Learning Kolmogorov-Arnold Neural Activation Functions by Infinite-Dimensional Optimization

Leon Khalyavin L.KHALYAVIN23@IMPERIAL.AC.UK
Department of Electrical and Electronic Engineering,
Imperial College London, SW72AZ London, U.K.

Alessio Moreschini A.MORESCHINI@IMPERIAL.AC.UK
Department of Electrical and Electronic Engineering,
Imperial College London, SW72AZ London, U.K.

Thomas Parisini T.PARISINI@IMPERIAL.AC.UK
Department of Electrical and Electronic Engineering,
Imperial College London, SW72AZ London, U.K.
Department of Electronic Systems, Aalborg University, Denmark.
Department of Engineering and Architecture, University of Trieste, Italy.

Editors: N. Ozay, L. Balzano, D. Panagou, A. Abate

Abstract

Inspired by the Kolmogorov-Arnold Representation Theorem, Kolmogorov-Arnold Networks (KAN) have recently reshaped the landscape of functional representation in the context of training univariate activation functions, thus achieving remarkable performance gains over traditional Multi-Layer Perceptron (MLPs). However, the parametrization of the activation functions in KANs hinders their scalability and usage in machine learning applications. In this article, we propose a novel infinite-dimensional optimization framework to learn the activation functions, enabling the achievement of boundedness, interpretability, and, seamless compatibility with training by backpropagation are all preserved for the network, therefore, reducing the number of required trainable parameters. Through functional representation examples, our results reveal superior accuracy in tasks such as functional representation.

Keywords: Functional Learning; Kolmogorov Theorem; Learning Neural Activation Functions; Infinite-dimensional Optimization.

1. Introduction

Following Vladimir Vapnik's interpretation of machine learning (ML) (Vapnik, 1998, 2000), the general conceptual ML problem can be reinterpreted in terms of a *function approximation* problem. In this context, the Kolmogorov-Arnold (KA) Representation Theorem — originally proved in Kolmogorov (1957) to tackle *Hilbert's 13th Problem* Hilbert (1902) — spurred significant interest and debates in the early '90s due to its potential relevance in *(i)* addressing the well-known *curse of dimensionality* when tackling general function approximation problems (Zoppoli et al., 2020, Sec. 2.5), and *(ii)* providing a theoretical justification for the early successes in the application of neural networks. KA representation theorem can be stated as follows: any function $f : \mathbb{R}^n \to \mathbb{R}$ can be *exactly represented* in terms of *only* $(2n+1)(n+1)$ univariate functions, that is:

$$f(x_1, \cdots, x_n) = \sum_{q=1}^{2n+1} g_q \left(\sum_{p=1}^{n} \psi_{q,p}(x_p) \right),$$

where[1] $g_q : \mathbb{R} \to \mathbb{R}$ and $\psi_{q,p} : \mathbb{R} \to \mathbb{R}$. The polynomial growth of the number $(2n+1)(n+1)$ of univariate functions with respect to the number n of arguments of the objective function f marks the possible relevance of the KA theorem in terms of taming the curse of dimensionality.

As recently remarked in Schmidt-Hieber (2021), the usefulness of the KA representation theorem in ML and, lately, in deep learning, has been the subject of intense debate since the late '80s. Researchers such as Girosi and Poggio (1989) have argued that the theoretical and practical relevance of the KA representation theorem is undermined due to the lack of a constructive proof and difficulties in designing the smooth functions ϕ. Conversely, other researchers such as Kůrková (1991) strongly disagreed with the statement, arguing for the strong relevance of the representation. Consequently, follow-up works have attempted to connect the KA representation to neural networks, often reformulating the problem as a two-hidden-layer network (Schmidt-Hieber, 2021). See, e.g., Lin and Unbehauen (1993); Sprecher and Draghici (2002); Kůrková (1992); Jürgen Braun (2009); Montanelli and Yang (2020); Polar and Poluektov (2021); Fakhoury et al. (2022).

The fundamental question with making the KA representation implementable is: *How should the nonlinear functions ϕ be learned?* In common deep learning architectures, such as Multi-Layer Perceptron (MLP) networks, often only the linear components need to be trained, leveraging the famous Universal Approximation Theorem, e.g. Hornik et al. (1989); LeCun et al. (2015). Attempts at learning the nonlinear components have been made, such as polynomial activation functions (Goyal et al., 2019), max-out units (Goodfellow et al., 2013), universal activation functions (Yuen et al., 2021), and others (Apicella et al., 2021). However, the main difficulty lies in changing these structures of activation functions, as the network often cannot guarantee universal approximation (Stinchcombe and White, 1990). Indeed, in research fields such as control systems, bounds on approximation errors are needed for example because stability guarantees are essential, which makes the usage of learned activation functions highly critical (Wang and Hill, 2006; Cai et al., 2021; Xiong et al., 2024).

The recent developments in deep learning of the Kolmogorov Arnold Network (KAN) by Liu et al. (2024) take inspiration from the KA representation theorem by learning the activation functions ϕ — represented as B-splines — on the edges rather than the linear components, as done with standard MLP networks. By only learning the nonlinear components, studies have found that KANs can significantly outperform MLPs on tasks such as functional representation, see, e.g., Yu et al. (2024); Koenig et al. (2024). Despite some promising results, the article Yu et al. (2024) also shows that KANs severely underperform on numerous machine learning problems, such as classification, image processing, and speech recognition, likely due to the number of parameters required to represent the activation functions ϕ. Furthermore, the representation of ϕ as B-splines introduces constraints, not necessarily matching the structure and the KA representation.

As with previous neural-network research, the KAN relies on discretizing the functional space. However, we believe that training should be done *directly* on the functional space with an infinite-dimensional operator. Specifically, Moreschini et al. (2024c) recently developed a promising infinite-dimensional optimization framework that leverages an infinite-dimensional operator — regarded as the Fréchet discrete gradient (Moreschini et al., 2024a). This operator enhances the discrete gradient operator of Gonzalez (1996) to infinite-dimensional spaces through a formal connection with the Fréchet derivative and its Riesz representation. As usual, in energy-based contexts (see, e.g. Moreschini et al. (2024b)), discrete gradient methods are used for approximating the solution

1. For simplicity, throughout the article, we generically refer to the univariate functions g_q and $\psi_{q,p}$ as $\phi : \mathbb{R} \to \mathbb{R}$.

of differential equations — in a geometric fashion — while preserving their first integral. In turn, the strength of optimization algorithms based on Fréchet discrete gradients is that they guarantee convergence of the optimization algorithm on infinite dimensional spaces for any (bounded) step size, even in abstract time domains following (Moreschini et al., 2025). We extend our infinite-dimensional optimization method to generate a closed and bounded functional subspace defined by a continuous function, a generalized Fréchet differentiable cost function, and a single parameter to be learned, representing all the functions found in the generalized gradient descent. With an initial function, the method leverages the Fréchet discrete gradient to find the direction of the update step in the functional space and the single parameter is can be trained through backpropagation to arrive at the desired function in a single step. Therefore, we here introduce a class of new activation functions that are parameterized by solving for an infinite dimensional optimization problem — representing many applicable activation functions with just a few parameters. We can also *maintain the interpretability of the activation function*, as cost functions are the most transferrable and robust representation of a task Ng and Russell (2000). Infinite dimensional input neural networks have been shown to work in Rossi and Conan-Guez (2005) and Nishikawa et al. (2022), but ours differs by representing the activation function in the infinite dimension, rather than the input. We demonstrate through a running example the effectiveness of this approach for designing activation functions and for achieving a significantly good overall performance of the neural network.

The rest of the article is structured as follows: Section 2 introduces the preliminaries and describes the activation functions. Section 3 applies the activation functions to a layered structure, and in Section 4 we demonstrate the performance of activation functions on functional representation examples and we compare them with MLP and KAN solutions. Finally, in Section 5 we offer some concluding remarks.

2. Neural Activation Functions

2.1. Notations and Preliminaries

We denote by $\mathbb{R}$ the set of real numbers and by $\mathbb{N}$ as the set of natural numbers. For a given $a, b \in \mathbb{R}$ and $n \in \mathbb{N}$, let $\mathscr{C}([a,b],\mathbb{R}^n)$ denote the set of all continuous functions $\phi : [a,b] \to \mathbb{R}^n$. For any two $\phi, \varphi \in \mathscr{C}([a,b],\mathbb{R}^n)$, we define the inner product as $\langle \phi, \varphi \rangle \coloneqq \int_a^b \varphi(s)^\top \phi(s) ds$. Consider the functional $S : \mathscr{C}([a,b],\mathbb{R}^n) \to \mathbb{R}$. For the rest of this article, we assume that ϕ is bounded on $[a,b]$. Functionals such as S have a special type of derivative, known as the Fréchet derivative $\nabla_F S : \mathscr{C}([a,b],\mathbb{R}^n) \to \mathbb{R}$. S is said to be Fréchet differentiable at $\phi_0 \in \mathscr{C}([a,b],\mathbb{R}^n)$ such that

$$\lim_{\|\phi\| \to 0^+} \frac{|S(\phi_0 + \phi) - S(\phi_0) - \nabla_F S(\phi_0)(\phi)|}{\|\phi\|} = 0,$$

where $|\cdot|$ is the Euclidean norm of a vector and $\|\phi\| \coloneqq \sqrt{\langle \phi, \phi \rangle}$. In our context, the Fréchet derivative can be represented by means of the Riesz representation $\nabla_R S \in \mathscr{C}([a,b],\mathbb{R}^n)$ such that $\nabla_F S(\phi_0)(\varphi) = \langle \nabla_R S(\phi_0), \varphi \rangle$ for every $\varphi \in \mathscr{C}([a,b],\mathbb{R}^n)$.

In infinite-dimensional optimization, we can represent S as a cost function, and we optimize with respect to ϕ using a *generalized gradient descent* of the form $\phi_{k+1} = \phi_k - \tau_k \nabla_R S(\phi_k)$, where ϕ_k is the iteration of the function and $\tau_k > 0$ is a time-varying positive step size assumed bounded for all k. Because the iterations are discrete, the convergence of the gradient descent is not guaranteed for all learning step size τ_k. To *guarantee the convergence*, and therefore a *robust implementation*, we use the novel Fréchet discrete gradient introduced in Moreschini et al. (2024a).

Given two functions, $\phi_{k+1}, \phi_k \in \mathscr{C}([a,b], \mathbb{R}^n)$, the Fréchet discrete gradient $\overline{\nabla} S(\phi, \varphi)$ satisfies the following equation

$$\begin{aligned} \langle \phi_{k+1} - \phi_k, \overline{\nabla} S(\phi_{k+1}, \phi_k) \rangle &= S(\phi_{k+1}) - S(\phi_k), \\ \lim_{\|\phi_{k+1} - \phi_k\| \to 0^+} \overline{\nabla} S(\phi_{k+1}, \phi_k) &= \nabla S(\phi_k). \end{aligned} \tag{1}$$

By implementing $\overline{\nabla} S(\phi_{k+1}, \phi_k)$, then for any lower semi-continuous, Fréchet differentiable function S, the dynamics

$$\phi_{k+1} = \phi_k - \tau_k \overline{\nabla} S(\phi_{k+1}, \phi_k). \tag{2}$$

converge for any $\tau_k > 0$, see (Moreschini et al., 2024a, Theorem 1). To better visualize the concepts, throughout the article, we use the following running example.

Example 1 *Assume that $S = (\phi_k - \phi_f)^2$, which represents the simple least squares problem. By using the Fréchet discrete gradient, we can find that $\overline{\nabla} S(\phi_{k+1}, \phi_k) = (\phi_{k+1} + \phi_k - 2\phi_f)$. Therefore, the optimization problem can be explicitly written as*

$$\phi_{k+1} = \phi_k - \frac{2\tau_k}{1 + \tau_k}(\phi_k - \phi_f). \tag{3}$$

2.2. Learning the Neural Activation Functions

Typically, activation functions represent the nonlinear components of a neural network. Most commonly, they are fixed univariate functions that apply element-wise. Trainable activation functions are often parameterized using a finite number of parameters. However, implementing expressive activation functions (such as those in the KAN) requires a large number of parameters, which can be computationally inefficient to store and train. The parameters in the network are optimized with respect to a global cost function, which we denote as $J : \mathbb{R}^d \to \mathbb{R}$, where d is a finite number of parameters (i.e., a mean-squared error). It is important to note that J and S are different cost functions: S is a function that optimizes each neural activation function and J optimizes the entire network. Therefore, we minimize J by solving sub-optimization problems defined by S for each neural activation function.

To maintain the representation of the activation function in infinite dimensions, we represent an activation function as a solution to the general optimization problem in (2) using our Fréchet discrete gradient. In fact, all solutions using (2) with any positive time-varying step size τ_k and a positive integer $K \geqslant 0$ exist in a closed and bounded functional subspace, since (2) is guaranteed to converge. Therefore, with the definition of S, τ_k and K will select a function on this closed and bounded subspace. In practice, evaluating for a large K may be impractical, as for each forward pass, we must iterate K times on every edge. Furthermore, because K is an integer, we cannot use backpropagation to directly train K, as it does not have a gradient. Instead, we optimize with respect to τ_k with a small fixed step size.

Proposition 1 *Given a strictly convex, non-negative, Fréchet differentiable function S and a differentiable and bounded function ϕ_0, then any function generated by (2) with a given $\tau_k \geqslant 0$ and $K \geqslant 0$ can be represented by the closed and bounded functional subspace defined by*

$$\phi = \phi_0 - \overline{\tau} \overline{\nabla} S(\phi, \phi_0). \tag{4}$$

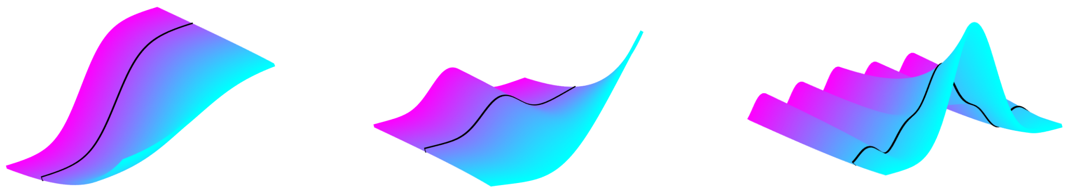


Figure 1: Trajectories generated by (3) with varying ϕ_0 (purple) and ϕ_f (cyan) with $\overline{\tau}$ selecting a function on the trajectory (black).

Proof The subspace generated by (4) is a closed and bounded subspace, as for any $\overline{\tau} \geqslant 0$, $S(\phi_0) \geqslant S(\phi)$ from Moreschini et al. (2024a). Since S is strictly convex and non-negative, applying (2) will result in a unique solution ϕ^* for any $\tau_k > 0$ with an initial ϕ_0 as shown in Moreschini et al. (2024a). Therefore, there exists a unique minimum and a τ^* that solves for that minimum in a single step by solving the implicit equation $\phi^* = \phi_0 - \tau^*\overline{\nabla}S(\phi^*, \phi_0)$. To represent a function ϕ_K, which is the solution to (2) after K steps, we can apply a similar approach by solving $\phi_K = \phi_0 - \overline{\tau}\overline{\nabla}S(\phi_K, \phi_0)$ for $\overline{\tau}$ implicitly. Because S is strictly convex, then $S(\phi^*) \leqslant S(\phi_K) \leqslant S(\phi_0)$ and therefore, a solution to ϕ_K for any $K \geqslant 0$ can be found in a single step by $\overline{\tau}$. ■

Example 2 *Consider the functions represented by (3). For any given ϕ_0 and ϕ_f, ϕ_K can be represented as $\phi_K = (1-\alpha)\phi_0 + \alpha\phi_f$, where $\alpha = \sum_{k=0}^{K} \frac{2\tau_k}{1+\tau_k}$. From this derivation, ϕ_K is the linear combination of ϕ_0 and ϕ_f. Using the expression*

$$\phi = \phi_0 - \frac{2\overline{\tau}}{1+\overline{\tau}}(\phi_0 - \phi_f), \tag{5}$$

we can represent $\phi = \phi_K$ by setting $\frac{2\overline{\tau}}{1+\overline{\tau}} = \alpha$.

By Proposition 1, we define a neural activation function as the solution of (2) for any positive $\tau_k \geqslant 0$ and $K \geqslant 0$, as in Definition 2 below.

Definition 2 (Neural Activation Function) *An activation function $\phi(S, \phi_0, \overline{\tau}) \in \mathscr{C}([a,b], \mathbb{R})$ is a differentiable and bounded function on the compact set $[a, b]$ existing in the subspace defined by (4), where $S : \mathscr{C}([a,b], \mathbb{R}) \to \mathbb{R}$ is a strictly convex, non-negative, twice Fréchet differentiable function, $\phi_0 \in \mathscr{C}([a,b], \mathbb{R})$ is a differentiable bounded function, and $\overline{\tau} \geqslant 0$.*

Example 3 *Figure 1 demonstrates all of the possible activation functions for a given ϕ_0 and ϕ_f by varying $\overline{\tau}$, as in (5). In this figure, we have 3 different selections of ϕ_0 and ϕ_f. From left to right, sigmoid(x) and $\tanh(x)$, from $\exp(-x^2)$ and $\exp(x)$, and $0.1\cos(5x)$ and $\exp(-x^2)$ are chosen as ϕ_0 and ϕ_f for each subfigure, respectively. It is evident that in this example, every activation function shows the linear combinations between ϕ_0 and ϕ_f. In each subfigure, $\overline{\tau}$ selects a function on the surface (represented in black). Therefore, with the initialization of ϕ_0 and ϕ_f, we can navigate the entire surface just with a single parameter $\overline{\tau}$.*

To learn an optimal neural activation function, we optimize with respect to $\overline{\tau}$. In this formulation, the Fréchet discrete gradient selects the direction in the infinite dimensional space and the trainable $\overline{\tau}$ finds the exact step size. Because the formulation is derived from a *guaranteed stable* *trajectory*, it follows that, for any $0 \leqslant \overline{\tau} < \infty$, the resulting ϕ is a bounded function, and hence it can be evaluated: we compute $\phi_0(x)$ at x and the output yields the equivalent representation of $\phi(x)$. By defining a neural activation function with the infinite-dimensional operator $\bar{\nabla}S(\phi, \phi_0)$, we guarantee boundedness on the compact set $[a, b]$ and direct relation with the cost functional S. Unlike other approximation methods such as kernels and B-splines, the neural activation function can represent the *exact* function that minimizes the functional S.

3. Layered Structure of Neural Activation Functions

In this section, we apply the neural activation functions previously defined to a network structure, inspired by the KA representation. Similar to Liu et al. (2024), we consider a fully connected edge layer (every node in layer l has a directed edge to every node in layer $l + 1$) where each edge from node i to node j represents a neural activation function. We represent the activation function with (4) just using a *single trainable parameter* $\overline{\tau}$, which is learned by backpropagation instead of B-splines as in the KAN. The structure of a neural network is defined by an integer array $[n_1, \cdots, n_L]$, where $n_l \in \mathbb{N}$ represents the number of nodes in each layer, $L \in \mathbb{N}$ represents the number of layers, and $l \in [1, L]$ and the function of the layer is defined by the user. In an MLP network, the layers are linear, where $y = \sigma(W_l x + b_l)$ and $W_l \in \mathbb{R}^{n_l \times n_{l+1}}$, $b_l \in \mathbb{R}^{n_{l+1}}$, x is the input to the layer, y is the output and $\sigma(\cdot)$ is a predefined element-wise nonlinear (or linear) activation function. To use the activation functions introduced in Definition 2, we combine functions into a neural activation layer, defined as in Definition 3 below.

Definition 3 (Neural Activation Layer) *An activation layer $\Phi_l : \mathbb{R}^{n_l} \to \mathbb{R}^{n_{l+1}}$ at l is a fully connected layer between n_l and n_{l+1} nodes. Each edge between nodes i and j represents a unique univariate activation function $\phi_{l,i,j} : \mathbb{R} \to \mathbb{R}$, where $\phi_{l,i,j} = \phi_{0,l,i,j} - \overline{\tau}_{i,j}\overline{\nabla}S_{l,i,j}(\phi_{l,i,j}, \phi_{0,l,i,j})$ and $\overline{\nabla}S_{l,i,j}(\phi_{l,i,j}, \phi_{0,l,i,j})$ is the Fréchet discrete gradient of S evaluated at $\phi_{l,i,j}$ and $\phi_{0,l,i,j}$ similar to* (4)*. At the output layers, n_{l+1}, a sum is taken at each node for all incoming edges.*

Each edge stores a unique $S_{l,i,j}$, $\phi_{0,l,i,j}$, and a trainable $\overline{\tau}_{l,i,j}$ in an activation layer. At the output node, a sum is taken of all the inputs[2]. According to Definition 3, the activation function between node i and j in layer l is given by:

$$\phi_{l,i,j} = \phi_{0,l,i,j} - \overline{\tau}_{l,i,j}\overline{\nabla}S_{l,i,j}(\phi_{l,i,j}, \phi_{0,l,i,j})\,.$$

Hence, it is immediate to see that the output of the activation layer is obtained as

$$\Phi_l = \begin{bmatrix} \sum_{i=1}^{n_l}(\phi_{0,l,i,1} - \overline{\tau}_{l,i,1}\overline{\nabla}S_{l,i,1}(\phi_{l,i,1}, \phi_{0,l,i,1})) \\ \vdots \\ \sum_{i=1}^{n_l}(\phi_{0,l,i,n_{l+1}} - \overline{\tau}_{l,i,n_{l+1}}\overline{\nabla}S_{l,i,n_{l+1}}(\phi_{l,i,n_{l+1}}, \phi_{0,l,i,n_{l+1}})) \end{bmatrix}. \tag{6}$$

By defining the neural activation layer in this way, we get the exact KA representation with two activation layers and a node structure of $[n, 2n + 1, 1]$. Furthermore, the definition of the neural activation layer allows for the flexibility of defining deep neural networks, by sequentially processing multiple neural activation layers, where the output. With such a sequential structure, the output of the previous neural activation layer can provide not only the following input, but

2. If the layer has a large amount of edges, a weighted sum can be taken instead (with possibly trainable weights), but that is beyond the scope of the present article.

it can also be used to define the following neural activation layer. For example, we can select $\phi_{0,l,i,j} = \sum_{k=1}^{n_{l-1}} \phi_{l-1,i,k}$, such that neural activation function at layer l depends on those learned at layer $l-1$. Thus, a function f can be learned by multiple neural activation layers and the representation $\hat{f}$ can be shown as

$$\hat{f}(x_1, \cdots, x_n) = \Phi_L(\Phi_{L-1}(\cdots(\Phi_1(x_1, \cdots, x_n)))).$$

Under standard differentiability conditions with respect to x and the learning parameters $\overline{\tau}$ and by implicit differentiations, the classical backpropagation procedure (see, Rumelhart et al. (1986)) can be used. Since ϕ_0 is once differentiable and S is twice differentiable, we can obtain that the backpropagation gradients for an activation function are the solutions of the following equations[3]:

$$\frac{\partial \phi}{\partial x} = \frac{\partial \phi_0}{\partial x} - \overline{\tau}\frac{\partial \bar{\nabla} S(\phi, \phi_0)}{\partial x}, \quad \frac{\partial \phi}{\partial \overline{\tau}} = -\bar{\nabla} S(\phi, \phi_0) - \overline{\tau}\frac{\partial \bar{\nabla} S(\phi, \phi_0)}{\partial \overline{\tau}}. \tag{7}$$

It is worth noting that (7) is an implicit expression. However, by choosing an S that can result in an explicit solution, we are also able to represent the gradient explicitly. It is also possible to train the functional subspace defined by $\overline{\nabla} S(\phi, \phi_0)$ and ϕ_0.

Example 4 *Following Example 3, assume that the activation function $\phi(S, \phi_0, \overline{\tau})$ is defined by* (5) *and that $\overline{\tau}$ is our trainable parameter. Also, without loss of generality*[4]*, suppose that the entire network is optimizing a least squares cost function of the form*

$$J = \frac{1}{N}\sum_{s=1}^{N}(\hat{y}_s - y_s)^2, \tag{8}$$

where N is the number of samples, $s \in \mathbb{N}$ is the sample index, y_s is the output data, and $\hat{y}_s$ is the predicted output of the model. Given that ϕ and ϕ_0 are differentiable, we obtain

$$\frac{\partial \phi}{\partial x_s} = \frac{\partial \phi_0}{\partial x_s} - \frac{2\overline{\tau}}{1+\overline{\tau}}\left(\frac{\partial \phi_0}{\partial x_s} - \frac{\partial \phi_f}{\partial x_s}\right), \quad \frac{\partial \phi}{\partial \overline{\tau}} = -\frac{2}{(1+\overline{\tau})^2}(\phi_0 - \phi_f).$$

Backpropagation can now be easily applied taking into account the summations in (6).

4. Numerical Examples

We present a numerical validation of our proposed methodology to demonstrate the trainability of a network of neural activation functions. To characterize the families of neural activation functions on each edge, we suitably select the functions $\phi_{0,l,i,j}$, and learn the parameters $\overline{\tau}_{l,i,j}$ via optimizing the cost functional (8). In the following examples, each of the neural activation functions is defined with $S_{l,i,j} = \phi_{l,i,j}^2$ (in consistency with the running example, but setting $\phi_{f,l,i,j} = 0$ for all x). In order to maintain interpretability, we select S to have the same structure of J (we approximate the final function f rather than use training data to directly define it). The minimization of the cost functional J, in turn boils down to exploiting our infinite-dimensional optimization methodology to learn the representation of the neural activation functions $\phi_{l,i,j}$. To illustrate the generality and versatility of the proposed approach, each example uses the same selection of $\phi_{0,l,i,j}$ to learn the activation functions.

3. Due to space limitation, the explicit version is not dealt with in this preliminary work.
4. More general cost functionals could be considered, not necessarily explicitly depending on y_s.

l	i	j	$S_{l,i,j}$	$\phi_{0,l,i,j}$	$\alpha_{l,i,j}$ for f	$\alpha_{l,i,j}$ for g	$\alpha_{l,i,j}$ for h
1	1	1	ϕ^2	$\sin(\pi x)$	**1.33e-4**	0.999	0.568
1	1	2	ϕ^2	$x^3 - 1$	0.857	**8.13e-3**	1.65e-2
1	1	3	ϕ^2	$3\exp(-x^2)$	0.933	0.597	0.671
1	1	4	ϕ^2	$\tanh(x)$	0.107	0.163	2.69e-5
1	1	5	ϕ^2	$0.5\cos(x)$	0.569	**0.916**	0.994
1	2	1	ϕ^2	x^2	**3.81e-3**	0.577	0.999
1	2	2	ϕ^2	$10x$	0.981	**0.491**	0.990
1	2	3	ϕ^2	$\sin(x) - 2$	0.913	0.676	0.649
1	2	4	ϕ^2	$5\tanh(x)$	0.956	0.719	0.729
1	2	5	ϕ^2	$\text{sigmoid}(3x)$	0.907	0.544	6.64e-2
2	1	1	ϕ^2	$\exp(x)$	**3.71e-3**	0.695	6.87e-2
2	2	1	ϕ^2	x	0.989	**0.187**	6.67e-5
2	3	1	ϕ^2	x^3	0.997	0.558	0.551
2	4	1	ϕ^2	$-\cos(x)$	0.844	0.481	0.342
2	5	1	ϕ^2	$3\sin(x)$	0.798	**0.660**	0.266

Table 1: Learned neural activation functions for the network.

For the sake of comparison, we first select the target function of the neural network learning task as in the second example in (Liu et al., 2024, Sec. 3.1), that is:

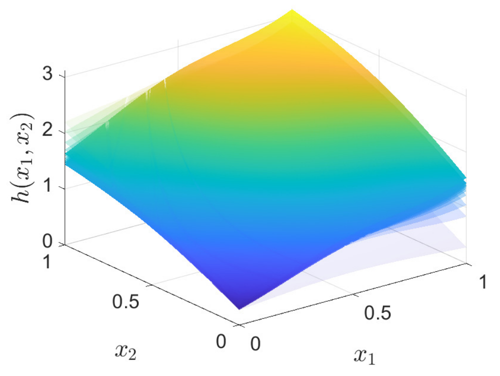
$$f(x_1, x_2) = \exp(x_2^2 + \sin(\pi x_1)), \text{ with } x_1, x_2 \in [0, 1]\,. \tag{9}$$

As an additional learning task — to show the effectiveness of our functional learning methodology[5] — we also train the neural network (thus learning its activation functions) to represent the functions:

$$g(x_1, x_2) = x_1^3 + 5x_2 + 3\sin(0.1\cos(x_1)) - 1, \quad h(x_1, x_2) = \sin(x_1) + \sin(x_2) + \sin(x_1 + x_2).$$

In this section, our goal is to represent f, g, and h with the KA structure by initializing a network with two neural activation layers and a node structure of [2,5,1]. From Liu et al. (2024), it is evident that f can be explicitly represented with a network of the structure of $[2, 1, 1]$ and selecting $\phi_{1,1,1} = \sin(\pi x)$, $\phi_{1,2,1} = x^2$, and $\phi_{2,1,1} = \exp(x)$. However, the native KAN relies on B-splines to represent such functions, thus reducing the infinite-dimensional framework to a finite-parametrization, which is what our approach aims to avoid. Hence, we represent these activation functions with neural activation functions (as in Definition 2). By selecting $\phi_{0,1,1,1} = \sin(\pi x)$, $\phi_{0,1,2,1} = x^2$, and $\phi_{0,2,1,1} = \exp(x)$, we ensure that a set of $\overline{\tau}$ exists that can exactly represent f, regardless of $\phi_{0,l,i,j}$ for the remainder of the edges. In doing so, we define the necessary functional subspaces to exactly represent f with just a single parameter per edge, rather than the many parameters required to represent each B-spline. The other functions $\phi_{0,l,i,j}$ are selected such that they can get the exact representation of g, with the intent to show how a single initialization of $\phi_{0,l,i,j}$ can represent different functions. Yet, the remainder of the $\phi_{0,l,i,j}$ are selected to *not* have an exact

5. The networks are optimized with the Adam optimizer, with $\overline{\tau}_{l,i,j}$ being the only parameters optimized, which is done so through backpropagation. Furthermore, the networks are also optimized using a stochastic optimization approach, with a batch of 100 training data points generated at every epoch (the evaluation dataset stays constant) and corrupted with additive zero-mean Gaussian measurement noise with the distribution $\eta \sim \mathcal{N}(0, 0.1)$.



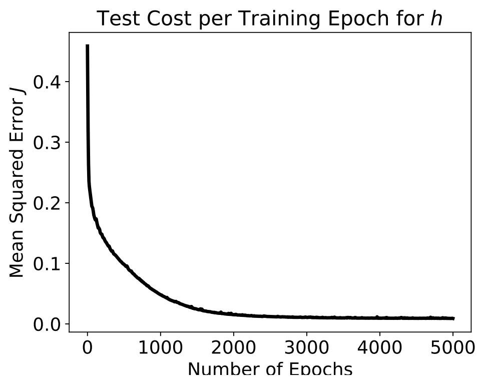


Figure 2: Convergence of the h surface every 500 epochs, from transparent to opaque (left) and behavior of the cost functions per epoch (right).

Model	# of Parameters	Test c .f. for f	Test c .f. for g	Test c .f. for h
ReLU MLP	201	3.21e-3	2.56e-4	1.95e-4
KAN	171	1.77e-4	1.21e-4	**9.53e-5**
Neural Activation Layers	**15**	**6.70e-5**	**9.79e-5**	8.62e-3

Table 2: Validation costs between with the MLP, KAN, and Neural Activation Layers.

representation of h, to demonstrate the convergence to a minimal solution, without prior knowledge of the function. Furthermore, because we only learn a single parameter per edge, we demonstrate that the learned solution *remains interpretable*. Consider that $\alpha_{l,i,j} \coloneqq 2(1+\overline{\tau}_{l,i,j})^{-1}\overline{\tau}_{l,i,j}$, such that we can represent $\phi_{l,i,j} = (1-\alpha_{l,i,j})\phi_{0,l,i,j}$. In this case, if $\alpha_{l,i,j}$ is trained to be large, then we infer that the contribution of $\phi_{0,l,i,j}$ to the representation of the function of the network is reduced (and vice-versa).

Table 1 shows the selected $\phi_{0,l,i,j}$ for every edge in the network. The neural activation functions highlighted in yellow can represent f exactly ($\phi_{1,1,1}$, $\phi_{1,2,1}$, and $\phi_{2,1,1}$) and the ones highlighted in cyan can represent g exactly ($\phi_{1,1,2}$, $\phi_{1,1,5}$, $\phi_{1,2,2}$, $\phi_{2,2,1}$, and $\phi_{2,5,1}$). Furthermore, Table 1 shows the trained parameters of the network for each example. With our network of neural activation functions, we can represent the learned output of f as $\hat{f}$ defined by

$$\hat{f} = \sum_{q=1}^{5}(1-\alpha_{2,q,1})\phi_{0,2,q,1}\left(\sum_{p=1}^{2}(1-\alpha_{1,q,p})\phi_{0,1,q,p}(x_p)\right), \tag{10}$$

where $\alpha_{l,i,j}$ and $\phi_{0,l,i,j}$ can be found in Table 1. The same argument applies equivalently to g and h.

We compare these results with an MLP, with a structure of [2,50,1] and a ReLU activation function, and the KAN, with the same structure as our network of $[2, 5, 1]$, and selecting the grid size to be 5 and the number of knots to be 3 (the number of parameters for each network is calculated according to Yu et al. (2024)). The validation costs are shown in Table 2 where we find that for f and g, we outperform both the MLP and KAN models — highlighting the primary advantage of training in infinite-dimensional spaces: we require significantly fewer trainable parameters while retaining accurate functional representations. However, due to the flexibility of the MLP and the B-splines in the KAN, our model performs worse in representing h. Even still, we retain an accurate

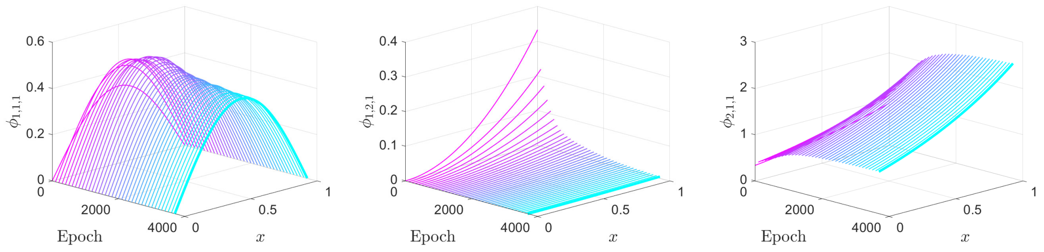


Figure 3: Learning activation functions $\phi_{1,1,1}$, $\phi_{1,2,1}$, and $\phi_{2,1,1}$ for h

representation, and converge to an optimal solution in Figure 2, without the expense of a large number of parameters. We can see that the surface (moving from transparent to opaque surfaces) converges to a surface that is closest to the representation (as seen by the decreased cost). This demonstrates that, regardless of the initialization of $S_{l,i,j}$ and $\phi_{0,l,i,j}$, we are able to find a local minimum within the set of possible functions that can be represented by the model. Furthermore, we see the convergence for a selection of neural activation functions in Figure 3. In the network, the input of $\phi_{2,1,1}$ is found by the output $\phi_{1,1,1} + \phi_{1,2,1}$ (similarly for other selections of neural activation functions). Despite the dependence of the neural activation functions on each other, through the definition of the neural activation function, each function is represented through a simple analytical expression and is guaranteed to be a bounded function on the compact set $[a, b]$, providing robustness throughout the training process.

5. Conclusions and Extensions

We have explored an exciting frontier in machine learning: *leveraging the fundamental KA Representation Theorem to train neural networks directly within an infinite-dimensional framework*. By structuring activation functions through infinite-dimensional optimization, we learn expressive and interpretable nonlinear components with as little as a single parameter. We have compared our results with popular deep learning techniques, such as the MLP and KAN, and demonstrated that with a suitable selection of trainable parameters, we can outperform those methods with the number of parameters reduced by an order of magnitude, whilst maintaining an analytical and interpretable expression. *De facto*, this framework unlocks unprecedented robustness and precision, empowering neural networks to seamlessly integrate analytically expressed functions and revolutionize their application across complex domains.

Selecting, or even learning, the functional spaces that are both accurate and easy to design remains an open challenge. Indeed, the selection of S and ϕ_0 parallels physics-informed machine learning (Brunton et al., 2016; Rigas et al., 2024) and inverse optimization/reinforcement learning (Ahuja and Orlin, 2001; Ng and Russell, 2000), as we find a cost function S and initial condition ϕ_0 that encapsulate task-specific representations. Such insights into the cost function can provide valuable interpretations of the underlying *value structure* of the network and establish a clear connection with the global objective J. In turn, functional spaces pave the way for robust theoretical guarantees and provide deeper insights into the behavior of networks, particularly when dealing with continuous data or functions rather than discrete points. This would ensure a deeper understanding of how small changes in the input function affect the network output, handle functions with certain smoothness properties — crucial in applications such as learning for control (Zoppoli et al., 2020) — and help to analyze and design networks potentially taming the *curse of dimensionality* (Barron, 1993).

Acknowledgments

This work has been supported by the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 739551 (KIOS CoE).

References

Ravindra K. Ahuja and James B. Orlin. Inverse optimization. *Operations Research*, 49(5):771–783, 2001.

Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, and Roberto Prevete. A survey on modern trainable activation functions. *Neural Networks*, 138:14–32, 2021.

Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.

Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

Mingyu Cai, Mohammadhosein Hasanbeig, Shaoping Xiao, Alessandro Abate, and Zhen Kan. Modular deep reinforcement learning for continuous motion planning with temporal logic. *IEEE Robotics and Automation Letters*, 6(4):7973–7980, 2021.

Daniele Fakhoury, Emanuele Fakhoury, and Hendrik Speleers. Exsplinet: An interpretable and expressive spline-based neural network. *Neural Networks*, 152:332–346, 2022.

Federico Girosi and Tomaso Poggio. Representation properties of networks: Kolmogorov's theorem is irrelevant. *Neural Computation*, 1(4):465–469, 1989.

Oscar Gonzalez. Time integration and discrete Hamiltonian systems. *Journal of Nonlinear Science*, 6:449–467, 1996.

Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. volume 28 of *Proceedings of Machine Learning Research*, pages 1319–1327, 2013.

Mohit Goyal, Rajan Goyal, and Brejesh Lall. Learning activation functions: A new paradigm for understanding neural networks. *arXiv preprint arXiv:1906.09529*, 2019.

David Hilbert. Mathematical problems. *Bulletin of the American Mathematical Society*, 8(10): 437–479, 1902.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Jürgen Braun. *An Application of Kolmogorov's Superposition Theorem to Function Reconstruction in Higher Dimensions*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2009.

Benjamin C Koenig, Suyong Kim, and Sili Deng. KAN-ODEs: Kolmogorov–Arnold network ordinary differential equations for learning dynamical systems and hidden physics. *Computer Methods in Applied Mechanics and Engineering*, 432:117397, 2024.

Andrei Nikolaivich Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 114:953–956, 1957.

Věra Kůrková. Kolmogorov's theorem is relevant. *Neural Computation*, 3(4):617–622, 1991.

Věra Kůrková. Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, 5(3): 501–506, 1992.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Ji-Nan Lin and Rolf Unbehauen. On the realization of a Kolmogorov network. *Neural Computation*, 5(1):18–20, 1993.

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. KAN: Kolmogorov-Arnold networks. *arXiv preprint arXiv:2404.19756*, 2024.

Hadrien Montanelli and Haizhao Yang. Error bounds for deep ReLU networks using the Kolmogorov–Arnold superposition theorem. *Neural Networks*, 129:1–6, 2020.

Alessio Moreschini, Gökhan Göksu, and Thomas Parisini. Fréchet discrete gradient and Hessian operators on infinite-dimensional spaces. *IFAC-PapersOnLine*, 58(5):78–83, 2024a.

Alessio Moreschini, Salvatore Monaco, and Dorothée Normand-Cyrot. Dirac structures for a class of port-Hamiltonian systems in discrete time. *IEEE Transactions on Automatic Control*, 69(3): 1999–2006, 2024b.

Alessio Moreschini, Matteo Scandella, and Thomas Parisini. Non-convex learning with guaranteed convergence: Perspectives on stochastic optimal control. In *63rd IEEE Conference on Decision and Control (CDC)*, pages 6002–6009, 2024c.

Alessio Moreschini, Michelangelo Bin, Alessandro Astolfi, and Thomas Parisini. A generalized passivity theory over abstract time domains. *IEEE Transactions on Automatic Control*, 70(1): 2–17, 2025.

Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning (ICML)*, page 663–670. Morgan Kaufmann Publishers Inc., 2000.

Naoki Nishikawa, Taiji Suzuki, Atsushi Nitanda, and Denny Wu. Two-layer neural network on infinite dimensional data: global optimization guarantee in the mean-field regime. In *Advances in Neural Information Processing Systems*, volume 35, pages 32612–32623. Curran Associates, Inc., 2022.

Andrew Polar and Michael Poluektov. A deep machine learning algorithm for construction of the Kolmogorov–Arnold representation. *Engineering Applications of Artificial Intelligence*, 99: 104137, 2021.

Spyros Rigas, Michalis Papachristou, Theofilos Papadopoulos, Fotios Anagnostopoulos, and Georgios Alexandridis. Adaptive training of grid-dependent physics-informed Kolmogorov-Arnold networks. *IEEE Access*, 2024.

Fabrice Rossi and Brieuc Conan-Guez. Functional multi-layer perceptron: a non-linear tool for functional data analysis. *Neural Networks*, 18(1):45–60, 2005.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Johannes Schmidt-Hieber. The Kolmogorov–Arnold representation theorem revisited. *Neural Networks*, 137:119–126, 2021.

David A. Sprecher and Sorin Draghici. Space-filling curves and Kolmogorov superposition-based neural networks. *Neural Networks*, 15(1):57–67, 2002.

Maxwell Stinchcombe and Halbert White. Approximating and learning unknown mappings using multilayer feedforward networks with bounded weights. In *1990 International Joint Conference on Neural Networks (IJCNN)*, volume 3, pages 7–16, 1990.

Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 2nd edition, 2000.

Cong Wang and David J. Hill. Learning from neural control. *IEEE Transactions on Neural Networks*, 17(1):130–146, 2006.

Zuxun Xiong, Han Wang, Liqun Zhao, and Antonis Papachristodoulou. Data-driven stable neural feedback loop design. *arXiv preprint arXiv:2405.02100*, 2024.

Runpeng Yu, Weihao Yu, and Xinchao Wang. KAN or MLP: A fairer comparison. *arXiv preprint arXiv:2407.16674*, 2024.

Brosnan Yuen, Minh Tu Hoang, Xiaodai Dong, and Tao Lu. Universal activation function for machine learning. *Scientific Reports*, 11(1):18757, 2021.

Riccardo Zoppoli, Marcello Sanguineti, Giorgio Gnecco, and Thomas Parisini. *Neural Approximations for Optimal Control and Decision*. Springer, 2020.