can be established using the Rodrigues' definition of Jacobi polynomials:

$$\mathcal{J}_n^{(\alpha,\beta)}(\zeta) = \frac{(-1)^n}{2^n n!}(1-\zeta)^{-\alpha}(1+\zeta)^{-\beta}\frac{\mathrm{d}^n}{\mathrm{d}\zeta^n}\left[(1-\zeta)^{\alpha}(1+\zeta)^{\beta}\left(1-\zeta^2\right)^n\right]. \tag{21}$$

Corollary 1. *The Jacobi polynomials at the boundary points can be approximated by [30]:*

$$\begin{aligned} &\mathcal{J}_n^{(\alpha,\beta)}(-1) = \frac{(-1)^n}{n!}\frac{\Gamma(n+1+\beta)}{\Gamma(1+\beta)}, \\ &|\mathcal{J}_n^{(\alpha,\beta)}(-1)| \approx n^{\beta} \qquad n \gg 1, \\ &\mathcal{J}_n^{(\alpha,\beta)}(+1) \approx n^{\alpha} \qquad n \gg 1. \end{aligned} \tag{22}$$

Theorem 3. *The derivatives of the Jacobi polynomials can be expressed in terms of themselves. Specifically:*

$$\frac{\mathrm{d}^m}{\mathrm{d}\zeta^m}\mathcal{J}_n^{(\alpha,\beta)}(\zeta) = \frac{\Gamma(n+m+1+\alpha+\beta)}{2^m\Gamma(m+1+\alpha+\beta)}P_{n-m}^{(\alpha+m,\beta+m)}(\zeta). \tag{23}$$

Proof. This property is a direct consequence of the orthogonality and the derivatives of the hypergeometric function:

$$\frac{\mathrm{d}^m}{\mathrm{d}\zeta^m}\,{}_2\mathcal{F}_1\left(\begin{matrix}a_1,a_2\\b_1\end{matrix};\zeta\right) = \frac{(a_1)_m(a_2)_m}{(b_1)_m}\,{}_2\mathcal{F}_1\left(\begin{matrix}a_1+m,a_2+m\\b_1+m\end{matrix};\zeta\right). \tag{24}$$

An alternative approach to the proof can be observed by directly employing the properties of Jacobi polynomials. By utilizing the tree term recurrence relation (17) (see Theorem 3.1 in [30]), we can derive the following result:

$$\frac{\mathrm{d}}{\mathrm{d}\zeta}\mathcal{J}_n^{(\alpha,\beta)}(\zeta) = \frac{(\alpha+\beta+n+1)}{2}\mathcal{J}_{n-1}^{(\alpha+1,\beta+1)}(\zeta). \tag{25}$$

By recursively applying this formula, we obtain:

$$\frac{\mathrm{d}^m}{\mathrm{d}\zeta^m}\mathcal{J}_n^{(\alpha,\beta)}(\zeta) = \mathrm{u}_{m,n}^{\alpha,\beta}P_{n-m}^{(\alpha+m,\beta+m)}(\zeta), \tag{26}$$

where

$$\mathrm{u}_{m,n}^{\alpha,\beta} = \frac{\Gamma(n+m+1+\alpha+\beta)}{2^m\Gamma(m+1+\alpha+\beta)}. \tag{27}$$

Theorem 4. *Jacobi polynomial of the degree* n *has* n *real distinct roots.*

Proof. The proof of this theorem can be done employing theorem 3.4 in [30] which uses the recurrence definition (Eq. (17)) of Jacobi polynomials. For more details, we refer the reader to [86,87].

Theorem 5. *The linear bijective mapping*

$$\varphi_{(d_0,d_1)}(\zeta) = \frac{2\zeta - d_0 - d_1}{d_1 - d_0}, \tag{28}$$

transforms the properties of the Jacobi polynomials from the interval $[-1, 1]$ *into the desired domain* $[d_0, d_1]$*. Moreover, for extending the polynomial space into fractional-order functions, one may use:*

$$\varphi_{(d_0,d_1,\gamma)}(\zeta) = \frac{2\zeta^{\gamma} - d_0 - d_1}{d_1 - d_0}. \tag{29}$$

Employing these mappings, the shifted fractional Jacobi functions can be generated using:

$$P_n^{(\alpha,\beta)}(\zeta) = \mathcal{J}_n^{(\alpha,\beta)}(\varphi_{(d_0,d_1,\gamma)}(\zeta)). \tag{30}$$

Proof. The proof of orthogonality can be readily established by performing a change of variables in the integration. Additionally, the derivative formula can be derived using the chain rule.

Examining the characteristics of Jacobi orthogonal polynomials as a potential activation function reveals their non-linearity and differentiability. Their derivatives can be easily computed using a formula derived from the function itself, simplifying calculations during backpropagation and reducing computational burden. Additionally, Jacobi polynomials inherently confine their output within a predetermined range, promoting network stability. Consequently, they offer promising features for activation functions in neural networks. With these insights, the following section introduces an architecture for incorporating Jacobi functions into neural network frameworks.

4. Fractional KAN

In this section, we present a novel block structure enabling the utilization of fractional Jacobi functions as activation functions within a neural network. This framework, empowers KANs or even deep learning architectures to determine optimal values for Jacobi polynomial parameters, namely α, β, and the fractional order γ. Following this, we will explore the intricacies of the Jacobi neural block integrated into the design of the neural network models employed in our experiments.

It is crucial to acknowledge that Jacobi polynomials, as per Theorem 1, require their input values to reside within a specific interval to yield meaningful output. However, the output produced by a neural network layer does not inherently adhere to this constraint, making it impractical to directly employ Jacobi polynomials as activation functions immediately after a fully connected layer. To tackle this challenge, two different approaches have been proposed: (1) Utilizing batch normalization and (2) Applying a bounded activation function [39].

The first option introduces certain complications, including increased time complexity [88], incompatibility with small batch sizes or online learning [89], and issues with saturating non-linearities [90,91]. On the other hand, the second approach offers lower computational complexity compared to batch normalization. It facilitates optimal backward computations, boasts easy implementation, and ensures consistent behavior in both the training and testing phases. In contrast, the first option exhibits varying behaviors between these phases. Therefore, in this paper, we opt for the second approach to address this case.

The choice of an activation function to confine the output of a layer within a bounded interval ($[d_0, d_1]$) can include options like the sigmoid function, hyperbolic tangent, Gaussian function, or any bounded-range activation function. While the selection of this function is arbitrary, in this paper, we opt for the sigmoid function ($\sigma(\zeta) = \frac{1}{1+\exp(-\zeta)}$) due to its range, which spans $(0, 1)$. This property of positivity enables the computation of fractional powers without encountering complex-valued numbers in fKAN.

To propose the fractional KAN, we first revisit the original KAN:

$$\hat{\chi}(\boldsymbol{\zeta}) = \sum_{q=0}^{2d}\boldsymbol{\Phi}_q\left(\sum_{p=1}^{d}\phi_{q,p}(\boldsymbol{\zeta}_p)\right). \tag{31}$$

In general, the fractional KAN can be obtained using a fractional univariate function $\phi_{p,q}(\cdot)$ such as fractional Jacobi functions, fractional Bernoulli functions, or even fractional B-splines [92]. This extension can be formulated as:

$$\hat{\chi}(\boldsymbol{\zeta}) = \sum_{q=0}^{2d}\boldsymbol{\Phi}_q^{(\gamma)}\left(\sum_{p=1}^{d}\phi_{q,p}^{(\gamma)}(\boldsymbol{\zeta}_p)\right), \tag{32}$$

where γ is the fractional order. In this paper, we choose the fractional Jacobi functions due to their flexibility, ease of calculation, and adaptability. Mathematically, for predefined values of α, β, and γ, a basis function can be formulated as:

$$\phi_{p,q}^{(\gamma)}(\boldsymbol{\zeta}_p) = \mathcal{J}_q^{(\alpha,\beta)}\left(\varphi_{(0,1;\gamma)}(\sigma(\boldsymbol{\zeta}_p))\right). \tag{33}$$

In order to allow this mathematical function to be tuned for given data during the optimization process of the networks, we can define these predefined parameters to be learned during training. However,

this requires some modifications. For example, to ensure that the Jacobi polynomial converges, its parameters (α, β) should be greater than -1, while a network weight can take any value. To address this, we use the well-known ELU activation function [93] with the ELU : $\mathbb{R} \to (-\kappa, \infty)$ property:

$$\mathrm{ELU}(\zeta; \kappa) = \begin{cases} \zeta & \text{if } \zeta > 0, \\ \kappa \times (e^{\zeta} - 1) & \text{if } \zeta \leq 0, \end{cases} \tag{34}$$

where κ is a parameter that controls the lower bound of the range of the ELU function. As a result of this definition, for parameters α and β, one can easily set κ to 1 to obtain meaningful Jacobi functions. However, the original definition of fractional Jacobi functions (17) is not normalized to attain values of ± 1 at the boundary points, which may lead to excessively high values (specifically n^{α} and n^{β} as defined in (22)). Constraining the values of α and β to lower limits may be beneficial in mitigating the issue of gradient explosion. In this context, one might consider employing a scaled hyperbolic tangent function, denoted as $\varphi_{-1,d_1,1}(\tanh(\alpha))$ and $\varphi_{-1,d_1',1}(\tanh(\beta))$, where d_1 and d_1' are problem-specific tuning parameters.

Furthermore, the fractional power γ needs to be positive to ensure the well-definedness of the Jacobi polynomials. For this purpose, one might consider using functions like ReLU$(\cdot)$ or ELU$(\cdot, 0)$. However, these functions allow the parameter to approach infinity, potentially causing instability issues such as Runge's phenomenon [30]. Instead, we propose using the sigmoid function again to constrain the values of γ between zero and one.

This approach, combined with polynomial degrees, enables the network to find an accurate solution for the given data. By fixing the basis function degree to n (denoted as $\mathcal{J}_n^{(\alpha,\beta)}(\cdot)$), the network can utilize a fractional degree between zero and n, exhibiting a "liquid" behavior that permits the network to explore the fractional space for approximating the desired function. Putting all of these together, the new basis function for fKAN can be expressed as:

$$\phi_{p,q}^{(\gamma)}(\boldsymbol{\zeta}_p) = \mathcal{J}_q^{(\mathrm{ELU}(\alpha,1),\mathrm{ELU}(\beta,1))}\left(\varphi_{(0,1;\sigma(\gamma))}(\sigma(\boldsymbol{\zeta}_p))\right), \tag{35}$$

with trainable parameters α, β, and γ. We call this adaptive activation the fractional Jacobi neural block, abbreviated as fJNB. A visualization of the fJNB can be seen in Fig. 2.

The next step in defining fKAN is to establish the summation bounds and the function $\boldsymbol{\Phi}_q(\cdot)$. In the general definition of KANs, the basis functions $\phi_{p,q}(\cdot)$ are local functions (e.g., obtained through B-splines or compact support RBFs), meaning they are non-zero only on a subset of the real line. To approximate a desired function, these local functions must be combined with other local functions in that domain. However, in the case of polynomials, especially orthogonal Jacobi polynomials, the function is non-zero except at some root points (see Theorem 4). This allows us to omit the outer summation and focus solely on the inner one. For the function $\boldsymbol{\Phi}_q(\cdot)$, now denoted as $\boldsymbol{\Phi}(\cdot)$, we can use a simple linear function with trainable weights θ that are adjusted during the network's backward phase. This approach is visualized in Fig. 3(a).

By closely examining the proposed fKAN, one can see that this approach focuses on a specific basis function (e.g., $q = 2$), which may not be suitable for a wide variety of applications. For example, in numerical analysis, the Taylor series guarantees that a function can be approximated by a linear combination of monomials. As the number of terms in this series tends to infinity, the approximation becomes increasingly accurate. Formally, for some known weights θ_n:

$$\chi(\zeta) = \sum_{n=0}^{\infty} \theta_n \zeta^n. \tag{36}$$

To simulate a similar approach in fKANs, we can use the following formulation:

$$\hat{\chi}(\boldsymbol{\zeta}) = \sum_{q=0}^{Q} \boldsymbol{\Phi}_q^{(\gamma)} \left(\sum_{p=1}^{d} \phi_{q,p}^{(\gamma)}(\boldsymbol{\zeta}_p) \right). \tag{37}$$

In this formulation, Q represents the maximum degree of the fKAN and $\phi_{q,p}^{(\gamma)}(\boldsymbol{\zeta}_p)$ is defined as in Eq. (35). Notably, the outer summation is independent of the input data. Additionally, the function $\boldsymbol{\Phi}_q(\cdot)$ can be defined to fuse the outputs of each fJNB using either a concatenation layer or an attention mechanism. This scheme is illustrated in Fig. 3(b).

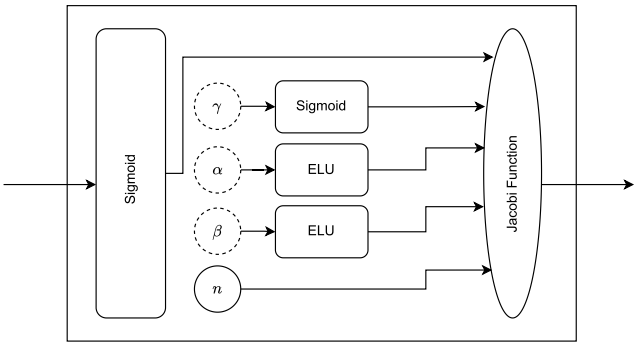


Fig. 2. The architecture of the fractional Jacobi neural block proposed in formula (35), featuring trainable parameters α, β, and γ. This block serves as an adaptive activation function, enabling the network to learn optimal values for these parameters during training. The fJNB offers flexibility, ease of calculation, and adaptability, making it suitable for various applications in neural network architectures.

5. Experiments

To evaluate the effectiveness of the proposed fKAN, a series of experiments were designed. In the following sections, we provide detailed descriptions of each experiment, covering a range of tasks including deep learning and physics-informed deep learning. The implementation of fKAN and the subsequent experiments are publicly available in the GitHub repository.[1] Additionally, we have created a Python package named `fkan` to simplify using our proposed method.

All experiments were implemented in Python using the latest version of Keras with TensorFlow as the backend for deep learning tasks and PyTorch for physics-informed tasks. The experiments were conducted on a personal computer equipped with an Intel Core-i3 10100 CPU, 16 GB of RAM, and an Nvidia 1650 GPU.

5.1. Deep learning tasks

In this section, we will conduct several common deep learning tasks to validate the proposed neural block's effectiveness in handling real-world applications. Specifically, we will assess the accuracy of the proposed fKAN in synthetic regression tasks. Next, we will examine a classification problem in CNNs using the proposed activation function, followed by the application of pretrained deep learning architectures for transfer learning. Subsequently, we will employ an autoencoder CNN network for the task of image denoising. Finally, a one-dimensional CNN (1D-CNN) will be employed for a sentiment analysis task.

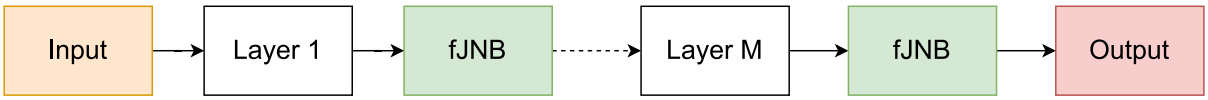
Regression task

For the first experiment, we consider the following function as the ground truth model:

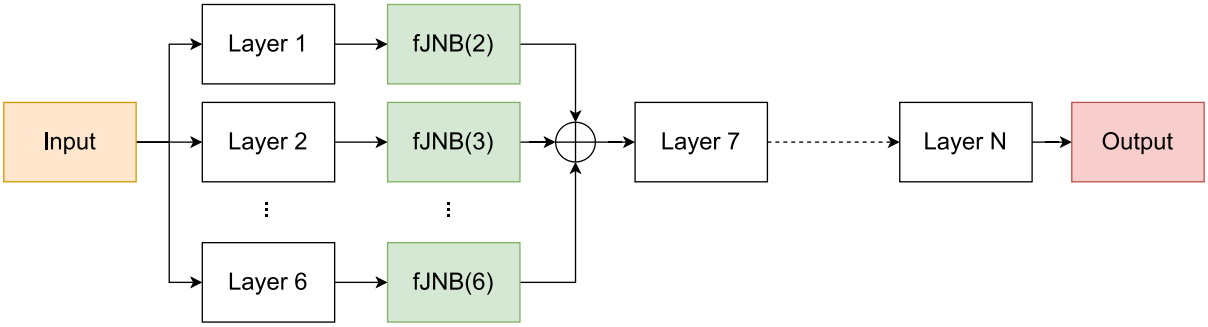
$$\chi(\zeta) := \sin(\pi\zeta) + 10\exp\left(\frac{\zeta}{5}\right) + \frac{\epsilon}{100}, \tag{38}$$

where ϵ is standard normal white noise. We sampled 50 equidistant points in the domain $[-2, 1]$ and split them into train and test sets in a hold-out manner with a test size of 33%. To simulate this problem, we designed a single-layer neural network with varying numbers of neurons. The loss function was set to mean squared error, the optimizer to Adam with a learning rate of 0.01, the maximum iterations to

[1] https://github.com/alirezaafzalaghaei/fKAN.



(a) Utilizing fJNB in a sequential architecture, where the value of q may vary for each block. This flexibility allows for the incorporation of different fractional Jacobi basis functions with varying degrees of complexity across successive blocks in the network. By adapting the value of q, the network can tailor its representation capabilities to better capture the intricacies of the underlying data.



(b) Employing fJNB with varying values of q concurrently, this deep network comprises six subnetworks, each equipped with distinct fJNBs. By incorporating fJNBs with different q values, the network can capture a diverse range of features simultaneously. The outputs of these fJNBs are then fused to produce the final prediction, leveraging the collective information gleaned from multiple representations.

Fig. 3. Two distinct structures of fKAN employing one or more fJNBs simultaneously.

500, and early stopping with a patience of 200. Fig. 4(a) shows a comparison between the accuracy of the predictions of this network with different activation functions. As can be seen, the proposed fKAN is more accurate than the well-known activation functions as well as a KAN.

For the next validation task, we compare the accuracy of the network for a multi-layer neural network with a fixed number of five neurons in each layer. The hyperparameters of the network are set the same as in the first experiment. The results are shown in Fig. 4(b) and Table 1. Again, it can be seen that the proposed network performs better than the well-known activation functions and alternatives, although there is a higher variance compared to other activation functions. As in the previous example, the KAN has the lowest variance, which can be attributed to its use of a quasi-Newton optimizer in each trial.

To examine the evolution of the Jacobi polynomial parameters (α, β, and γ), we selected a single-layer network with a fixed 5 neurons in the hidden layer, increased the number of samples to 250, and removed the early stopping procedure. Fig. 4(c) depicts the values of these parameters during the training phase. It can be seen that these trainable parameters converge to optimal values in about 250 iterations.

The evaluation of the time complexity of fKAN is conducted through various aspects. In the simplest approach, we compare the CPU and GPU time required to compute each activation function for a fixed input dataset. To achieve this, we generated a random matrix of shape $(1000, 1000)$ and evaluated different activation functions. The results are presented in Fig. 4(d). It can be observed that the proposed activation function in the CPU implementation is approximately two times slower than the sigmoid function and three times slower than the fastest activation function, ReLU.

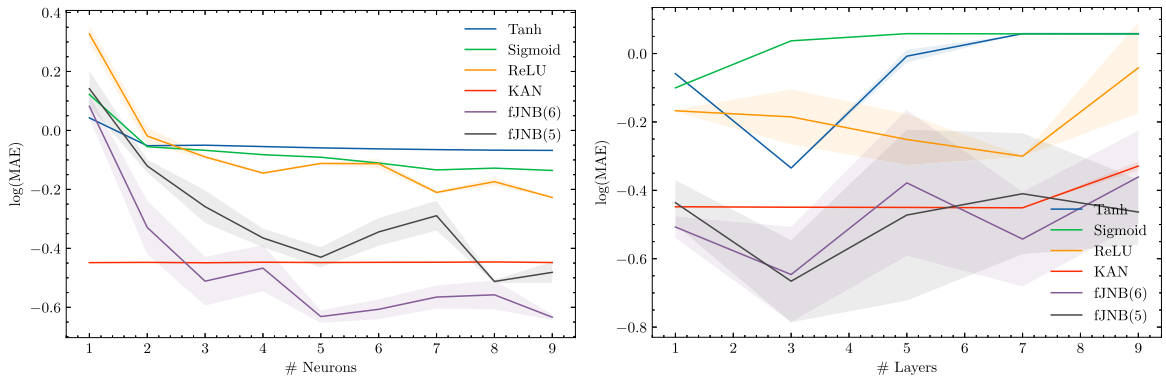
To analyze the time complexity of fJNB across different input sizes, we executed it on matrices of size $(10^i, 10^{i-1})$ for $i = 1, \ldots, 5$ on the GPU and compared it with the ReLU, hyperbolic tangent, and sigmoid functions. This comparison is illustrated in Fig. 5(a). It is evident that fJNB is about ten times slower than the sigmoid function. Furthermore, when incorporating this function as the activation function in a multi-layer network, we plotted the CPU time per epoch for training the network, which consists of five neurons per layer. This time complexity plot is shown in Fig. 5(b). It can be seen that the proposed activation function exhibits a slightly steeper growth slope compared to ReLU.

Although this computational time is higher than that of classic fixed activation functions, the proposed method is more efficient than KAN architectures in terms of both accuracy and time. To demonstrate this, we utilized a simple linear regression dataset generated by the `make_regression` function from the scikit-learn library, consisting of 1000 samples, 2 features, and a Gaussian noise level set to one. An 80-20 train-test split was applied to validate the results. A straightforward architecture of $[2, 4, 1]$ was employed to fit the network. The results of this simulation are reported in Table 2. This table includes the original KAN with splines for $k = 2, 3, 4, 5, 6$, wavelet KAN with wavelet types Mexican hat, Shannon, Meyer, dog, and Morlet, rational KAN for k from 2 to 6, Fast KAN with RBFs, MLP-KAN, Chebyshev KAN with degrees ranging from 2 to 6, and fKAN with fJNB for degrees 2 to 6, thereby supporting our claim regarding the time efficiency of orthogonal functions compared to alternatives like splines and wavelets.

Furthermore, it can be observed that the Chebyshev KANs are faster than other KANs due to their simpler definition (they appear at the bottom of the hierarchy in Fig. 1). However, in terms of numerical stability, the Chebyshev KANs are less accurate than others, while the fKAN, with a CPU time comparable to that of the Chebyshev KAN (the second best), demonstrates the lowest error rate. Regarding trainable parameters, the fJNB has the fewest parameters, whereas the MLP-KAN method has the largest number, with approximately 175 times more parameters.

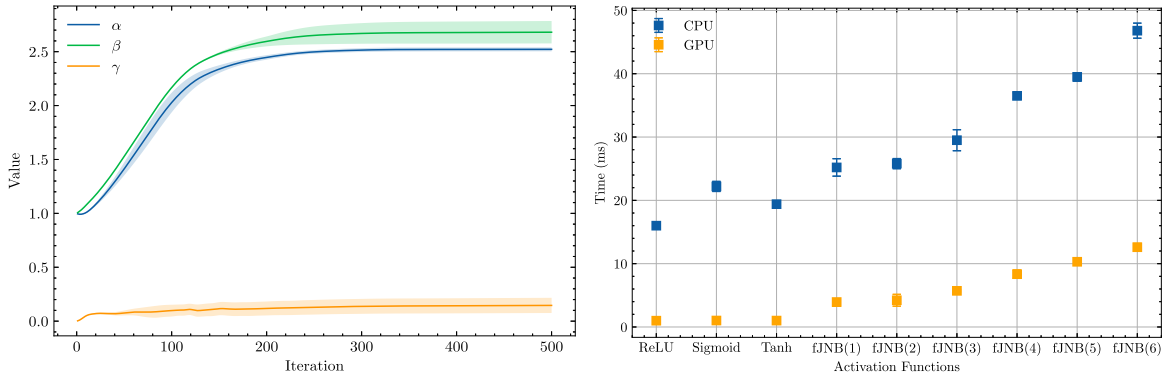
Image classification

To assess the effectiveness of using generalized Jacobi polynomials as the activation function within the fKAN framework, we utilize a selection of well-established benchmark datasets. These include the MNIST handwritten digit classification dataset [96], which consists of a training set with 60,000 examples and a test set of 10,000 examples, featuring 28×28 grayscale images representing digits from zero to nine, totaling 10 classes. Additionally, we incorporate the CIFAR-10 and CIFAR-100 datasets [97], which contain 50,000 training and 10,000 test color images of size 32×32, representing 10 and 100 classes, respectively. Furthermore, we utilize the cropped Street View



(a) Comparison of prediction accuracy for a single-layer neural network with different activation functions. The proposed activation function demonstrates higher accuracy than well-known functions and KAN.

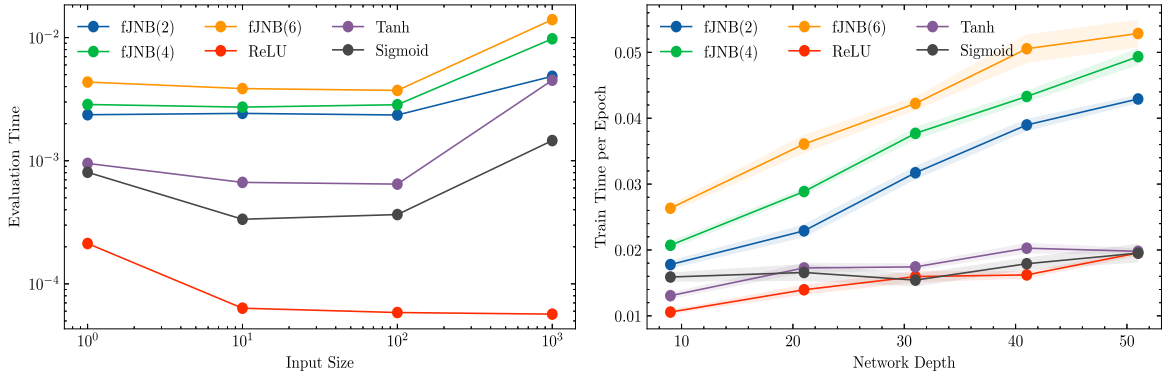
(b) Accuracy comparison for an MLP with a fixed number of neurons per layer. The proposed network performs better than traditional activation functions and alternatives, though with higher variance.



(c) Evolution of Jacobi polynomial parameters (α, β, and γ) during training. The parameters converge to optimal values after approximately 250 iterations.

(d) Comparison of CPU and GPU time required to compute various activation functions. The times are the mean of 100 different runs.

Fig. 4. The results of simulation of a one-dimensional function regression task for different activation functions. The proposed activation function demonstrates higher accuracy than well-known functions as well as KAN.



(a) GPU evaluation time for different activation functions across various input sizes.

(b) Time required to train one epoch of a network with a given depth.

Fig. 5. Time complexity analysis of the fJNB activation function compared to other alternatives.

House Numbers (SVHN) dataset [98], which includes 73,257 training and 26,032 test color images, also of size 32 × 32, representing 10 classes. Lastly, the mini ImageNet dataset [99] is employed, containing 60,000 color images of dimensions 84 × 84 across 100 classes, following a hold-out train-test split with a test size of 15,000. In all cases, 10% of the training data is reserved for early stopping and validation purposes.

We employ a straightforward CNN architecture, as illustrated in Fig. 6 (using MNIST as an example), and train it using the cross-entropy loss function with a batch size of 512. An exception is made for the mini ImageNet dataset, where we incorporate batch normalization in the first layer and apply global average pooling prior to the classifier

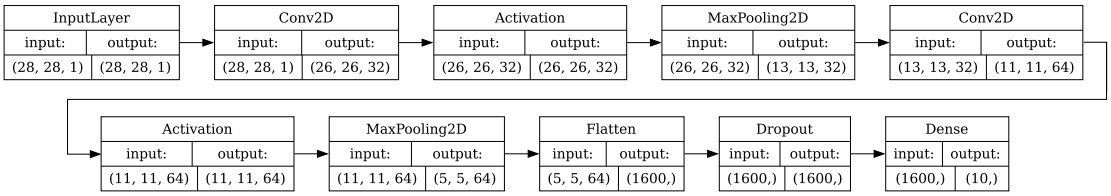


Fig. 6. The CNN architecture of the proposed method for MNIST classification. It is important to note that for other datasets, the input and output shapes of the layers will vary to accommodate the specific characteristics of each dataset.

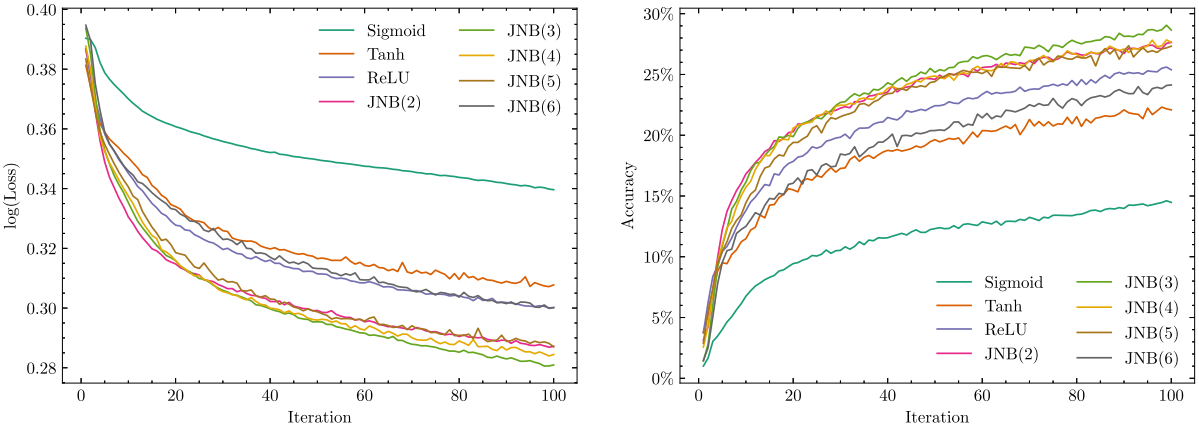


Fig. 7. The loss function and accuracy of the proposed method in comparison to well-known activation functions for classifying the mini ImageNet dataset.

Table 1
Comparison of mean absolute error for multi-layer neural networks with different activation functions for the regression task.

Act. Func.	No. of hidden layers				
	1	3	5	7	9
SOFTPLUS	0.4390	0.4630	0.4850	0.468	1.7600
SILU	0.6740	0.5190	0.5100	0.337	1.0400
SELU	0.7410	0.9240	0.2290	**0.171**	0.1270
Leaky ReLU	0.3490	0.2860	0.7060	0.152	0.3540
GELU	0.7130	0.6250	0.7190	0.283	0.1240
ELU	0.6630	0.7520	0.4330	0.786	0.6100
Tanh	0.7940	0.2640	1.2600	1.260	1.2600
Sigmoid	0.7930	1.2400	1.2600	1.260	1.2400
ReLU	0.4290	0.7300	0.3430	0.353	1.7200
fJNB(6)	**0.0861**	**0.0186**	0.0459	0.177	0.2950
fJNB(5)	0.6140	0.6990	**0.0168**	0.203	**0.0218**

block of the network to reduce the number of parameters.

In the architecture depicted in Fig. 6, we varied the activation function to include sigmoid, hyperbolic tangent, ReLU, or fJNB. For each activation function, the network was trained using the Adam optimizer with Keras's default learning rate over 30 epochs, except for the mini ImageNet dataset, where the training duration was extended to 100 epochs. The accuracy comparisons of these methods are presented in Table 3. Additionally, Fig. 7 illustrates the training process of the network with different activation functions. The results indicate that the fractional Jacobi neural block outperforms the alternatives, even within a convolutional deep learning framework.

Transfer learning

In this section, we explore the application of transfer learning to classify the Oxford Flowers dataset [100]. The dataset comprises 1020 training samples and 6149 images resized to $224 \times 224 \times 3$ across 102 different classes. To classify this data, we employ two different architectures that are pre-trained on the ImageNet dataset: the minimalistic MobileNetV3 Small [101] and ResNet-18 [102]. For the training, we freeze the weights of the feature extraction component and construct a classifier using an MLP network. For MobileNet, we considered various network depths ranging from 1 to 10, each with 240 neurons per layer. The activation function for the layers is set to either fJNB or ReLU, which is known for deep architectures. The model is trained with a batch size of 64 for only 10 epochs with Adam optimizer. Table 4 presents the macro F1 score of the model predictions on the test data. As shown, fJNB(2) achieves the best results in all cases. When the depth is less than three, ReLU demonstrates the second-best performance; however, increasing the depth leads to a decline in ReLU's performance, while fJNB(3) becomes the second-best model.

For the ResNet network, we adopt a similar approach to the previous one, with the only modification being the setting of the number of hidden MLP layers to one. The classification report for this approach is provided in Table 5. It can be observed that in this model, fJNB with degrees of two or three consistently yields the best and second-best results across all metrics.

Image denoising

In the next experiment, we develop a CNN specifically designed for image denoising tasks, utilizing the CIFAR-10 and Fashion MNIST datasets as two well-known benchmark problems. The Fashion MNIST dataset serves as a more complex alternative to the original MNIST data, comprising 60,000 samples of 28×28 grayscale images categorized into 10 distinct fashion classes. As with the previous task, we scale the input data to a range between zero and one before feeding it into the deep model.

The architecture of our CNN for this task is illustrated in Fig. 8. We begin by pre-training this CNN with the training data. This pre-training step is crucial for enabling the network to learn the underlying structure and distribution of the clean data, thus enhancing its understanding of image features, which is beneficial for the subsequent denoising task. After pre-training, we train the network using pairs of noisy and clean images. Noisy images are generated by adding Gaussian noise with a noise factor of 0.3, serving as input, while the corresponding clean images are used as output. This process teaches the network to effectively map noisy images to their denoised versions. In both the pre-training and denoising training phases, we use a batch size of 512 and

Table 2
Comparison of various KAN architectures for a regression task. Each row presents the averaged output of the respective model across its different parameters. The table demonstrates that fKAN achieves the highest prediction accuracy while utilizing significantly fewer parameters and exhibiting better time efficiency than all alternatives, except for the Chebyshev KANs.

Model	Time		MAE		Number of parameters	
	Mean	Std.	Mean	Std.	Mean	Std.
fKAN	0.60	0.163	$\mathbf{2.43 \times 10^{+01}}$	$9.87 \times 10^{+00}$	**20.00**	0.000
KAN [4]	2.37	0.704	$1.64 \times 10^{+02}$	$3.35 \times 10^{+02}$	272.00	37.947
rKAN [94]	0.83	0.175	$2.63 \times 10^{+01}$	$9.84 \times 10^{+00}$	20.00	0.000
ChebKAN [39,83]	**0.24**	0.071	$1.05 \times 10^{+06}$	$2.35 \times 10^{+06}$	45.00	12.649
wKAN [18]	0.78	0.041	$1.17 \times 10^{+04}$	$2.61 \times 10^{+04}$	58.00	0.000
fastKAN [82]	0.65	–	$2.48 \times 10^{+01}$	–	141.00	–
MLP-KAN [95]	2.56	–	$3.19 \times 10^{+01}$	–	3468.00	–

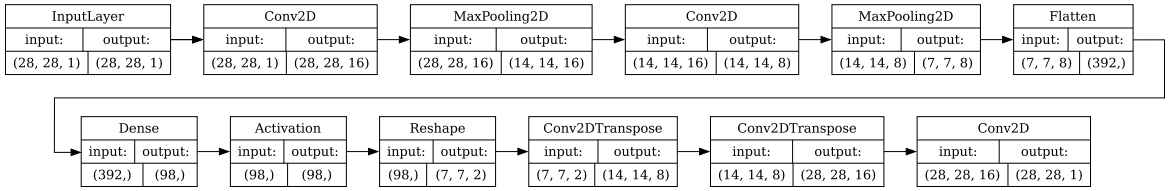


Fig. 8. The architecture of the autoencoder used for denoising Fashion MNIST images. One of the activation functions-sigmoid, tanh, ReLU, or fJNB with q values ranging from 2 to 6—is selected to allow for flexible adaptation to different non-linearities in the data. For the CIFAR-10 dataset, the input and output shapes will differ, but the architecture remains consistent.

Table 3
Performance comparison of different activation functions in a CNN for classifying the MNIST, CIFAR-10, CIFAR-100, SVHN, and mini ImageNet datasets.

Act. Fun.	MNIST	CIFAR10	CIFAR100	SVHN	mini ImageNet
Sigmoid	98.09 ± 0.09	49.60 ± 0.46	26.18 ± 0.43	82.76 ± 0.27	14.50 ± 0.15
Tanh	98.90 ± 0.07	65.84 ± 0.50	38.66 ± 0.39	84.50 ± 0.11	22.18 ± 0.21
ReLU	99.14 ± 0.04	$\mathbf{68.71} \pm 0.51$	40.69 ± 0.53	87.13 ± 0.24	25.67 ± 0.67
fJNB(2)	99.13 ± 0.05	67.27 ± 0.38	40.31 ± 0.70	86.35 ± 0.21	27.84 ± 0.20
fJNB(3)	99.20 ± 0.08	65.21 ± 0.47	38.79 ± 0.27	87.29 ± 0.18	$\mathbf{28.91} \pm 0.36$
fJNB(4)	$\mathbf{99.23} \pm 0.05$	66.60 ± 0.29	40.23 ± 0.25	87.42 ± 0.13	28.20 ± 0.51
fJNB(5)	99.20 ± 0.05	67.14 ± 0.54	$\mathbf{40.79} \pm 0.37$	87.36 ± 0.08	27.42 ± 0.37
fJNB(6)	99.02 ± 0.12	65.70 ± 0.65	38.76 ± 0.61	$\mathbf{87.89} \pm 0.24$	23.70 ± 0.94

train the CNN over 20 epochs using the Adam optimizer with Keras's default learning rate.

After the training phase, we evaluate the performance of the trained network by comparing the denoising results of the CNN with the ground truth clean images. The results are quantitatively presented in Tables 6 and 7, which include PSNR and SSIM metrics to assess the quality of the denoised images. These metrics are defined as:

$$\mathrm{PSNR}(\zeta, \hat{\zeta}) = 10 \log_{10} \left(\frac{\mathrm{MAX}^2\{\zeta\} \cdot MN}{\|\zeta - \hat{\zeta}\|_F^2} \right), \tag{39}$$

where $\|\cdot\|_F$ is the Frobenius norm, $\zeta, \hat{\zeta} \in [0,1]^{M \times N}$, and

$$\mathrm{SSIM}(\zeta, \hat{\zeta}) = \frac{(2\mu_\zeta \mu_{\hat{\zeta}} + C_1)(2\sigma_{\zeta\hat{\zeta}} + C_2)}{(\mu_\zeta^2 + \mu_{\hat{\zeta}}^2 + C_1)(\sigma_\zeta^2 + \sigma_{\hat{\zeta}}^2 + C_2)}, \tag{40}$$

in which μ_ζ and $\mu_{\hat{\zeta}}$ are the means of ζ and $\hat{\zeta}$, $\sigma_{\zeta\hat{\zeta}}$ is the covariance between ζ and $\hat{\zeta}$, σ_ζ^2 and $\sigma_{\hat{\zeta}}^2$ are the variances of ζ and $\hat{\zeta}$, respectively. The constants C_1 and C_2 are used to stabilize the division.

Sentiment analysis

For the final deep learning experiment, we evaluate the efficiency of the fJNB in a sentiment analysis task using the well-known IMDB dataset, which includes 20,000 samples for training, 5000 for validation, and 25,000 for testing. The text data is preprocessed using a custom standardization function to remove HTML tags and punctuation. Subsequently, the text is vectorized with a maximum of 20,000 tokens and a sequence length of 500. This processed data is fed into a deep classifier model, which is a 1D CNN comprising an embedding layer, dropout layers, convolutional layers, global max pooling, a dense layer, and an output layer. The full architecture is depicted in Fig. 9. Similar to the previous examples, we use the Adam optimizer, with a batch size of 512 and 10 epochs for training the network. The performance of the model is evaluated using binary classification metrics, including accuracy, precision, recall, and ROC-AUC, as reported in Table 8.

5.2. Physics-informed deep learning tasks

To demonstrate the effectiveness of the proposed architecture in solving physical problems, we simulate various types of differential equations within a physics-informed loss function scheme. Mathematically, for a functional operator of the form $\mathcal{N}(\chi) = S$, we define the residual function as $\Re(\zeta) = \mathcal{N}(\chi)(\zeta) - S(\zeta)$ and then formulate the loss function of the network as:

$$\mathrm{Loss}(\zeta) = \Re(\zeta)^{\mathrm{T}} \Re(\zeta) + \mathfrak{B}^2 + \mathfrak{I}^2, \tag{41}$$

where $\mathfrak{B}$ and $\mathfrak{I}$ represent the network errors for the known boundary and initial conditions, respectively, and $\zeta \in \mathbb{R}^{N \times d}$ is a vector containing N training points in the d-dimensional problem domain. To compute the derivatives of the network with respect to the input parameters, we utilize the standard backpropagation algorithm. For fractional-order derivatives, we employ the Caputo operational matrix of differentiation as proposed by Taheri et al. [103]. This method approximates the fractional derivative using an L1 finite difference scheme in a matrix format, which is then multiplied by the network output to obtain the fractional derivative of the network.

In the following experiments, we simulate various ordinary, partial, and fractional-order differential equations. Given the complexity of these problems, we utilize the second proposed architecture (Fig. 3(b)), which incorporates multiple fJNBs simultaneously in a network architecture with a concatenation operator. This approach ensures that different polynomial degrees are included in approximating the function.

Ordinary differential equations

For the first example, we consider the well-known Lane–Emden differential equation in its standard form. This benchmark problem is a second-order singular differential equation given by:

$$\begin{aligned} &\frac{\mathrm{d}^2}{\mathrm{d}\zeta^2}\chi(\zeta) + \frac{2}{\zeta}\frac{\mathrm{d}}{\mathrm{d}\zeta}\chi(\zeta) + \chi^m(\zeta) = 0, \\ &\chi(0) = 1, \quad \chi'(0) = 0, \end{aligned} \tag{42}$$

Table 4
The macro F1 score of predictions from the MobileNetV3 Small model for classifying the Oxford Flowers dataset.

Act. Fun.	Depth of classifier									
	1	2	3	4	5	6	7	8	9	10
ReLU	75.73	74.62	69.09	61.35	53.04	51.55	44.54	42.32	32.86	26.59
fJNB(2)	**76.75**	**75.02**	**74.00**	**73.24**	**71.05**	**70.16**	**65.97**	**67.46**	**65.01**	**60.94**
fJNB(3)	75.10	69.92	66.93	61.82	59.43	57.64	52.31	46.94	47.11	37.47
fJNB(4)	71.51	62.73	57.35	53.91	50.39	44.16	46.72	35.70	10.92	02.62
fJNB(5)	65.75	55.10	52.00	46.65	42.90	32.10	24.91	19.40	0.99	01.22

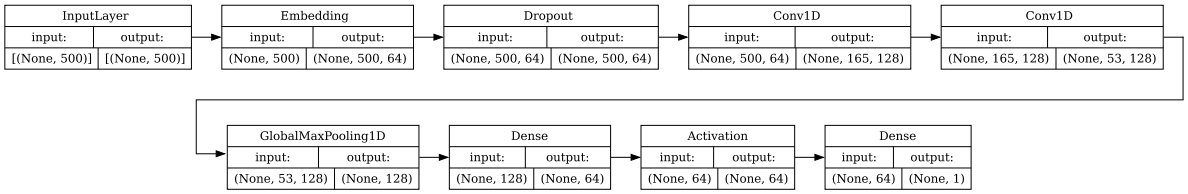


Fig. 9. The architecture of the 1D-CNN used for sentiment analysis on the IMDB dataset. One of the activation functions-sigmoid, tanh, ReLU, or fJNB with q values ranging from 1 to 6—is selected to allow for flexible adaptation to different non-linearities in the data.

Table 5
Classification report for transfer learning using the ResNet-18 architecture with a classifier featuring one hidden layer and various activation functions.

Act. Fun.	Accuracy	Macro			Weighted		
		Precision	Recall	F1	Precision	Recall	F1
RELU	78.19	77.48	80.48	77.67	81.01	78.19	78.04
fJNB(2)	**80.13**	78.77	**82.42**	**79.69**	**82.27**	**80.13**	**80.23**
fJNB(3)	79.88	**78.82**	81.68	79.28	82.16	79.88	80.08
fJNB(4)	78.53	77.04	80.87	77.67	81.15	78.53	78.63
fJNB(5)	76.65	75.45	79.10	76.04	79.38	76.65	76.81
fJNB(6)	73.95	71.56	75.70	72.57	76.20	73.95	74.01

where m is an integer parameter, typically ranging from zero to five. For this problem, we use 1500 equidistant data points as input. Six fJNB blocks are applied, each to a layer containing 10 neurons. The final layer of the network takes the inputs from 60 different activations of the network and outputs a single real value as the approximation for the Lane–Emden equation. Optimization of this network is performed using the limited memory variant of the quasi-Newton Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm.

The simulation results for this problem, considering $m = 0, 1, \ldots, 5$, are depicted in Fig. 10. The first root of the predicted function for each value of m holds physical significance and serves as a criterion for accuracy assessment. Comparison with a similar neural network approach [104] demonstrates significant accuracy improvements (see Table 9).

Partial differential equations

As a benchmark partial differential equation, we choose the renowned one-dimensional Burgers equation, defined as follows [106]:

$$\begin{aligned} &\frac{\partial}{\partial\tau}\chi(\zeta,\tau) + m_0\chi(\zeta,\tau)\frac{\partial}{\partial\zeta}\chi(\zeta,\tau) - m_1\frac{\partial^2}{\partial\tau^2}\chi(\zeta,\tau) = 0,\\ &\chi(\zeta,0) = \frac{m_2}{m_0} + 2\frac{m_1}{m_0}\tanh(x). \end{aligned} \tag{43}$$

The exact solution to this problem is given by $\chi(\zeta,\tau) = \frac{m_2}{m_0} + 2\frac{m_1}{m_0}\tanh(x - m_2\tau)$ for predefined parameters m_0, m_1, and m_2. To address this problem, we utilize a neural network with a similar architecture to the previous example, with the exception of having two hidden neurons per layer before the fJNB and an input dimension of two to accommodate both time and space variables. For these two variables, we fed the network with a Cartesian product of 100 equidistant points in $[0, 1]$. For this example, we compare the predicted solution with the exact solution for two common parameter choices m_0, m_1, and m_2 as used by Khater et al. [106]. The simulation results are depicted in Fig. 11, demonstrating accurate predictions.

Fractional differential equations

For the final experiment involving physics-informed neural networks, we investigate a fractional delay differential equation represented as [107]:

$$\begin{aligned} &\frac{\mathrm{d}^{0.3}}{\mathrm{d}\zeta^{0.3}}\chi(\zeta) = \chi(\zeta-1) - \chi(\zeta) + 1 - 3\zeta + 3\zeta^2 + \frac{2000\zeta^{2.7}}{1071\Gamma(0.7)},\\ &\chi(0) = 0, \end{aligned} \tag{44}$$

where $\Gamma(\cdot)$ denotes the gamma function and the fractional derivative is defined using the Caputo fractional derivative operator [107,108]:

$$\frac{\mathrm{d}^{\alpha}}{\mathrm{d}\zeta^{\alpha}}\chi(\zeta) = \frac{1}{\Gamma(1-\alpha)}\int_0^{\zeta}\frac{\chi'(\tau)}{(\zeta-\tau)^{\alpha}}d\tau, \quad 0 < \alpha < 1. \tag{45}$$

For simulating this problem, we adopt a similar approach to the previous examples. Specifically, we design the architecture as follows: Each of the first 6 layers connected to fJNBs employs 10 neurons. Subsequently, a concatenation layer is employed. Next, a layer with a weight size of 60×10 is used to reduce the network features to 10. Then, three fully connected layers are employed, followed by a final regression neuron. The loss function is computed using the residual of this equation:

$$\mathfrak{R}(\boldsymbol{\zeta}) = \left[D^{0.3}\hat{\chi}(\boldsymbol{\zeta})\right] - \left[\hat{\chi}(\boldsymbol{\zeta}-1) - \hat{\chi}(\boldsymbol{\zeta}) + 1 - 3\boldsymbol{\zeta} + 3\boldsymbol{\zeta}^2 + \frac{2000}{1071\Gamma(0.7)}\boldsymbol{\zeta}^{2.7}\right], \tag{46}$$

where D^{α} is a lower triangular operational matrix of the derivative for the Caputo derivative defined in Taheri et al. [103]. The network weights are then optimized using the L-BFGS algorithm. The predicted solution, the residual with respect to the exact solution, and the network residual loss using this architecture are reported in Fig. 12. In this example, we observe fluctuations in the residual errors, which, based on experiments, we found are a direct result of the architectures with multiple fJNBs.

6. Conclusion

This paper introduces a novel extension of the Kolmogorov–Arnold Network framework, incorporating fractional-order orthogonal Jacobi polynomials as basis functions for approximation. Through Eq. (35) and Figs. 3(a) and 3(b), we demonstrate the integration of this new basis function into the KAN architecture. Comparisons with the Learnable Activation Network [4] highlight the similarities and advantages of our approach.

We observe that the proposed basis function exhibits key characteristics of effective activation functions, including non-linearity, simplicity, and straightforward derivatives. Our study further showcases how the fractional order of these polynomials can be leveraged within neural networks, with parameters adaptively tuned during training.

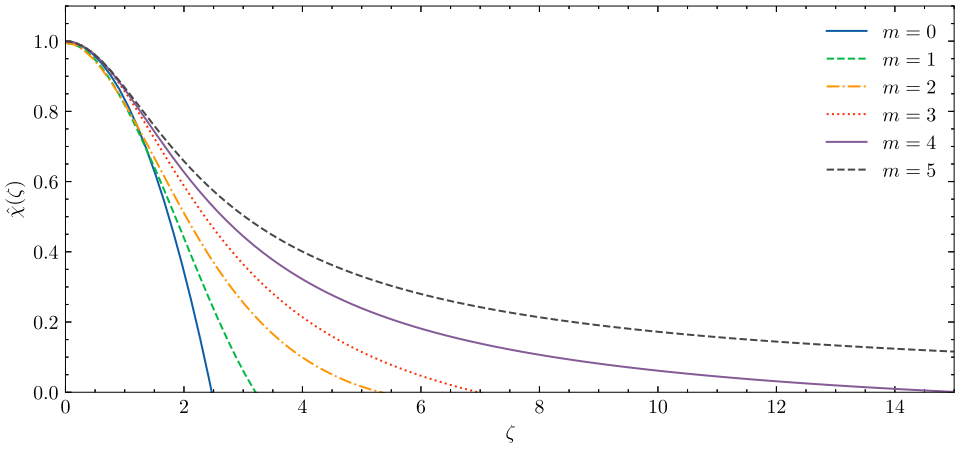


Fig. 10. The simulated results of the Lane–Emden differential equation using the fKAN.

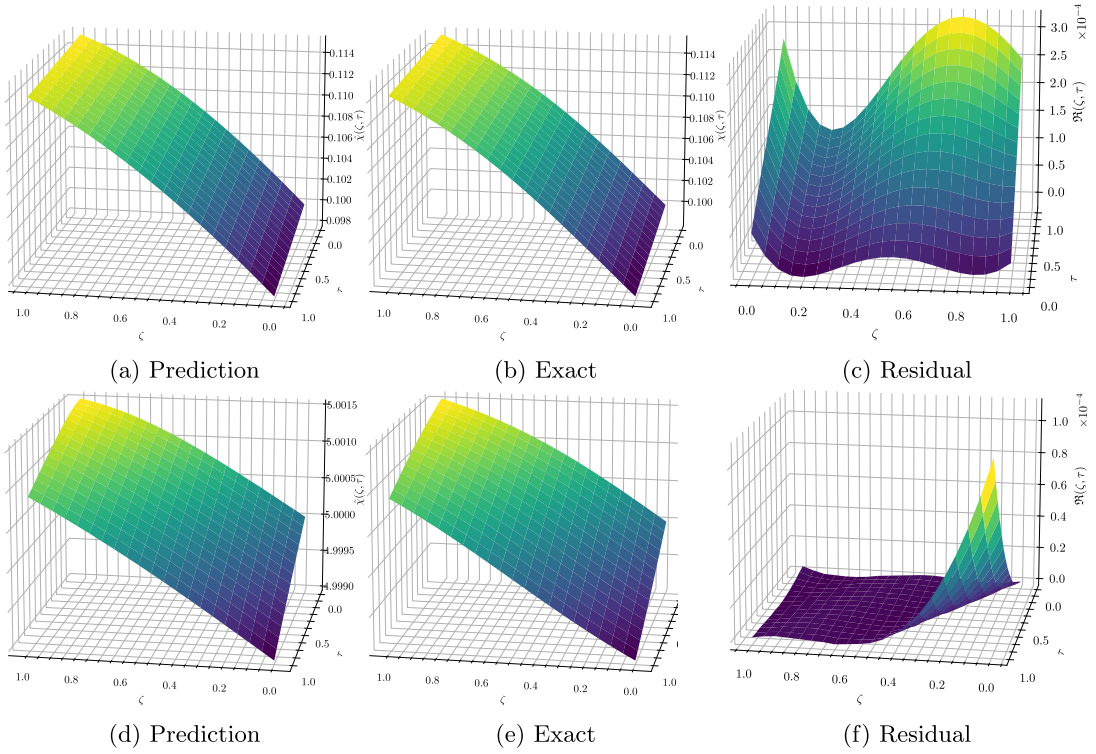


Fig. 11. Simulation results of the Burgers PDE using the fJNB with two sets of parameters: $m_0, m_1, m_2 = 1, 0.01, 0.1$ (top) and $m_0, m_1, m_2 = 0.1, 0.0001, 0.5$ (bottom).

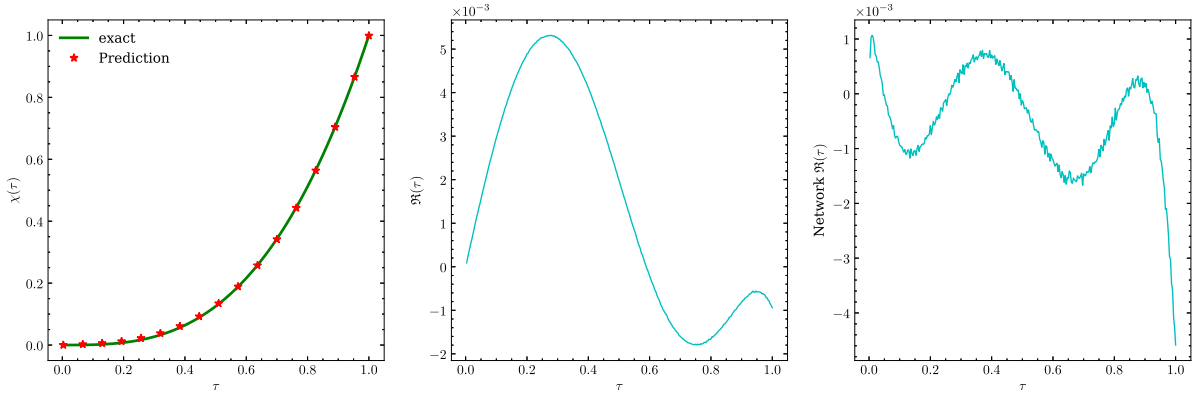


Fig. 12. The simulation results for the fractional delay differential equation using the fJNB architecture.

Table 6
Comparison of the accuracy of various activation functions for the Fashion MNIST image denoising task.

Act. Func.	Pre-train			Denoising		
	MSE	PSNR	SSIM	MSE	PSNR	SSIM
Sigmoid	0.018 ± 0.003	18.026 ± 0.709	0.579 ± 0.043	0.018 ± 0.001	17.813 ± 0.347	0.564 ± 0.019
Tanh	0.017 ± 0.002	18.078 ± 0.545	0.583 ± 0.035	0.018 ± 0.001	17.830 ± 0.261	0.562 ± 0.020
ReLU	0.017 ± 0.001	18.124 ± 0.227	0.577 ± 0.013	0.019 ± 0.001	17.543 ± 0.186	0.537 ± 0.014
fJNB(5)	0.013 ± 0.001	**19.205** ± 0.300	0.647 ± 0.015	0.017 ± 0.001	18.159 ± 0.308	0.585 ± 0.017
fJNB(6)	**0.013** ± 0.001	19.171 ± 0.337	**0.648** ± 0.019	**0.016** ± 0.001	**18.206** ± 0.174	**0.587** ± 0.009

Table 7
Comparison of the accuracy of various activation functions for the CIFAR10 image denoising task.

Act. Func.	Pre-train			Denoising		
	MSE	PSNR	SSIM	MSE	PSNR	SSIM
Sigmoid	0.014 ± 0.001	19.238 ± 0.361	0.574 ± 0.030	0.014 ± 0.001	18.871 ± 0.395	0.533 ± 0.019
Tanh	0.014 ± 0.001	19.182 ± 0.474	0.554 ± 0.015	0.015 ± 0.001	18.684 ± 0.358	0.513 ± 0.011
ReLU	0.016 ± 0.001	18.476 ± 0.332	0.514 ± 0.028	0.017 ± 0.001	18.197 ± 0.190	0.485 ± 0.017
fJNB(5)	0.012 ± 0.002	19.766 ± 0.578	**0.590** ± 0.020	0.013 ± 0.002	**19.294** ± 0.466	0.535 ± 0.021
fJNB(6)	**0.012** ± 0.001	**19.792** ± 0.404	0.588 ± 0.030	**0.013** ± 0.001	19.184 ± 0.338	**0.535** ± 0.020

Table 8
Binary classification metrics for the IMDB dataset using different activation functions.

Act. Func.	Loss	Accuracy	Precision	Recall	ROC-AUC
Sigmoid	0.604 ± 0.039	85.198 ± 0.468	81.440 ± 1.511	91.266 ± 1.532	91.631 ± 0.422
Tanh	0.610 ± 0.060	85.604 ± 0.818	82.364 ± 2.790	90.912 ± 2.705	91.894 ± 0.505
ReLU	0.808 ± 0.064	83.826 ± 1.141	78.197 ± 2.124	**93.995 ± 1.358**	91.080 ± 0.404
fJNB(1)	0.724 ± 0.074	84.462 ± 1.156	79.967 ± 3.160	92.387 ± 3.090	91.241 ± 0.501
fJNB(2)	0.671 ± 0.044	85.205 ± 0.591	80.559 ± 1.374	92.882 ± 1.126	91.306 ± 0.478
fJNB(3)	0.646 ± 0.057	84.828 ± 0.761	80.100 ± 1.787	92.808 ± 1.404	91.576 ± 0.416
fJNB(4)	**0.559 ± 0.062**	**85.645 ± 0.762**	**83.395 ± 3.549**	89.498 ± 3.916	**92.497 ± 0.404**
fJNB(5)	0.691 ± 0.100	84.247 ± 1.479	79.283 ± 3.401	93.190 ± 2.584	91.454 ± 0.825
fJNB(6)	0.627 ± 0.107	84.455 ± 1.658	80.079 ± 4.017	92.371 ± 3.215	91.648 ± 0.985

Table 9
Comparison of the first roots of the predicted solution with the exact roots from [105] and the approximated results from a similar neural network approach [104].

m	Exact [105]	fKAN approximate	fKAN error	GEPINN [104] Error
0	2.44948974	2.44945454	3.52×10^{-5}	$\mathbf{1.40 \times 10^{-7}}$
1	3.14159265	3.14160132	$\mathbf{8.67 \times 10^{-6}}$	4.83×10^{-3}
2	4.35287460	4.35288394	$\mathbf{9.34 \times 10^{-6}}$	8.93×10^{-3}
3	6.89684860	6.89684915	$\mathbf{5.55 \times 10^{-7}}$	1.88×10^{-2}
4	14.9715463	14.9712493	$\mathbf{2.97 \times 10^{-4}}$	5.08×10^{-2}

A comprehensive series of experiments across various deep learning tasks, including MLP and 1D/2D-CNN architectures, underscores the superior performance of the fractional KAN over traditional KAN and other activation functions. However, we acknowledge limitations such as increased time complexity compared to simpler activation functions and reduced interpretability relative to KAN due to the global nature of the basis functions. Future work may explore local basis function variants, like fractional B-splines, to address these limitations.

Funding statement

No external funding received for this project.

Research data

The raw data required to reproduce the above findings are openly available to download.

Code availability

The complete code necessary to reproduce the results presented in this paper is publicly available in our GitHub repository: https://github.com/alirezaafzalaghaei/fKAN. Additionally, the fKAN method can be accessed as a package on the Python Package Index (PyPI) under the name `fkan`.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The raw data required to reproduce the above findings are openly available to download.

References

[1] Charu C. Aggarwal, et al., Neural Networks and Deep Learning, Vol. 10, Springer, 2018, p. 3, 978.
[2] Ian Goodfellow, Deep Learning, MIT Press, 2016.
[3] Ziming Liu, Pingchuan Ma, Yixuan Wang, Wojciech Matusik, Max Tegmark, Kan 2.0: Kolmogorov-arnold networks meet science, 2024, arXiv preprint arXiv:2408.10205.
[4] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, Max Tegmark, Kan: Kolmogorov-arnold networks, 2024, arXiv preprint arXiv:2404.19756.
[5] Shriyank Somvanshi, Syed Aaqib Javed, Md. Monzurul Islam, Diwas Pandit, Subasish Das, A survey on Kolmogorov-Arnold network, 2024, arXiv preprint arXiv:2411.06078.
[6] Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, Roberto Prevete, A survey on modern trainable activation functions, Neural Netw. 138 (2021) 14–32.
[7] Z. Hu, The study of neural network control system, Control Decis. 7 (1992) 361–366.
[8] Chyi-Tsong Chen, Wei-Der Chang, A feedforward neural network with function shape autotuning, Neural Netw. 9 (4) (1996) 627–641.
[9] Francesco Piazza, Aurelio Uncini, Massimo Zenobi, et al., Artificial neural networks with adaptive polynomial activation function, 1992.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1026–1034.

[11] Yevgeniy Bodyanskiy, Serhii Kostiuk, Learnable extended activation function for deep neural networks, Int. J. Comput. (2023) 311–318, (Oct. 2023).

[12] Martin Arjovsky, Amar Shah, Yoshua Bengio, Unitary evolution recurrent neural networks, in: International Conference on Machine Learning, PMLR, 2016, pp. 1120–1128.

[13] Evgenii Pishchik, Trainable activations for image classification, 2023.

[14] Simone Scardapane, Michele Scarpiniti, Danilo Comminiello, Aurelio Uncini, Learning activation functions from data using cubic spline interpolation, in: Neural Advances in Processing Nonlinear Dynamic Signals 27, Springer, 2019, pp. 73–83.

[15] Daniele Fakhoury, Emanuele Fakhoury, Hendrik Speleers, ExSpliNet: An interpretable and expressive spline-based neural network, Neural Netw. 152 (2022) 332–346.

[16] Sebastian Neumayer, Alexis Goujon, Pakshal Bohra, Michael Unser, Approximation of Lipschitz functions using deep spline neural networks, SIAM J. Math. Data Sci. 5 (2) (2023) 306–322.

[17] Pakshal Bohra, Joaquim Campos, Harshit Gupta, Shayan Aziznejad, Michael Unser, Learning activation functions in deep (spline) neural networks, IEEE Open J. Signal Process. 1 (2020) 295–309.

[18] Zavareh Bozorgasl, Hao Chen, Wav-kan: Wavelet kolmogorov-arnold networks, 2024, arXiv preprint arXiv:2405.12832.

[19] Jinfeng Xu, Zheyu Chen, Jinze Li, Shuo Yang, Wei Wang, Xiping Hu, Edith C.-H. Ngai, FourierKAN-GCF: Fourier Kolmogorov-Arnold network–An effective and efficient feature transformation for graph collaborative filtering, 2024, arXiv preprint arXiv:2406.01034.

[20] Remi Genet, Hugo Inzirillo, Tkan: Temporal kolmogorov-arnold networks, 2024, arXiv preprint arXiv:2405.07344.

[21] Moein E. Samadi, Younes Müller, Andreas Schuppert, Smooth Kolmogorov Arnold networks enabling structural knowledge representation, 2024, arXiv preprint arXiv:2405.11318.

[22] Diab W. Abueidda, Panos Pantidis, Mostafa E. Mobasher, Deepokan: Deep operator network based on kolmogorov arnold networks for mechanics problems, 2024, arXiv preprint arXiv:2405.19143.

[23] Benjamin C. Koenig, Suyong Kim, Sili Deng, KAN-ODEs: Kolmogorov–Arnold network ordinary differential equations for learning dynamical systems and hidden physics, Comput. Methods Appl. Mech. Engrg. 432 (2024) 117397.

[24] Alireza Afzal Aghaei, KANtrol: A physics-informed Kolmogorov-Arnold network framework for solving multi-dimensional and fractional optimal control problems, 2024, arXiv preprint arXiv:2409.06649.

[25] Seyd Teymoor Seydi, Exploring the potential of polynomial basis functions in Kolmogorov-Arnold networks: A comparative study of different groups of polynomials, 2024, arXiv preprint arXiv:2406.02583.

[26] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, Shimon Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, Neural Netw. 6 (6) (1993) 861–867.

[27] Maxwell Stinchcombe, Halbert White, Approximating and learning unknown mappings using multilayer feedforward networks with bounded weights, in: 1990 IJCNN International Joint Conference on Neural Networks, IEEE, 1990, pp. 7–16.

[28] Sankatha Prasad Singh, J.H.W. Burry, B. Watson, Approximation Theory and Spline Functions, vol. 136, Springer Science & Business Media, 2012.

[29] Walter Gautschi, Orthogonal Polynomials: Computation and Approximation, OUP Oxford, 2004.

[30] Jie Shen, Tao Tang, Li-Lian Wang, Spectral Methods: Algorithms, Analysis and Applications, vol. 41, Springer Science & Business Media, 2011.

[31] Alireza Afzal Aghaei, Kourosh Parand, Ali Nikkhah, Shakila Jaberi, Solving Falkner-Skan type equations via Legendre and Chebyshev neural blocks, 2023, arXiv preprint arXiv:2308.03337.

[32] M. Deepthi, G.N.V.R. Vikram, P. Venkatappareddy, Development of a novel activation function based on Chebyshev polynomials: an aid for classification and denoising of images, J. Supercomput. (2023) 1–17.

[33] Fawaz E. Alsaadi, Amirreza Yasami, Hajid Alsubaie, Ahmed Alotaibi, Hadi Jahanshahi, Control of a hydraulic generator regulating system using Chebyshev-neural-network-based non-singular fast terminal sliding mode method, Mathematics 11 (1) (2023) 168.

[34] P. Venkatappareddy, Jayanth Culli, Siddharth Srivastava, Brejesh Lall, A Legendre polynomial based activation function: An aid for modeling of max pooling, Digit. Signal Process. 115 (2021) 103093.

[35] Cong Wang, Hongli Zhang, Ping Ma, Wind power forecasting based on singular spectrum analysis and a new hybrid Laguerre neural network, Appl. Energy 259 (2020) 114139.

[36] Jagdish Chandra Patra, Pramod Kumar Meher, Goutam Chakraborty, Development of Laguerre neural-network-based intelligent sensors for wireless sensor networks, IEEE Trans. Instrum. Meas. 60 (3) (2010) 725–734.

[37] A. Ebrahimzadeh, M. Ahmadi, M. Safarnejad, Classification of ECG signals using Hermite functions and MLP neural networks, J. AI Data Min. 4 (1) (2016) 55–65.

[38] Yuhe Guo, Zhewei Wei, Graph neural networks with learnable and optimal polynomial bases, 2023, arXiv preprint arXiv:2302.12432.

[39] Sidharth SS, Chebyshev polynomial-based Kolmogorov-Arnold networks: An efficient architecture for nonlinear function approximation, 2024, arXiv preprint arXiv:2405.07200.

[40] Chunyu Guo, Lucheng Sun, Shilong Li, Zelong Yuan, Chao Wang, Physics-informed Kolmogorov-Arnold network with Chebyshev polynomials for fluid mechanics, 2024, arXiv preprint arXiv:2411.04516.

[41] Farinaz Mostajeran, Salah A. Faroughi, Epi-ckans: Elasto-plasticity informed kolmogorov-arnold networks using chebyshev polynomials, 2024, arXiv preprint arXiv:2410.10897.

[42] Ali Kashefi, Kolmogorov-Arnold PointNet: Deep learning for prediction of fluid fields on irregular geometries, 2024, arXiv preprint arXiv:2408.02950.

[43] B.C. Carlson, B-splines, hypergeometric functions, and Dirichlet averages, J. Approx. Theory 67 (3) (1991) 311–325.

[44] Brigitte Forster, Peter Massopust, Splines of complex order: Fourier, filters and fractional derivatives, Sampl. Theory Signal Image Process. 10 (1) (2011) 89–109.

[45] Qian Tao, Zhen Wang, Wenyuan Yu, Yaliang Li, Zhewei Wei, LONGNN: Spectral GNNs with learnable orthonormal basis, 2023, arXiv preprint arXiv:2303.13750.

[46] Hongliang Liu, Huini Liu, Jie Xu, Lijuan Li, Jingwen Song, Jacobi neural network method for solving linear differential-algebraic equations with variable coefficients, Neural Process. Lett. 53 (5) (2021) 3357–3374.

[47] Jamal Amani Rad, Kourosh Parand, Snehashish Chakraverty, Learning with Fractional Orthogonal Kernel Classifiers in Support Vector Machines: Theory, Algorithms and Applications, Springer, 2023.

[48] Kourosh Parand, Mehdi Delkhosh, Accurate solution of the Thomas–Fermi equation using the fractional order of rational Chebyshev functions, J. Comput. Appl. Math. 317 (2017) 624–642.

[49] Kourosh Parand, Mehdi Delkhosh, Solving Volterra's population growth model of arbitrary order using the generalized fractional order of the Chebyshev functions, Ric. Mat. 65 (2016) 307–328.

[50] Zeinab Hajimohammadi, Fatemeh Baharifard, Ali Ghodsi, Kourosh Parand, Fractional Chebyshev deep neural network (FCDNN) for solving differential models, Chaos Solitons Fractals 153 (2021) 111530.

[51] Michael Unser, Thierry Blu, Fractional splines and wavelets, SIAM Rev. 42 (1) (2000) 43–67.

[52] Thierry Blu, Michael Unser, The fractional spline wavelet transform: definition end implementation, in: 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 00CH37100), Vol. 1, IEEE, 2000, pp. 512–515.

[53] K. Parand, H. Yousefi, M. Delkhosh, A numerical approach to solve Lane-Emden type equations by the fractional order of rational Bernoulli functions, Romanian J. Phys. 62 (104) (2017) 1–24.

[54] Kourosh Parand, Pooria Mazaheri, Hossein Yousefi, Mehdi Delkhosh, Fractional order of rational Jacobi functions for solving the non-linear singular Thomas-Fermi equation, Eur. Phys. J. Plus 132 (2017) 1–13.

[55] Kourosh Parand, A.A. Aghaei, M. Jani, A. Ghodsi, Parallel LS-SVM for the numerical simulation of fractional Volterra's population model, Alex. Eng. J. 60 (6) (2021) 5637–5647.

[56] Amir Hosein Hadian Rasanan, Nastaran Bajalan, Kourosh Parand, Jamal Amani Rad, Simulation of nonlinear fractional dynamics arising in the modeling of cognitive decision making using a new fractional neural network, Math. Methods Appl. Sci. 43 (3) (2020) 1437–1466.

[57] Xiyuan Wang, Muhan Zhang, How powerful are spectral graph neural networks, in: International Conference on Machine Learning, PMLR, 2022, pp. 23341–23362.

[58] Susmita Mall, Snehashish Chakraverty, Application of Legendre neural network for solving ordinary differential equations, Appl. Soft Comput. 43 (2016) 347–356.

[59] Liying Ma, Khashayar Khorasani, Constructive feedforward neural networks using Hermite polynomial activation functions, IEEE Trans. Neural Netw. 16 (4) (2005) 821–833.

[60] Gerasimos G. Rigatos, Spyros G. Tzafestas, Feed-forward neural networks using Hermite polynomial activation functions, in: Advances in Artificial Intelligence: 4th Helenic Conference on AI, SETN 2006, Heraklion, Crete, Greece, May 18-20, 2006. Proceedings 4, Springer, 2006, pp. 323–333.

[61] Leif E. Peterson, Kirill V. Larin, Image classification of artificial fingerprints using Gabor wavelet filters, self-organising maps and Hermite/Laguerre neural networks, Int. J. Knowl. Eng. Soft Data Paradigms 1 (3) (2009) 239–256.

[62] Yinghao Chen, Hanyu Yu, Xiangyu Meng, Xiaoliang Xie, Muzhou Hou, Julien Chevallier, Numerical solving of the generalized Black-Scholes differential equation using Laguerre neural network, Digit. Signal Process. 112 (2021) 103003.

[63] Sergio A. Dorado-Rojas, Bhanukiran Vinzamuri, Luigi Vanfretti, Orthogonal laguerre recurrent neural networks, in: Mach. Learn. and the Phys. Sci. Workshop at the 34th Conf. on Neural Info. Proc. Syst., NeurIPS, 2020.

[64] Emad M. Alsaedi, Alaa Kadhim Farhan, Mayadah W. Falah, Bashra Kadhim Oleiwi, Classification of encrypted data using deep learning and Legendre polynomials, in: The International Conference on Innovations in Computing Research, Springer, 2022, pp. 331–345.

[65] Allan Pinkus, Approximation theory of the MLP model in neural networks, Acta Numer. 8 (1999) 143–195.
[66] Takato Nishijima, Universal approximation theorem for neural networks, 2021, arXiv preprint arXiv:2102.10993.
[67] Wataru Kumagai, Akiyoshi Sannai, Universal approximation theorem for equivariant maps by group cnns, 2020, arXiv preprint arXiv:2012.13882.
[68] Gustaf Gripenberg, Approximation by neural networks with a bounded number of nodes at each level, J. Approx. Theory 122 (2) (2003) 260–266.
[69] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, Liwei Wang, The expressive power of neural networks: A view from the width, Adv. Neural Inf. Process. Syst. 30 (2017).
[70] Patrick Kidger, Terry Lyons, Universal approximation with deep narrow networks, in: Conference on Learning Theory, PMLR, 2020, pp. 2306–2327.
[71] Zhen Qin, Xiaodong Han, Weixuan Sun, Bowen He, Dong Li, Dongxu Li, Yuchao Dai, Lingpeng Kong, Yiran Zhong, Toeplitz neural network for sequence modeling, in: The Eleventh International Conference on Learning Representations, 2023, URL: https://openreview.net/forum?id=IxmWsm4xrua.
[72] Ameya Jagtap, George Em Karniadakis, How important are activation functions in regression and classification? A survey, performance comparison, and future directions, J. Mach. Learn. Model. Comput. (2022).
[73] Jürgen Braun, Michael Griebel, On a constructive proof of Kolmogorov's superposition theorem, Constr. Approx. 30 (2009) 653–675.
[74] Andrei Nikolaevich Kolmogorov, On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition, in: Doklady Akademii Nauk, Vol. 114, Russian Academy of Sciences, 1957, pp. 953–956, 5.
[75] Carla Manni, Fabio Roman, Hendrik Speleers, Generalized B-splines in isogeometric analysis, in: Approximation Theory XV: San Antonio 2016 15, Springer, 2017, pp. 239–267.
[76] Ashis Paul, Rajarshi Bandyopadhyay, Jin Hee Yoon, Zong Woo Geem, Ram Sarkar, SinLU: Sinu-sigmoidal linear unit, Mathematics 10 (3) (2022) 337.
[77] S. Dzhenzher, A. Skopenkov, A structured proof of Kolmogorov's Superposition Theorem, 2021, arXiv preprint arXiv:2105.00408.
[78] Johannes Schmidt-Hieber, The Kolmogorov–Arnold representation theorem revisited, Neural Netw. 137 (2021) 119–126.
[79] G.G. Lorentz, Metric entropy, widths, and superpositions of functions, Amer. Math. Monthly 69 (6) (1962) 469–485.
[80] David A. Sprecher, On the structure of continuous functions of several variables, Trans. Amer. Math. Soc. 115 (1965) 340–355.
[81] Federico Girosi, Tomaso Poggio, Representation properties of networks: Kolmogorov's theorem is irrelevant, Neural Comput. 1 (4) (1989) 465–469.
[82] Ziyao Li, Kolmogorov-Arnold networks are radial basis function networks, 2024, arXiv preprint arXiv:2405.06721.
[83] SynodicMonth, ChebyKAN, 2024, https://github.com/SynodicMonth/ChebyKAN.
[84] S. Benzoubeir, A. Hmamed, H. Qjidaa, Hypergeometric Laguerre moment for handwritten digit recognition, in: 2009 International Conference on Multimedia Computing and Systems, IEEE, 2009, pp. 449–453.
[85] Nelson Vieira, Bicomplex neural networks with hypergeometric activation functions, Adv. Appl. Clifford Algebr. 33 (2) (2023) 20.
[86] J. Arvesú, K. Driver, L.L. Littlejohn, Zeros of Jacobi and ultraspherical polynomials, Ramanujan J. (2021) 1–20.
[87] Wolfgang Schweizer, Orthogonal polynomials: General aspects, in: Special Functions in Physics with MATLAB, Springer International Publishing, Cham, ISBN: 978-3-030-64232-7, 2021, pp. 181–197, http://dx.doi.org/10.1007/978-3-030-64232-7_15.
[88] Christian Garbin, Xingquan Zhu, Oge Marques, Dropout vs. batch normalization: an empirical study of their impact to deep learning, Multimedia Tools Appl. 79 (19) (2020) 12777–12815.
[89] Susanna Lange, Kyle Helfrich, Qiang Ye, Batch normalization preconditioning for neural network training, 2022, arXiv:2108.01110 [cs.LG].
[90] Sergey Ioffe, Christian Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: International Conference on Machine Learning, pmlr, 2015, pp. 448–456.
[91] Johan Bjorck, Carla Gomes, Bart Selman, Kilian Q. Weinberger, Understanding batch normalization, 2018, arXiv:1806.02375 [cs.LG].
[92] Francesca Pitolli, A fractional B-spline collocation method for the numerical solution of fractional predator-prey models, Fractal Fract. 2 (1) (2018) 13.
[93] Djork-Arné Clevert, Thomas Unterthiner, Sepp Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), 2015, arXiv preprint arXiv:1511.07289.
[94] Alireza Afzal Aghaei, rKAN: Rational Kolmogorov-Arnold networks, 2024, arXiv preprint arXiv:2406.14495.
[95] Zhangyanbo, MLP-KAN, 2024, https://github.com/Zhangyanbo/MLP-KAN.
[96] Yann LeCun, The MNIST database of handwritten digits, 1998, http://yann.lecun.com/exdb/mnist/.
[97] Krizhevsky Alex, Learning multiple layers of features from tiny images, 2009, https://www.cs.toronto.edu/kriz/learning-features-2009-TR.pdf.
[98] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y. Ng, et al., Reading digits in natural images with unsupervised feature learning, in: NIPS Workshop on Deep Learning and Unsupervised Feature Learning, Vol. 2011, Granada, 2011, p. 4, 2.
[99] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al., Matching networks for one shot learning, Adv. Neural Inf. Process. Syst. 29 (2016).
[100] Maria-Elena Nilsback, Andrew Zisserman, Automated flower classification over a large number of classes, in: 2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing, IEEE, 2008, pp. 722–729.
[101] Brett Koonce, Brett Koonce, MobileNetV3, in: Convolutional Neural Networks with Swift for Tensorflow: Image Recognition and Dataset Categorization, Springer, 2021, pp. 125–144.
[102] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[103] Tayebeh Taheri, Alireza Afzal Aghaei, Kourosh Parand, Accelerating fractional PINNs using operational matrices of derivative, 2024, arXiv preprint arXiv:2401.14081.
[104] Hassan Dana Mazraeh, Kourosh Parand, GEPINN: An innovative hybrid method for a symbolic solution to the Lane-Emden type equation based on grammatical evolution and physics-informed neural networks, Astron. Comput. (2024) 100846.
[105] George Paul Horedt, Polytropes: Applications in Astrophysics and Related Fields, vol. 306, Springer Science & Business Media, 2004.
[106] A.H. Khater, R.S. Temsah, MM2474624 Hassan, A Chebyshev spectral collocation method for solving Burgers'-type equations, J. Comput. Appl. Math. 222 (2) (2008) 333–350.
[107] Tayebeh Taheri, Alireza Afzal Aghaei, Kourosh Parand, Bridging machine learning and weighted residual methods for delay differential equations of fractional order, Appl. Soft Comput. 149 (2023) 110936.
[108] Ali Nosrati Firoozsalari, Hassan Dana Mazraeh, Alireza Afzal Aghaei, Kourosh Parand, deepFDEnet: A novel neural network architecture for solving fractional differential equations, 2023, arXiv preprint arXiv:2309.07684.

Alireza Afzal Aghaei is a graduate student in Computer Science and an independent researcher who studies the intersection of machine learning and mathematical computation. His work focuses on developing innovative methods for solving differential and integral equations using physics-informed machine learning algorithms, particularly leveraging support vector machines and neural networks enhanced with orthogonal functions. Proficient in programming languages, mathematical software, and data science tools, he has authored several publications across diverse domains, ranging from computational physics to medical diagnosis using artificial intelligence.