symmetry

IMPACT FACTOR 2.2

CITESCORE 5.3

Article

Symmetry in the Algebra of Learning: Dual Numbers and the Jacobian in K-Nets

Agustin Solis-Winkler, J. Raymundo Marcial-Romero and J. A. Hernández-Servín

Special Issue

Symmetry/Asymmetry and Its Applications in Deep Learning and Artificial Intelligence Methods

Edited by

Prof. Dr. Eréndira Rendón-Lara and Prof. Dr. Rosa María Valdovinos-Rosas

MDPI

https://doi.org/10.3390/sym17081293

symmetry

MDPI

Article

Symmetry in the Algebra of Learning: Dual Numbers and the Jacobian in K-Nets

Agustin Solis-Winkler, J. Raymundo Marcial-Romero and J. A. Hernández-Servín *

Facultad de Ingeniería, Universidad Autónoma del Estado de México, Toluca 50110, Mexico; asolisw@uaemex.mx (A.S.W.); jrmarcialr@uaemex.mx (J.R.M.R.)
* Correspondence: joseph_servin@uaemex.mx

Abstract

The black-box nature of deep machine learning hinders the extraction of knowledge in science. To address this issue, a proposal for a neural network (k-net) based on the Kolmogorov–Arnold Representation Theorem is presented, pursuing to be an alternative to the traditional Multilayer Perceptron. In its core, the algorithmic nature of neural networks lies in the fundamental symmetry between forward-mode and reverse-mode accumulation techniques, both of which rely on the chain rule of partial derivatives. These methods are essential for computing gradients of functions, an operation that is at the core of the training process of neural networks. Automatic differentiation addresses the need for accurate and efficient calculation of derivative values in scientific computing; procedural programs are thus transformed into the computation of the required derivatives at the same numerical arguments. This work formalizes the algebraic structure of neural network computations by framing the training process within the domain of hyperdual numbers. Specifically, it defines a Kolmogorov–Arnold-inspired neural network (k-net) using dual numbers by extending the univariate functions and their compositions that appear in the representation theorem. This approach focuses on computation of the Jacobian and the ability to implement such procedures algorithmically, without sacrificing accuracy and mathematical rigor, while exploiting the inherent symmetry of the dual number formalism.

Keywords: symmetry; dual numbers; machine learning; k-nets; neural networks

Academic Editor: Zhengqiu Zhang

Received: 8 July 2025
Revised: 29 July 2025
Accepted: 4 August 2025
Published: 11 August 2025

Citation: Solis-Winkler, A.; Marcial-Romero, J.R.; Hernández-Servín, J.A. Symmetry in the Algebra of Learning: Dual Numbers and the Jacobian in K-Nets. *Symmetry* **2025**, *17*, 1293. https://doi.org/10.3390/sym17081293

1. Introduction

Traditional neural networks have demonstrated remarkable capabilities across various application fields, from image recognition to natural language processing and generation. The main building block of these networks is the Multilayer Perceptron (MLP), grounded on the Universal Approximation Theorem. According to this theorem, MLPs can approximate any continuous function given enough neurons in the hidden layer [1,2]. However, their approximation capabilities rely on the use of nonlinear activation functions to approximate arbitrary continuous functions, which complicates the interpretability of the model [3]. One of the main limitation of MLPs, particularly when deep architectures are used, is the lack of explainability. These models are often regarded as black boxes, due to the difficulty of interpreting the way they make decisions from given inputs [4]. This lack of transparency poses significant challenges in application domains such as medicine, security, and financial decision-making, where the need to understand the process behind a decision may be as important as the prediction itself.

The field of Explainable Artificial Intelligence aims to address this issue by developing methods that enable the interpretation of deep learning models [5]. However, the complexity of their architectures and the nonlinearity of their activation functions represent a challenge for this field. While current approaches, such as activation visualizations, input attribution techniques, and surrogate models provide partial insights, they often rely on heuristics that may not guarantee complete interpretability.

The newly proposed Kolmogorov–Arnold Networks (k-nets), or simply k-nets, based on the Kolmogorov–Arnold Representation Theorem, seeks to address this issue as an alternative to traditional Multilayer Perceptron [3]. Kolmogorov published the proof in 1957 [6], including Arnold's results [7], to solve the Hilbert's 13th problem proving whether a solution exists for all 7th-degree equations using continuous functions of two arguments (the variant for algebraic functions is still unresolved). These networks fundamentally employ forward and reverse accumulation techniques based on the chain rule of partial derivatives to compute function gradients, which are fundamental in the optimization process. Automatic differentiation provides accurate and efficient calculations of derivative values, transforming procedural programs into derivative computations at the same numerical arguments [8].

The formal algebraic framework, for addressing rigorously such differentiation problems, lies in the algebraic structure of dual numbers. This work aims to formally define k-nets in terms of dual numbers [9] by extending their function compositions into the dual number domain. This approach focuses on the Jacobian and the implementation of these procedures algorithmically without losing accuracy and mathematical rigor. In this context, networks based on the Kolmogorov–Arnold Representation Theorem have emerged as a promising alternative to enhance the explainability of neural models. This theorem states that any continuous function of multiple variables can be expressed as a combination of univariate functions, allowing for a more interpretable decomposition of the relationship between inputs and outputs. The ability to structure inference in a more explicit manner may provide advantages in terms of interpretability and decision transparency.

This article explores the use of k-nets in the context of dual numbers, examining how this combination can contribute to both computational efficiency and model explainability. Weights on k-nets are represented by univariate functions, typically by spline functions; in terms of parameter optimization, k-nets introduce learnable optimization functions instead of fixed linear weights, which could adapt to different data patterns [10]. Recently, the Kolmogorov–Arnold theorem (KAT) has gained increased attention thanks to the work of [3]. They suggest a performance increase in these networks by introducing learnable univariate functions [3]. From this point, new research efforts in the field had followed [10]. Among neural network architectures, the Multilayer Perceptron (MLP), a fully connected network that extends the original Rosenblatt Perceptron, is the building block of current deep learning architectures [3]. The MLP approximation of nonlinear functions is supported by the Universal Approximation Theorem [2]. However, its use has significant drawbacks, and it is difficult to interpret without the use of post-analysis tools [11].

The Kolmogorov–Arnold Networks, or k-nets, mentioned since 1987 by [12] by introducing the "Kolmogorov's Mapping Neural Network Existence Theorem" based on the work due to [13], establish that any continuous function can be represented by a neural network with two hidden layers, and are inspired by the Kolmogorov–Arnold Representation Theorem [6]. k-nets resemble an MLP because of a fully connected structure. However, instead of using fixed activation functions on each node, a k-net uses learnable activation functions on edges, replacing weights with learnable single variable functions. Each k-net node adds incoming values without adding any nonlinear function [3].

This paper is therefore organized as follows: the first part introduces Kolmogorov–Arnold Networks (k-nets) defined entirely on the Kolmogorov–Arnold Representation Theorem, as composition of linear combination of simple univariate functions. While the theorem guarantees the existence of such functions, their exact form is not specified and they are often difficult to interpret or apply in practice. Therefore, the first task is to select a suitable set of simple functions that can be used to represent and numerically approximate and implement the desired mappings. Section 2 formally defines k-nets and introduces the use of dual numbers to compute the Jacobian matrix of the network functions that is a fundamental component of the training of k-nets; that is, the method to optimize all parameters that define them. Section 4 is dedicated to an empirical evaluation of the theoretical proposal of the model, with a direct non-optimized implementation. The results are compared with other approaches inspired by the Kolmogorov–Arnold Representation Theorem, specifically the KAN model proposed by Liu et al. [3].

The contributions of this work are (i) to show the theoretical feasibility of building neural networks based on the mentioned representation theorem, and also the feasibility of implementing them, as an alternative to the traditional architecture by replacing the "activation functions" in a more flexible manner, despite current limitations in performance and efficiency that prevent their practical use yet; (ii) to introduce the algebraic expressivity of dual numbers in the training of neural networks, since it has been proven elsewhere to perform well in the calculations of the Jacobian to optimize the loss function; (iii) a practical implementation of the theoretical framework that shows the viability of the new proposal in practical applications; and finally, the possibility of practical use of splines in defining neural networks. Finally, the paper concludes with a discussion and conclusions derived from key findings.

2. Theoretical Framework

The original Kolmogorov–Arnold Representation Theorem states that if f is a multivariate continuous function on a bounded domain, then f can be rewritten as a finite composition of continuous functions of a single variable and the binary operation of addition. Being more specific, for a continuous function $f : [0,1]^n \to \mathbb{R}$, $f(\boldsymbol{x}) = f(x_1, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \phi_{qp}(x_p) \right)$. Here, $\phi_{qp} : [0,1] \to \mathbb{R}$ and $\Phi_q : \mathbb{R} \to \mathbb{R}$ are continuous real univariate functions. This means that the $(2n+1)$ Φ_q and the n ϕ_{qp} univariate functions are enough for an exact representation of an n-variate function [14,15]. As every other function can be written using univariate functions and sum, addition becomes the only true multivariate function [3]. Although the Kolmogorov–Arnold Representation Theorem (KAN) does not explicitly specify the form of the functions Φ_q and ϕ_{pq}, it establishes that they can be approximated through superpositions of a fixed form involving polynomials in a single variable and summation [6]. This property guarantees the existence of such functions and suggests that effective approximations can be constructed using polynomials. Despite the theoretical power of the Kolmogorov–Arnold theorem, its practical application in neural networks has been largely overlooked due to several limitations [10]. One key challenge is that the one-dimensional functions involved in the representation can be highly non-smooth or even fractal, making them difficult to learn in practice [3]. This spurious behavior led to the theorem being regarded as formally grounded but impractical for machine learning applications [16]. Additionally, compared to Multilayer Perceptrons (MLPs), Kolmogorov–Arnold Networks (k-nets) require numerous function superpositions, resulting in significantly higher computational complexity [10]. Early attempts to leverage the theorem for neural network implementations in the late 20th century were constrained by the limited computational power and lack of efficient optimization algorithms at the time. Sprecher's reformulation [13] proposed an alternative

representation of the Kolmogorov–Arnold theorem. He demonstrated that all inner functions ϕ_{qp} could be replaced by translations and scalings of a single function ψ, such that $\phi_{qp}(x_p) \approx \lambda^p \psi(x_p + \varepsilon q) + q$, where λ^p are constants independent of the function f, and ε is a small constant. Additionally, Lorentz [17] showed that all outer functions Φ_q can be replaced by a single function Φ, allowing the full representation to be rewritten as in (1):

$$f(x_1, \ldots, x_n) = \sum_{q=0}^{2n} \Phi\left(\sum_{p=1}^{n} \lambda^{qp} \psi(x_p + \varepsilon q) \right). \tag{1}$$

Sprecher's reformulation, in addition to Lorentz's work, laid the groundwork for the result later presented by Hecht-Nielsen [12], known as the "Kolmogorov Mapping Neural Network Existence Theorem". It states that any continuous mapping $f : [0,1]^n \to \mathbb{R}^m$, $f(x) = y$ can be implemented by a neural network structured as follows: $z_q = \sum_{p=1}^{n} \lambda^q \psi(x_p + \epsilon q) + q$, $q = 0, ..., 2n$; $y_i = \sum_{q=0}^{2n} \Phi_i(z_q)$, $i = 1, \ldots, m$. Here, the real constants λ^q and the continuous real monotonic increasing function ψ are independent of f, constant ϵ is an arbitrary rational number, $0 < \epsilon$. The function ψ can be chosen to satisfy a Lipschitz condition, and the outer functions Φ_i depend on f and ϵ. Each hidden unit z_q is a weighted sum of translated inputs passed through ψ, forming a hidden layer of $2n + 1$ units. Each output y_i is a sum of shared functional transformations applied to simple univariate compositions, Sprecher introduces a modification to the original on requirements, obtaining a representation that no longer requires the final constant q [18]. This results in the representation in Equation (1), which will serve as the foundation for experimental evaluation of the model. Although Equation (1) is a suitable starting point for defining k-nets, it relies on specific assumptions that, while potentially valid in the long term, may limit the exploration of other alternative ways for choosing inner functions. In fact, Equation (1) is the one implemented in the present study and compared against other proposals in the literature. The formal definition of k-nets is provided in the next section, adopting a more general approach.

2.1. Kolmogorov–Arnold Neural Networks (K-Nets)

All considerations discussed in previous sections are accounted for in the implementation of k-nets. However, to maintain generality, we define k-nets without imposing any specific approximation scheme, thereby preserving the possibility to select the functions Φ's and ϕ in some other way. For instance, in [15], authors propose an algorithm for constructing the inner functions in a way that preserves the spirit of the original proof of the theorem by Arnold and Kolmogorov. While the authors claim that the original construction is not suitable for algorithmic implementation, they demonstrate that it is possible to define the ϕ_{pq} inner functions to be Lipschitz continuous, a property that is often desirable in the design of neural network activation functions [1] in the context of deep learning algorithms.

The Kolmogorov–Arnold Representation Theorem expresses a function $f : \mathbb{R}^n \to \mathbb{R}$ in terms of functions Φ_q, ϕ_{qp} of the form $f = \sum_q \Phi_q \sum_p \phi_{qp}(x_p)$. The idea is to define a neural network based on such representation. Let $\zeta_{kq} = (\phi_{kq1}, ..., \phi_{kqp}) : \mathbb{R}^n \to \mathbb{R}^p$ be a function $\zeta_k = \zeta \circ \Phi_{kq} \circ \zeta \circ \zeta_{kq} : \mathbb{R}^n \to \mathbb{R}$ for every k. In this composition diagram (Figure 1), ζ simply represents the sum over the last index p and q. Index k is being used to allow the composition diagram to obtain a single output $\zeta_k : \mathbb{R}^n \to \mathbb{R}$. The function ζ_k corresponds to a neural network with n inputs and one output, two hidden layers, clearly represented by Φ_q (or one single Φ) and ϕ_{qp} (or one single ϕ times some constants λ^{qp}); this is one of the main differences with other neural networks.

$$\begin{array}{ccccc} (\phi_{kq1}(x_1),..,\phi_{kqp}(x_p)) & \xrightarrow{\zeta} & \sum_p \phi_{kqp} & \xrightarrow{\Phi_{kq}} & \Phi_{kq}\big(\sum_p \phi_{kqp}(x_p)\big) \\ \zeta_{kq}\uparrow & & & & \downarrow\zeta \\ (x_1,...,x_n) & & & & \zeta_k(x_1,...,x_n) = \sum_q \Phi_{kq}\big(\sum_p \phi_{kqp}(x_p)\big) \end{array}$$

Figure 1. Composition diagram for a k-net.

Thus, for every k, we have an output vector $Z_k = (\zeta_{k1}, \ldots, \zeta_{km})$, then we can define a neural network as follows $Z = Z_L \circ \cdots \circ Z_0$. The mapping Z is a composition of $Z_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_{i+1}}$ maps in such a way that domain of Z_i must coincide with the codomain of Z_{i-1}; some authors in the literature called property model layer structure; this is represented by a vector of integers $[d_0, \ldots, d_L]$, where L is the number of layers that define the network. In our case, d_0 is the number n, but since the number of layers is changing, it is customary to use indices.

Definition 1. *Let $f : [0,1]^n \to \mathbb{R}$ be a continuous function for which exists univariate function $\Phi : [0,1] \to \mathbb{R}$, $\phi_{qp} : [0,1] \to \mathbb{R}$ such that $f = \sum_q \Phi_q(\sum_p \phi(x_p))$, $1 \leq q \leq 2n+1$, $1 \leq p \leq n$, then a k-net is a composition mapping $Z = Z_0 \circ \cdots \circ Z_L$, where $Z_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_{i+1}}$, and $d_{i-1} = d_i$. The coordinates of each $Z_i = (\zeta_1, \ldots, \zeta_{d_i})$ are $\zeta_k = \zeta \circ \Phi_{kq} \circ \zeta \circ \zeta_{kq} : \mathbb{R}^{d_i} \to \mathbb{R}$, as defined in the diagram of Figure 1. The model layer structure is the vector $[d_0, \ldots, d_L]$ of dimensions of the domain of every Z_i for each layer.*

A neural network would not be complete without its training process. For that, we introduce the concept of dual numbers and how it can be integrated in our formulation of k-nets. The use of dual numbers is not new: it is within the area of artificial intelligence called automatic differentiation (AD), and it has been proven to be a promising technique to solve differential equations [19]. However, the algebraic representation as well as their implementation remains a challenge; firstly, because it is difficult to represent any real function in the dual domain, and secondly, it computationally lies in the realm of meta-programming that it is not always possible to achieve in any language. Dual numbers are used in the implementation of the so-called forward mode of AD [20]. The key idea relies on the fact that dual numbers can encode not only the value of a function but also its derivative, without the need for symbolic manipulation or finite difference approximations. Before introducing dual numbers, let us discuss the need for efficient calculation of gradient of functions, necessary for data training in a network. Data training consists essentially in finding the local or global minimum of a map $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$; that is, find θ^* such that $\mathcal{L}(\theta^*) \leq \mathcal{L}(\theta)$ for all $\theta \in \mathbb{R}^n$. Thus, training a k-net $Z(x)$ as in Definition 1 means to find a minimum set of parameters, denoted by $\{\theta_i\}$, which are defined by the functions Φ_q and ϕ_{qp} from the definition of Z. To do so, we assume some initial experimental dataset $S = \{x_i, y_i\}$; to fit a neural k-net $Z(x, \theta) : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}$, where m is well-defined and depends on the number n, which is the number of functions Φ's and ϕ's used in defining Z. Thus, the method of gradient descent to solve the minimization problem $\min_\theta \mathcal{L}_S(Z(x, \theta, y))$, the preferred method to train neural networks, is to update θ of the following form $\theta = \theta - \eta \nabla_\theta \mathcal{L}$. Just to clarify the training, consider for example, the mean square method $\mathcal{L}_S = (1/2m) \sum_i (Z(x_i, \theta) - y_i)^2$, and therefore, the $\nabla_\theta \mathcal{L}_S = \frac{1}{m} \sum_i (Z(x_i, \theta) - y_i)$ $\nabla_\theta Z(x_i, \theta)$ from which we can infer that training a k-net is translated into the direct calculation of the gradient of Z mapping itself. The above discussion will be elaborated in more detail in the results section, where some other functions are considered for training. Therefore, one of the main purposes of this paper is not only to define a neural network but to introduce the mechanism of calculating the gradient of k-nets by using dual numbers in forward-mode automatic differentiation, since it is required by the method of gradient

descent; this is particularly challenging since it has to be repeated thousands of times. Reverse mode cannot be directly implemented using dual numbers, as they do not have a structure that allows backpropagation.

2.2. The Jacobian of Z K-Nets

Let $Z : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function; thus, when evaluating Z on a dual number $t = x + \varepsilon$, using the Taylor expansion [21]

$$Z(t) = Z(x) + \sum_{j=1}^{d} \frac{\partial Z(x)}{\partial t_j} + \frac{1}{2!} \sum_{j=1}^{d} \sum_{k=1}^{d} \frac{\partial^2 Z(x)}{\partial t_j \partial t_k} (t_j - x_j)(t_k - x_k) + \cdots \tag{2}$$

and the property $\varepsilon^2 = 0$, we obtain $Z(x + \varepsilon) = Z(x) + J_Z(x)\varepsilon$, where $J_Z(x)$ denotes the Jacobian matrix of Z; since we are assuming that image of mapping Z has dimension one, it would be more appropriate to write ∇Z because the second term in Equation (2) is precisely the gradient of Z; in other words, $J_Z(x) = \nabla Z \in \mathbb{R}^{d \times 1}$. Let us recall that the third term in Equation (2) is known as the Hessian matrix. This result indicates that the dual component of $Z(x + \varepsilon)$ directly contains the Jacobian or gradient (for dimension one) $J_Z(x)$. Therefore, by overloading arithmetic operations to work with dual numbers, one can propagate derivatives alongside function evaluations in a computational graph, enabling efficient and exact gradient computation [19,20]. The above procedure shows a very intuitive way of looking at dual numbers and its relation to its first derivatives in several variables. However, a k-net Z is multidimensional, thus extending t to a vector; the Taylor expansion in Equation (2) contains also terms of second degree and above, and the simple substitution $t = x + \varepsilon$ will be a first-degree approximation only. To consider terms of more degrees or derivatives of higher degree, for example, second degree in Equation (2), we must define hyperdual numbers as an algebra of polynomials [8].

Definition 2. *Let $d \in \mathbb{N}$, $N \geq 1$ be and consider the algebra $\mathbb{R}[\varepsilon_1, \ldots, \varepsilon_d]/I_N$, where $I_N = \langle \{\varepsilon_1^{k_1} \cdots \varepsilon_d^{k_d} \mid k_i \in \mathbb{N} \text{ with } \sum_i k_i > N\} \rangle$ is the ideal generated by all monomials of order $N + 1$. Then,*

$$\mathbb{D}^d = \frac{\mathbb{R}[\varepsilon_1, \ldots, \varepsilon_d]}{I_N} = \left\{ \sum_{k_1 + \cdots + k_d = 0}^{k_1 + \cdots + k_d = N} a_{k_1 \cdots k_d} \varepsilon_1^{k_1}, \ldots, \varepsilon_d^{k_d} \right\} \tag{3}$$

is the algebra of dual numbers (3). The degree of a monomial of type $\varepsilon_1^{k_1} \cdots \varepsilon_d^{k_d}$ is defined as $\sum_i k_i$, denoted by deg.

In order to use Definition 2, it is assumed that the mappings Z are polynomials; this appears to be rather restrictive, however, as we show in the results. It is the way that this new proposal of neural network may result in a practical alternative. In the next section, it is shown that a practical k-net can be accomplished by assuming that $\phi_{kq}(x) = b(x) + \sum_i c_{kqi} B_i(x)$, where $b(x) = (x/1 + e^{-x})$ and B_i are splines of some degree. To lift $b(x)$ to dual numbers, we must calculate $\hat{b}(x + \varepsilon) = b(x) + b'(x)\varepsilon$; this can always be achieved using Definition 2 and Equation (2). Now, since B_i are splines and therefore polynomials, we can assume that $Z(x)$ is a polynomial in $\mathbb{R}[x_1, \ldots, x_d]$. From the above discussion, we can establish a proposition. The proof is incomplete; considering the purpose and the length of the paper, we sketch the proof using an example to show why the proposition is true. To sketch the proof, we make use of one of the parabolic singularities studied by Arnold [22], which is $Z(x, \varepsilon) = Z(x_1, x_2, x_3) = x_1^3 + x_2^3 + x_3^3 + a x_1 x_2 x_3$. The idea of the proof is to show that $Z(x_1 + \varepsilon_1, x_2 + \varepsilon_2, x_3 + \varepsilon_3)$ in the ring $\mathbb{D}^3$; in other words, the coefficients of $(Z(x, \varepsilon) \bmod I_N)$—the remainder of $Z(x + \varepsilon)$—for some N, are precisely the partial derivatives of $Z(x)$.

Proposition 1. *If* $Z \in \mathbb{R}[x_1, ..., x_d]$, *then* $Z(x+\varepsilon) \in \frac{\mathbb{R}[x_1,...,x_d]}{I_N}$ *is the truncated polynomial of the Taylor series expansion of* $Z_N(t+\varepsilon)$, *Equation (2), up to N terms. In other words,* $\hat{Z}(x+\varepsilon) = Z(x+\varepsilon) \bmod I_N = Z_N(x+\varepsilon)$. *In particular, if* $N = 2$, *then* $(Z(x+\varepsilon) \bmod I_2)$ *becomes*

$$\hat{Z}(x+\varepsilon) = Z(x) + \sum_{j=1}^{d} \frac{\partial Z(x)}{\partial x_j}\varepsilon_j + \frac{1}{2!}\sum_{j=1}^{d}\sum_{k=1}^{d} \frac{\partial^2 X}{\partial x_j \partial x_k} \cdot \varepsilon_j \varepsilon_k \tag{4}$$

Proof. The proof of Proposition 1 should be straightforward from Definition 2 and Equation (2). Nevertheless, let us assume that we are interested in evaluating the following expression $Z(x_1, x_2, x_3) = x_1^3 + x_2^3 + x_3^3 + a x_1 x_2 x_3$ in the ring of dual numbers. The polynomial is taken from [22], but any polynomial should work just fine. By using ∂^2_{ij} as a shorthand for $\frac{\partial^2}{\partial x_i \partial x_j}$, we know, from direct calculation of Z, that Jacobian $(J_Z)_i = \partial_i Z$ and the Hessian $(H_Z)_{ij} = \partial^2_{ij} Z$ are given by (5)

$$J_Z(x) = \begin{bmatrix} 3x_1^2 + a x_2 x_3 \\ 3x_2^2 + a x_1 x_3 \\ 3x_3^2 + a x_1 x_2 \end{bmatrix}, \quad H_Z = \begin{bmatrix} 6x_1 & a x_3 & a x_2 \\ a x_3 & 6x_2 & a x_1 \\ a x_2 & a x_1 & 6x_3 \end{bmatrix}. \tag{5}$$

After expanding quadratic and cubic polynomials, and grouping terms in ascending order by total degree ($\deg(\varepsilon_1^{k_1}\varepsilon_2^{k_2}\varepsilon_3^{k_3} = k_1 + k_2 + k_3$), we have that $Z(x+e) = [x_1^3 + x_2^3 + x_3^3 + a x_1 x_2 x_3] + [(3x_1^2 + a x_2 x_3) \cdot \varepsilon_1 + (3x_2^2 + a x_1 x_3) \cdot \varepsilon_2 + (3x_3^2 + a x_1 x_2) \cdot \varepsilon_3] + [3x_1\varepsilon_1^2 + 3x_2\varepsilon_2^2 + 3x_3\varepsilon_3^2 + a x_3\varepsilon_1\varepsilon_2 + a x_2\varepsilon_1\varepsilon_3 + a x_1\varepsilon_2\varepsilon_3] + [\varepsilon_1^3 + \varepsilon_2^3 + \varepsilon_3^3 + a\varepsilon_1\varepsilon_2\varepsilon_3]$. By making $R_Z = \varepsilon_1^3 + \varepsilon_2^3 + \varepsilon_3^3 + a\varepsilon_1\varepsilon_2\varepsilon_3$, homogeneous polynomial of degree 3, we see that the homogeneous polynomial of degree 1 is $Z(x)$, whereas the homogeneous polynomials of degree 1 and 2 correspond to $[\partial_1 Z \cdot \varepsilon_1 + \partial_2 Z \cdot \varepsilon_2 + \partial_3 Z \cdot \varepsilon_3]$, $[\frac{1}{2}\sum_i \partial_i Z \cdot \varepsilon_i + \sum_i \sum_j \partial_{ij} Z \cdot \varepsilon_{ij}]$, respectively; then, $Z(x+\epsilon)$ can be rewritten as

$$Z(x+\varepsilon) \quad = \quad Z(x) + \sum_{i=1}^{3} \partial_i Z \cdot \varepsilon_i + \frac{1}{2!}\sum_{i=1}^{3}\sum_{j=1}^{3} \partial_{ij} \cdot \varepsilon_{ij} + R_Z \tag{6}$$

The coefficient of the third term in variable ε_1 corresponds to first entry in $\partial_1 Z$, the coefficient of the fourth and fifth term correspond to second ($\partial_2 Z$) and third ($\partial_3 Z$) entry in matrix J_Z, respectively, and this will be true for any polynomial in any number of variables. Analogously, the coefficients of monomials of degree 2 in Equation (6) for Z correspond to every entry in the Hessian matrix for i, j; the residual homogeneous polynomial R_Z on variables $\{\varepsilon_i\}$ contain monomials of total degree greater than 2; that is, $\deg(e_1^{k_1} e_2^{k_1} e_3^{k_3}) = k_1 + k_2 + k_3$ with $k_1, k_2 \in \{1, 2\}$ is either 3 or greater in general. From Definition 2, the ideal I_2 is the set of all monomials $\{\varepsilon_1^{k_1}\varepsilon_2^{k_2}\varepsilon_3^{k_3} | k_1 + k_2 + k_3 \geq 3\}$. Since the goal is to look at Z in the domain of hyperdual numbers, we must consider Z module the ideal I_2; that is, Z in the quotient $\mathbb{D}^2 = \mathbb{R}[\varepsilon_1, \varepsilon_2, \varepsilon_3]/I_2$ corresponds to $\hat{Z} = Z(x) + \sum_{j=1}^{3} \partial_i Z \cdot \varepsilon_i + \frac{1}{2!}\sum_{j=1}^{3}\sum_{k=1}^{3} \partial_{jk} Z \cdot \varepsilon_j \varepsilon_k$. This is true because $(Z \bmod I_2)$ (the remainder) is equivalent to making $\varepsilon_1^{k_1}\varepsilon_2^{k_2}\varepsilon_3^{k_3} = 0$ as long as $k_1 + k_2 + k_3 \geq 3$, which is the case for all monomial of R_Z; therefore, $R_Z = 0$. By comparing $\hat{Z}$ (4) and Taylor expansion of Z around $t + \varepsilon$, we see that the coefficients of Z as polynomial on variables $\{\varepsilon_i\}$ correspond precisely to the derivatives of the mapping Z; thus, the dual part of a polynomial function always carries the derivative of the function. □

In contrast to the more traditional numerical methods based on finite differences that are inherently prone to rounding or truncation errors, auto-differentiation is theoretically exact, and computationally inexpensive, compared to symbolic algorithms, making use

of the fact that all computer calculations are executed as a series of elementary arithmetic operations and elementary functions to compute derivatives of arbitrary order by applying the chain rule.

2.3. A Practical K-Net

Based on the foundational work of Kolmogorov, Sprecher, Hecht-Nielsen, and Lorentz, a neural network architecture that explicitly reflects the structure of the representation theorem is proposed, where each layer of the network follows Definition 1 and the Hecht-Nielsen model. With a more flexible combination of the coefficients λ^{qp} and selecting a ψ function with enough internal variability, it is possible to simplify giving the following approximation (7):

$$\lambda^{qp}\psi(x_p + \epsilon p) \approx \alpha_{qp}\phi(x_p). \tag{7}$$

Sprecher [18] also makes reference to the work of Kúrková, where functions ψ and Φ, are approximated with sums of sigmoidal functions. Following Liu's [3] proposal for the ϕ functions, grade 3 B-splines with learnable coefficients are used, and ReLU activation with learnable parameters is used for the Φ functions. An inner layer (8), based on $2n + 1$ units, which computes for each unit $q = 0, \ldots, 2n$:

$$z_q = \sum_{p=1}^{n} \alpha_{qp}\, \phi_{qp}(x_p), \tag{8}$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is a cubic B-spline with learnable coefficients specific to the pair (qp), and $\alpha_{qp} \in \mathbb{R}$ are trainable parameters. The inner layer is implemented as `InnerLayer`, as shown in Figure 2. An outer layer (9) that maps the inner functions z_q to the output y_i for $i = 1, \ldots, m$:

$$y_i = \sum_{q=0}^{2n} w_{iq}\Phi_i(z_q), \tag{9}$$

where each w_{iq} is a trainable scalar weight, and $\Phi_i : \mathbb{R} \to \mathbb{R}$ is a fixed univariate nonlinear activation function such as ReLU or sigmoid. These weights determine the contribution of each transformed inner unit z_q to the output y_i. The outer layer module is shown in Figure 2, labeled as `OuterLayer`.

These two layers are grouped into a composite module called `FullLayer`, which realizes a full Kolmogorov–Arnold representation, yielding a single scalar output. When multiple outputs are needed, multiple instances of `FullLayer` are used in parallel, each receiving the same input vector $\boldsymbol{x}$. Figure 2 shows the organization of a full layer. For a single layer k-net, $(\zeta_1(\boldsymbol{x}), \ldots, \zeta_m(\boldsymbol{x}))$ is the output of the model. Model layer structure is represented by an integer array $[d_0, d_1, \ldots, d_L]$, where $d_0 = n$ is the number of inputs to the model, $d_L = m$ is the output dimension, and each layer is defined by (d_{l-1}, d_l), where d_{l-1} is the number of inputs to the layer l, and d_l is the number of outputs. The layer then will have d_l blocks, each one combining an inner layer followed by an outer layer, realizing a Kolmogorov–Arnold representation. Making the network deeper is achieved by stacking several layers with the `FullLayer` module where the input of each subsequent layer is taken from the output of the previous one. This hierarchical composition helps to preserve the constructive approximation capabilities of the architecture.

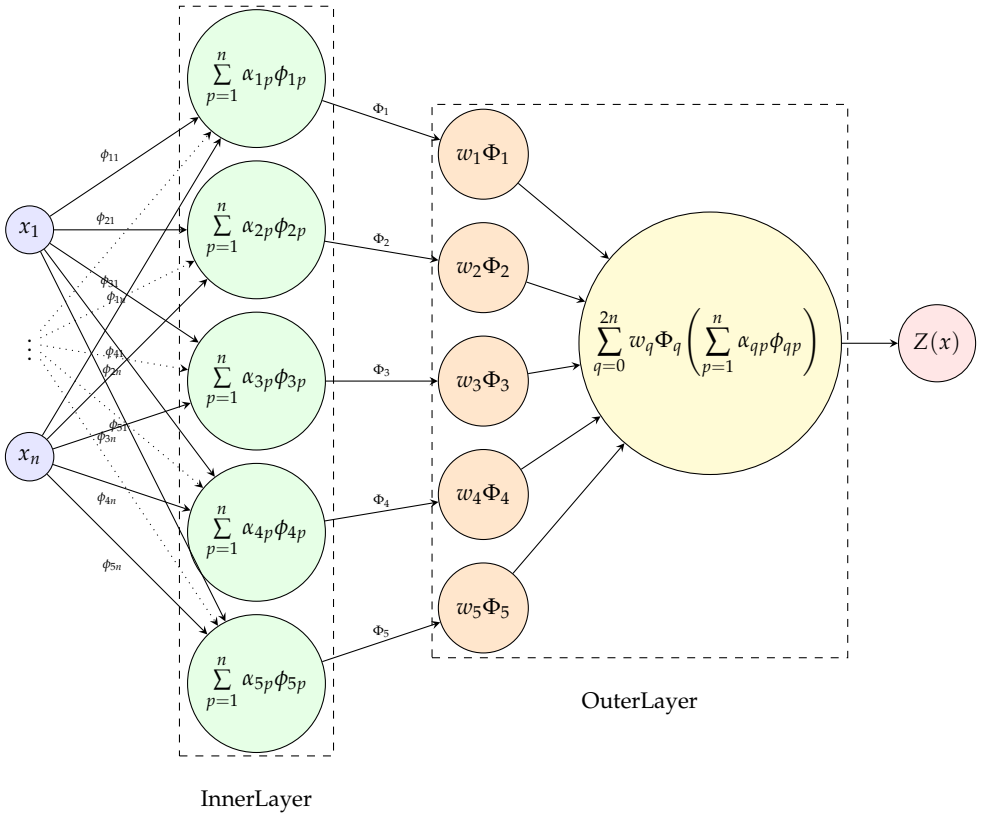


Figure 2. Kolmogorov–Arnold representation implemented as a combination of inner and outer layers.

3. Methods

To validate the proposed framework, the steps as shown below are followed:

1. Define k-nets in the dual number and automatic differentiation framework.
2. Extend function compositions to dual numbers, ensuring mathematical consistency.
3. Implement gradient-based optimization using dual numbers to enhance computational efficiency.
4. Compare performance and interpretability against Liu's Kolmogorov–Arnold-inspired proposal KAN [3].

The key variables of study include model accuracy (evaluating predictive performance), gradient computation efficiency (measuring computational cost in training), and interpretability (assessing the transparency of the learned representation). The other technical consideration for the implementation is the fact that Bernstein polynomials can be evaluated over dual numbers. Samanci [23] introduced a formulation of Bernstein polynomials defined over the ring of dual numbers $\mathbb{D} = \mathbb{R}[\varepsilon]/\langle\varepsilon^2\rangle$, enabling the simultaneous evaluation of a function and its derivative via forward-mode automatic differentiation. Note that from Definition 2, the ideal $I_1 = \{\varepsilon_1^{k_1} | k_1 > 1\}$, which corresponds to polynomials of type Equation (2) modulus the ideal $\langle\varepsilon^2\rangle$. The dual Bernstein basis functions are defined as $B_{in}(x + y\varepsilon) = \binom{n}{i}(x + y\varepsilon)^i(1 - x - y\varepsilon)^{n-i}$, which expands to $B_{in}(x + y\varepsilon) = B_{in}(x) + B'_{in}(x)\, y\, \varepsilon$, where $B_{in}(x)$ is the classical Bernstein basis and $B'_{in}(x)$ its derivative. This formulation allows for differentiability-aware modeling using Bernstein polynomials, and serves as a foundation for our generalization of B-spline basis functions to dual inputs. Following the extension of Bernstein polynomials to dual variables as presented by Samanci [23], it is possible to generalize B-spline basis functions to operate over the ring of dual numbers $\mathbb{D} = \mathbb{R}[\varepsilon]/(\varepsilon^2)$. This extension enables forward-mode automatic differentiation by evaluating both the value and the derivative of a B-spline function

in a single computation. Let us recall that a B-spline of degree k is defined recursively applying (10) and (11):

$$B_{i0}(x) = \begin{cases} 1 & \text{if } t_i \leq x < t_{i+1}, \\ 0 & \text{otherwise,} \end{cases} \tag{10}$$

$$B_{ik}(x) = \frac{x - t_i}{t_{i+k} - t_i} B_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x), \tag{11}$$

where $\{t_i\}$ is a non-decreasing knot sequence; to extend B-splines to dual inputs, let $x = a + b\varepsilon \in \mathbb{D}$. Since each recursive step involves affine transformations and multiplication operations that are well-defined in $\mathbb{D}$, the recurrence relation naturally extends to the dual setting. Thus, the dual B-spline basis $B_{ik}(a + b\varepsilon) \in \mathbb{D}$ provides both the function value and its derivative with respect to x at a. This extension preserves the locality, partition of unity, and continuity properties of the B-spline basis, while enriching it with derivative information, making it suitable for applications in automatic differentiation and learning architectures involving differentiable programming. Since Bernstein polynomials are a special case of B-splines with fixed degree and uniform knot vector on $[0, 1]$, Samanci's construction can be viewed as the zero-degree instance of this more general dual B-spline framework.

3.1. Training Procedure

In this implementation, the internal univariate functions ϕ_{qp} of `InnerLayer` (Figure 2) are represented using cubic B-splines, as proposed by Liu [3]. These functions are defined by a fixed set of knots and allow flexible approximation of continuous transformations on each input coordinate. Each ϕ_{qp} is represented as a linear combination of six basis B-splines (corresponding to the number of internal basis functions for degree 3 and 10 knots). For the external layer `OuterLayer`, each function Φ_q is implemented using a nonlinear activation function, such as ReLU, sigmoid or the identity. These functions are applied after the summation over the internal spline-transformed inputs and provide the necessary nonlinearity for function approximation. To enable gradient-based training, a forward-mode automatic differentiation strategy based on dual numbers was implemented, considering the work of [23]. Each scalar or tensor value in the network is extended to a dual number of the form $a + b\varepsilon$, where $\varepsilon^2 = 0$. This allows the network to compute derivatives of the loss with respect to each model parameter directly by evaluating the dual component of the forward pass. This formulation preserves the interpretable structure of the Kolmogorov–Arnold representation by explicitly modeling each intermediate transformation as a sum of univariate functions. The inner functions $\phi(x_p)$ are applied independently to each input variable, and the summation across p introduces controlled linear mixing. The outer layer aggregates the resulting activations via nonlinear univariate mappings, enabling a flexible yet interpretable composition. The training of the Kolmogorov–Arnold Network exploits the modularity and interpretability of its architecture. Let $D = \{(x^{(i)}, y^{(i)})\}$, $i = 1 \ldots N$ be a dataset of N input-output pairs, where $x^{(i)} \in \mathbb{R}^n$ and $y^{(i)} \in \mathbb{R}^m$. Each forward pass computes the hidden activations $z_q^{(i)}$ and output predictions $\hat{y}^{(i)}$ using the defined inner and outer layers. The prediction loss is quantified using a loss function $\mathcal{L}$, typically cross-entropy (12) for classification or mean squared error (MSE) (13) for regression:

- Cross-entropy. Measures the difference between two probability distributions; normally, the true labels on the dataset and the predicted value produced by the classifier.

$$\mathcal{L}_{CE}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{m} y_j^{(i)} log\, \hat{y}_j^{(i)} \tag{12}$$

- Mean Squared Error. On regression, measures the average squared difference between the predicted value and the actual target.

$$\mathcal{L}_{MSE}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} \| y^{(i)} - \hat{y}^{(i)} \|^2 \tag{13}$$

For each training epoch:

1. Compute $\hat{y}^{(i)}$ for each input $x^{(i)}$ via the model.
2. Use dual numbers to compute the forward-mode derivatives of $\mathcal{L}$ regarding all learnable parameters (B-spline coefficients and weights λ^{qp}).
3. Update parameters using gradient descent as in (14):

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}, \tag{14}$$

where $\eta > 0$ is the learning rate.

This training procedure is implemented in Python 3 using custom modules for dual number arithmetic, spline evaluation, and layer composition. Metrics such as loss and accuracy for both training and test datasets are recorded and visualized per epoch to monitor convergence.

3.2. Computational Complexity of the K-Net Model

The computational cost of training a Kolmogorov–Arnold Network (k-net) model using forward-mode automatic differentiation grows significantly with the architecture size. Each weight update requires a full forward pass with dual numbers, which makes the number of training steps a critical factor. The operation count can be estimated as follows: for each `FullLayer` in the network, the number of dual evaluations required depends on P—the number of inputs to the layer, Q—the number of outputs (i.e., number of full KA blocks), K—the number of B-spline coefficients per univariate basis function ϕ_{qp} (i.e., number of basis functions, which depends on the knot vector length as $K = m - d - 1$, where m is the number of knots and d is the spline degree), and B—the number of training samples in the batch. Each internal function ϕ_{qp} contributes $(2P + 1) \cdot P \cdot K$ operations, and each output function Φ_q contributes $(2P + 1)$ operations. Therefore, the total number of dual forward evaluations per layer is, according to (15),

$$\text{Ops}_{\text{layer}} = B \cdot Q \cdot [(2P + 1) \cdot P \cdot K + (2P + 1)] \tag{15}$$

For a full network with L layers defined by dimensions `layer_dims` $= [d_0, d_1, \ldots, d_L]$, the total number of operations per epoch is

$$\text{Ops}_{\text{total}} = B \cdot \sum_{\ell=1}^{L} d_\ell \cdot [(2d_{\ell-1} + 1) \cdot d_{\ell-1} \cdot K + (2d_{\ell-1} + 1)] \tag{16}$$

4. Results and Discussion

To evaluate the performance of the k-net architecture, experiments were conducted on regression and classification tasks. Regression was tested training two different models on the Diabetes dataset, and for classification, two distinct models were trained on the Iris dataset. Table 1 presents the model configurations used for the experiments, including model name and estimated operation counts per epoch for each configuration. Total number of operations is directly related with the dimensions of the input, and is based on Equation (16). As explained in Section 3.1, grade 3 B-splines were used ($d = 3$), with ten

knots ($m = 10$), resulting in six B-spline coefficients (i.e., $K = 6$), as explained in Section 3.2, and full-batch training.

Table 1. Number of dual forward evaluations per epoch for each k-net model configuration.

Task	Layer Dims	Model Name	Batch Size	Total Ops per Epoch
Regression	[10, 1]	k-net-1r	356	456,036
Regression	[10, 2, 1]	k-net-2r	356	935,212
Classification	[4, 2, 3]	k-net-1c	120	77,400
Classification	[4, 3, 3, 3]	k-net-2c	120	176,760

For regression tasks, the mean squared error (MSE), as defined in Equation (13), is used as the loss function. To report a more interpretable performance metric, the root mean squared error (RMSE) in Equation (17) is used. RMSE is calculated as the square root of the MSE:

$$\text{RMSE} = \sqrt{\mathcal{L}_{\text{MSE}}}. \tag{17}$$

This value reflects the average magnitude of prediction error in the original target scale, and is particularly useful for comparing models across datasets or tasks.

For classification tasks, the softmax function shown in Equation (18) is applied to the model output vector $\hat{\boldsymbol{y}}^{(i)} \in \mathbb{R}^m$ (where m is the output dimension) to obtain class probabilities:

$$\text{softmax}(\hat{y}_j^{(i)}) = \frac{\exp(\hat{y}_j^{(i)})}{\sum_{k=1}^{m} \exp(\hat{y}_k^{(i)})}. \tag{18}$$

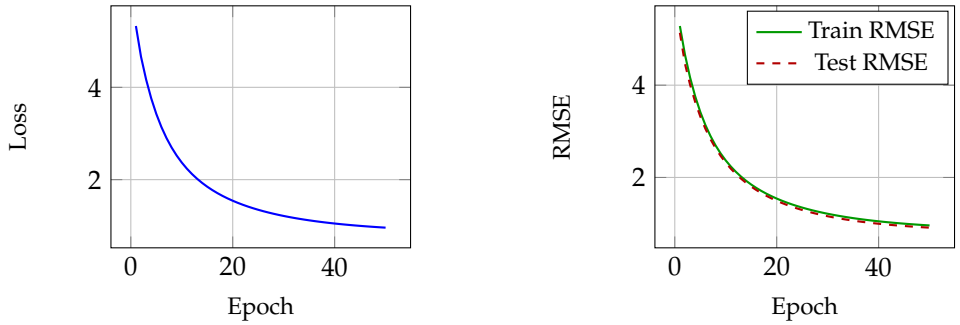
The loss function used is the categorical cross-entropy, defined for one-hot encoded ground truth $\boldsymbol{y}^{(i)}$ as defined in Equation (12). To evaluate classification performance, we use accuracy as in Equation (19), computed as the proportion of correct predictions:

$$\text{accuracy} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\left[\arg\max_j \hat{p}_j^{(i)} = \arg\max_j y_j^{(i)}\right], \tag{19}$$

where $\mathbb{I}[\cdot]$ is the indicator function.

4.1. Regressions

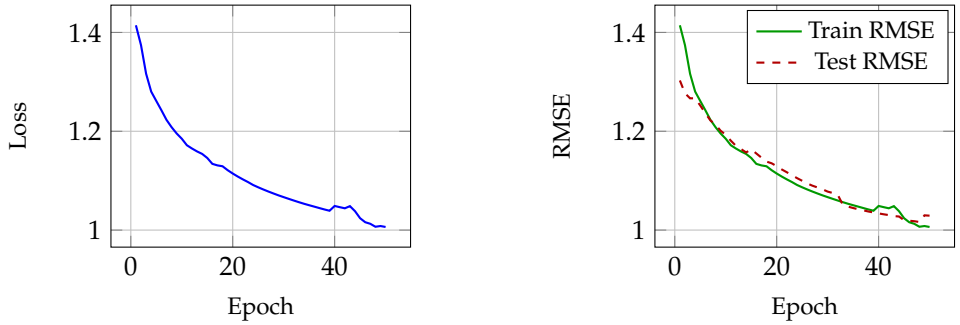
The Diabetes dataset consists of 442 samples with 10 input features and a single continuous target. Two model configurations were tested: A single-layer k-net with architecture $[10, 1]$, whose performance is shown in Figure 3, and a two-layer k-net with architecture $[10, 2, 1]$, the performance of which can be observed in Figure 4. Loss for the first architecture is shown in Figure 3a and the mean squared error (RMSE) over epochs for this single layer k-net, equivalent to the KAT can be seen in Figure 3b. As can be observed, the model learns over epochs, loss decreases up to 0.960 as well as the training RMSE, and RMSE over the test set goes down up to 0.918. For the second configuration—a two layer k-net— Figure 4a shows loss and Figure 4b illustrates RMSE. Notice the increase in the loss and RMSE around epoch number 40. This model requires twice the number of evaluations compared to the single layer, as shown in Table 1, requiring almost four times more training time, and its results are less satisfactory, indicating that a deeper k-net does not necessarily imply better generalization.



(**a**) Training loss for `[10, 1]` (**b**) RMSE for `[10, 1]`

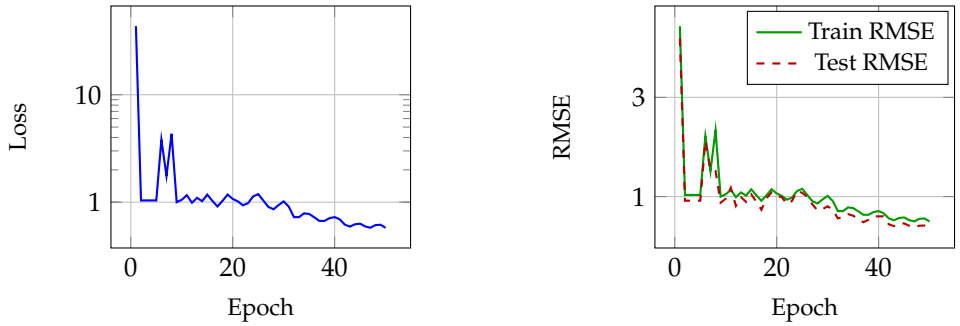
Figure 3. Performance on the Diabetes dataset using k-net model with architecture [10, 1].

The configuration used was [10, 2, 1]. Performance of KAN model is shown in Figure 5. Loss can be observed in Figure 5a; although variations of the loss during training can be observed, it is reduced consistently, and RMSE for training and test is shown in Figure 5b, behavior of the RMSE is consistent with loss.



(**a**) Training loss for `[10, 2, 1]` (**b**) RMSE for `[10, 2, 1]`

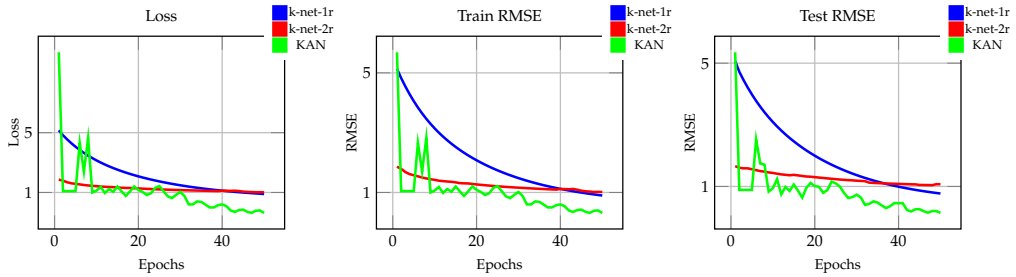
Figure 4. Performance on the Diabetes dataset using k-net model with architecture [10, 2, 1].



(**a**) Training loss for `KAN [10, 2, 1]` (**b**) RMSE for `KAN [10, 2, 1]`

Figure 5. Performance on the Diabetes dataset using KAN model with architecture [10, 2, 1].

For the regression task, in Figure 6, loss and error of the models are compared. Figure 6a presents the evolution of the loss for the three models evaluated on the Diabetes dataset. KAN achieves a lower loss compared to both k-net-1r and k-net-2r. Both k-net configurations exhibit an almost similar final loss. Logarithmic scale is used for better visualization. Figure 6b shows the training RMSE. Notice that KAN model is slightly better than both k-net models, with a final RMSE of 0.759. Training RMSE for the k-net models is around 1. For test RMSE, Figure 6c compares test RMSE. Here, the first k-net model has a slightly better performance than the second. The KAN model performs better than both k-net models.



(**a**) Comparing Loss (**b**) Comparing training RMSE (**c**) Comparing test RMSE

Figure 6. Evolution of loss and RMSE for the k-net and KAN models on regression.

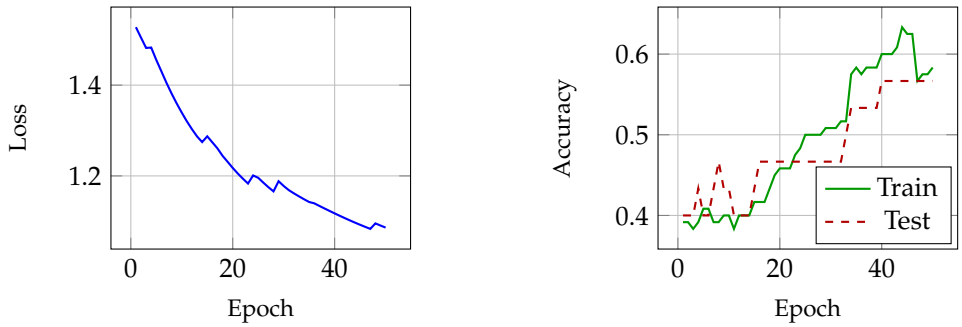
Table 2 summarizes results including loss, training, and testing RMSE. Training time is displayed in the last column of the table. As can be observed, both k-net models have an almost similar performance while KAN model loss is almost half of k-net and its test RMSE is 77% of the lowest k-net error, indicating a better approximation.

Table 2. Performance comparison of k-net and KAN models on the Diabetes dataset.

Model	Architecture	Final Loss	Train RMSE	Test RMSE	Epochs	Time (h)
k-net-1r	[10, 1]	0.960	0.960	0.918	50	40.7
k-net-2r	[10, 2, 1]	1.006	1.006	1.029	50	153.0
KAN	[10, 2, 1]	0.576	0.759	0.707	50	0.001

4.2. Classification

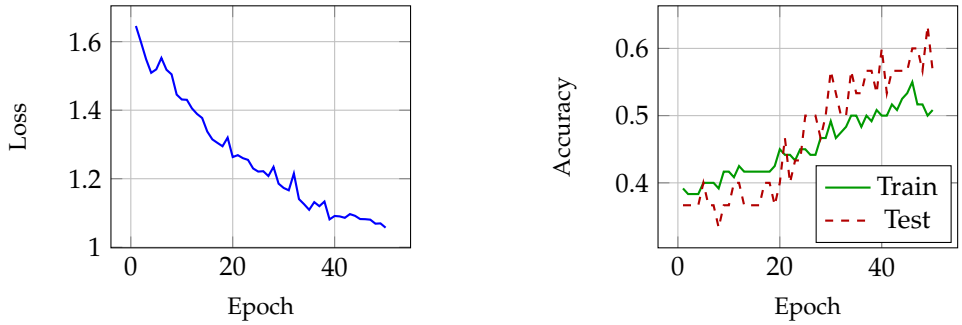
The Iris dataset includes 150 samples, each with 4 input features and a categorical label for one of three classes. Two k-net architectures were tested: a two-layer k-net with shape $[4, 2, 3]$, whose results are shown in Figure 7, and a three-layer k-net with $[4, 3, 3, 3]$ displayed in Figure 8. Cross-entropy loss and classification accuracy are shown below for each case. For the first architecture, Figure 7a shows decreasing loss across the 50 epochs. Training and test accuracy can be observed in Figure 7b, where a lot of variation and a few plateaus occur as accuracy improves.



(**a**) Training loss for `[4, 2, 3]` (**b**) Accuracy for `[4, 2, 3]`

Figure 7. Performance on the Iris dataset using k-net model with architecture [4, 2, 3].

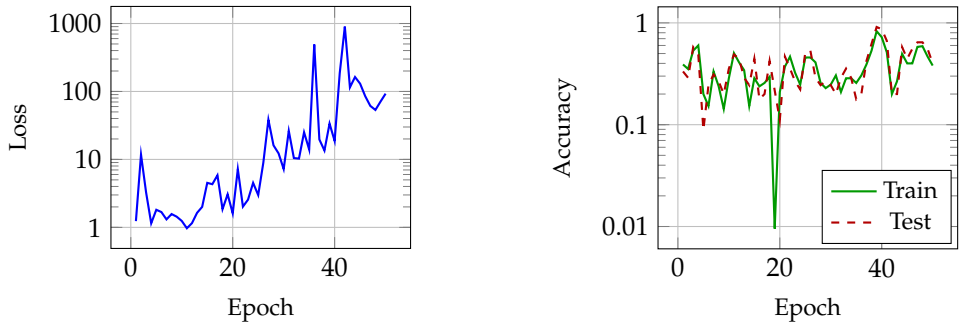
In Figure 8, results of the second k-net architecture are shown. During the 50 epochs in Figure 8a, loss is reduced with higher variation than the first model, as shown in Figure 7a. In Figure 8b, accuracy for the test set improves more than training accuracy.



(**a**) Training loss for `[4, 3, 3, 3]` (**b**) Accuracy for `[4, 3, 3, 3]`

Figure 8. Performance on the Iris dataset using k-net model with architecture [4, 3, 3, 3].

The model used was [4, 3, 3, 3]. Figure 9 shows the performance of this model. For this case, Figure 9a shows that loss during training increases, and Figure 9b shows that accuracy is not improved.



(**a**) Training loss for `KAN [4, 3, 3, 3]` (**b**) Accuracy for `KAN [4, 3, 3, 3]`

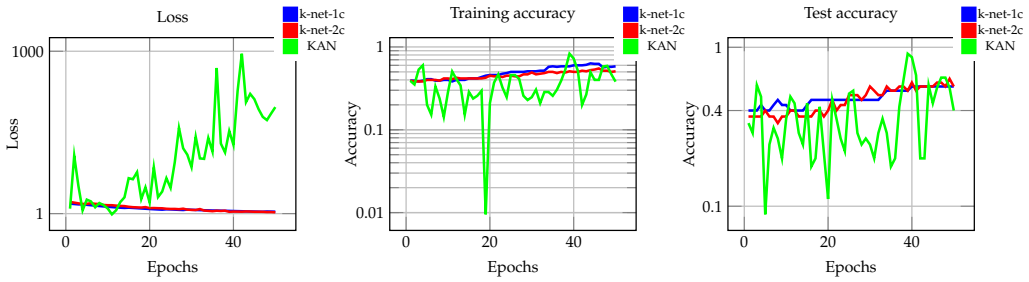
Figure 9. Performance on the Iris dataset using KAN model with architecture [4, 3, 3, 3].

In Figure 10, evolution of loss and accuracy for the three models used on classification is presented. Figure 10a compares the change in loss during training on the Iris dataset. Both k-net models achieve a similar loss (note that k-net-2c plot overlaps k-net-1c in the graph). Interestingly, KAN loss increases over epochs. Figure 10b shows the training accuracy. k-net-1c achieves better training accuracy than k-net-2c despite having a simpler architecture, suggesting that the additional layer in k-net-2c may not contribute effectively to learning on this dataset. The KAN model exhibits high variation, with an accuracy lower than both k-net models. Figure 10c compares the test accuracy of the models. Although k-net-2c exhibits some variability across epochs, its final accuracy is similar to that of k-net-1c. KAN presents very high variability. These results indicate that, for this classification task, increasing the architectural complexity of k-net does not yield improved performance.

The performance comparison of k-net and KAN models on the Iris dataset is displayed in Table 3, which summarizes results, including loss, training, and testing accuracy. Training time is observed in the last column.

The comparative evaluation of Kolmogorov–Arnold Networks (k-nets) and KAN model across classification (Iris) and regression (Diabetes) tasks reveals consistent performance on both k-net models. Notably, adding depth to the k-net architecture—such as moving from a two-layer (k-net-1c) to a three-layer (k-net-2c) modeling—does not yield performance improvements, suggesting diminishing returns from increased architectural complexity under the current training setup. In regression tasks, both k-net variants and KAN achieved very similar results. Moreover, the deeper k-net model underperformed relative to the simpler one, indicating that increased depth may introduce over-parametrization

or training instability when the internal transformations are constrained to a fixed functional form (e.g., B-splines). These observations highlight several limitations in the current implementation of k-net, which is based on the classical Hecht-Nielsen interpretation of the Kolmogorov–Arnold theorem. While theoretically elegant, the structured decomposition employed in k-net may impose rigidity that restricts the expressiveness or learning dynamics of the model when optimized via standard gradient descent. The performance plateau observed in deeper k-net models may also reflect suboptimal gradient flow, potentially exacerbated by the sequential structure of the spline-based inner representations. Despite these limitations, the k-net framework offers a unique architectural bias grounded in formal functional decomposition. This may be advantageous for interpretability, as the intermediate transformations are explicitly defined and independently accessible. However, such transparency appears to come at a cost in terms of representational flexibility and optimization efficiency when compared to conventional deep learning architectures.



(**a**) Comparing loss (**b**) Comparing training accuracy (**c**) Comparing test accuracy

Figure 10. Evolution of loss and accuracy for the k-net and KAN models on classification.

Table 3. Performance comparison of k-net and KAN models on Iris dataset.

Model	Architecture	Final Loss	Train Accuracy	Test Accuracy	Epochs	Time (h)
k-net-1c	[4, 2, 3]	1.086	0.583	0.566	50	17.7
k-net-2c	[4, 3, 3, 3]	1.057	0.508	0.566	50	87.1
KAN	[4, 3, 3, 3]	26.011	0.285	0.333	50	0.001

5. Conclusions

This paper presents a mathematically grounded definition of Kolmogorov–Arnold Networks (k-nets), formulated as a neural architecture inspired by the representation theorem of Kolmogorov and Arnold. The proposed definition exploits the linearity of the representation and is independent of the specific class of univariate functions satisfying certain Lipschitz conditions employed, offering a flexible and generalizable foundation for network construction. The principal contribution of this study is twofold: (i) the formalization of k-nets as a class of neural networks derived directly from the mapping of several variable functions in terms of simpler functions of one variable, and (ii) the exploration of forward-mode automatic differentiation using dual numbers as a viable training mechanism within this framework. In addition to establishing the theoretical foundation, this study presents a practical implementation of k-nets using cubic B-splines as internal univariate functions, ReLU as external function, and evaluates their learning performance on standard classification and regression tasks.

The results show that, although k-nets are capable of learning and exhibit convergence during training, their predictive accuracy, convergence speed, and generalization performance consistently fall short when compared to standard Multilayer Perceptrons (MLPs). Furthermore, increasing the depth of k-nets did not improve the outcome, and

in some cases, led to diminished performance. Despite these limitations, the structured nature of k-nets offers a valuable degree of interpretability: each internal transformation corresponds to an explicitly defined univariate function, which can be independently analyzed. This property opens new possibilities for incorporating alternative univariate function representations—beyond traditional sigmoidal activations—thereby expanding the design space of explainable neural models. Moreover, the use of dual numbers for gradient computation highlights an alternative to classical backpropagation techniques, and may be of particular interest in domains such as differential equation solving, as an alternative to traditional methods such as finite-differences.

This study also serves as a practical validation of the k-net framework, keeping its definition as close as possible to the original theorem, showing that such models are not merely theoretical constructs, but can be trained and compared to existing architectures. While their performance under standard training regimes may not yet match modern neural architectures, k-nets provide a promising direction for structured and interpretable learning frameworks, where each transformation can be analyzed independently. Future research may explore enhanced initialization schemes, dynamic adaptation of internal functions, alternative representation of the internal functions, different optimization strategies, and hybrid architectures that preserve interpretability while improving empirical performance. Ultimately, addressing the trade-off between mathematical structure and expressive power remains a central challenge in the development of this model class.

Author Contributions: Conceptualization, J.R.M.-R. and J.A.H.-S.; Software, A.S.-W.; Formal analysis, J.A.H.-S.; Investigation, A.S.-W.; Writing—original draft, A.S.-W.; Writing—review and editing, J.R.M.-R. and J.A.H.-S. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partially supported by the Mexican National Council of Science, Technology and Humanities (CONAHCyT) through its scholarship program for students to obtain a MSc or PhD degree, scholarship number 66c4d44c0834050f707a8789; as well as the division of National System of Researchers (SNI) that provides support for many researchers accross the country.

Data Availability Statement: The programs developed to implement the model—including the tested examples reported in this article and the result tables used—are available at http://github.com/aswinkler/k-net, accessed on 3 August 2025.

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

References

1. Petersen, P.; Zech, J. Mathematical Theory of Deep Learning. *arXiv* **2024**, arXiv:2407.18384. [CrossRef]
2. Hornik, K.; Stinchcombe, M.; White, H. Multilayer Feedforward Networks Are Universal Approximators. *Neural Netw.* **1989**, *2*, 359–366. [CrossRef]
3. Liu, Z.; Wang, Y.; Vaidya, S.; Ruehle, F.; Halverson, J.; Soljačić, M.; Hou, T.Y.; Tegmark, M. KAN: Kolmogorov-Arnold Networks. *arXiv* **2025**, arXiv:2404.19756. [CrossRef]
4. Tsimenidis, S. Limitations of Deep Neural Networks: A Discussion of G. Marcus' Critical Appraisal of Deep Learning. *arXiv* **2020**, arXiv:2012.15754. [CrossRef]
5. Zhao, H.; Chen, H.; Yang, F.; Liu, N.; Deng, H.; Cai, H.; Wang, S.; Yin, D.; Du, M. Explainability for Large Language Models: A Survey. *arXiv* **2023**, arXiv:2309.01029. [CrossRef]
6. Kolmogorov, A.N. On the Representation of Contuous Functions of Several Variables as Superposition of Continuous Functions of a Smaller Number of Variables. *Dokl. Akad. Nauk USR* **1956**, *5*, 953–956.
7. Arnol'd, V. On Functions of Three Variables. *Dokl. Akad. Nauk USR* **1957**, *4*, 679–681.
8. Hoffmann, P.H.W. A Hitchhiker's Guide to Automatic Differentiation. *Numer. Algorithms* **2016**, *72*, 775–811. [CrossRef]
9. Kandasamy, W.B.V.; Smarandache, F. *Dual Numbers*; Zip Publishing: Las Vegas, NV, USA, 2012.
10. Ji, T.; Hou, Y.; Zhang, D. A Comprehensive Survey on Kolmogorov Arnold Networks (KAN). *arXiv* **2025**, arXiv:2407.11075. [CrossRef]

11. Cunningham, H.; Ewart, A.; Riggs, L.; Huben, R.; Sharkey, L. Sparse Autoencoders Find Highly Interpretable Features in Language Models. *arXiv* **2023**, arXiv:2309.08600. [CrossRef]
12. Nielsen, R.H. Kolmogorov's Mapping Neural Network Existence Theorem. In Proceedings of the IEEE First International Conference on Neural Networks, San Diego, CA, USA, 21–24 June 1987; IEEE: Piscataway, NJ, USA, 1987; Volume III, pp. 11–13.
13. Sprecher, D.A. On the Structure of Continuous Functions of Several Variables. *Trans. Am. Math. Soc.* **1965**, *115*, 340–355. [CrossRef]
14. Schmidt-Hieber, J. The Kolmogorov–Arnold Representation Theorem Revisited. *Neural Netw.* **2021**, *137*, 119–126. [CrossRef] [PubMed]
15. Actor, J.; Knepley, M.G. An Algorithm for Computing Lipschitz Inner Functions in Kolmogorov's Superposition Theorem. *arXiv* **2017**, arXiv:1712.08286. [CrossRef]
16. Poggio, T.; Banburski, A.; Liao, Q. Theoretical Issues in Deep Networks: Approximation, Optimization and Generalization. *arXiv* **2019**, arXiv:1908.09375. [CrossRef]
17. Lorentz, G.G. Metric Entropy, Widths, and Superpositions of Functions. *Am. Math. Mon.* **1962**, *69*, 469–485. [CrossRef]
18. Sprecher, D.A. A Numerical Implementation of Kolmogorov's Superpositions. *Neural Netw.* **1996**, *9*, 765–772. [CrossRef] [PubMed]
19. Vigliotti, A.; Auricchio, F. Automatic Differentiation for Solid Mechanics. *Arch. Comput. Methods Eng.* **2020**, *28*, 875–895. [CrossRef]
20. Baydin, A.G.; Pearlmutter, B.A.; Radul, A.A.; Siskind, J.M. Automatic Differentiation in Machine Learning: A Survey. *arXiv* **2018**, arXiv:1502.05767. [CrossRef]
21. Bustamante, J. *Bernstein Operators and Their Properties*; Springer International Publishing: Cham, Switzerland, 2017. [CrossRef]
22. Arnold, V.I. Local Normal Forms of Functions. *Invent. Math.* **1976**, *35*, 87–109. [CrossRef]
23. Samanci, H.K. Generalized Dual-Variable Bernstein Polyniomials. *Konuralp J. Math.* **2017**, *5*, 56–67.