KAN: Kolmogorov–Arnold Networks

Ziming Liu[1,4*] **Yixuan Wang**[2] **Sachin Vaidya**[1] **Fabian Ruehle**[3,4]
James Halverson[3,4] **Marin Soljačić**[1,4] **Thomas Y. Hou**[2] **Max Tegmark**[1,4]
[1] Massachusetts Institute of Technology
[2] California Institute of Technology
[3] Northeastern University
[4] The NSF Institute for Artificial Intelligence and Fundamental Interactions

Abstract

Inspired by the Kolmogorov-Arnold representation theorem, we propose Kolmogorov-Arnold Networks (KANs) as promising alternatives to Multi-Layer Perceptrons (MLPs). While MLPs have *fixed* activation functions on *nodes* ("neurons"), KANs have *learnable* activation functions on *edges* ("weights"). KANs have no linear weights at all – every weight parameter is replaced by a univariate function parametrized as a spline. We show that this seemingly simple change makes KANs outperform MLPs in terms of accuracy and interpretability, on small-scale AI + Science tasks. For accuracy, smaller KANs can achieve comparable or better accuracy than larger MLPs in function fitting tasks. Theoretically and empirically, KANs possess faster neural scaling laws than MLPs. For interpretability, KANs can be intuitively visualized and can easily interact with human users. Through two examples in mathematics and physics, KANs are shown to be useful "collaborators" helping scientists (re)discover mathematical and physical laws. In summary, KANs are promising alternatives for MLPs, opening opportunities for further improving today's deep learning models which rely heavily on MLPs.

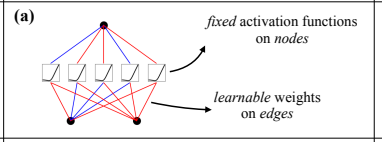
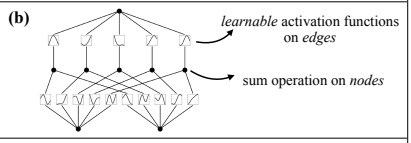
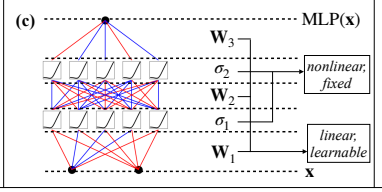
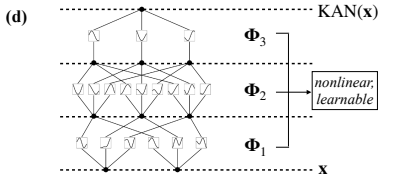
Model	**Multi-Layer Perceptron (MLP)**	**Kolmogorov-Arnold Network (KAN)**
Theorem	**Universal Approximation Theorem**	**Kolmogorov-Arnold Representation Theorem**
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$
Model (Shallow)		
Formula (Deep)	$\mathrm{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\mathrm{KAN}(\mathbf{x}) = (\mathbf{\Phi}_3 \circ \mathbf{\Phi}_2 \circ \mathbf{\Phi}_1)(\mathbf{x})$
Model (Deep)		

Figure 0.1: Multi-Layer Perceptrons (MLPs) vs. Kolmogorov-Arnold Networks (KANs)

*zmliu@mit.edu

Preprint. Under review.

1 Introduction

Multi-layer perceptrons (MLPs) [1, 2, 3], also known as fully-connected feedforward neural networks, are foundational building blocks of today's deep learning models. The importance of MLPs can never be overstated, since they are the default models in machine learning for approximating nonlinear functions, due to their expressive power guaranteed by the universal approximation theorem [3]. However, are MLPs the best nonlinear regressors we can build? Despite the prevalent use of MLPs, they have significant drawbacks. In transformers [4] for example, MLPs consume almost all non-embedding parameters and are typically less interpretable (relative to attention layers) without post-analysis tools [5].

We propose a promising alternative to MLPs, called Kolmogorov-Arnold Networks (KANs). Whereas MLPs are inspired by the universal approximation theorem, KANs are inspired by the Kolmogorov-Arnold representation theorem [6, 7, 8]. Like MLPs, KANs have fully-connected structures. However, while MLPs place fixed activation functions on *nodes* ("neurons"), KANs place learnable activation functions on *edges* ("weights"), as illustrated in Figure 0.1. As a result, KANs have no linear weight matrices at all: instead, each weight parameter is replaced by a learnable 1D function parametrized as a spline. KANs' nodes simply sum incoming signals without applying any non-linearities. One might worry that KANs are hopelessly expensive, since each MLP's weight parameter becomes KAN's spline function. Fortunately, KANs usually allow much smaller computation graphs than MLPs.

Unsurprisingly, the possibility of using Kolmogorov-Arnold representation theorem to build neural networks has been studied [9, 10, 11, 12, 13, 14, 15, 16]. However, most work has stuck with the original depth-2 width-$(2n+1)$ representation, and many did not have the chance to leverage more modern techniques (e.g., back propagation) to train the networks. In [12], a depth-2 width-$(2n+1)$ representation was investigated, with breaking of the curse of dimensionality observed both empirically and with an approximation theory given compositional structures of the function. Our contribution lies in generalizing the original Kolmogorov-Arnold representation to arbitrary widths and depths, revitalizing and contextualizing it in today's deep learning world, as well as using extensive empirical experiments to highlight its potential for AI + Science due to its accuracy and interpretability.

Despite their elegant mathematical interpretation, KANs are nothing more than combinations of splines and MLPs, leveraging their respective strengths and avoiding their respective weaknesses. Splines are accurate for low-dimensional functions, easy to adjust locally, and able to switch between different resolutions. However, splines have a serious curse of dimensionality (COD) problem, because of their inability to exploit compositional structures. MLPs, on the other hand, suffer less from COD thanks to their feature learning, but are less accurate than splines in low dimensions, because of their inability to optimize univariate functions. The link between MLPs using ReLU-k as activation functions and splines have been established in [17, 18]. To learn a function accurately, a model should not only learn the compositional structure (*external* degrees of freedom), but should also approximate well the univariate functions (*internal* degrees of freedom). KANs are such models since they have MLPs on the outside and splines on the inside. As a result, KANs can not only learn features (thanks to their external similarity to MLPs), but can also optimize these learned features to great accuracy (thanks to their internal similarity to splines). For example, given a high dimensional function

$$f(x_1, \cdots, x_N) = \exp\left(\frac{1}{N}\sum_{i=1}^{N}\sin^2(x_i)\right), \tag{1.1}$$

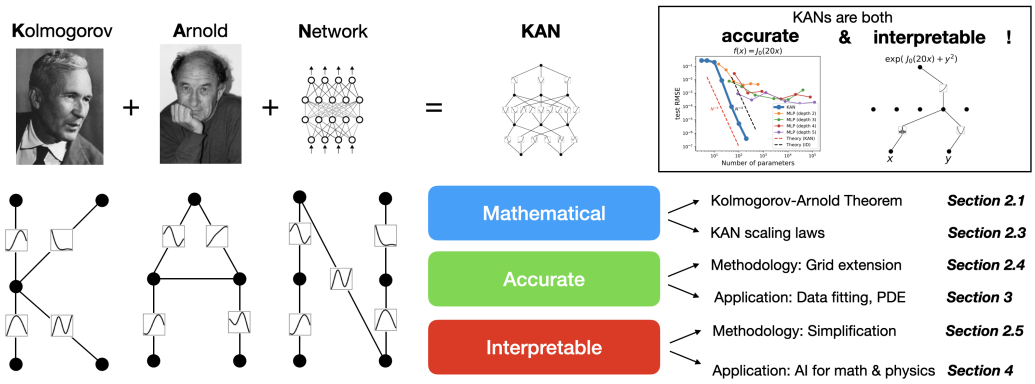


Figure 2.1: Our proposed Kolmogorov-Arnold networks are in honor of two great late mathematicians, Andrey Kolmogorov and Vladimir Arnold. KANs are mathematically sound, accurate and interpretable.

splines would fail for large N due to COD; MLPs can potentially learn the the generalized additive structure, but they are very inefficient for approximating the exponential and sine functions with say, ReLU activations. In contrast, KANs can learn both the compositional structure and the univariate functions quite well, hence outperforming MLPs by a large margin (see Figure 3.1).

Throughout this paper, we will use extensive numerical experiments to show that KANs can lead to accuracy and interpretability improvement over MLPs, at least on small-scale AI + Science tasks. The organization of the paper is illustrated in Figure 2.1. In Section 2, we introduce the KAN architecture and its mathematical foundation, introduce network simplification techniques to make KANs interpretable, and introduce a grid extension technique to make KANs more accurate. In Section 3, we show that KANs are more accurate than MLPs for data fitting: KANs can beat the curse of dimensionality when there is a compositional structure in data, achieving better scaling laws than MLPs. We also demonstrate the potential of KANs in PDE solving via a simple example of the Poisson equation. In Section 4, we show that KANs are interpretable and can be used for scientific discoveries. We use two examples from mathematics (knot theory) and physics (Anderson localization) to demonstrate that KANs can be helpful "collaborators" for scientists to (re)discover math and physical laws. Section 5 summarizes related works. In Section 6, we conclude by discussing broad impacts and future directions. Codes are available at `https://github.com/KindXiaoming/pykan` and can also be installed via `pip install pykan`.

2 Kolmogorov–Arnold Networks (KAN)

Multi-Layer Perceptrons (MLPs) are inspired by the universal approximation theorem. We instead focus on the Kolmogorov-Arnold representation theorem, which can be realized by a new type of neural network called Kolmogorov-Arnold networks (KAN). We review the Kolmogorov-Arnold theorem in Section 2.1, to inspire the design of Kolmogorov-Arnold Networks in Section 2.2. In Section 2.3, we provide theoretical guarantees for the expressive power of KANs and their neural scaling laws, relating them to existing approximation and generalization theories in the literature. In Section 2.4, we propose a grid extension technique to make KANs increasingly more accurate. In Section 2.5, we propose simplification techniques to make KANs interpretable.

2.1 Kolmogorov-Arnold Representation theorem

Vladimir Arnold and Andrey Kolmogorov established that if f is a multivariate continuous function on a bounded domain, then f can be written as a finite composition of continuous functions of a

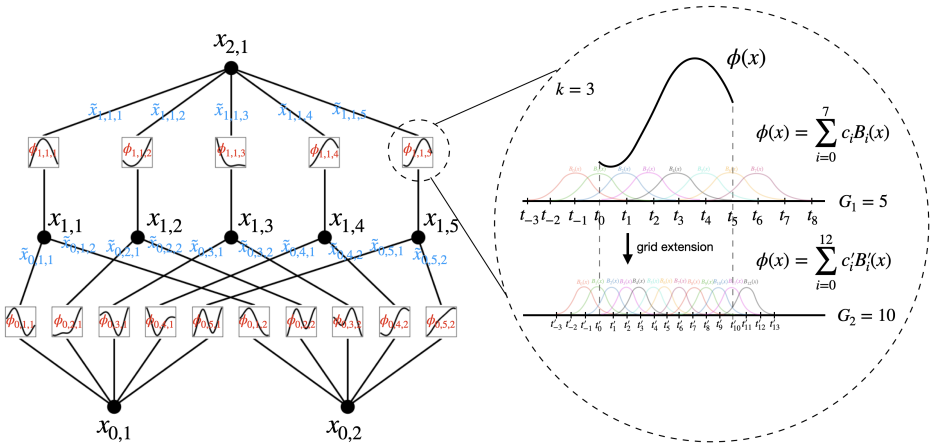


Figure 2.2: Left: Notations of activations that flow through the network. Right: an activation function is parameterized as a B-spline, which allows switching between coarse-grained and fine-grained grids.

single variable and the binary operation of addition. More specifically, for a smooth $f : [0, 1]^n \to \mathbb{R}$,

$$f(\mathbf{x}) = f(x_1, \cdots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right), \tag{2.1}$$

where $\phi_{q,p} : [0, 1] \to \mathbb{R}$ and $\Phi_q : \mathbb{R} \to \mathbb{R}$. In a sense, they showed that the only true multivariate function is addition, since every other function can be written using univariate functions and sum. One might naively consider this great news for machine learning: learning a high-dimensional function boils down to learning a polynomial number of 1D functions. However, these 1D functions can be non-smooth and even fractal, so they may not be learnable in practice [19, 20]. Because of this pathological behavior, the Kolmogorov-Arnold representation theorem was basically sentenced to death in machine learning, regarded as theoretically sound but practically useless [19, 20].

However, we are more optimistic about the usefulness of the Kolmogorov-Arnold theorem for machine learning. First of all, we need not stick to the original Eq. (2.1) which has only two-layer nonlinearities and a small number of terms ($2n + 1$) in the hidden layer: we will generalize the network to arbitrary widths and depths. Secondly, most functions in science and daily life are often smooth and have sparse compositional structures, potentially facilitating smooth Kolmogorov-Arnold representations. The philosophy here is close to the mindset of physicists, who often care more about typical cases rather than worst cases. After all, our physical world and machine learning tasks must have structures to make physics and machine learning useful or generalizable at all [21].

2.2 KAN architecture

Suppose we have a supervised learning task consisting of input-output pairs $\{\mathbf{x}_i, y_i\}$, where we want to find f such that $y_i \approx f(\mathbf{x}_i)$ for all data points. Eq. (2.1) implies that we are done if we can find appropriate univariate functions $\phi_{q,p}$ and Φ_q. This inspires us to design a neural network which explicitly parametrizes Eq. (2.1). Since all functions to be learned are univariate functions, we can parametrize each 1D function as a B-spline curve, with learnable coefficients of local B-spline basis functions (see Figure 2.2 right). Now we have a prototype of KAN, whose computation graph is exactly specified by Eq. (2.1) and illustrated in Figure 0.1 (b) (with the input dimension $n = 2$), appearing as a two-layer neural network with activation functions placed on edges instead of nodes (simple summation is performed on nodes), and with width $2n + 1$ in the middle layer.

As mentioned, such a network is known to be too simple to approximate any function arbitrarily well in practice with smooth splines! We therefore generalize our KAN to be wider and deeper. It is not immediately clear how to make KANs deeper, since Kolmogorov-Arnold representations correspond to two-layer KANs. To the best of our knowledge, there is not yet a "generalized" version of the theorem that corresponds to deeper KANs.

The breakthrough occurs when we notice the analogy between MLPs and KANs. In MLPs, once we define a layer (which is composed of a linear transformation and nonlinearties), we can stack more layers to make the network deeper. To build deep KANs, we should first answer: "what is a KAN layer?" It turns out that a KAN layer with $n_{\rm in}$-dimensional inputs and $n_{\rm out}$-dimensional outputs can be defined as a matrix of 1D functions

$$\mathbf{\Phi} = \{\phi_{q,p}\}, \qquad p = 1, 2, \cdots, n_{\rm in}, \qquad q = 1, 2 \cdots, n_{\rm out}, \tag{2.2}$$

where the functions $\phi_{q,p}$ have trainable parameters, as detaild below. In the Kolmogov-Arnold theorem, the inner functions form a KAN layer with $n_{\rm in} = n$ and $n_{\rm out} = 2n+1$, and the outer functions form a KAN layer with $n_{\rm in} = 2n+1$ and $n_{\rm out} = 1$. So the Kolmogorov-Arnold representations in Eq. (2.1) are simply compositions of two KAN layers. Now it becomes clear what it means to have deeper Kolmogorov-Arnold representations: simply stack more KAN layers!

Let us introduce some notation. This paragraph will be a bit technical, but readers can refer to Figure 2.2 (left) for a concrete example and intuitive understanding. The shape of a KAN is represented by an integer array

$$[n_0, n_1, \cdots, n_L], \tag{2.3}$$

where n_i is the number of nodes in the $i^{\rm th}$ layer of the computational graph. We denote the $i^{\rm th}$ neuron in the $l^{\rm th}$ layer by (l, i), and the activation value of the (l, i)-neuron by $x_{l,i}$. Between layer l and layer $l+1$, there are $n_l n_{l+1}$ activation functions: the activation function that connects (l, i) and $(l+1, j)$ is denoted by

$$\phi_{l,j,i}, \quad l = 0, \cdots, L-1, \quad i = 1, \cdots, n_l, \quad j = 1, \cdots, n_{l+1}. \tag{2.4}$$

The pre-activation of $\phi_{l,j,i}$ is simply $x_{l,i}$; the post-activation of $\phi_{l,j,i}$ is denoted by $\tilde{x}_{l,j,i} \equiv \phi_{l,j,i}(x_{l,i})$. The activation value of the $(l+1, j)$ neuron is simply the sum of all incoming post-activations:

$$x_{l+1,j} = \sum_{i=1}^{n_l} \tilde{x}_{l,j,i} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}), \qquad j = 1, \cdots, n_{l+1}. \tag{2.5}$$

In matrix form, this reads

$$\mathbf{x}_{l+1} = \underbrace{\begin{pmatrix} \phi_{l,1,1}(\cdot) & \phi_{l,1,2}(\cdot) & \cdots & \phi_{l,1,n_l}(\cdot) \\ \phi_{l,2,1}(\cdot) & \phi_{l,2,2}(\cdot) & \cdots & \phi_{l,2,n_l}(\cdot) \\ \vdots & \vdots & & \vdots \\ \phi_{l,n_{l+1},1}(\cdot) & \phi_{l,n_{l+1},2}(\cdot) & \cdots & \phi_{l,n_{l+1},n_l}(\cdot) \end{pmatrix}}_{\mathbf{\Phi}_l} \mathbf{x}_l, \tag{2.6}$$

where $\mathbf{\Phi}_l$ is the function matrix corresponding to the $l^{\rm th}$ KAN layer. A general KAN network is a composition of L layers: given an input vector $\mathbf{x}_0 \in \mathbb{R}^{n_0}$, the output of KAN is

$$\mathrm{KAN}(\mathbf{x}) = (\mathbf{\Phi}_{L-1} \circ \mathbf{\Phi}_{L-2} \circ \cdots \circ \mathbf{\Phi}_1 \circ \mathbf{\Phi}_0)\mathbf{x}. \tag{2.7}$$

We can also rewrite the above equation to make it more analogous to Eq. (2.1), assuming output dimension $n_L = 1$, and define $f(\mathbf{x}) \equiv \mathrm{KAN}(\mathbf{x})$:

$$f(\mathbf{x}) = \sum_{i_{L-1}=1}^{n_{L-1}} \phi_{L-1,i_L,i_{L-1}} \left(\sum_{i_{L-2}=1}^{n_{L-2}} \cdots \left(\sum_{i_2=1}^{n_2} \phi_{2,i_3,i_2} \left(\sum_{i_1=1}^{n_1} \phi_{1,i_2,i_1} \left(\sum_{i_0=1}^{n_0} \phi_{0,i_1,i_0}(x_{i_0}) \right) \right) \right) \cdots \right), \tag{2.8}$$

which is quite cumbersome. In contrast, our abstraction of KAN layers and their visualizations are cleaner and intuitive. The original Kolmogorov-Arnold representation Eq. (2.1) corresponds to a 2-Layer KAN with shape $[n, 2n+1, 1]$. Notice that all the operations are differentiable, so we can train KANs with back propagation. For comparison, an MLP can be written as interleaving of affine transformations $\mathbf{W}$ and non-linearities σ:

$$\mathrm{MLP}(\mathbf{x}) = (\mathbf{W}_{L-1} \circ \sigma \circ \mathbf{W}_{L-2} \circ \sigma \circ \cdots \circ \mathbf{W}_1 \circ \sigma \circ \mathbf{W}_0)\mathbf{x}. \tag{2.9}$$

It is clear that MLPs treat linear transformations and nonlinearities separately as $\mathbf{W}$ and σ, while KANs treat them all together in $\mathbf{\Phi}$. In Figure 0.1 (c) and (d), we visualize a three-layer MLP and a three-layer KAN, to clarify their differences.

Implementation details. Although a KAN layer Eq. (2.5) looks extremely simple, it is non-trivial to make it well optimizable. The key tricks are:

(1) Residual activation functions. We include a basis function $b(x)$ (similar to residual connections) such that the activation function $\phi(x)$ is the sum of the basis function $b(x)$ and the spline function:

$$\phi(x) = w_b b(x) + w_s \mathrm{spline}(x). \tag{2.10}$$

We set

$$b(x) = \mathrm{silu}(x) = x/(1 + e^{-x}) \tag{2.11}$$

in most cases. $\mathrm{spline}(x)$ is parametrized as a linear combination of B-splines such that

$$\mathrm{spline}(x) = \sum_i c_i B_i(x) \tag{2.12}$$

where c_is are trainable (see Figure 2.2 for an illustration). In principle w_b and w_s are redundant since it can be absorbed into $b(x)$ and $\mathrm{spline}(x)$. However, we still include these factors (which are by default trainable) to better control the overall magnitude of the activation function.

(2) Initialization scales. Each activation function is initialized to have $w_s = 1$ and $\mathrm{spline}(x) \approx 0$ [2]. w_b is initialized according to the Xavier initialization, which has been used to initialize linear layers in MLPs.

(3) Update of spline grids. We update each grid on the fly according to its input activations, to address the issue that splines are defined on bounded regions but activation values can evolve out of the fixed region during training [3].

Parameter count. For simplicity, let us assume a network

(1) of depth L,

[2] This is done by drawing B-spline coefficients $c_i \sim \mathcal{N}(0, \sigma^2)$ with a small σ, typically we set $\sigma = 0.1$.

[3] Other possibilities are: (a) the grid is learnable with gradient descent, e.g., [22]; (b) use normalization such that the input range is fixed. We tried (b) at first but its performance is inferior to our current approach.

(2) with layers of equal width $n_0 = n_1 = \cdots = n_L = N$,

(3) with each spline of order k (usually $k = 3$) on G intervals (for $G + 1$ grid points).

Then there are in total $O(N^2L(G + k)) \sim O(N^2LG)$ parameters. In contrast, an MLP with depth L and width N only needs $O(N^2L)$ parameters, which appears to be more efficient than KAN. Fortunately, KANs usually require much smaller N than MLPs, which not only saves parameters, but also achieves better generalization (see e.g., Figure 3.1 and 3.3) and facilitates interpretability. We remark that for 1D problems, we can take $N = L = 1$ and the KAN network in our implementation is nothing but a spline approximation. For higher dimensions, we characterize the generalization behavior of KANs with a theorem below.

2.3 KAN's Approximation Abilities and Scaling Laws

Recall that in Eq. (2.1), the 2-Layer width-$(2n + 1)$ representation may be non-smooth. However, deeper representations may bring the advantages of smoother activations. For example, the 4-variable function

$$f(x_1, x_2, x_3, x_4) = \exp\left(\sin(x_1^2 + x_2^2) + \sin(x_3^2 + x_4^2)\right) \tag{2.13}$$

can be smoothly represented by a $[4, 2, 1, 1]$ KAN which is 3-Layer, but may not admit a 2-Layer KAN with smooth activations. To facilitate an approximation analysis, we still assume smoothness of activations, but allow the representations to be arbitrarily wide and deep, as in Eq. (2.7). To emphasize the dependence of our KAN on the finite set of grid points, we use $\mathbf{\Phi}_l^G$ and $\Phi_{l,i,j}^G$ below to replace the notation $\mathbf{\Phi}_l$ and $\Phi_{l,i,j}$ used in Eq. (2.5) and (2.6).

Theorem 2.1 (Approximation theory, KAT)**.** *Let $\mathbf{x} = (x_1, x_2, \cdots, x_n)$. Suppose that a function $f(\mathbf{x})$ admits a representation*

$$f = (\mathbf{\Phi}_{L-1} \circ \mathbf{\Phi}_{L-2} \circ \cdots \circ \mathbf{\Phi}_1 \circ \mathbf{\Phi}_0)\mathbf{x}\,, \tag{2.14}$$

as in Eq. (2.7)*, where each one of the $\Phi_{l,i,j}$ are $(k + 1)$-times continuously differentiable. Then there exists a constant C depending on f and its representation, such that we have the following approximation bound in terms of the grid size G: there exist k-th order B-spline functions $\Phi_{l,i,j}^G$ such that for any $0 \le m \le k$, we have the bound*

$$\|f - (\mathbf{\Phi}_{L-1}^G \circ \mathbf{\Phi}_{L-2}^G \circ \cdots \circ \mathbf{\Phi}_1^G \circ \mathbf{\Phi}_0^G)\mathbf{x}\|_{C^m} \le CG^{-k-1+m}\,. \tag{2.15}$$

Here we adopt the notation of C^m-norm measuring the magnitude of derivatives up to order m:

$$\|g\|_{C^m} = \max_{|\beta|\le m} \sup_{x\in[0,1]^n} \left|D^\beta g(x)\right|.$$

Proof. By the classical 1D B-spline theory [23] and the fact that $\Phi_{l,i,j}$ as continuous functions can be uniformly bounded on a bounded domain, we know that there exist finite-grid B-spline functions $\Phi_{l,i,j}^G$ such that for any $0 \le m \le k$,

$$\|(\Phi_{l,i,j} \circ \mathbf{\Phi}_{l-1} \circ \mathbf{\Phi}_{l-2} \circ \cdots \circ \mathbf{\Phi}_1 \circ \mathbf{\Phi}_0)\mathbf{x} - (\Phi_{l,i,j}^G \circ \mathbf{\Phi}_{l-1} \circ \mathbf{\Phi}_{l-2} \circ \cdots \circ \mathbf{\Phi}_1 \circ \mathbf{\Phi}_0)\mathbf{x}\|_{C^m} \le CG^{-k-1+m}\,,$$

with a constant C independent of G. We fix those B-spline approximations. Therefore we have that the residue R_l defined via

$$R_l \coloneqq (\mathbf{\Phi}_{L-1}^G \circ \cdots \circ \mathbf{\Phi}_{l+1}^G \circ \mathbf{\Phi}_l \circ \mathbf{\Phi}_{l-1} \circ \cdots \circ \mathbf{\Phi}_0)\mathbf{x} - (\mathbf{\Phi}_{L-1}^G \circ \cdots \circ \mathbf{\Phi}_{l+1}^G \circ \mathbf{\Phi}_l^G \circ \mathbf{\Phi}_{l-1} \circ \cdots \circ \mathbf{\Phi}_0)\mathbf{x}$$

satisfies

$$\|R_l\|_{C^m} \leq CG^{-k-1+m}\,,$$

with a constant independent of G. Finally notice that

$$f - (\mathbf{\Phi}_{L-1}^G \circ \mathbf{\Phi}_{L-2}^G \circ \cdots \circ \mathbf{\Phi}_1^G \circ \mathbf{\Phi}_0^G)\mathbf{x} = R_{L-1} + R_{L-2} + \cdots + R_1 + R_0\,,$$

we know that (2.15) holds. □

We know that asymptotically, provided that the assumption in Theorem 2.1 holds, KANs with finite grid size can approximate the function well with a residue rate **independent of the dimension, hence beating curse of dimensionality!** This comes naturally since we only use splines to approximate 1D functions. In particular, for $m = 0$, we recover the accuracy in L^∞ norm, which in turn provides a bound of RMSE on the finite domain, which gives a scaling exponent $k + 1$. Of course, the constant C is dependent on the representation; hence it will depend on the dimension. We will leave the discussion of the dependence of the constant on the dimension as a future work.

We remark that although the Kolmogorov-Arnold theorem Eq. (2.1) corresponds to a KAN representation with shape $[d, 2d+1, 1]$, its functions are not necessarily smooth. On the other hand, if we are able to identify a smooth representation (maybe at the cost of extra layers or making the KAN wider than the theory prescribes), then Theorem 2.1 indicates that we can beat the curse of dimensionality (COD). This should not come as a surprise since we can inherently learn the structure of the function and make our finite-sample KAN approximation interpretable.

Neural scaling laws: comparison to other theories. Neural scaling laws are the phenomenon where test loss decreases with more model parameters, i.e., $\ell \propto N^{-\alpha}$ where ℓ is test RMSE, N is the number of parameters, and α is the scaling exponent. A larger α promises more improvement by simply scaling up the model. Different theories have been proposed to predict α. Sharma & Kaplan [24] suggest that α comes from data fitting on an input manifold of intrinsic dimensionality d. If the model function class is piecewise polynomials of order k ($k = 1$ for ReLU), then the standard approximation theory implies $\alpha = (k + 1)/d$ from the approximation theory. This bound suffers from the curse of dimensionality, so people have sought other bounds independent of d by leveraging compositional structures. In particular, Michaud et al. [25] considered computational graphs that only involve unary (e.g., squared, sine, exp) and binary ($+$ and $\times$) operations, finding $\alpha = (k + 1)/d^* = (k + 1)/2$, where $d^* = 2$ is the maximum arity. Poggio et al. [19] leveraged the idea of compositional sparsity and proved that given function class W_m (function whose derivatives are continuous up to m-th order), one needs $N = O(\epsilon^{-\frac{2}{m}})$ number of parameters to achieve error ϵ, which is equivalent to $\alpha = \frac{m}{2}$. Our approach, which assumes the existence of smooth Kolmogorov-Arnold representations, decomposes the high-dimensional function into several 1D functions, giving $\alpha = k+1$ (where k is the piecewise polynomial order of the splines). We choose $k = 3$ cubic splines so $\alpha = 4$ which is the largest and best scaling exponent compared to other works. We will show in Section 3.1 that this bound $\alpha = 4$ can in fact be achieved empirically with KANs, while previous work [25] reported that MLPs have problems even saturating slower bounds (e.g., $\alpha = 1$) and plateau quickly. Of course, we can increase k to match the smoothness of functions, but too high k might be too oscillatory, leading to optimization issues.

Comparison between KAT and UAT. The power of fully-connected neural networks is justified by the universal approximation theorem (UAT), which states that given a function and error tolerance $\epsilon > 0$, a two-layer network with $k > N(\epsilon)$ neurons can approximate the function within error ϵ. However, the UAT guarantees no bound for how $N(\epsilon)$ scales with ϵ. Indeed, it suffers from the COD, and N has been shown to grow exponentially with d in some cases [21]. The difference between

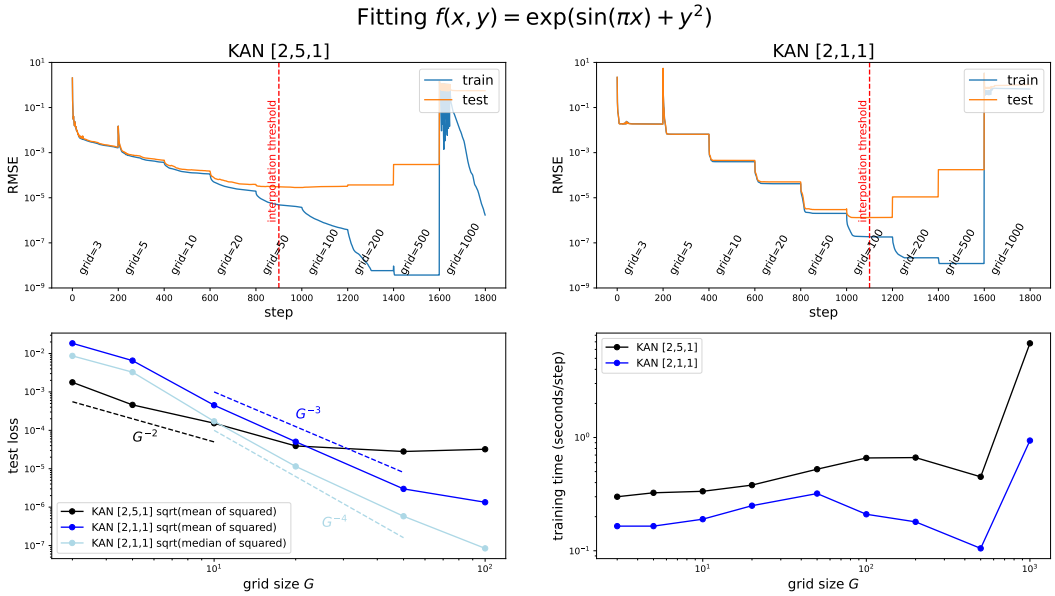


Figure 2.3: We can make KANs more accurate by grid extension (fine-graining spline grids). Top left (right): training dynamics of a $[2,5,1]$ ($[2,1,1]$) KAN. Both models display staircases in their loss curves, i.e., loss suddenly drops then plateaus after grid extension. Bottom left: test RMSE follows scaling laws against grid size G. Bottom right: training time scales favorably with grid size G.

KAT and UAT is a consequence that KANs take advantage of the intrinsically low-dimensional representation of the function while MLPs do not. In KAT, we highlight quantifying the approximation error in the compositional space. In the literature, generalization error bounds, taking into account finite samples of training data, for a similar space have been studied for regression problems; see [26, 27], and also specifically for MLPs with ReLU activations [28]. On the other hand, for general function spaces like Sobolev or Besov spaces, the nonlinear n-widths theory [29, 30, 31] indicates that we can never beat the curse of dimensionality, while MLPs with ReLU activations can achieve the tight rate [32, 33, 34]. This fact again motivates us to consider functions of compositional structure, the much "nicer" functions that we encounter in practice and in science, to overcome the COD. Compared with MLPs, we may use a smaller architecture in practice, since we learn general nonlinear activation functions; see also [28] where the depth of the ReLU MLPs needs to reach at least $\log n$ to have the desired rate, where n is the number of samples. Indeed, we will show that KANs are nicely aligned with symbolic functions while MLPs are not.

2.4 For accuracy: Grid Extension

In principle, a spline can be made arbitrarily accurate to a target function as the grid can be made arbitrarily fine-grained. This good feature is inherited by KANs. By contrast, MLPs do not have the notion of "fine-graining". Admittedly, increasing the width and depth of MLPs can lead to improvement in performance ("neural scaling laws"). However, these neural scaling laws are slow (discussed in the last section). They are also expensive to obtain, because models of varying sizes are trained independently. By contrast, for KANs, one can first train a KAN with fewer parameters and then extend it to a KAN with more parameters by simply making its spline grids finer, without the need to retraining the larger model from scratch.

We next describe how to perform grid extension (illustrated in Figure 2.2 right), which is basically fitting a new fine-grained spline to an old coarse-grained spline. Suppose we want to approximate a 1D function f in a bounded region $[a,b]$ with B-splines of order k. A coarse-grained

grid with G_1 intervals has grid points at $\{t_0 = a, t_1, t_2, \cdots, t_{G_1} = b\}$, which is augmented to $\{t_{-k}, \cdots, t_{-1}, t_0, \cdots, t_{G_1}, t_{G_1+1}, \cdots, t_{G_1+k}\}$. There are $G_1 + k$ B-spline basis functions, with the i^{th} B-spline $B_i(x)$ being non-zero only on $[t_{-k+i}, t_{i+1}]$ $(i = 0, \cdots, G_1 + k - 1)$. Then f on the coarse grid is expressed in terms of linear combination of these B-splines basis functions $f_{\text{coarse}}(x) = \sum_{i=0}^{G_1+k-1} c_i B_i(x)$. Given a finer grid with G_2 intervals, f on the fine grid is correspondingly $f_{\text{fine}}(x) = \sum_{j=0}^{G_2+k-1} c_j' B_j'(x)$. The parameters c_j's can be initialized from the parameters c_i by minimizing the distance between $f_{\text{fine}}(x)$ to $f_{\text{coarse}}(x)$ (over some distribution of x):

$$\{c_j'\} = \underset{\{c_j'\}}{\text{argmin}} \underset{x\sim p(x)}{\mathbb{E}} \left(\sum_{j=0}^{G_2+k-1} c_j' B_j'(x) - \sum_{i=0}^{G_1+k-1} c_i B_i(x) \right)^2, \tag{2.16}$$

which can be implemented by the least squares algorithm. We perform grid extension for all splines in a KAN independently.

Toy example: staricase-like loss curves. We use a toy example $f(x, y) = \exp(\sin(\pi x) + y^2)$ to demonstrate the effect of grid extension. In Figure 2.3 (top left), we show the train and test RMSE for a $[2, 5, 1]$ KAN. The number of grid points starts as 3, increases to a higher value every 200 LBFGS steps, ending up with 1000 grid points. It is clear that every time fine graining happens, the training loss drops faster than before (except for the finest grid with 1000 points, where optimization ceases to work probably due to bad loss landscapes). However, the test losses first go down then go up, displaying a U-shape, due to the bias-variance tradeoff (underfitting vs. overfitting). We conjecture that the optimal test loss is achieved at the interpolation threshold when the number of parameters match the number of data points. Since our training samples are 1000 and the total parameters of a $[2, 5, 1]$ KAN is $15G$ (G is the number of grid intervals), we expect the interpolation threshold to be $G = 1000/15 \approx 67$, which roughly agrees with our experimentally observed value $G \sim 50$.

Small KANs generalize better. Is this the best test performance we can achieve? Notice that the synthetic task can be represented exactly by a $[2, 1, 1]$ KAN, so we train a $[2, 1, 1]$ KAN and present the training dynamics in Figure 2.3 top right. Interestingly, it can achieve even lower test losses than the $[2, 5, 1]$ KAN, with clearer staircase structures and the interpolation threshold is delayed to a larger grid size as a result of fewer parameters. This highlights a subtlety of choosing KAN architectures. If we do not know the problem structure, how can we determine the minimal KAN shape? In Section 2.5, we will propose a method to auto-discover such minimal KAN architecture via regularization and pruning.

Scaling laws: comparison with theory. We are also interested in how the test loss decreases as the number of grid parameters increases. In Figure 2.3 (bottom left), a [2,1,1] KAN scales roughly as $\text{test RMSE} \propto G^{-3}$. However, according to the Theorem 2.1, we would expect $\text{test RMSE} \propto G^{-4}$. We found that the errors across samples are not uniform. This is probably attributed to boundary effects [25]. In fact, there are a few samples that have significantly larger errors than others, making the overall scaling slow down. If we plot the square root of the *median* (not *mean*) of the squared losses, we get a scaling closer to G^{-4}. Despite this suboptimality (probably due to optimization), KANs still have much better scaling laws than MLPs, for data fitting (Figure 3.1) and PDE solving (Figure 3.3). In addition, the training time scales favorably with the number of grid points G, shown in Figure 2.3 bottom right [4].

[4] When $G = 1000$, training becomes significantly slower, which is specific to the use of the LBFGS optimizer with line search. We conjecture that the loss landscape becomes bad for $G = 1000$, so line search with trying to find an optimal step size within maximal iterations without early stopping.

External vs Internal degrees of freedom. A new concept that KANs highlights is a distinction between external versus internal degrees of freedom (parameters). The computational graph of how nodes are connected represents external degrees of freedom ("dofs"), while the grid points inside an activation function are internal degrees of freedom. KANs benefit from the fact that they have both external dofs and internal dofs. External dofs (that MLPs also have but splines do not) are responsible for learning compositional structures of multiple variables. Internal dofs (that splines also have but MLPs do not) are responsible for learning univariate functions.

2.5 For Interpretability: Simplifying KANs and Making them interactive

One loose end from the last subsection is that we do not know how to choose the KAN shape that best matches the structure of a dataset. For example, if we know that the dataset is generated via the symbolic formula $f(x,y) = \exp(\sin(\pi x)+y^2)$, then we know that a $[2,1,1]$ KAN is able to express this function. However, in practice we do not know the information a priori, so it would be nice to have approaches to determine this shape automatically. The idea is to start from a large enough KAN and train it with sparsity regularization followed by pruning. We will show that these pruned KANs are much more interpretable than non-pruned ones. To make KANs maximally interpretable, we propose a few simplification techniques in Section 2.5.1, and an example of how users can interact with KANs to make them more interpretable in Section 2.5.2.

2.5.1 Simplification techniques

1. Sparsification. For MLPs, L1 regularization of linear weights is used to favor sparsity. KANs can adapt this high-level idea, but need two modifications:

(1) There is no linear "weight" in KANs. Linear weights are replaced by learnable activation functions, so we should define the L1 norm of these activation functions.

(2) We find L1 to be insufficient for sparsification of KANs; instead an additional entropy regularization is necessary (see Appendix C for more details).

We define the L1 norm of an activation function ϕ to be its average magnitude over its N_p inputs, i.e.,

$$|\phi|_1 \equiv \frac{1}{N_p}\sum_{s=1}^{N_p}\left|\phi(x^{(s)})\right|. \tag{2.17}$$

Then for a KAN layer $\mathbf{\Phi}$ with $n_{\rm in}$ inputs and $n_{\rm out}$ outputs, we define the L1 norm of $\mathbf{\Phi}$ to be the sum of L1 norms of all activation functions, i.e.,

$$|\mathbf{\Phi}|_1 \equiv \sum_{i=1}^{n_{\rm in}}\sum_{j=1}^{n_{\rm out}}|\phi_{i,j}|_1. \tag{2.18}$$

In addition, we define the entropy of $\mathbf{\Phi}$ to be

$$S(\mathbf{\Phi}) \equiv -\sum_{i=1}^{n_{\rm in}}\sum_{j=1}^{n_{\rm out}}\frac{|\phi_{i,j}|_1}{|\mathbf{\Phi}|_1}\log\left(\frac{|\phi_{i,j}|_1}{|\mathbf{\Phi}|_1}\right). \tag{2.19}$$

The total training objective $\ell_{\rm total}$ is the prediction loss $\ell_{\rm pred}$ plus L1 and entropy regularization of all KAN layers:

$$\ell_{\rm total} = \ell_{\rm pred} + \lambda\left(\mu_1\sum_{l=0}^{L-1}|\mathbf{\Phi}_l|_1 + \mu_2\sum_{l=0}^{L-1}S(\mathbf{\Phi}_l)\right), \tag{2.20}$$

where μ_1, μ_2 are relative magnitudes usually set to $\mu_1 = \mu_2 = 1$, and λ controls overall regularization magnitude.

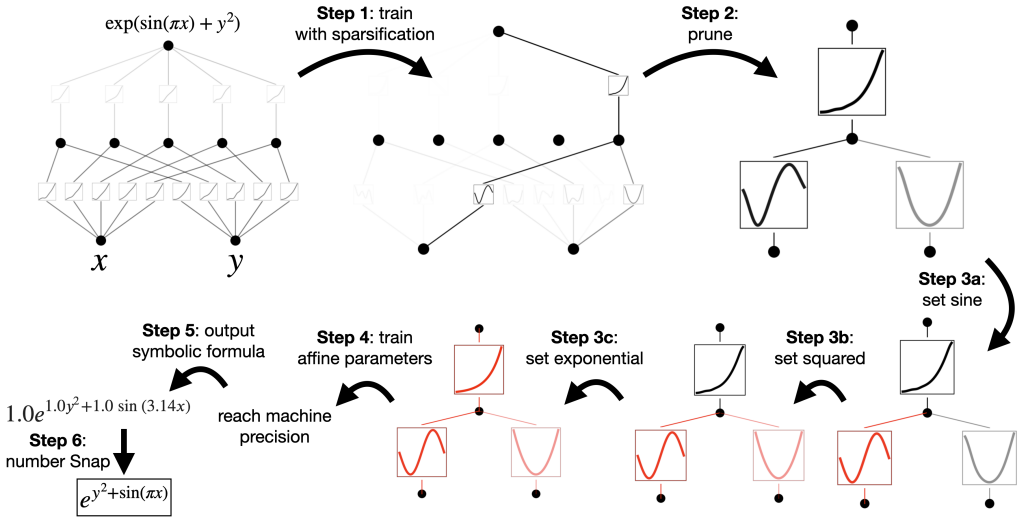


Figure 2.4: An example of how to do symbolic regression with KAN.

2. Visualization. When we visualize a KAN, to get a sense of magnitudes, we set the transparency of an activation function $\phi_{l,i,j}$ proportional to $\tanh(\beta A_{l,i,j})$ where $\beta = 3$. Hence, functions with small magnitude appear faded out to allow us to focus on important ones.

3. Pruning. After training with sparsification penalty, we may also want to prune the network to a smaller subnetwork. We sparsify KANs on the node level (rather than on the edge level). For each node (say the i^{th} neuron in the l^{th} layer), we define its incoming and outgoing score as

$$I_{l,i} = \max_k(|\phi_{l-1,i,k}|_1), \qquad O_{l,i} = \max_j(|\phi_{l+1,j,i}|_1), \tag{2.21}$$

and consider a node to be important if both incoming and outgoing scores are greater than a threshold hyperparameter $\theta = 10^{-2}$ by default. All unimportant neurons are pruned.

4. Symbolification. In cases where we suspect that some activation functions are in fact symbolic (e.g., $\cos$ or $\log$), we provide an interface to set them to be a specified symbolic form, `fix_symbolic(l,i,j,f)` can set the (l, i, j) activation to be f. However, we cannot simply set the activation function to be the exact symbolic formula, since its inputs and outputs may have shifts and scalings. So, we obtain preactivations x and postactivations y from samples, and fit affine parameters (a, b, c, d) such that $y \approx cf(ax + b) + d$. The fitting is done by iterative grid search of a, b and linear regression.

Besides these techniques, we provide additional tools that allow users to apply more fine-grained control to KANs, listed in Appendix A.

2.5.2 A toy example: how humans can interact with KANs

Above we have proposed a number of simplification techniques for KANs. We can view these simplification choices as buttons one can click on. A user interacting with these buttons can decide which button is most promising to click next to make KANs more interpretable. We use an example below to showcase how a user could interact with a KAN to obtain maximally interpretable results.

Let us again consider the regression task

$$f(x, y) = \exp\left(\sin(\pi x) + y^2\right). \tag{2.22}$$

Given data points (x_i, y_i, f_i), $i = 1, 2, \cdots, N_p$, a hypothetical user Alice is interested in figuring out the symbolic formula. The steps of Alice's interaction with the KANs are described below (illustrated in Figure 2.4):

Step 1: Training with sparsification. Starting from a fully-connected $[2, 5, 1]$ KAN, training with sparsification regularization can make it quite sparse. 4 out of 5 neurons in the hidden layer appear useless, hence we want to prune them away.

Step 2: Pruning. Automatic pruning is seen to discard all hidden neurons except the last one, leaving a $[2, 1, 1]$ KAN. The activation functions appear to be known symbolic functions.

Step 3: Setting symbolic functions. Assuming that the user can correctly guess these symbolic formulas from staring at the KAN plot, they can set

```
fix_symbolic(0,0,0,'sin')
fix_symbolic(0,1,0,'x^2')
fix_symbolic(1,0,0,'exp').
```
(2.23)

In case the user has no domain knowledge or no idea which symbolic functions these activation functions might be, we provide a function `suggest_symbolic` to suggest symbolic candidates.

Step 4: Further training. After symbolifying all the activation functions in the network, the only remaining parameters are the affine parameters. We continue training these affine parameters, and when we see the loss dropping to machine precision, we know that we have found the correct symbolic expression.

Step 5: Output the symbolic formula. `Sympy` is used to compute the symbolic formula of the output node. The user obtains $1.0e^{1.0y^2+1.0\sin(3.14x)}$, which is the true answer (we only displayed two decimals for π).

Remark: Why not symbolic regression (SR)? It is reasonable to use symbolic regression for this example. However, symbolic regression methods are in general brittle and hard to debug. They either return a success or a failure in the end without outputting interpretable intermediate results. In contrast, KANs do continuous search (with gradient descent) in function space, so their results are more continuous and hence more robust. Moreover, users have more control over KANs as compared to SR due to KANs' transparency. The way we visualize KANs is like displaying KANs' "brain" to users, and users can perform "surgery" (debugging) on KANs. This level of control is typically unavailable for SR. We will show examples of this in Section 4.4. More generally, when the target function is not symbolic, symbolic regression will fail but KANs can still provide something meaningful. For example, a special function (e.g., a Bessel function) is impossible to SR to learn unless it is provided in advance, but KANs can use splines to approximate it numerically anyway (see Figure 4.1 (d)).

3 KANs are accurate

In this section, we demonstrate that KANs are more effective at representing functions than MLPs in various tasks (regression and PDE solving). When comparing two families of models, it is fair to compare both their accuracy (loss) and their complexity (number of parameters). We will show that KANs display more favorable Pareto Frontiers than MLPs. Moreover, in Section 3.5, we show that KANs can naturally work in continual learning without catastrophic forgetting.

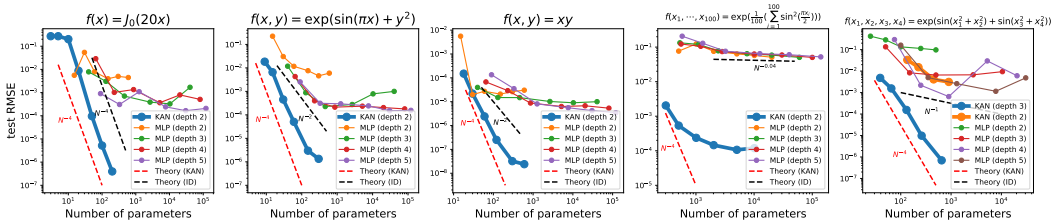


Figure 3.1: Compare KANs to MLPs on five toy examples. KANs can almost saturate the fastest scaling law predicted by our theory ($\alpha = 4$), while MLPs scales slowly and plateau quickly.

3.1 Toy datasets

In Section 2.3, our theory suggested that test RMSE loss ℓ scales as $\ell \propto N^{-4}$ with model parameters N. However, this relies on the existence of a Kolmogorov-Arnold representation. As a sanity check, we construct five examples we know have smooth KA representations:

(1) $f(x) = J_0(20x)$, which is the Bessel function. Since it is a univariate function, it can be represented by a spline, which is a $[1, 1]$ KAN.

(2) $f(x, y) = \exp(\sin(\pi x) + y^2)$. We know that it can be exactly represented by a $[2, 1, 1]$ KAN.

(3) $f(x, y) = xy$. We know from Figure 4.1 that it can be exactly represented by a $[2, 2, 1]$ KAN.

(4) A high-dimensional example $f(x_1, \cdots, x_{100}) = \exp(\frac{1}{100}\sum_{i=1}^{100} \sin^2(\frac{\pi x_i}{2}))$ which can be represented by a $[100, 1, 1]$ KAN.

(5) A four-dimensional example $f(x_1, x_2, x_3, x_4) = \exp(\frac{1}{2}(\sin(\pi(x_1^2 + x_2^2)) + \sin(\pi(x_3^2 + x_4^2))))$ which can be represented by a $[4, 4, 2, 1]$ KAN.

We train these KANs by increasing grid points every 200 steps, in total covering $G = \{3, 5, 10, 20, 50, 100, 200, 500, 1000\}$. We train MLPs with different depths and widths as baselines. Both MLPs and KANs are trained with LBFGS for 1800 steps in total. We plot test RMSE as a function of the number of parameters for KANs and MLPs in Figure 3.1, showing that KANs have better scaling curves than MLPs, especially for the high-dimensional example. For comparison, we plot the lines predicted from our KAN theory as red dashed ($\alpha = k+1 = 4$), and the lines predicted from Sharma & Kaplan [24] as black-dashed ($\alpha = (k+1)/d = 4/d$). KANs can almost saturate the steeper red lines, while MLPs struggle to converge even as fast as the slower black lines and plateau quickly. We also note that for the last example, the 2-Layer KAN $[4, 9, 1]$ behaves much worse than the 3-Layer KAN (shape $[4, 2, 2, 1]$). This highlights the greater expressive power of deeper KANs, which is the same for MLPs: deeper MLPs have more expressive power than shallower ones. Note that we have adopted the vanilla setup where both KANs and MLPs are trained with LBFGS without advanced techniques, e.g., switching between Adam and LBFGS, or boosting [35]. We leave the comparison of KANs and MLPs in advanced setups for future work.

3.2 Special functions

One caveat for the above results is that we assume knowledge of the "true" KAN shape. In practice, we do not know the existence of KA representations. Even when we are promised that such a KA representation exists, we do not know the KAN shape a priori. Special functions in more than one variables are such cases, because it would be (mathematically) surprising if multivariate special functions (e.g., a Bessel function $f(\nu, x) = J_\nu(x)$) could be written in KA represenations, involving only univariate functions and sums). We show below that:

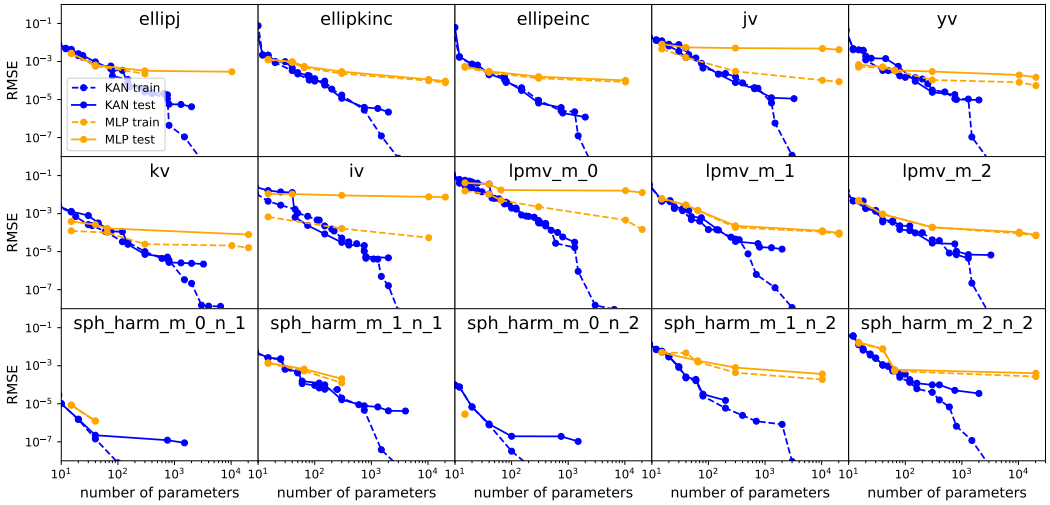


Figure 3.2: Fitting special functions. We show the Pareto Frontier of KANs and MLPs in the plane spanned by the number of model parameters and RMSE loss. Consistently accross all special functions, KANs have better Pareto Frontiers than MLPs. The definitions of these special functions are in Table 1.

(1) Finding (approximate) compact KA representations of special functions is possible, revealing novel mathematical properties of special functions from the perspective of Kolmogorov-Arnold representations.

(2) KANs are more efficient and accurate in representing special functions than MLPs.

We collect 15 special functions common in math and physics, summarized in Table 1. We choose MLPs with fixed width 5 or 100 and depths swept in $\{2, 3, 4, 5, 6\}$. We run KANs both with and without pruning. *KANs without pruning*: We fix the shape of KAN, whose width are set to 5 and depths are swept in {2,3,4,5,6}. *KAN with pruning*. We use the sparsification ($\lambda = 10^{-2}$ or 10^{-3}) and pruning technique in Section 2.5.1 to obtain a smaller KAN pruned from a fixed-shape KAN. Each KAN is initialized to have $G = 3$, trained with LBFGS, with increasing number of grid points every 200 steps to cover $G = \{3, 5, 10, 20, 50, 100, 200\}$. For each hyperparameter combination, we run 3 random seeds.

For each dataset and each model family (KANs or MLPs), we plot the Pareto frontier [5], in the (number of parameters, RMSE) plane, shown in Figure 3.2. KANs' performance is shown to be consistently better than MLPs, i.e., KANs can achieve lower training/test losses than MLPs, given the same number of parameters. Moreover, we report the (surprisingly compact) shapes of our auto-discovered KANs for special functions in Table 1. On one hand, it is interesting to interpret what these compact representations mean mathematically (we include the KAN illustrations in Figure F.1 and F.2 in Appendix F). On the other hand, these compact representations imply the possibility of breaking down a high-dimensional lookup table into several 1D lookup tables, which can potentially save a lot of memory, with the (almost negligible) overhead to perform a few additions at inference time.

3.3 Feynman datasets

The setup in Section 3.1 is when we clearly know "true" KAN shapes. The setup in Section 3.2 is when we clearly do **not** know "true" KAN shapes. This part investigates a setup lying in the middle:

[5] Pareto frontier is defined as fits that are optimal in the sense of no other fit being both simpler and more accurate.

Name	scipy.special API	Minimal KAN shape test RMSE $< 10^{-2}$	Minimal KAN test RMSE	Best KAN shape	Best KAN test RMSE	MLP test RMSE
Jacobian elliptic functions	$\mathrm{ellipj}(x,y)$	[2,2,1]	7.29×10^{-3}	[2,3,2,1,1,1]	$\mathbf{1.33 \times 10^{-4}}$	6.48×10^{-4}
Incomplete elliptic integral of the first kind	$\mathrm{ellipkinc}(x,y)$	[2,2,1,1]	1.00×10^{-3}	[2,2,1,1,1]	$\mathbf{1.24 \times 10^{-4}}$	5.52×10^{-4}
Incomplete elliptic integral of the second kind	$\mathrm{ellipeinc}(x,y)$	[2,2,1,1]	8.36×10^{-5}	[2,2,1,1]	$\mathbf{8.26 \times 10^{-5}}$	3.04×10^{-4}
Bessel function of the first kind	$\mathrm{jv}(x,y)$	[2,2,1]	4.93×10^{-3}	[2,3,1,1,1]	$\mathbf{1.64 \times 10^{-3}}$	5.52×10^{-3}
Bessel function of the second kind	$\mathrm{yv}(x,y)$	[2,3,1]	1.89×10^{-3}	[2,2,2,1]	$\mathbf{1.49 \times 10^{-5}}$	3.45×10^{-4}
Modified Bessel function of the second kind	$\mathrm{kv}(x,y)$	[2,1,1]	4.89×10^{-3}	[2,2,1]	$\mathbf{2.52 \times 10^{-5}}$	1.67×10^{-4}
Modified Bessel function of the first kind	$\mathrm{iv}(x,y)$	[2,4,3,2,1,1]	9.28×10^{-3}	[2,4,3,2,1,1]	$\mathbf{9.28 \times 10^{-3}}$	1.07×10^{-2}
Associated Legendre function $(m=0)$	$\mathrm{lpmv}(0,x,y)$	[2,2,1]	5.25×10^{-5}	[2,2,1]	$\mathbf{5.25 \times 10^{-5}}$	1.74×10^{-2}
Associated Legendre function $(m=1)$	$\mathrm{lpmv}(1,x,y)$	[2,4,1]	6.90×10^{-4}	[2,4,1]	$\mathbf{6.90 \times 10^{-4}}$	1.50×10^{-3}
Associated Legendre function $(m=2)$	$\mathrm{lpmv}(2,x,y)$	[2,2,1]	4.88×10^{-3}	[2,3,2,1]	$\mathbf{2.26 \times 10^{-4}}$	9.43×10^{-4}
spherical harmonics $(m=0,n=1)$	$\mathrm{sph\_harm}(0,1,x,y)$	[2,1,1]	2.21×10^{-7}	[2,1,1]	$\mathbf{2.21 \times 10^{-7}}$	1.25×10^{-6}
spherical harmonics $(m=1,n=1)$	$\mathrm{sph\_harm}(1,1,x,y)$	[2,2,1]	7.86×10^{-4}	[2,3,2,1]	$\mathbf{1.22 \times 10^{-4}}$	6.70×10^{-4}
spherical harmonics $(m=0,n=2)$	$\mathrm{sph\_harm}(0,2,x,y)$	[2,1,1]	1.95×10^{-7}	[2,1,1]	$\mathbf{1.95 \times 10^{-7}}$	2.85×10^{-6}
spherical harmonics $(m=1,n=2)$	$\mathrm{sph\_harm}(1,2,x,y)$	[2,2,1]	4.70×10^{-4}	[2,2,1,1]	$\mathbf{1.50 \times 10^{-5}}$	1.84×10^{-3}
spherical harmonics $(m=2,n=2)$	$\mathrm{sph\_harm}(2,2,x,y)$	[2,2,1]	1.12×10^{-3}	[2,2,3,2,1]	$\mathbf{9.45 \times 10^{-5}}$	6.21×10^{-4}

Table 1: Special functions

Feynman Eq.	Original Formula	Dimensionless formula	Variables	Human-constructed KAN shape	Pruned KAN shape (smallest shape that achieves RMSE $< 10^{-2}$)	Pruned KAN shape (lowest loss)	Human-constructed KAN loss (lowest test RMSE)	Pruned KAN loss (lowest test RMSE)	Unpruned KAN loss (lowest test RMSE)	MLP loss (lowest test RMSE)
I.6.2	$\exp(-\frac{\theta^2}{2\sigma^2})/\sqrt{2\pi\sigma^2}$	$\exp(-\frac{\theta^2}{2\sigma^2})/\sqrt{2\pi\sigma^2}$	θ, σ	[2,2,1,1]	[2,2,1]	[2,2,1,1]	7.66×10^{-5}	$\mathbf{2.86 \times 10^{-5}}$	4.60×10^{-5}	1.45×10^{-4}
I.6.2b	$\exp(-\frac{(\theta-\theta_1)^2}{2\sigma^2})/\sqrt{2\pi\sigma^2}$	$\exp(-\frac{(\theta-\theta_1)^2}{2\sigma^2})/\sqrt{2\pi\sigma^2}$	θ, θ_1, σ	[3,2,2,1,1]	[3,4,1]	[3,2,2,1,1]	1.22×10^{-3}	$\mathbf{4.45 \times 10^{-4}}$	1.25×10^{-3}	7.40×10^{-4}
I.9.18	$\frac{Gm_1m_2}{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}$	$\frac{a}{(b-1)^2+(c-d)^2+(e-f)^2}$	a, b, c, d, e, f	[6,4,2,1,1]	[6,4,1,1]	[6,4,1,1]	1.48×10^{-3}	$\mathbf{1.48 \times 10^{-3}}$	8.62×10^{-3}	6.56×10^{-3}	1.59×10^{-3}
I.12.11	$q(E_f + B v \sin\theta)$	$1 + a \sin\theta$	a, θ	[2,2,2,1]	[2,2,1]	[2,2,1]	2.07×10^{-3}	1.39×10^{-3}	9.13×10^{-4}	$\mathbf{6.71 \times 10^{-4}}$
I.13.12	$Gm_1m_2(\frac{1}{r_2} - \frac{1}{r_1})$	$a(\frac{1}{b} - 1)$	a, b	[2,2,1]	[2,2,1]	[2,2,1]	7.22×10^{-3}	4.81×10^{-3}	2.72×10^{-3}	$\mathbf{1.42 \times 10^{-3}}$
I.15.3x	$\frac{x - ut}{\sqrt{1-(\frac{v}{c})^2}}$	$\frac{1-a}{\sqrt{1-b^2}}$	a, b	[2,2,1,1]	[2,1,1]	[2,2,1,1,1]	7.35×10^{-3}	1.58×10^{-3}	1.14×10^{-3}	$\mathbf{8.54 \times 10^{-4}}$
I.16.6	$\frac{u+v}{1+\frac{uv}{c^2}}$	$\frac{a+b}{1+ab}$	a, b	[2,2,2,2,2,1]	[2,2,1]	[2,2,1]	1.06×10^{-3}	1.19×10^{-3}	1.53×10^{-3}	$\mathbf{6.20 \times 10^{-4}}$
I.18.4	$\frac{m_1r_1+m_2r_2}{m_1+m_2}$	$\frac{1+ab}{1+a}$	a, b	[2,2,2,1,1]	[2,2,1]	[2,2,1]	3.92×10^{-4}	$\mathbf{1.50 \times 10^{-4}}$	1.32×10^{-3}	3.68×10^{-4}
I.26.2	$\arcsin(n \sin\theta_2)$	$\arcsin(n \sin\theta_2)$	n, θ_2	[2,2,2,1,1]	[2,2,1]	[2,2,2,1,1]	1.22×10^{-1}	$\mathbf{7.90 \times 10^{-4}}$	8.63×10^{-4}	1.24×10^{-3}
I.27.6	$\frac{1}{\frac{1}{d_1} + \frac{n}{d_2}}$	$\frac{1}{1+ab}$	a, b	[2,2,1,1]	[2,1,1]	[2,1,1]	2.22×10^{-4}	$\mathbf{1.94 \times 10^{-4}}$	2.14×10^{-4}	2.46×10^{-4}
I.29.16	$\sqrt{x_1^2 + x_2^2 - 2x_1x_2\cos(\theta_1 - \theta_2)}$	$\sqrt{1 + a^2 - 2a\cos(\theta_1 - \theta_2)}$	a, θ_1, θ_2	[3,2,2,3,2,1,1]	[3,2,2,1]	[3,2,3,1]	2.36×10^{-1}	3.99×10^{-3}	$\mathbf{3.20 \times 10^{-3}}$	4.64×10^{-3}
I.30.3	$I_* 0 \frac{\sin^2(\frac{n\theta}{2})}{\sin^2(\frac{\theta}{2})}$	$\frac{\sin^2(\frac{n\theta}{2})}{\sin^2(\frac{\theta}{2})}$	n, θ	[2,3,2,2,1,1]	[2,4,3,1]	[2,3,2,3,1,1]	3.85×10^{-1}	$\mathbf{1.03 \times 10^{-3}}$	1.11×10^{-2}	1.50×10^{-2}
I.30.5	$\arcsin(\frac{\lambda}{nd})$	$\arcsin(\frac{a}{n})$	a, n	[2,1,1]	[2,1,1]	[2,1,1,1,1,1]	2.23×10^{-4}	$\mathbf{3.49 \times 10^{-5}}$	6.92×10^{-5}	9.45×10^{-5}
I.37.4	$I_* = I_1 + I_2 + 2\sqrt{I_1 I_2}\cos\delta$	$1 + a + 2\sqrt{a}\cos\delta$	a, δ	[2,3,2,1]	[2,2,1]	[2,2,1]	7.57×10^{-5}	$\mathbf{4.91 \times 10^{-6}}$	3.41×10^{-4}	5.67×10^{-4}
I.40.1	$n_0 \exp(-\frac{mgx}{k_bT})$	$n_0 e^{-a}$	n_0, a	[2,1,1]	[2,2,1]	[2,2,1,1,1,2,1]	3.45×10^{-3}	5.01×10^{-4}	$\mathbf{3.12 \times 10^{-4}}$	3.99×10^{-4}
I.44.4	$nk_bT\ln(\frac{V_2}{V_1})$	$n \ln a$	n, a	[2,2,1]	[2,2,1]	[2,2,1]	$\mathbf{2.30 \times 10^{-5}}$	2.43×10^{-5}	1.10×10^{-4}	3.99×10^{-4}
I.50.26	$x_1(\cos(\omega t) + \alpha\cos^2(wt))$	$\cos a + \alpha\cos^2 a$	a, α	[2,2,3,1]	[2,3,1]	[2,3,2,1]	$\mathbf{1.52 \times 10^{-4}}$	5.82×10^{-4}	4.90×10^{-4}	1.53×10^{-3}
II.2.42	$\frac{k(T_2-T_1)A}{d}$	$(a-1)b$	a, b	[2,2,1]	[2,2,1]	[2,2,2,1]	8.54×10^{-4}	7.22×10^{-4}	1.22×10^{-3}	$\mathbf{1.81 \times 10^{-4}}$
II.6.15a	$\frac{3}{4\pi\epsilon}\frac{pz}{r^5}\sqrt{x^2+y^2}$	$\frac{1}{4\pi}c\sqrt{a^2+b^2}$	a, b, c	[3,2,2,2,1]	[3,2,1,1]	[3,2,1,1]	2.61×10^{-3}	3.28×10^{-3}	1.35×10^{-3}	$\mathbf{5.92 \times 10^{-4}}$
II.11.7	$n_0(1 + \frac{p_d E_f \cos\theta}{k_bT})$	$n_0(1 + a\cos\theta)$	n_0, a, θ	[3,3,3,2,2,1]	[3,3,1,1]	[3,3,1,1]	7.10×10^{-3}	8.52×10^{-3}	5.03×10^{-3}	$\mathbf{5.92 \times 10^{-4}}$
II.11.27	$\frac{n\alpha}{1-\frac{n\alpha}{3}}\epsilon E_f$	$\frac{n\alpha}{1-\frac{n\alpha}{3}}$	n, α	[2,2,1,2,1]	[2,1,1]	[2,2,1]	2.67×10^{-5}	4.40×10^{-5}	$\mathbf{1.43 \times 10^{-5}}$	7.18×10^{-5}
II.35.18	$\frac{n_0}{\exp(\frac{\mu m B}{k_bT}) + \exp(-\frac{\mu m B}{k_bT})}$	$\frac{n_0}{\exp(a) + \exp(-a)}$	n_0, a	[2,1,1]	[2,1,1]	[2,1,1,1]	4.13×10^{-4}	1.58×10^{-4}	$\mathbf{7.71 \times 10^{-5}}$	7.92×10^{-5}
II.36.38	$\frac{\mu_m B}{k_bT} + \frac{\mu_m \alpha M}{\epsilon c^2 k_bT}$	$a + ab$	a, α, b	[3,3,1]	[3,2,1]	[3,2,1]	2.85×10^{-3}	$\mathbf{1.15 \times 10^{-3}}$	3.03×10^{-3}	2.15×10^{-3}
II.38.3	$\frac{YAx}{d}$	$\frac{a}{b}$	a, b	[2,1,1]	[2,1,1]	[2,2,1,1,1]	1.47×10^{-4}	$\mathbf{8.78 \times 10^{-5}}$	6.43×10^{-4}	5.26×10^{-4}
III.9.52	$\frac{p_d E_f}{h}\frac{\sin^2((\omega-\omega_0)t/2)}{((\omega-\omega_0)t/2)^2}$	$a\frac{\sin^2(\frac{b-c}{2})}{(\frac{b-c}{2})^2}$	a, b, c	[3,2,3,1,1]	[3,3,2,1]	[3,3,2,1,1,1]	4.43×10^{-2}	3.90×10^{-3}	2.11×10^{-2}	$\mathbf{9.07 \times 10^{-4}}$
III.10.19	$\mu_m\sqrt{B_x^2 + B_y^2 + B_z^2}$	$\sqrt{1 + a^2 + b^2}$	a, b	[2,1,1]	[2,1,1]	[2,1,2,1]	2.54×10^{-3}	1.18×10^{-3}	8.16×10^{-4}	$\mathbf{1.67 \times 10^{-4}}$
III.17.37	$\beta(1 + \alpha\cos\theta)$	$\beta(1 + \alpha\cos\theta)$	α, β, θ	[3,3,3,2,2,1]	[3,3,1]	[3,3,1]	1.10×10^{-3}	5.03×10^{-4}	$\mathbf{4.12 \times 10^{-4}}$	6.80×10^{-4}

Table 2: Feynman dataset

Given the structure of the dataset, we may construct KANs by hand, but we are not sure if they are optimal. In this regime, it is interesting to compare human-constructed KANs and auto-discovered KANs via pruning (techniques in Section 2.5.1).

Feynman dataset. The Feynman dataset collects many physics equations from Feynman's textbooks [36, 37]. For our purpose, we are interested in problems in the `Feynman_no_units` dataset that have at least 2 variables, since univariate problems are trivial for KANs (they simplify to 1D splines). A sample equation from the Feynman dataset is the relativisic velocity addition formula

$$f(u, v) = (u + v)/(1 + uv). \tag{3.1}$$

The dataset can be constructed by randomly drawing $u_i \in (-1, 1)$, $v_i \in (-1, 1)$, and computing $f_i = f(u_i, v_i)$. Given many tuples (u_i, v_i, f_i), a neural network is trained and aims to predict f from u and v. We are interested in (1) how well a neural network can perform on test samples; (2) how much we can learn about the structure of the problem from neural networks.

We compare four kinds of neural networks:

(1) Human-constructued KAN. Given a symbolic formula, we rewrite it in Kolmogorov-Arnold representations. For example, to multiply two numbers x and y, we can use the identity $xy = \frac{(x+y)^2}{4} - \frac{(x-y)^2}{4}$, which corresponds to a $[2, 2, 1]$ KAN. The constructued shapes are listed in the "Human-constructed KAN shape" in Table 2.

(2) KANs without pruning. We fix the KAN shape to width 5 and depths are swept over {2,3,4,5,6}.

(3) KAN with pruning. We use the sparsification ($\lambda = 10^{-2}$ or 10^{-3}) and the pruning technique from Section 2.5.1 to obtain a smaller KAN from a fixed-shape KAN from (2).

(4) MLPs with fixed width 5, depths swept in $\{2, 3, 4, 5, 6\}$, and activations chosen from $\{\mathrm{Tanh}, \mathrm{ReLU}, \mathrm{SiLU}\}$.

Each KAN is initialized to have $G = 3$, trained with LBFGS, with increasing number of grid points every 200 steps to cover $G = \{3, 5, 10, 20, 50, 100, 200\}$. For each hyperparameter combination, we try 3 random seeds. For each dataset (equation) and each method, we report the results of the best model (minimal KAN shape, or lowest test loss) over random seeds and depths in Table 2. We find that MLPs and KANs behave comparably on average. For each dataset and each model family (KANs or MLPs), we plot the Pareto frontier in the plane spanned by the number of parameters and RMSE losses, shown in Figure D.1 in Appendix D. We conjecture that the Feynman datasets are too simple to let KANs make further improvements, in the sense that variable dependence is usually smooth or monotonic, which is in contrast to the complexity of special functions which often demonstrate oscillatory behavior.

Auto-discovered KANs are smaller than human-constructed ones. We report the pruned KAN shape in two columns of Table 2; one column is for the minimal pruned KAN shape that can achieve reasonable loss (i.e., test RMSE smaller than 10^{-2}); the other column is for the pruned KAN that achieves lowest test loss. For completeness, we visualize all 54 pruned KANs in Appendix D (Figure D.2 and D.3). It is interesting to observe that auto-discovered KAN shapes (for both minimal and best) are usually smaller than our human constructions. This means that KA representations can be more efficient than we imagine. At the same time, this may make interpretability subtle because information is being squashed into a smaller space than what we are comfortable with.

Consider the relativistic velocity composition $f(u, v) = \frac{u+v}{1+uv}$, for example. Our construction is quite deep because we were assuming that multiplication of u, v would use two layers (see Figure 4.1 (a)), inversion of $1 + uv$ would use one layer, and multiplication of $u + v$ and $1/(1 + uv)$ would use another two layers[6], resulting a total of 5 layers. However, the auto-discovered KANs are only 2 layers deep! In hindsight, this is actually expected if we recall the rapidity trick in relativity: define the two "rapidities" $a \equiv \mathrm{arctanh}\ u$ and $b \equiv \mathrm{arctanh}\ v$. The relativistic composition of velocities are simple additions in rapidity space, i.e., $\frac{u+v}{1+uv} = \tanh(\mathrm{arctanh}\ u + \mathrm{arctanh}\ v)$, which can be realized by a two-layer KAN. Pretending we do not know the notion of rapidity in physics, we could potentially discover this concept right from KANs without trial-and-error symbolic manipulations. The interpretability of KANs which can facilitate scientific discovery is the main topic in Section 4.

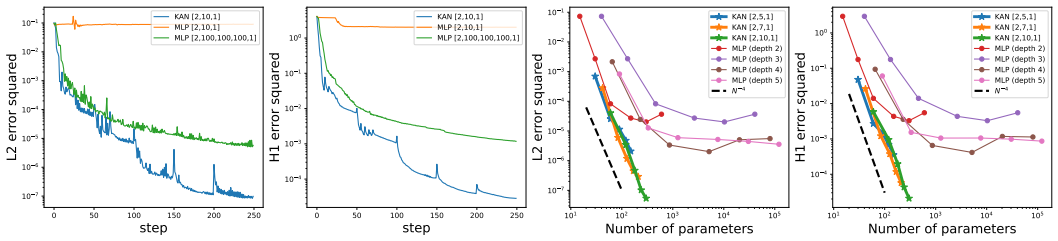


Figure 3.3: The PDE example. We plot L2 squared and H1 squared losses between the predicted solution and ground truth solution. First and second: training dynamics of losses. Third and fourth: scaling laws of losses against the number of parameters. KANs converge faster, achieve lower losses, and have steeper scaling laws than MLPs.

3.4 Solving partial differential equations

We consider a Poisson equation with zero Dirichlet boundary data. For $\Omega = [-1, 1]^2$, consider the PDE

$$\begin{aligned} u_{xx} + u_{yy} &= f \quad \text{in } \Omega\,, \\ u &= 0 \quad \text{on } \partial\Omega\,. \end{aligned} \tag{3.2}$$

We consider the data $f = -\pi^2(1+4y^2)\sin(\pi x)\sin(\pi y^2) + 2\pi\sin(\pi x)\cos(\pi y^2)$ for which $u = \sin(\pi x)\sin(\pi y^2)$ is the true solution. We use the framework of physics-informed neural networks (PINNs) [38, 39] to solve this PDE, with the loss function given by

$$\text{loss}_{\text{pde}} = \alpha \text{loss}_i + \text{loss}_b := \alpha \frac{1}{n_i}\sum_{i=1}^{n_i} |u_{xx}(z_i) + u_{yy}(z_i) - f(z_i)|^2 + \frac{1}{n_b}\sum_{i=1}^{n_b} u^2\,,$$

where we use loss_i to denote the interior loss, discretized and evaluated by a uniform sampling of n_i points $z_i = (x_i, y_i)$ inside the domain, and similarly we use loss_b to denote the boundary loss, discretized and evaluated by a uniform sampling of n_b points on the boundary. α is the hyperparameter balancing the effect of the two terms.

We compare the KAN architecture with that of MLPs using the same hyperparameters $n_i = 10000$, $n_b = 800$, and $\alpha = 0.01$. We measure both the error in the L^2 norm and energy (H^1) norm and see that KAN achieves a much better scaling law with a smaller error, using smaller networks and fewer parameters; see Figure 3.3. A 2-Layer width-10 KAN is 100 times more accurate than a 4-Layer width-100 MLP (10^{-7} vs 10^{-5} MSE) and 100 times more parameter efficient (10^2 vs 10^4 parameters). Therefore we speculate that KANs might have the potential of serving as a good neural network representation for model reduction of PDEs. However, we want to note that our implementation of KANs are typically 10x slower than MLPs to train. The ground truth being a symbolic formula might be an unfair comparison for MLPs since KANs are good at representing symbolic formulas. In general, KANs and MLPs are good at representing different function classes of PDE solutions, which needs detailed future study to understand their respective boundaries.

3.5 Continual Learning

Catastrophic forgetting is a serious problem in current machine learning [40]. When a human masters a task and switches to another task, they do not forget how to perform the first task. Unfortunately, this is not the case for neural networks. When a neural network is trained on task 1 and then shifted to being trained on task 2, the network will soon forget about how to perform task 1. A key difference between artificial neural networks and human brains is that human brains have function-

[6] Note that we cannot use the logarithmic construction for division, because u and v here might be negative numbers.

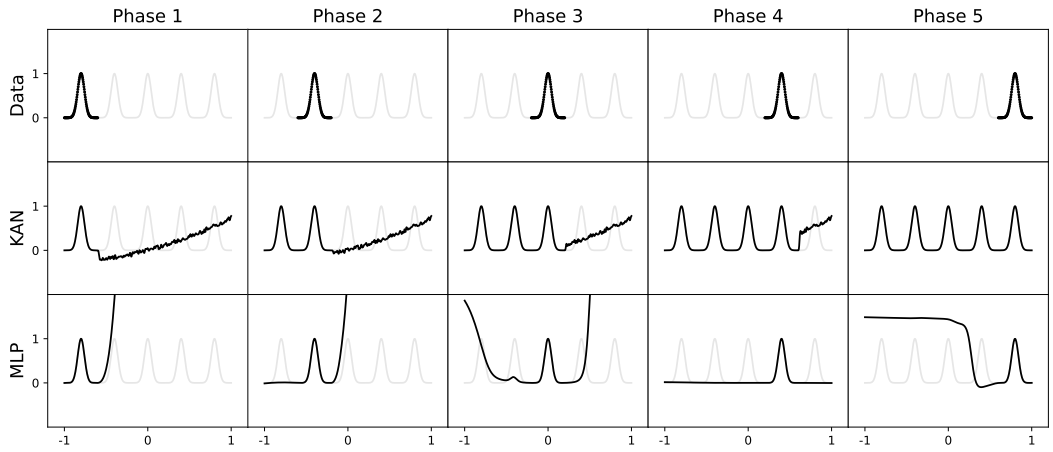


Figure 3.4: A toy continual learning problem. The dataset is a 1D regression task with 5 Gaussian peaks (top row). Data around each peak is presented sequentially (instead of all at once) to KANs and MLPs. KANs (middle row) can perfectly avoid catastrophic forgetting, while MLPs (bottom row) display severe catastrophic forgetting.

ally distinct modules placed locally in space. When a new task is learned, structure re-organization only occurs in local regions responsible for relevant skills [41, 42], leaving other regions intact. Most artificial neural networks, including MLPs, do not have this notion of locality, which is probably the reason for catastrophic forgetting.

We show that KANs have local plasticity and can avoid catastrophic forgetting by leveraging the locality of splines. The idea is simple: since spline bases are local, a sample will only affect a few nearby spline coefficients, leaving far-away coefficients intact (which is desirable since far-away regions may have already stored information that we want to preserve). By contrast, since MLPs usually use global activations, e.g., ReLU/Tanh/SiLU etc., any local change may propagate uncontrollably to regions far away, destroying the information being stored there.

We use a toy example to validate this intuition. The 1D regression task is composed of 5 Gaussian peaks. Data around each peak is presented sequentially (instead of all at once) to KANs and MLPs, as shown in Figure 3.4 top row. KAN and MLP predictions after each training phase are shown in the middle and bottom rows. As expected, KAN only remodels regions where data is present on in the current phase, leaving previous regions unchanged. By contrast, MLPs remodels the whole region after seeing new data samples, leading to catastrophic forgetting.

Here we simply present our preliminary results on an extremely simple example, to demonstrate how one could possibly leverage locality in KANs (thanks to spline parametrizations) to reduce catastrophic forgetting. However, it remains unclear whether our method can generalize to more realistic setups, especially in high-dimensional cases where it is unclear how to define "locality". In future work, We would also like to study how our method can be connected to and combined with SOTA methods in continual learning [43, 44].

4 KANs are interpretable

In this section, we show that KANs are interpretable and interactive thanks to the techniques we developed in Section 2.5. We want to test the use of KANs not only on synthetic tasks (Section 4.1 and 4.2), but also in real-life scientific research. We demonstrate that KANs can (re)discover both highly non-trivial relations in knot theory (Section 4.3) and phase transition boundaries in condensed

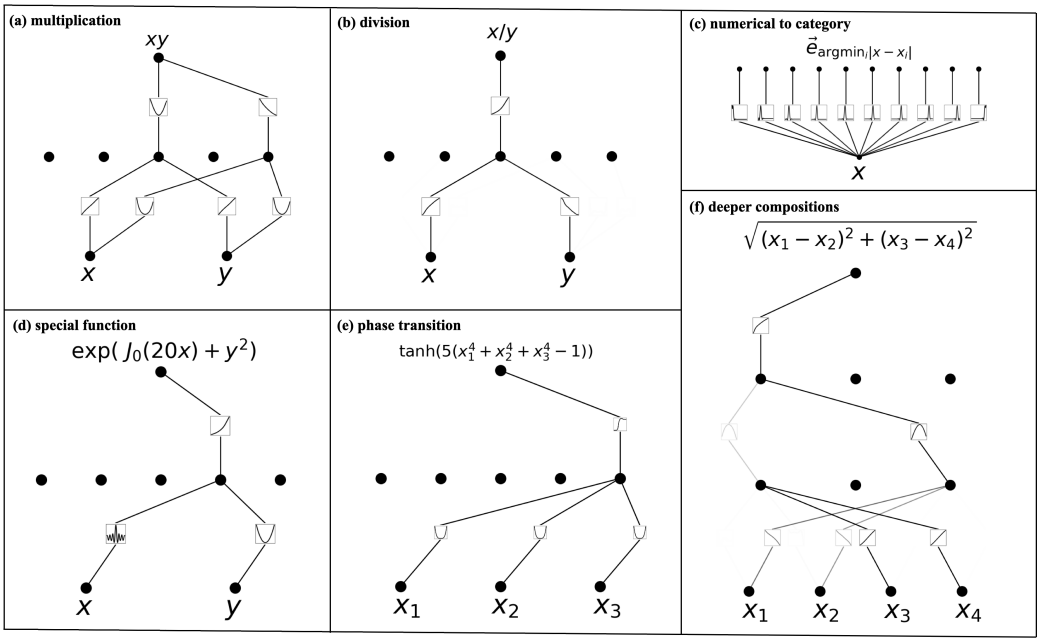


Figure 4.1: KANs are interepretable for simple symbolic tasks

matter physics (Section 4.4). KANs could potentially be the foundation model for AI + Science due to their accuracy (last section) and interpretability (this section).

4.1 Supervised toy datasets

We first examine KANs' ability to reveal the compositional structures in symbolic formulas. Six examples are listed below and their KANs are visualized in Figure 4.1. KANs are able to reveal the compositional structures present in these formulas, as well as learn the correct univariate functions.

(a) Multiplication $f(x,y) = xy$. A $[2,5,1]$ KAN is pruned to a $[2,2,1]$ KAN. The learned activation functions are linear and quadratic. From the computation graph, we see that the way it computes xy is leveraging $2xy = (x+y)^2 - (x^2+y^2)$.

(b) Division of positive numbers $f(x,y) = x/y$. A $[2,5,1]$ KAN is pruned to a $[2,1,1]$ KAN. The learned activation functions are logarithmic and exponential functions, and the KAN is computing x/y by leveraging the identity $x/y = \exp(\log x - \log y)$.

(c) Numerical to categorical. The task is to convert a real number in $[0,1]$ to its first decimal digit (as one hots), e.g., $0.0618 \to [1,0,0,0,0,\cdots]$, $0.314 \to [0,0,0,1,0,\cdots]$. Notice that activation functions are learned to be spikes located around the corresponding decimal digits.

(d) Special function $f(x,y) = \exp(J_0(20x) + y^2)$. One limitation of symbolic regression is that it will never find the correct formula of a special function if the special function is not provided as prior knowledge. KANs can learn special functions – the highly wiggly Bessel function $J_0(20x)$ is learned (numerically) by KAN.

(e) Phase transition $f(x_1,x_2,x_3) = \tanh(5(x_1^4 + x_2^4 + x_3^4 - 1))$. Phase transitions are of great interest in physics, so we want KANs to be able to detect phase transitions and to identify the correct order parameters. We use the tanh function to simulate the phase transition behavior, and the order parameter is the combination of the quartic terms of x_1, x_2, x_3. Both the quartic

dependence and tanh dependence emerge after KAN training. This is a simplified case of a localization phase transition discussed in Section 4.4.

(f) Deeper compositions $f(x_1, x_2, x_3, x_4) = \sqrt{(x_1 - x_2)^2 + (x_3 - x_4)^2}$. To compute this, we would need the identity function, squared function, and square root, which requires at least a three-layer KAN. Indeed, we find that a $[4, 3, 3, 1]$ KAN can be auto-pruned to a $[4, 2, 1, 1]$ KAN, which exactly corresponds to the computation graph we would expect.

More examples from the Feynman dataset and the special function dataset are visualized in Figure D.2, D.3, F.1, F.2 in Appendices D and F.

4.2 Unsupervised toy dataset

Often, scientific discoveries are formulated as supervised learning problems, i.e., given input variables $x_1, x_2, \cdots, x_d$ and output variable(s) y, we want to find an interpretable function f such that $y \approx f(x_1, x_2, \cdots, x_d)$. However, another type of scientific discovery can be formulated as unsupervised learning, i.e., given a set of variables $(x_1, x_2, \cdots, x_d)$, we want to discover a structural relationship between the variables. Specifically, we want to find a non-zero f such that

$$f(x_1, x_2, \cdots, x_d) \approx 0. \tag{4.1}$$

For example, consider a set of features (x_1, x_2, x_3) that satisfies $x_3 = \exp(\sin(\pi x_1) + x_2^2)$. Then a valid f is $f(x_1, x_2, x_3) = \sin(\pi x_1) + x_2^2 - \log(x_3) = 0$, implying that points of (x_1, x_2, x_3) form a 2D submanifold specified by $f = 0$ instead of filling the whole 3D space.

If an algorithm for solving the unsupervised problem can be devised, it has a considerable advantage over the supervised problem, since it requires only the sets of features $S = (x_1, x_2, \cdots, x_d)$. The supervised problem, on the other hand, tries to predict subsets of features in terms of the others, i.e. it splits $S = S_{\text{in}} \cup S_{\text{out}}$ into input and output features of the function to be learned. Without domain expertise to advise the splitting, there are $2^d - 2$ possibilities such that $|S_{\text{in}}| > 0$ and $|S_{\text{out}}| > 0$. This exponentially large space of supervised problems can be avoided by using the unsupervised approach. This unsupervised learning approach will be valuable to the knot dataset in Section 4.3. A Google Deepmind team [45] manually chose signature to be the target variable, otherwise they would face this combinatorial problem described above. This raises the question whether we can instead tackle the unsupervised learning directly. We present our method and a toy example below.

We tackle the unsupervised learning problem by turning it into a supervised learning problem on all of the d features, without requiring the choice of a splitting. The essential idea is to learn a function $f(x_1, \ldots, x_d) = 0$ such that f is not the 0-function. To do this, similar to contrastive learning, we define positive samples and negative samples: positive samples are feature vectors of real data. Negative samples are constructed by feature corruption. To ensure that the overall feature distribution for each topological invariant stays the same, we perform feature corruption by random permutation of each feature across the entire training set. Now we want to train a network g such that $g(\mathbf{x}_{\text{real}}) = 1$ and $g(\mathbf{x}_{\text{fake}}) = 0$ which turns the problem into a supervised problem. However, remember that we originally want $f(\mathbf{x}_{\text{real}}) = 0$ and $f(\mathbf{x}_{\text{fake}}) \neq 0$. We can achieve this by having $g = \sigma \circ f$ where $\sigma(x) = \exp(-\frac{x^2}{2w^2})$ is a Gaussian function with a small width w, which can be conveniently realized by a KAN with shape $[..., 1, 1]$ whose last activation is set to be the Gaussian function σ and all previous layers form f. Except for the modifications mentioned above, everything else is the same for supervised training.

Now we demonstrate that the unsupervised paradigm works for a synthetic example. Let us consider a 6D dataset, where (x_1, x_2, x_3) are dependent variables such that $x_3 = \exp(\sin(x_1) + x_2^2)$; (x_4, x_5)

Figure 4.2: Unsupervised learning of a toy task. KANs can identify groups of dependent variables, i.e., (x_1, x_2, x_3) and (x_4, x_5) in this case.

are dependent variables with $x_5 = x_4^3$; x_6 is independent of the other variables. In Figure 4.2, we show that for seed = 0, KAN reveals the functional dependence among x_1,x_2, and x_3; for another seed = 2024, KAN reveals the functional dependence between x_4 and x_5. Our preliminary results rely on randomness (different seeds) to discover different relations; in the future we would like to investigate a more systematic and more controlled way to discover a complete set of relations. Even so, our tool in its current status can provide insights for scientific tasks. We present our results with the knot dataset in Section 4.3.

4.3 Application to Mathematics: Knot Theory

Knot theory is a subject in low-dimensional topology that sheds light on topological aspects of three-manifolds and four-manifolds and has a variety of applications, including in biology and topological quantum computing. Mathematically, a knot K is an embedding of S^1 into S^3. Two knots K and K' are topologically equivalent if one can be deformed into the other via deformation of the ambient space S^3, in which case we write $[K] = [K']$. Some knots are topologically trivial, meaning that they can be smoothly deformed to a standard circle. Knots have a variety of deformation-invariant features f called topological invariants, which may be used to show that two knots are topologically inequivalent, $[K] \neq [K']$ if $f(K) \neq f(K')$. In some cases the topological invariants are geometric in nature. For instance, a hyperbolic knot K has a knot complement $S^3 \setminus K$ that admits a canonical hyperbolic metric g such that $\text{vol}_g(K)$ is a topological invariant known as the hyperbolic volume. Other topological invariants are algebraic in nature, such as the Jones polynomial.

Given the fundamental nature of knots in mathematics and the importance of its applications, it is interesting to study whether ML can lead to new results. For instance, in [46] reinforcement learning was utilized to establish ribbonness of certain knots, which ruled out many potential counterexamples to the smooth 4d Poincaré conjecture.

Supervised learning In [45], supervised learning and human domain experts were utilized to arrive at a new theorem relating algebraic and geometric knot invariants. In this case, gradient saliency identified key invariants for the supervised problem, which led the domain experts to make a conjecture that was subsequently refined and proven. We study whether a KAN can achieve good interpretable results on the same problem, which predicts the signature of a knot. Their main results from studying the knot theory dataset are:

(1) They use network attribution methods to find that the signature σ is mostly dependent on meridinal distance μ (real μ_r, imag μ_i) and longitudinal distance λ.

(2) Human scientists later identified that σ has high correlation with the $\text{slope} \equiv \text{Re}(\frac{\lambda}{\mu}) = \frac{\lambda\mu_r}{\mu_r^2+\mu_i^2}$ and derived a bound for $|2\sigma - \text{slope}|$.

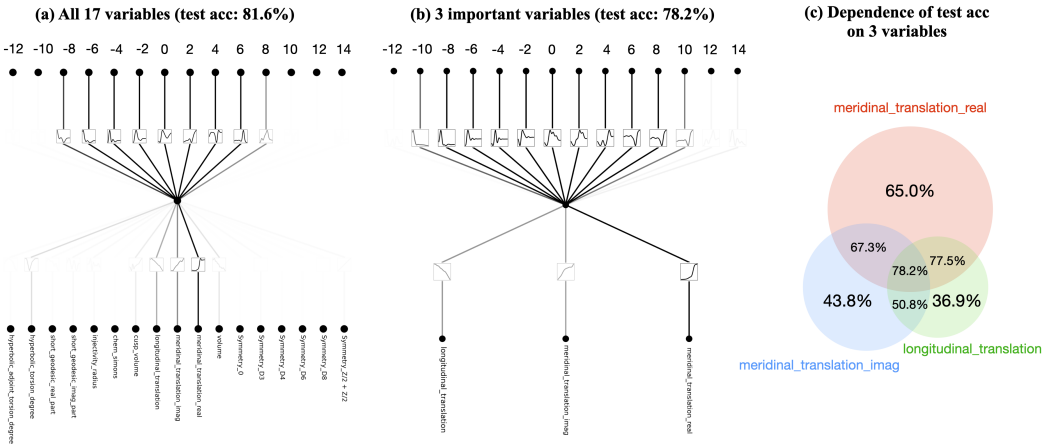


Figure 4.3: Knot dataset, supervised mode. With KANs, we rediscover Deepmind's results that signature is mainly dependent on meridinal translation (real and imaginary parts).

Method	Architecture	Parameter Count	Accuracy
Deepmind's MLP	4 layer, width-300	3×10^5	78.0%
KANs	2 layer, $[17, 1, 14]$ ($G = 3$, $k = 3$)	2×10^2	81.6%

Table 3: KANs can achieve better accuracy than MLPs with much fewer parameters in the signature classification problem. Soon after our preprint was first released, Prof. Shi Lab from Georgia tech discovered that an MLP with only 60 parameters is sufficient to achieve 80% accuracy (public but unpublished results). This is good news for AI + Science because this means perhaps many AI + Science tasks are not that computationally demanding than we might think (either with MLPs or with KANs), hence many new scientific discoveries are possible even on personal laptops.

We show below that KANs not only rediscover these results with much smaller networks and much more automation, but also present some interesting new results and insights.

To investigate (1), we treat 17 knot invariants as inputs and signature as outputs. Similar to the setup in [45], signatures (which are even numbers) are encoded as one-hot vectors and networks are trained with cross-entropy loss. We find that an extremely small $[17, 1, 14]$ KAN is able to achieve 81.6% test accuracy (while Deepmind's 4-layer width-300 MLP achieves 78% test accuracy). The $[17, 1, 14]$ KAN ($G = 3$, $k = 3$) has ≈ 200 parameters, while the MLP has $\approx 3 \times 10^5$ parameters, shown in Table 3. It is remarkable that KANs can be both more accurate and much more parameter efficient than MLPs at the same time. In terms of interpretability, we scale the transparency of each activation according to its magnitude, so it becomes immediately clear which input variables are important without the need for feature attribution (see Figure 4.3 left): signature is mostly dependent on μ_r, and slightly dependent on μ_i and λ, while dependence on other variables is small. We then train a $[3, 1, 14]$ KAN on the three important variables, obtaining test accuracy 78.2%. Our results have one subtle difference from results in [45]: they find that signature is mostly dependent on μ_i, while we find that signature is mostly dependent on μ_r. This difference could be due to subtle algorithmic choices, but has led us to carry out the following experiments: (a) ablation studies. We show that μ_r contributes more to accuracy than μ_i (see Figure 4.3): for example, μ_r alone can achieve 65.0% accuracy, while μ_i alone can only achieve 43.8% accuracy. (b) We find a symbolic formula (in Table 4) which only involves μ_r and λ, but can achieve 77.8% test accuracy.

To investigate (2), i.e., obtain the symbolic form of σ, we formulate the problem as a regression task. Using auto-symbolic regression introduced in Section 2.5.1, we can convert a trained KAN

Id	Formula	Discovered by	test acc	r^2 with Signature	r^2 with DM formula
A	$\frac{\lambda\mu_r}{(\mu_r^2+\mu_i^2)}$	Human (DM)	83.1%	0.946	1
B	$-0.02\sin(4.98\mu_i+0.85)+0.08\lvert 4.02\mu_r+6.28\rvert-0.52-0.04e^{-0.88(1-0.45\lambda)^2}$	[3, 1] KAN	62.6%	0.837	0.897
C	$0.17\tan(-1.51+0.1e^{-1.43(1-0.4\mu_i)^2+0.09e^{-0.06(1-0.21\lambda)^2}}+1.32e^{-3.18(1-0.43\mu_r)^2})$	[3, 1, 1] KAN	71.9%	0.871	0.934
D	$-0.09+1.04\exp(-9.59(-0.62\sin(0.61\mu_r+7.26))-0.32\tan(0.03\lambda-6.59)+1-0.11e^{-1.77(0.31-\mu_i)^2)^2}-1.09e^{-7.6(0.65(1-0.01\lambda)^3}+0.27\operatorname{atan}(0.53\mu_i-0.6)+0.09+\exp(-2.58(1-0.36\mu_r)^2))$	[3, 2, 1] KAN	84.0%	0.947	0.997
E	$\frac{4.76\lambda\mu_r}{3.09\mu_i+6.05\mu_r^2+3.54\mu_i^2}$	[3,2,1] KAN + Pade approx	82.8%	0.946	0.997
F	$\frac{2.94-2.92(1-0.10\mu_r)^2}{0.32(0.18-\mu_r)^2+5.36(1-0.04\lambda)^2+0.50}$	[3, 1] KAN/[3, 1] KAN	77.8%	0.925	0.977

Table 4: Symbolic formulas of signature as a function of meridinal translation μ (real μ_r, imag μ_i) and longitudinal translation λ. In [45], formula A was discovered by human scientists inspired by neural network attribution results. Formulas B-F are auto-discovered by KANs. KANs can trade-off between simplicity and accuracy (B, C, D). By adding more inductive biases, KAN is able to discover formula E which is not too dissimilar from formula A. KANs also discovered a formula F which only involves two variables (μ_r and λ) instead of all three variables, with little sacrifice in accuracy.

into symbolic formulas. We train KANs with shapes $[3,1]$, $[3,1,1]$, $[3,2,1]$, whose corresponding symbolic formulas are displayed in Table 4 B-D. It is clear that by having a larger KAN, both accuracy and complexity increase. So KANs provide not just a single symbolic formula, but a whole Pareto frontier of formulas, trading off simplicity and accuracy. However, KANs need additional inductive biases to further simplify these equations to rediscover the formula from [45] (Table 4 A). We have tested two scenarios: (1) in the first scenario, we assume the ground truth formula has a multi-variate Pade representation (division of two multi-variate Taylor series). We first train $[3,2,1]$ and then fit it to a Pade representation. We can obtain Formula E in Table 4, which bears similarity with Deepmind's formula. (2) We hypothesize that the division is not very interpretable for KANs, so we train two KANs (one for the numerator and the other for the denominator) and divide them manually. Surprisingly, we end up with the formula F (in Table 4) which only involves μ_r and λ, although μ_i is also provided but ignored by KANs.

So far, we have rediscovered the main results from [45]. It is remarkable to see that KANs made this discovery very intuitive and convenient. Instead of using feature attribution methods (which are great methods), one can instead simply stare at visualizations of KANs. Moreover, automatic symbolic regression also makes the discovery of symbolic formulas much easier.

In the next part, we propose a new paradigm of "AI for Math" not included in the Deepmind paper, where we aim to use KANs' unsupervised learning mode to discover more relations (besides signature) in knot invariants.

Unsupervised learning As we mentioned in Section 4.2, unsupervised learning is the setup that is more promising since it avoids manual partition of input and output variables which have combinatorially many possibilities. In the unsupervised learning mode, we treat all 18 variables (including signature) as inputs such that they are on the same footing. Knot data are positive samples, and we randomly shuffle features to obtain negative samples. An $[18,1,1]$ KAN is trained to classify whether a given feature vector belongs to a positive sample (1) or a negative sample (0). We manually set the second layer activation to be the Gaussian function with a peak one centered at zero, so positive samples will have activations at (around) zero, implicitly giving a relation among knot invariants $\sum_{i=1}^{18} g_i(x_i) = 0$ where x_i stands for a feature (invariant), and g_i is the corresponding

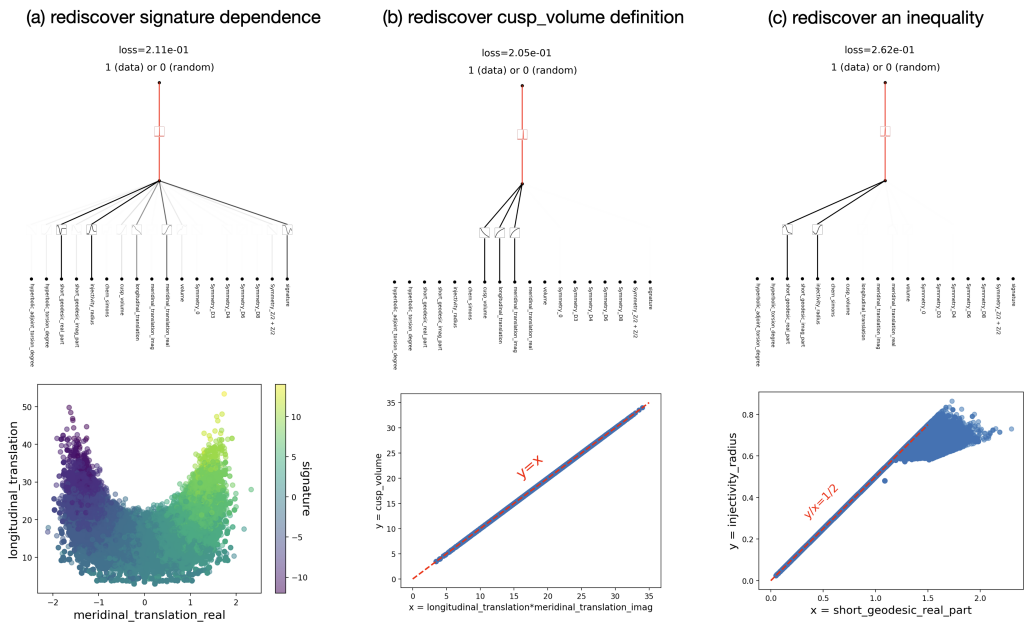


Figure 4.4: Knot dataset, unsupervised mode. With KANs, we rediscover three mathematical relations in the knot dataset.

activation function which can be readily read off from KAN diagrams. We train the KANs with $\lambda = \{10^{-2}, 10^{-3}\}$ to favor sparse combination of inputs, and $\text{seed} = \{0, 1, \cdots, 99\}$. All 200 networks can be grouped into three clusters, with representative KANs displayed in Figure 4.4. These three groups of dependent variables are:

(1) The first group of dependent variables is signature, real part of meridinal distance, and longitudinal distance (plus two other variables which can be removed because of (3)). This is the signature dependence studied above, so it is very interesting to see that this dependence relation is rediscovered again in the unsupervised mode.

(2) The second group of variables involve cusp volume V, real part of meridinal translation μ_r and longitudinal translation λ. Their activations all look like logarithmic functions (which can be verified by the implied symbolic functionality in Section 2.5.1). So the relation is $-\log V + \log \mu_r + \log \lambda = 0$ which is equivalent to $V = \mu_r \lambda$, which is true by definition. It is, however, reassuring that we discover this relation without any prior knowledge.

(3) The third group of variables includes the real part of short geodesic g_r and injectivity radius. Their activations look qualitatively the same but differ by a minus sign, so it is conjectured that these two variables have a linear correlation. We plot 2D scatters, finding that $2r$ upper bounds g_r, which is also a well-known relation [47].

It is interesting that KANs' unsupervised mode can rediscover several known mathematical relations. The good news is that the results discovered by KANs are probably reliable; the bad news is that we have not discovered anything new yet. It is worth noting that we have chosen a shallow KAN for simple visualization, but deeper KANs can probably find more relations if they exist. We would like to investigate how to discover more complicated relations with deeper KANs in future work.

4.4 Application to Physics: Anderson localization

Anderson localization is the fundamental phenomenon in which disorder in a quantum system leads to the localization of electronic wave functions, causing all transport to be ceased [48]. In one and two dimensions, scaling arguments show that all electronic eigenstates are exponentially localized for an infinitesimal amount of random disorder [49, 50]. In contrast, in three dimensions, a critical energy forms a phase boundary that separates the extended states from the localized states, known as a mobility edge. The understanding of these mobility edges is crucial for explaining various fundamental phenomena such as the metal-insulator transition in solids [51], as well as localization effects of light in photonic devices [52, 53, 54, 55, 56]. It is therefore necessary to develop microscopic models that exhibit mobility edges to enable detailed investigations. Developing such models is often more practical in lower dimensions, where introducing quasiperiodicity instead of random disorder can also result in mobility edges that separate localized and extended phases. Furthermore, experimental realizations of analytical mobility edges can help resolve the debate on localization in interacting systems [57, 58]. Indeed, several recent studies have focused on identifying such models and deriving exact analytic expressions for their mobility edges [59, 60, 61, 62, 63, 64, 65].

Here, we apply KANs to numerical data generated from quasiperiodic tight-binding models to extract their mobility edges. In particular, we examine three classes of models: the Mosaic model (MM) [63], the generalized Aubry-André model (GAAM) [62] and the modified Aubry-André model (MAAM) [60]. For the MM, we testify KAN's ability to accurately extract mobility edge as a 1D function of energy. For the GAAM, we find that the formula obtained from a KAN closely matches the ground truth. For the more complicated MAAM, we demonstrate yet another example of the symbolic interpretability of this framework. A user can simplify the complex expression obtained from KANs (and corresponding symbolic formulas) by means of a "collaboration" where the human generates hypotheses to obtain a better match (e.g., making an assumption of the form of certain activation function), after which KANs can carry out quick hypotheses testing.

To quantify the localization of states in these models, the inverse participation ratio (IPR) is commonly used. The IPR for the k^{th} eigenstate, $\psi^{(k)}$, is given by

$$\text{IPR}_k = \frac{\sum_n |\psi_n^{(k)}|^4}{\left(\sum_n |\psi_n^{(k)}|^2\right)^2} \tag{4.2}$$

where the sum runs over the site index. Here, we use the related measure of localization – the fractal dimension of the states, given by

$$D_k = -\frac{\log(\text{IPR}_k)}{\log(N)} \tag{4.3}$$

where N is the system size. $D_k = 0(1)$ indicates localized (extended) states.

Mosaic Model (MM) We first consider a class of tight-binding models defined by the Hamiltonian [63]

$$H = t\sum_n \left(c_{n+1}^\dagger c_n + \text{H.c.}\right) + \sum_n V_n(\lambda,\phi) c_n^\dagger c_n, \tag{4.4}$$

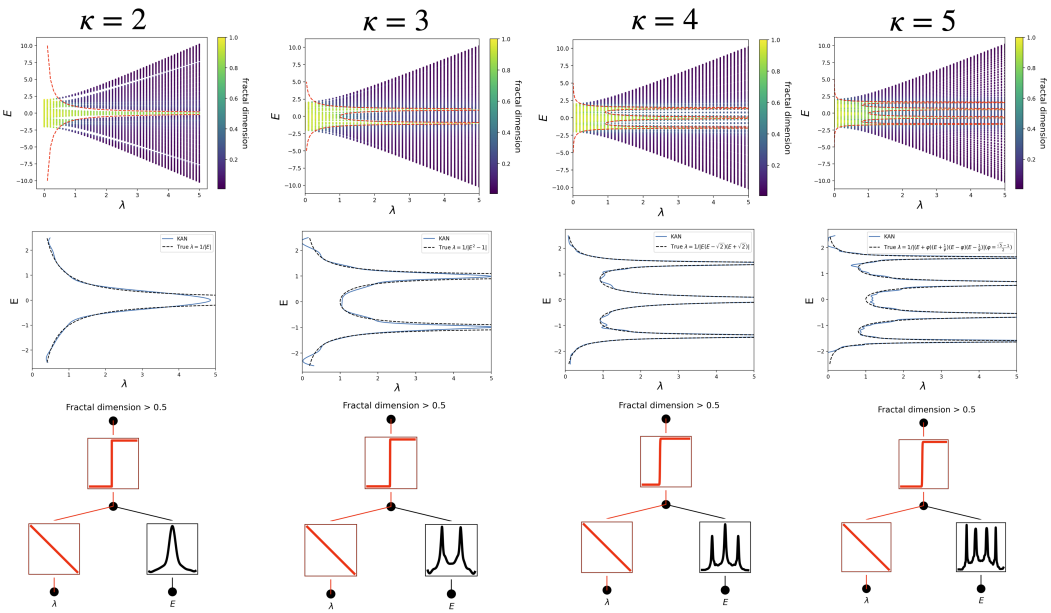


Figure 4.5: Results for the Mosaic Model. Top: phase diagram. Middle and Bottom: KANs can obtain both qualitative intuition (bottom) and extract quantitative results (middle). $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

where t is the nearest-neighbor coupling, $c_n(c_n^\dagger)$ is the annihilation (creation) operator at site n and the potential energy V_n is given by

$$V_n(\lambda, \phi) = \begin{cases} \lambda \cos(2\pi n b + \phi) & j = m\kappa \\ 0, & \text{otherwise,} \end{cases} \tag{4.5}$$

To introduce quasiperiodicity, we set b to be irrational (in particular, we choose b to be the golden ratio $\frac{1+\sqrt{5}}{2}$). κ is an integer and the quasiperiodic potential occurs with interval κ. The energy (E) spectrum for this model generically contains extended and localized regimes separated by a mobility edge. Interestingly, a unique feature found here is that the mobility edges are present for an arbitrarily strong quasiperiodic potential (i.e. there are always extended states present in the system that co-exist with localized ones).

The mobility edge can be described by $g(\lambda, E) \equiv \lambda - |f_\kappa(E)| = 0$. $g(\lambda, E) > 0$ and $g(\lambda, E) < 0$ correspond to localized and extended phases, respectively. Learning the mobility edge therefore hinges on learning the "order parameter" $g(\lambda, E)$. Admittedly, this problem can be tackled by many other theoretical methods for this class of models [63], but we will demonstrate below that our KAN framework is ready and convenient to take in assumptions and inductive biases from human users.

Let us assume a hypothetical user Alice, who is a new PhD student in condensed matter physics, and she is provided with a $[2, 1]$ KAN as an assistant for the task. Firstly, she understands that this is a classification task, so it is wise to set the activation function in the second layer to be sigmoid by using the `fix_symbolic` functionality. Secondly, she realizes that learning the whole 2D function $g(\lambda, E)$ is unnecessary because in the end she only cares about $\lambda = \lambda(E)$ determined by $g(\lambda, E) = 0$. In so doing, it is reasonable to assume $g(\lambda, E) = \lambda - h(E) = 0$. Alice simply sets the activation function of λ to be linear by again using the `fix_symbolic` functionality. Now Alice trains the KAN network and conveniently obtains the mobility edge, as shown in Figure 4.5. Alice can get both intuitive qualitative understanding (bottom) and quantitative results (middle), which well match the ground truth (top).

System	Origin	Mobility Edge Formula	Accuracy
GAAM	Theory	$\alpha E + 2\lambda - 2 = 0$	99.2%
	KAN auto	$\cancel{1.52E^2} + 21.06\alpha E + \cancel{0.66E} + \cancel{3.55\alpha^2} + \cancel{0.91\alpha} + 45.13\lambda - 54.45 = 0$	99.0%
MAAM	Theory	$E + \exp(p) - \lambda\cosh p = 0$	98.6%
	KAN auto	$13.99\sin(0.28\sin(0.87\lambda + 2.22) - 0.84\arctan(0.58E - 0.26) + 0.85\arctan(0.94p + 0.13) - 8.14) - 16.74 + 43.08\exp(-0.93(0.06(0.13 - p)^2 - 0.27\tanh(0.65E + 0.25) + 0.63\arctan(0.54\lambda - 0.62) + 1)^2) = 0$	97.1%
	KAN man (step 2) + auto	$4.19(0.28\sin(0.97\lambda + 2.17) - 0.77\arctan(0.83E - 0.19) + \arctan(0.97p + 0.15) - 0.35)^2 - 28.93 + 39.27\exp(-0.6(0.28\cosh^2(0.49p - 0.16) - 0.34\arctan(0.65E + 0.51) + 0.83\arctan(0.54\lambda - 0.62) + 1)^2) = 0$	97.7%
	KAN man (step 3) + auto	$-4.63E - 10.25(-0.94\sin(0.97\lambda - 6.81) + \tanh(0.8p - 0.45) + 0.09)^2 + 11.78\sin(0.76p - 1.41) + 22.49\arctan(1.08\lambda - 1.32) + 31.72 = 0$	97.7%
	KAN man (step 4A)	$6.92E - 6.23(-0.92\lambda - 1)^2 + 2572.45(-0.05\lambda + 0.95\cosh(0.11p + 0.4) - 1)^2 - 12.96\cosh^2(0.53p + 0.16) + 19.89 = 0$	96.6%
	KAN man (step 4B)	$7.25E - 8.81(-0.83\lambda - 1)^2 - 4.08(-p - 0.04)^2 + 12.71(-0.71\lambda + (0.3p + 1)^2 - 0.86)^2 + 10.29 = 0$	95.4%

Table 5: Symbolic formulas for two systems GAAM and MAAM, ground truth ones and KAN-discovered ones.

Generalized Andre-Aubry Model (GAAM) We next consider a class of tight-binding models defined by the Hamiltonian [62]

$$H = t\sum_n \left(c_{n+1}^\dagger c_n + \text{H.c.}\right) + \sum_n V_n(\alpha, \lambda, \phi) c_n^\dagger c_n, \tag{4.6}$$

where t is the nearest-neighbor coupling, $c_n(c_n^\dagger)$ is the annihilation (creation) operator at site n and the potential energy V_n is given by

$$V_n(\alpha, \lambda, \phi) = 2\lambda \frac{\cos(2\pi n b + \phi)}{1 - \alpha\cos(2\pi n b + \phi)}, \tag{4.7}$$

which is smooth for $\alpha \in (-1, 1)$. To introduce quasiperiodicity, we again set b to be irrational (in particular, we choose b to be the golden ratio). As before, we would like to obtain an expression for the mobility edge. For these models, the mobility edge is given by the closed form expression [62, 64],

$$\alpha E = 2(t - \lambda). \tag{4.8}$$

We randomly sample the model parameters: ϕ, α and λ (setting the energy scale $t = 1$) and calculate the energy eigenvalues as well as the fractal dimension of the corresponding eigenstates, which forms our training dataset.

Here the "order parameter" to be learned is $g(\alpha, E, \lambda, \phi) = \alpha E + 2(\lambda - 1)$ and mobility edge corresponds to $g = 0$. Let us again assume that Alice wants to figure out the mobility edge but only has access to IPR or fractal dimension data, so she decides to use KAN to help her with the task. Alice wants the model to be as small as possible, so she could either start from a large model and use auto-pruning to get a small model, or she could guess a reasonable small model based on her understanding of the complexity of the given problem. Either way, let us assume she arrives at a $[4, 2, 1, 1]$ KAN. First, she sets the last activation to be sigmoid because this is a classification problem. She trains her KAN with some sparsity regularization to accuracy 98.7% and visualizes the trained KAN in Figure 4.6 (a) step 1. She observes that ϕ is not picked up on at all, which makes her realize that the mobility edge is independent of ϕ (agreeing with Eq. (4.8)). In addition, she observes that almost all other activation functions are linear or quadratic, so she turns on automatic symbolic

snapping, constraining the library to be only linear or quadratic. After that, she immediately gets a network which is already symbolic (shown in Figure 4.6 (a) step 2), with comparable (even slightly better) accuracy 98.9%. By using `symbolic_formula` functionality, Alice conveniently gets the symbolic form of g, shown in Table 5 GAAM-KAN auto (row three). Perhaps she wants to cross out some small terms and snap coefficient to small integers, which takes her close to the true answer.

This hypothetical story for Alice would be completely different if she is using a symbolic regression method. If she is lucky, SR can return the exact correct formula. However, the vast majority of the time SR does not return useful results and it is impossible for Alice to "debug" or interact with the underlying process of symbolic regression. Furthermore, Alice may feel uncomfortable/inexperienced to provide a library of symbolic terms as prior knowledge to SR before SR is run. By constrast in KANs, Alice does not need to put any prior information to KANs. She can first get some clues by staring at a trained KAN and only then it is her job to decide which hypothesis she wants to make (e.g., "all activations are linear or quadratic") and implement her hypothesis in KANs. Although it is not likely for KANs to return the correct answer immediately, KANs will always return something useful, and Alice can collaborate with it to refine the results.

Modified Andre-Aubry Model (MAAM) The last class of models we consider is defined by the Hamiltonian [60]

$$H = \sum_{n \neq n'} t e^{-p|n-n'|} \left(c_n^\dagger c_{n'} + \text{H.c.} \right) + \sum_n V_n(\lambda, \phi) c_n^\dagger c_n, \tag{4.9}$$

where t is the strength of the exponentially decaying coupling in space, $c_n(c_n^\dagger)$ is the annihilation (creation) operator at site n and the potential energy V_n is given by

$$V_n(\lambda, \phi) = \lambda \cos(2\pi n b + \phi), \tag{4.10}$$

As before, to introduce quasiperiodicity, we set b to be irrational (the golden ratio). For these models, the mobility edge is given by the closed form expression [60],

$$\lambda \cosh(p) = E + t = E + t_1 \exp(p) \tag{4.11}$$

where we define $t_1 \equiv t\exp(-p)$ as the nearest neighbor hopping strength, and we set $t_1 = 1$ below.

Let us assume Alice wants to figure out the mobility edge for MAAM. This task is more complicated and requires more human wisdom. As in the last example, Alice starts from a $[4, 2, 1, 1]$ KAN and trains it but gets an accuracy around 75% which is less than acceptable. She then chooses a larger $[4, 3, 1, 1]$ KAN and successfully gets 98.4% which is acceptable (Figure 4.6 (b) step 1). Alice notices that ϕ is not picked up on by KANs, which means that the mobility edge is independent of the phase factor ϕ (agreeing with Eq. (4.11)). If Alice turns on the automatic symbolic regression (using a large library consisting of exp, tanh etc.), she would get a complicated formula in Tabel 5-MAAM-KAN auto, which has 97.1% accuracy. However, if Alice wants to find a simpler symbolic formula, she will want to use the manual mode where she does the symbolic snapping by herself. Before that she finds that the $[4, 3, 1, 1]$ KAN after training can then be pruned to be $[4, 2, 1, 1]$, while maintaining 97.7% accuracy (Figure 4.6 (b)). Alice may think that all activation functions except those dependent on p are linear or quadratic and snap them to be either linear or quadratic manually by using `fix_symbolic`. After snapping and retraining, the updated KAN is shown in Figure 4.6 (c) step 3, maintaining 97.7% accuracy. From now on, Alice may make two different choices based on her prior knowledge. In one case, Alice may have guessed that the dependence on p is $\cosh$, so she sets the activations of p to be $\cosh$ function. She retrains KAN and gets 96.9% accuracy (Figure 4.6 (c) Step 4A). In another case, Alice does not know the $\cosh$ p dependence, so she pursues simplicity

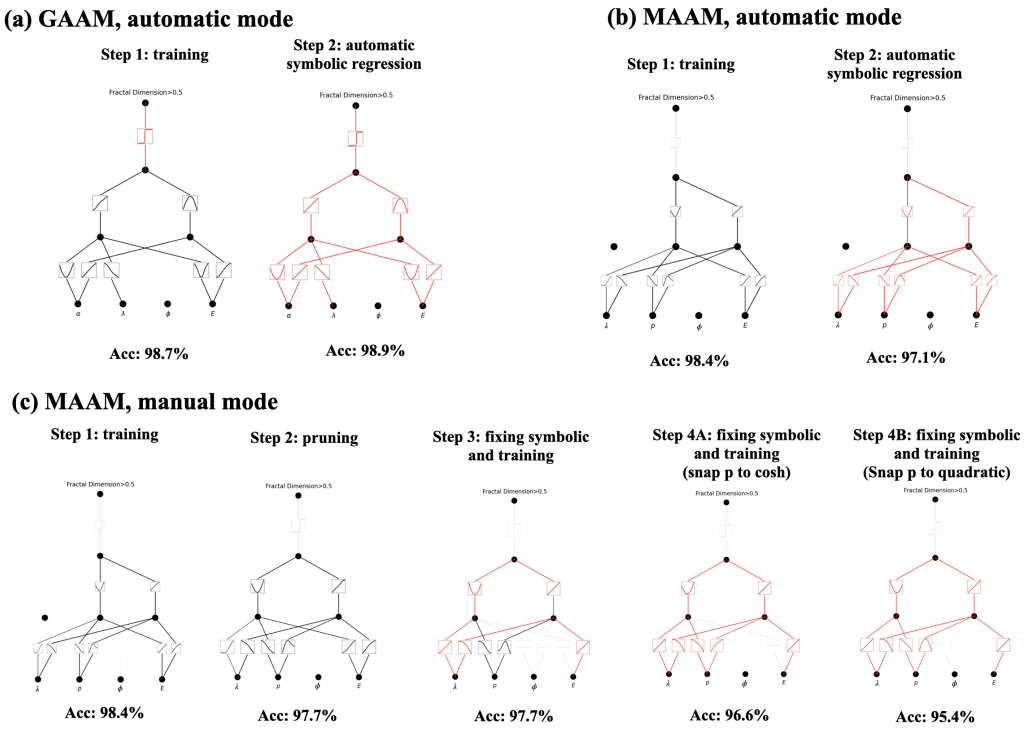


Figure 4.6: Human-KAN collaboration to discover mobility edges of GAAM and MAAM. The human user can choose to be lazy (using the auto mode) or more involved (using the manual mode). More details in text.

and again assumes the functions of p to be quadratic. She retrains KAN and gets 95.4% accuracy (Figure 4.6 (c) Step 4B). If she tried both, she would realize that $\cosh$ is better in terms of accuracy, while quadratic is better in terms of simplicity. The formulas corresponding to these steps are listed in Table 5. It is clear that the more manual operations are done by Alice, the simpler the symbolic formula is (which slight sacrifice in accuracy). KANs have a "knob" that a user can tune to trade-off between simplicity and accuracy (sometimes simplicity can even lead to better accuracy, as in the GAAM case).

5 Related works

Kolmogorov-Arnold theorem and neural networks. The connection between the Kolmogorov-Arnold theorem (KAT) and neural networks is not new in the literature [66, 67, 9, 10, 11, 12, 13, 14, 68, 69], but the pathological behavior of inner functions makes KAT appear unpromising in practice [66]. Most of these prior works stick to the original 2-layer width-$(2n+1)$ networks, which were limited in expressive power and many of them are even predating back-propagation. Therefore, most studies were built on theories with rather limited or artificial toy experiments. More broadly speaking, KANs are also somewhat related to generalized additive models (GAMs) [70], graph neural networks [71] and kernel machines [72]. The connections are intriguing and fundamental but might be out of the scope of the current paper. Our contribution lies in generalizing the Kolmogorov network to arbitrary widths and depths, revitalizing and contexualizing them in today's deep learning stream, as well as highlighting its potential role as a foundation model for AI + Science.

Neural Scaling Laws (NSLs). NSLs are the phenomena where test losses behave as power laws against model size, data, compute etc [73, 74, 75, 76, 24, 77, 78, 79]. The origin of NSLs still

remains mysterious, but competitive theories include intrinsic dimensionality [73], quantization of tasks [78], resource theory [79], random features [77], compositional sparsity [66], and maximu arity [25]. This paper contributes to this space by showing that a high-dimensional function can surprisingly scale as a 1D function (which is the best possible bound one can hope for) if it has a smooth Kolmogorov-Arnold representation. Our paper brings fresh optimism to neural scaling laws, since it promises the fastest scaling exponent ever. We have shown in our experiments that this fast neural scaling law can be achieved on synthetic datasets, but future research is required to address the question whether this fast scaling is achievable for more complicated tasks (e.g., language modeling): Do KA representations exist for general tasks? If so, does our training find these representations in practice?

Mechanistic Interpretability (MI). MI is an emerging field that aims to mechanistically understand the inner workings of neural networks [80, 81, 82, 83, 84, 85, 86, 87, 5]. MI research can be roughly divided into passive and active MI research. Most MI research is passive in focusing on understanding existing neural networks trained with standard methods. Active MI research attempts to achieve interpretability by designing intrinsically interpretable architectures or developing training methods to explicitly encourage interpretability [86, 87]. Our work lies in the second category, where the model and training method are by design interpretable.

Learnable activations. The idea of learnable activations in neural networks is not new in machine learning. Trainable activations functions are learned in a differentiable way [88, 14, 89, 90] or searched in a discrete way [91]. Activation function are parametrized as polynomials [88], splines [14, 92, 93], sigmoid linear unit [89], or neural networks [90]. KANs use B-splines to parametrize their activation functions. We also present our preliminary results on learnable activation networks (LANs), whose properties lie between KANs and MLPs and their results are deferred to Appendix B to focus on KANs in the main paper.

Symbolic Regression. There are many off-the-shelf symbolic regression methods based on genetic algorithms (Eureka [94], GPLearn [95], PySR [96]), neural-network based methods (EQL [97], OccamNet [98]), physics-inspired method (AI Feynman [36, 37]), and reinforcement learning-based methods [99]. KANs are most similar to neural network-based methods, but differ from previous works in that our activation functions are continuously learned before symbolic snapping rather than manually fixed [94, 98].

Physics-Informed Neural Networks (PINNs) and Physics-Informed Neural Operators (PINOs). In Subsection 3.4, we demonstrate that KANs can replace the paradigm of using MLPs for imposing PDE loss when solving PDEs. We refer to Deep Ritz Method [100], PINNs [38, 39, 101] for PDE solving, and Fourier Neural operator [102], PINOs [103, 104, 105], DeepONet [106] for operator learning methods learning the solution map. There is potential to replace MLPs with KANs in all the aforementioned networks.

AI for Mathematics. As we saw in Subsection 4.3, AI has recently been applied to several problems in Knot theory, including detecting whether a knot is the unknot [107, 108] or a ribbon knot [46], and predicting knot invariants and uncovering relations among them [109, 110, 111, 45]. For a summary of data science applications to datasets in mathematics and theoretical physics see e.g. [112, 113], and for ideas how to obtain rigorous results from ML techniques in these fields, see [114].

6 Discussion

In this section, we discuss KANs' limitations and future directions from the perspective of mathematical foundation, algorithms and applications.

Mathematical aspects: Although we have presented preliminary mathematical analysis of KANs (Theorem 2.1), our mathematical understanding of them is still very limited. The Kolmogorov-Arnold representation theorem has been studied thoroughly in mathematics, but the theorem corresponds to KANs with shape $[n, 2n+1, 1]$, which is a very restricted subclass of KANs. Does our empirical success with deeper KANs imply something fundamental in mathematics? An appealing generalized Kolmogorov-Arnold theorem could define "deeper" Kolmogorov-Arnold representations beyond depth-2 compositions, and potentially relate smoothness of activation functions to depth. Hypothetically, there exist functions which cannot be represented smoothly in the original (depth-2) Kolmogorov-Arnold representations, but might be smoothly represented with depth-3 or beyond. Can we use this notion of "Kolmogorov-Arnold depth" to characterize function classes?

Algorithmic aspects: We discuss the following:

(1) Accuracy. Multiple choices in architecture design and training are not fully investigated so alternatives can potentially further improve accuracy. For example, spline activation functions might be replaced by radial basis functions or other local kernels. Adaptive grid strategies can be used.

(2) Efficiency. One major reason why KANs run slowly is because different activation functions cannot leverage batch computation (large data through the same function). Actually, one can interpolate between activation functions being all the same (MLPs) and all different (KANs), by grouping activation functions into multiple groups ("multi-head"), where members within a group share the same activation function.

(3) Hybrid of KANs and MLPs. KANs have two major differences compared to MLPs:

   (i) activation functions are on edges instead of on nodes,

   (ii) activation functions are learnable instead of fixed.

   Which change is more essential to explain KAN's advantage? We present our preliminary results in Appendix B where we study a model which has (ii), i.e., activation functions are learnable (like KANs), but not (i), i.e., activation functions are on nodes (like MLPs). Moreover, one can also construct another model with fixed activations (like MLPs) but on edges (like KANs).

(4) Adaptivity. Thanks to the intrinsic locality of spline basis functions, we can introduce adaptivity in the design and training of KANs to enhance both accuracy and efficiency: see the idea of multi-level training like multigrid methods as in [115, 116], or domain-dependent basis functions like multiscale methods as in [117].

Application aspects: We have presented some preliminary evidences that KANs are more effective than MLPs in science-related tasks, e.g., fitting physical equations and PDE solving. We would like to apply KANs to solve Navier-Stokes equations, density functional theory, or any other tasks that can be formulated as regression or PDE solving. We would also like to apply KANs to machine-learning-related tasks, which would require integrating KANs into current architectures, e.g., transformers – one may propose "kansformers" which replace MLPs by KANs in transformers.

KAN as a "language model" for AI + Science The reason why large language models are so transformative is because they are useful to anyone who can speak natural language. The language of science is functions. KANs are composed of interpretable functions, so when a human user stares at a KAN, it is like communicating with it using the language of functions. This paragraph aims to promote the AI-Scientist-Collaboration paradigm rather than our specific tool KANs. Just like people use different languages to communicate, we expect that in the future KANs will be just one

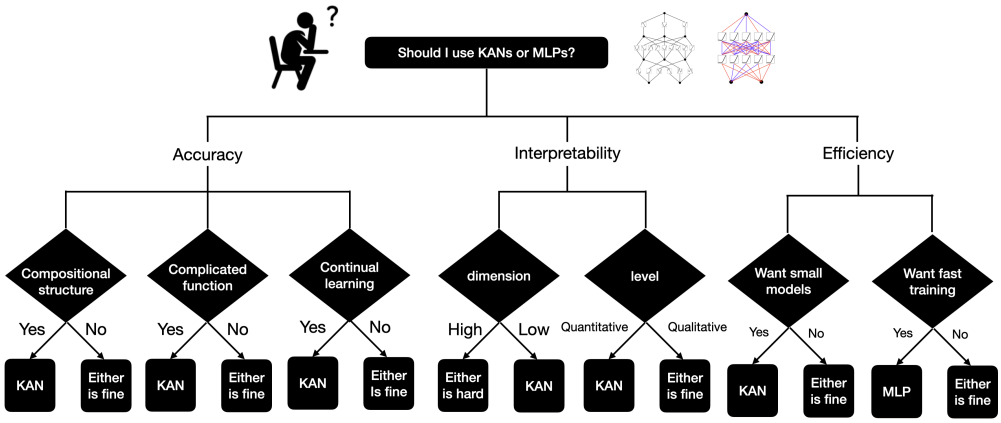


Figure 6.1: Should I use KANs or MLPs?

of the languages for AI + Science, although KANs will be one of the very first languages that would enable AI and human to communicate. However, enabled by KANs, the AI-Scientist-Collaboration paradigm has never been this easy and convenient, which leads us to rethink the paradigm of how we want to approach AI + Science: Do we want AI scientists, or do we want AI that helps scientists? The intrinsic difficulty of (fully automated) AI scientists is that it is hard to make human preferences quantitative, which would codify human preferences into AI objectives. In fact, scientists in different fields may feel differently about which functions are simple or interpretable. As a result, it is more desirable for scientists to have an AI that can speak the scientific language (functions) and can conveniently interact with inductive biases of individual scientist(s) to adapt to a specific scientific domain.

Final takeaway: Should I use KANs or MLPs?

Currently, the biggest bottleneck of KANs lies in its slow training. KANs are usually 10x slower than MLPs, given the same number of parameters. We should be honest that we did not try hard to optimize KANs' efficiency though, so we deem KANs' slow training more as an engineering problem to be improved in the future rather than a fundamental limitation. If one wants to train a model fast, one should use MLPs. In other cases, however, KANs should be comparable or better than MLPs, which makes them worth trying. The decision tree in Figure 6.1 can help decide when to use a KAN. In short, if you care about interpretability and/or accuracy, and slow training is not a major concern, we suggest trying KANs, at least for small-scale AI + Science problems.

Acknowledgement

We would like to thank Mikail Khona, Tomaso Poggio, Pingchuan Ma, Rui Wang, Di Luo, Sara Beery, Catherine Liang, Yiping Lu, Nicholas H. Nelsen, Nikola Kovachki, Jonathan W. Siegel, Hongkai Zhao, Juncai He, Shi Lab (Humphrey Shi, Steven Walton, Chuanhao Yan) and Matthieu Darcy for fruitful discussion and constructive suggestions. Z.L., F.R., J.H., M.S. and M.T. are supported by IAIFI through NSF grant PHY-2019786. The work of FR is in addition supported by the NSF grant PHY-2210333 and by startup funding from Northeastern University. Y.W and T.H are supported by the NSF Grant DMS-2205590 and the Choi Family Gift Fund. S. V. and M. S. acknowledge support from the U.S. Office of Naval Research (ONR) Multidisciplinary University Research Initiative (MURI) under Grant No. N00014-20-1-2325 on Robust Photonic Materials with Higher-Order Topological Protection.