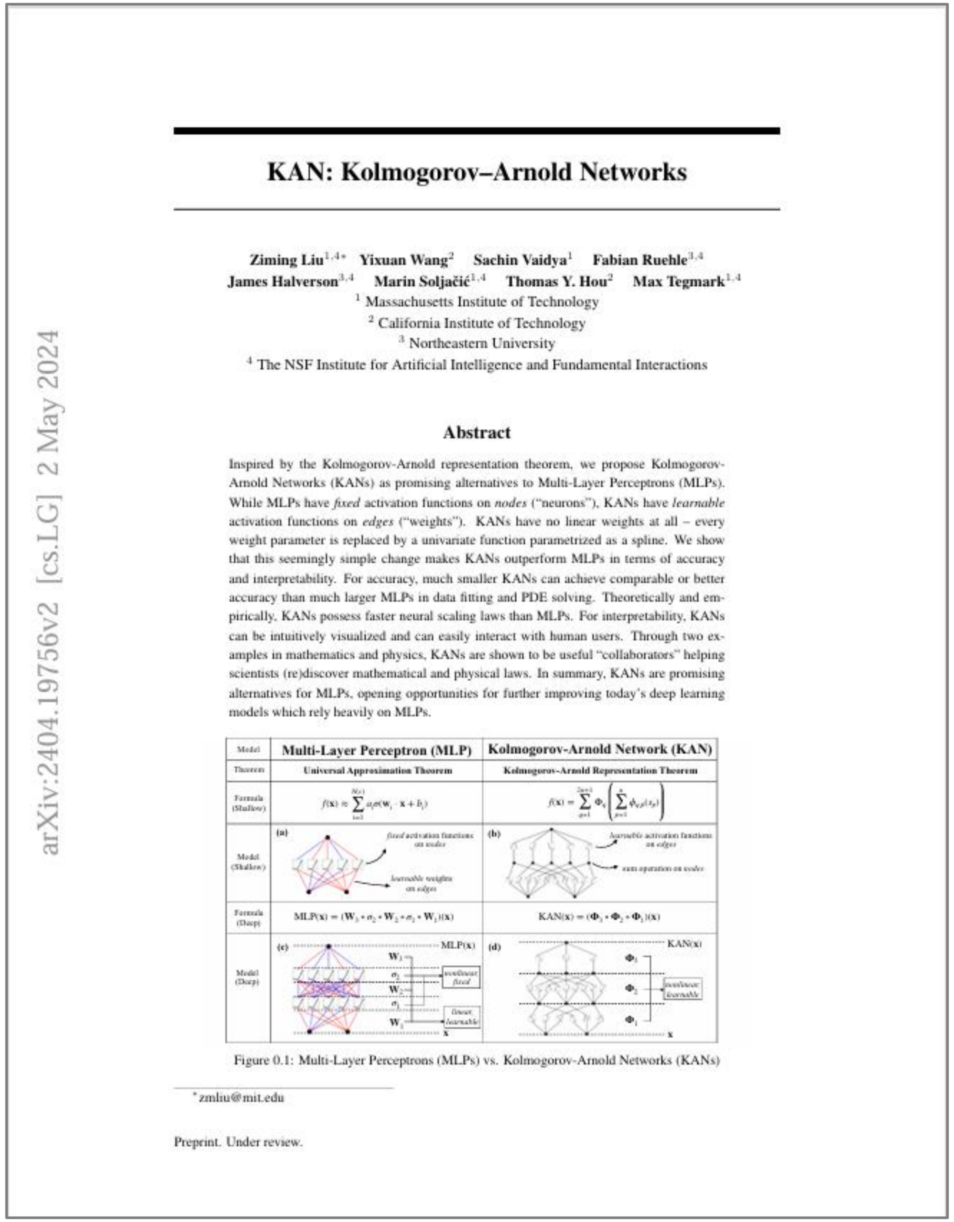
KAN: Kolmogorov-Arnold Networks

May 30, 2024

KAN: Kolmogorov-Arnold Networks

Ziming Liu et al., MIT+

https://arxiv.org/abs/2404.19756


arXiv:2404.19756v2 [cs.LG] 2 May 2024

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. For accuracy, much smaller KANs can achieve comparable or better accuracy than much larger MLPs in data fitting and PDE solving. 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)	(a) fixed activation functions on nodes; learnable weights on edges	(b) learnable activation functions on edges; sum operation on nodes
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)	(c) MLP(x), $\mathbf{W}_3$, σ_2, $\mathbf{W}_2$, σ_1, $\mathbf{W}_1$, x; nonlinear, fixed; linear, learnable	(d) KAN(x), $\mathbf{\Phi}_3$, $\mathbf{\Phi}_2$, $\mathbf{\Phi}_1$, x; nonlinear, learnable

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

*zmliu@mit.edu

Preprint. Under review.

KAN overview

- Most deep neural network architectures are built on the multi-layer perceptron (MLP) building block
  - If you draw a graph of nodes and edges, the nodes do interesting work with linear weights combining inputs and a single nonlinearity on their output
- The Kolmogorov-Arnold Network (KAN) building block is motivated by the Kolmogorov-Arnold representation theorem
  - In a graph of nodes and edges, the edges do interesting work, calculating complex nonlinear functions
- Authors fiddled and wound up building edge functions with B-splines
- Claim more expressivity per parameter, but also more expensive
- Paper only solved toy problems, so TBD whether it works real world

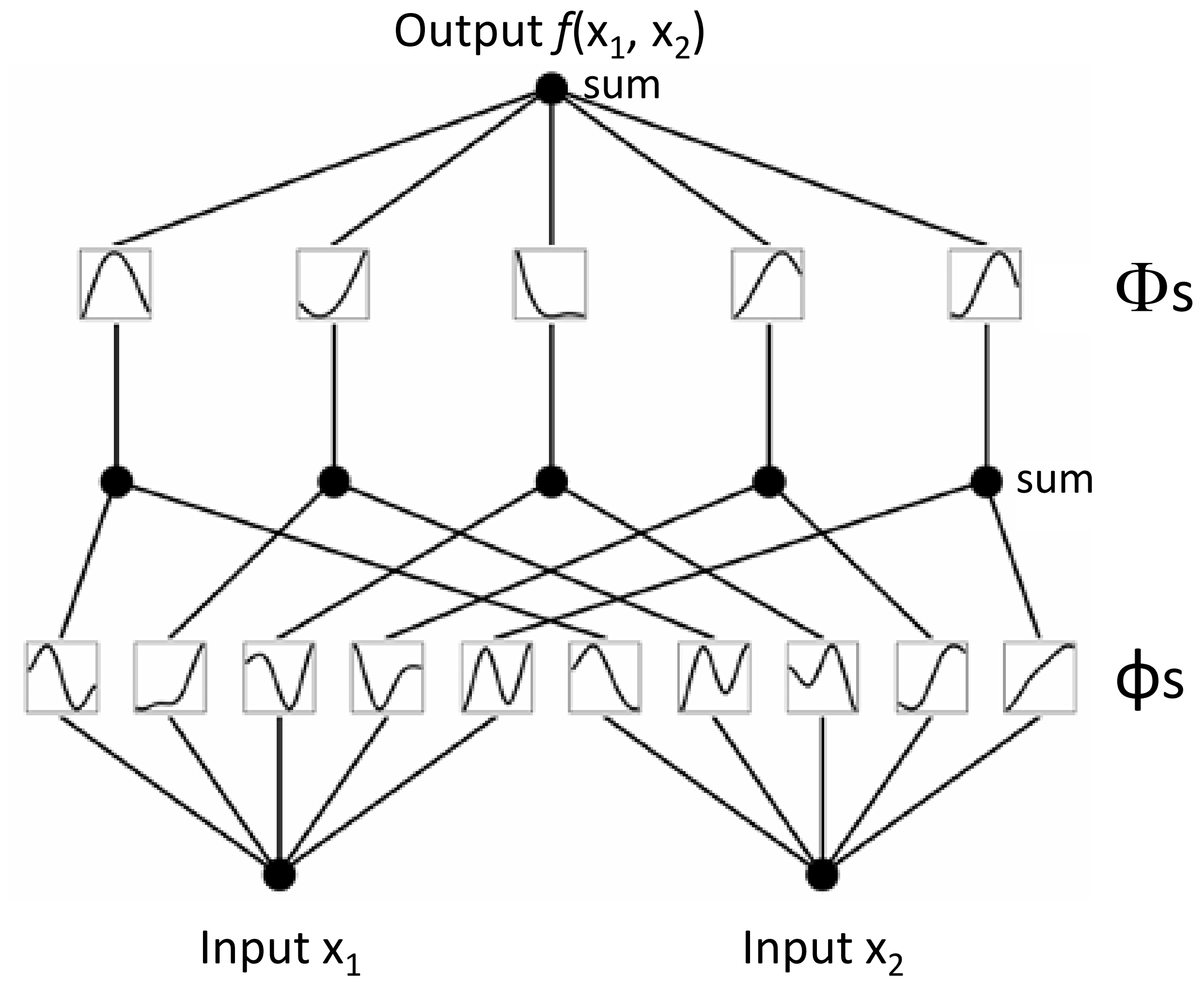
K-A representation theorem

- MLPs have universal approximation theorem
  - Infinitely wide, single hidden layer can approximate any function
- Kolmogorov-Arnold representation theorem
  - For a multivariate continuous function in *n* dimensions on a bounded domain
  - Can create $2n+1$ single variable functions named Φ_q
  - For each of these Φ_q, create *n* more single variable functions, one per dimension, naming these $\phi_{q,p}$
  - So, $2n+1$ Φ_q and $(2n+1)(n)=2n^2+n$ $\phi_{q,p}$ for a total of $2n^2+3n+1$ functions
  - Our original function is the sum of the $2n+1$ Φ_q, each called on the sum of its respective *n* input functions named $\phi_{q,p}$. In other words:

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

K-A representation example

- Any 2D function $f(x_1, x_2)$ can be represented by 15 1D functions
- You need 10 $\phi_{q,p}()$ functions, 5 that operate on x_1, and 5 that operate on x_2
- You need 5 $\Phi_q()$ functions that each operate on the sum of a pair of $\phi_{q,p}()$
- Our function $f()$ is the sum of the five $\Phi_q()$



KAN motivation

- K-A representation theorem sounds like great news:
  High dimensional functions are much harder to learn, so we can just learn O(n^2) univariate functions instead, right?
- Bad news is the theorem allows functions that are not smooth and possibly fractal
- For these problematic functions, gradient descent isn't going to work
- But authors were optimistic that maybe real world problems don't require those pathological functions, so we can try this anyway
- MLP universal approx. is only one hidden layer in middle, but deep networks work better. K-A rep. theorem also looks like one hidden layer, but maybe this idea will also work better if wider and deeper?

KAN architecture

- In the K-A representation theorem for n dimensions, you always have $(n+1)(n)$ functions being summed n at a time, feeding into $n+1$ functions, which are summed into your final output
- The above structure has width $n+1$ in the middle and two steps
- More generally, you can vary the width and increase the steps
- We define a KAN layer for n_{in}-dimensional inputs and n_{out}-dimensional outputs as $(n_{\text{in}})\cdot(n_{\text{out}})$ univariate functions summed n_{in} at a time
- The K-A representation theorem for n dimensions has two KAN layers
  - The first layer is always $n_{\text{in}} = n$ and $n_{\text{out}} = 2n+1$
  - The second layer is always $n_{\text{in}} = 2n+1$ and $n_{\text{out}} = 1$

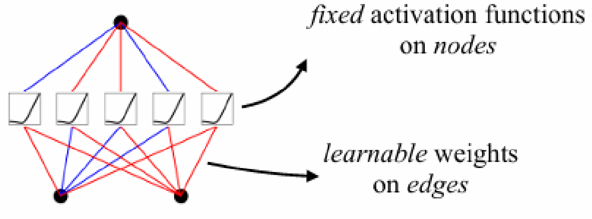
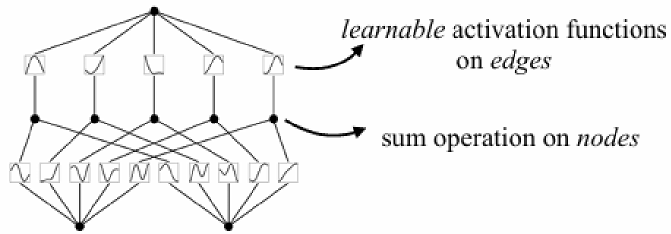
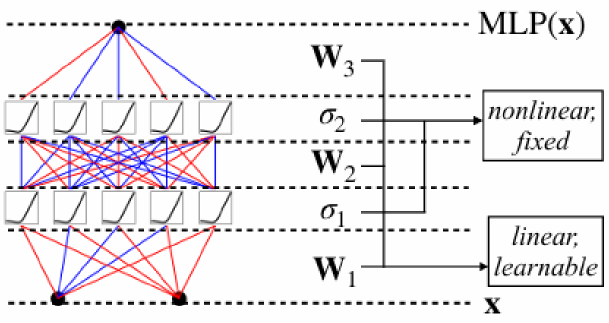
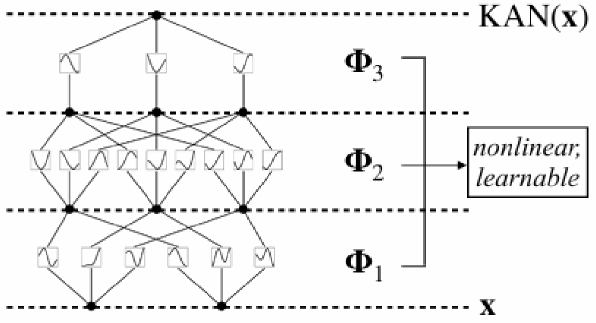
Nodes and edges

- Both MLPs and KANs can be drawn as graphs with nodes and edges
  - In both, the input/output nodes represent the input/output dimensionality
- In MLPs:
  - The hidden nodes do interesting work with linear weights combining inputs and a single nonlinearity on their output
  - Edges are uninteresting, applying identity function connecting output to input
- In KANs:
  - The edges do interesting work, calculating complex nonlinear functions
  - Nodes are simple, summing their inputs without any weighting
- With $n_{\text{in}} = 4$ and $n_{\text{out}} = 16$ for a single MLP or KAN layer:
  - The MLP will have $4 \cdot 16 = 64$ weights, optional 16 biases, 1 activation, 16 outputs
  - The KAN will have $4 \cdot 16 = 64$ functions, summed in groups of 4, for 16 outputs

KAN paper terminology

- Note that the KAN paper tries to make KANs as analogous to MLPs as possible, and also tries to make KANs sound simple and beautiful
- Composing linear functions does not increase expressivity, so in MLPs, you need a nonlinearity, named the *activation function,* from biology
  - The KAN paper calls its univariate functions activation functions, which is potentially confusing, and is a departure from the K-A representation theorem
- In many places, the KAN paper also buries the details that the nodes perform unweighted sums of functions (e.g., composition of layers)
- Also, both capital Φ and lowercase ϕ are used to mean different things
  - The Φ used when describing the KAN layer is NOT the same as the Φ in the K-A representation theorem

MLP vs KAN comparison

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)	**(a)** 	**(b)** 
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)	**(c)** 	**(d)** 

KAN details

- The KAN uses splines to approximate 1D functions
  - Unlike polynomial regression, B-splines will behave well between data points
  - Splines have local behavior, so you can optimize nearest control points only
- In each KAN layer, the $(n_{\text{in}})\cdot(n_{\text{out}})$ learnable univariate functions, named ϕ, are parameterized as: $w(\text{silu}(x) + \text{spline}(x))$
  - No explanation is given for why the silu(x) is there
- Each spline is the sum of weighted B-splines: $\text{spline}(x) = \sum_i c_i B_i(x)$
- When cubic splines are used, order k=3. With G grid points, each function will have about G parameters
  - Grid extension – if inputs exceed initial range, grid will be shifted/extended
- For L layers all same width N, when using G grid points, the total number of parameters will be around $O(N^2LG)$ parameters

Support for interpretability

- Sparsification – simple L1 regularization is not enough, but with entropy regularization, you can encourage many functions to be zero
- Visualization – author proves tool to see thumbnails of activations
- Pruning – even after sparsification may want to prune nodes with low input and output values
- Symbolification – you can say that an activation looks close to known functions such as cos or log, so author provides a fitting technique
    - This trick works for the toy examples that are known to be composed of only a few known functions, but seems to me to be useless for anything real world

Symbolic regression example

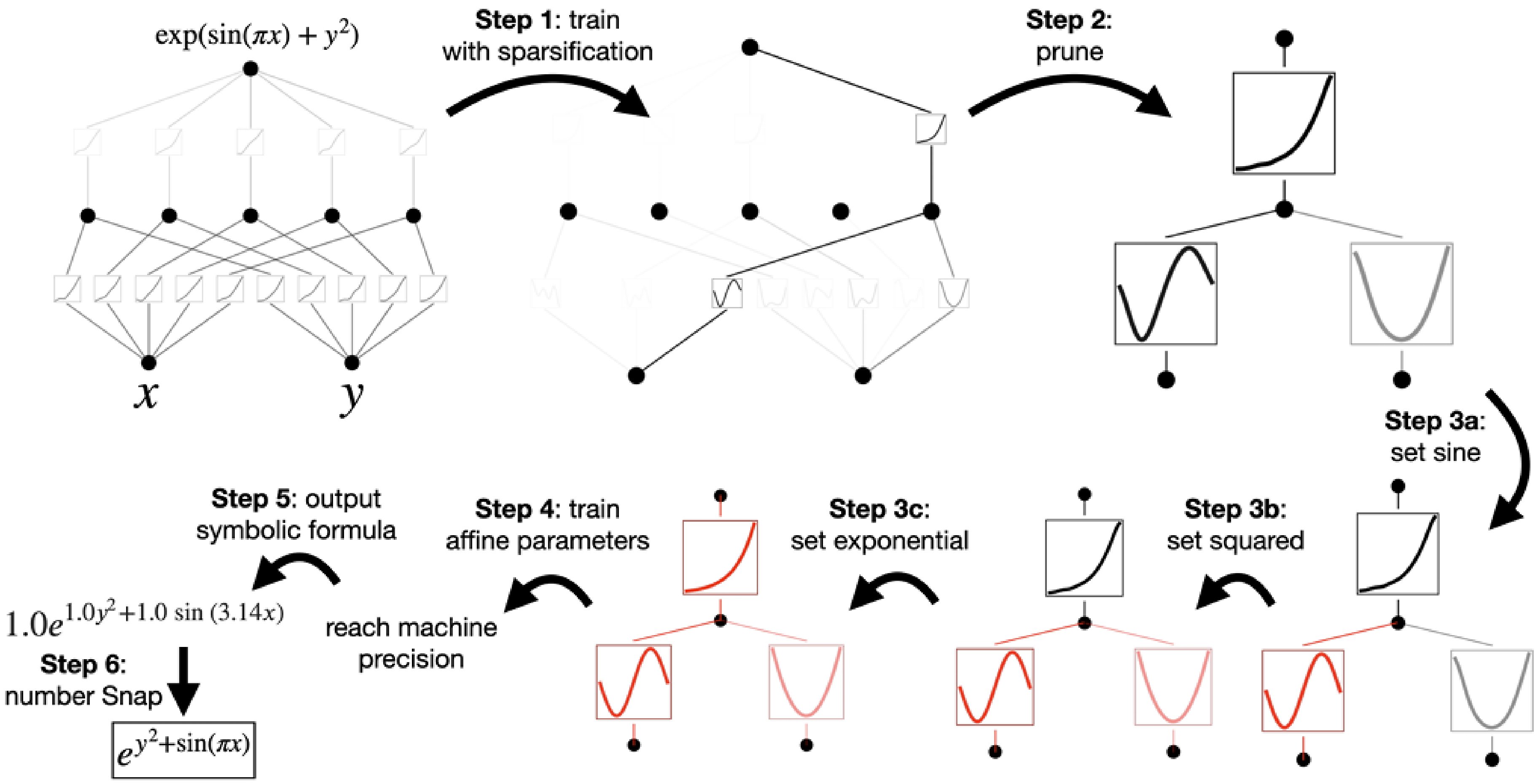


Figure 2.4: An example of how to do symbolic regression with KAN.

Accurate results on toy datasets [1]

- First, they tried toy functions such as $\exp(\sin(\pi x)+y^2)$
  - If you pick a KAN architecture that closely matches the toy function and increase the number of grid points, it will approximate the known functions fairly well
- Also tried special functions
  - I think these are all kind of simple & smooth

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}

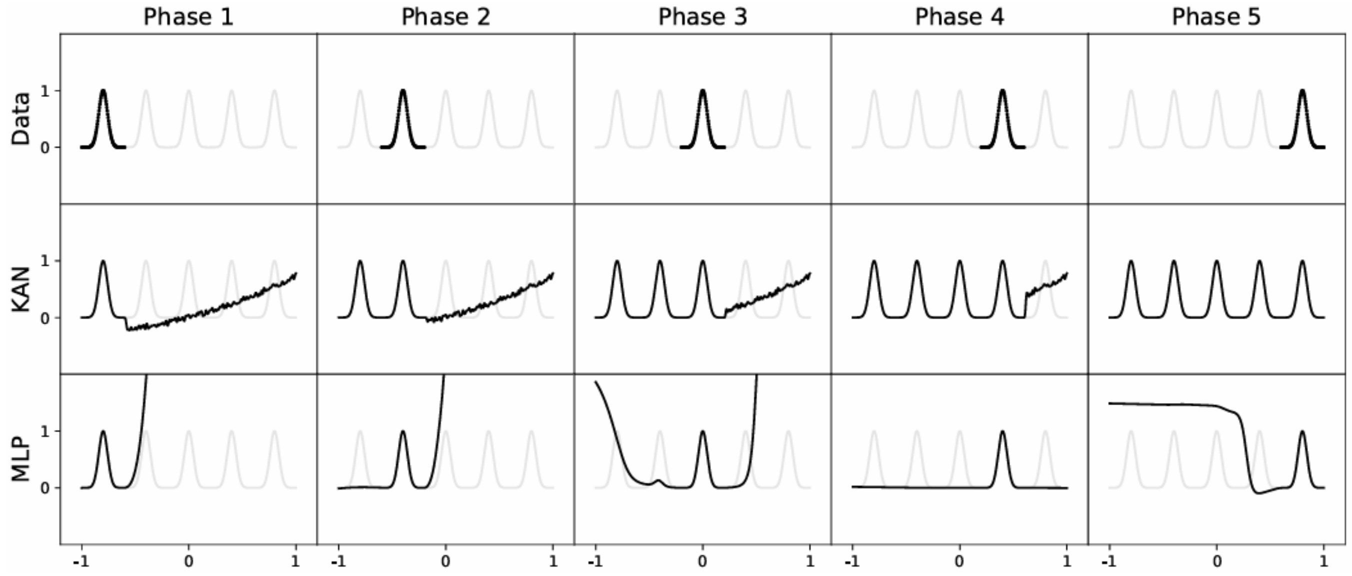
Accurate results on toy datasets [2]

- Tested on Fenman datasets
  - Tried hand crafted KAN shape, but pruning tended to beat hand crafted

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]	$\mathbf{1.48 \times 10^{-3}}$	8.62×10^{-3}	6.56×10^{-3}	1.59×10^{-3}
I.12.11	$q(E_f + Bv\sin\theta)$	$1 + a\sin\theta$	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	$\frac{x-ut}{\sqrt{1-(\frac{u}{c})^2}}$	$\frac{1-a}{\sqrt{1-b^2}}$	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	$I_* = I_1 + I_2 + 2\sqrt{I_1I_2}\cos\delta$	$1 + a + 2\sqrt{a}\cos\delta$	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_0e^{-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}

Continual learning

- Continual learning example of a Gaussian mixture of 5 modes presented sequentially
  - When new data extends the domain, the KAN's function grids will also extend
  - Very unclear if this has any relationship to generalize to avoid catastrophic forgetting for realistic problems



Interpretability examples

- Again, for toy functions, they were able to find KANs which built interpretable calculations, such as $2xy = (x + y)^2 - (x^2 + y^2)$
- Paper also talks about some unsupervised learning task, a knot theory problem, and Anderson localization from physics.

- Paper proved these KANs can learn. But are they suited for real world problems, such as CIFAR-10 classification?
- MLPs work fundamentally differently because they are made of simpler parts. Not sure if the KAN claims of better accuracy are because they compared to MLPs that weren't big enough.

KAN conclusion

- KANs loosely use K-A representation theorem as inspiration for composing 1D functions to estimate multivariate function
- Use splines to approximate 1D functions, and go wider and deeper
- In the paper, they mostly solve toy problems
- Seems useful to explore more complexity than linear weights of MLP
  - Authors suggest possibility of combining properties of MLPs and KANs
- Much hype, so many people are now trying to go beyond the original paper and implement KANs to solve MNIST, to replace portions of transformer, etc.

References

- Author’s KAN repo
  https://github.com/KindXiaoming/pykan
- Kolmogorov–Arnold representation theorem
  Wikipedia
  https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Arnold_representation_theorem
- B-spline
  Wikipedia
  https://en.wikipedia.org/wiki/B-spline