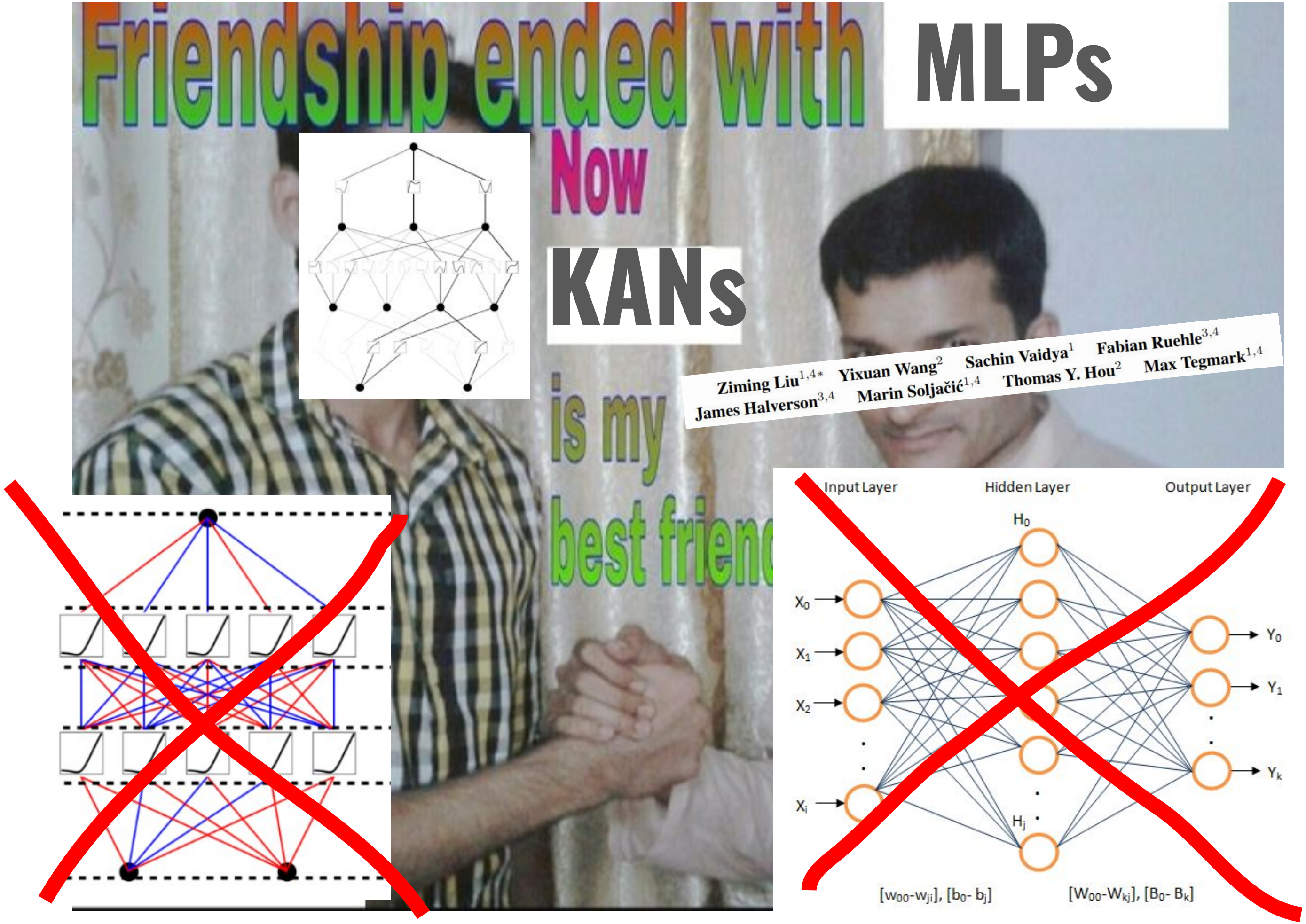
KAN: Kolmogorov–Arnold Networks

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

May 2024


Friendship ended with
MLPs
Now
KANs
is my
best frienc
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]
Input Layer
Hidden Layer
Output Layer
H_0
X_0
X_1
X_2
X_i
H_j
Y_0
Y_1
Y_k
$[w_{00}-w_{ji}]$, $[b_0- b_j]$
$[W_{00}-W_{kj}]$, $[B_0- B_k]$

Section 1: Motivation

Motivation

- Use Kolmogorov-Arnold representation theorem instead of universal approximation theorem

- More interpretable than MLPs
    - Can even do symbolic regression

- More parameter efficient than MLPs

Representation theorems

- MLPs: Universal Approximation Theorem
  - *Any multivariate continuous function on a bounded domain,* $f : [0,1]^n \to \mathbb{R}$ *can be approximated arbitrarily well by an MLP with a wide-enough hidden layer.*

Representation theorems

- MLPs: Universal Approximation Theorem
  - *Any multivariate continuous function on a bounded domain,* $f : [0,1]^n \to \mathbb{R}$ *can be approximated arbitrarily well by an MLP with a wide-enough hidden layer.*

- KANs: Kolmogorov-Arnold Representation Theorem
  - *Any multivariate continuous function on a bounded domain,* $f : [0,1]^n \to \mathbb{R}$ *can be written exactly using only addition and a finite composition of continuous univariate functions.*

Representation theorems

- MLPs: Universal Approximation Theorem
  - *Any multivariate continuous function on a bounded domain,* $f : [0,1]^n \to \mathbb{R}$ *can be approximated arbitrarily well by an MLP with a wide-enough hidden layer.*

- KANs: Kolmogorov-Arnold Representation Theorem
  - *Any multivariate continuous function on a bounded domain,* $f : [0,1]^n \to \mathbb{R}$ *can be written <u>exactly</u> using only addition and a finite composition of continuous univariate functions. Specifically, with* $\phi_{q,p} : [0,1] \to \mathbb{R}$ and $\Phi_q : \mathbb{R} \to \mathbb{R}$

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

Limitations

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

$$\begin{aligned} f &: [0,1]^n \to \mathbb{R} \\ \Phi_q &: \mathbb{R} \to \mathbb{R} \\ \phi_{q,p} &: [0,1] \to \mathbb{R} \end{aligned}$$

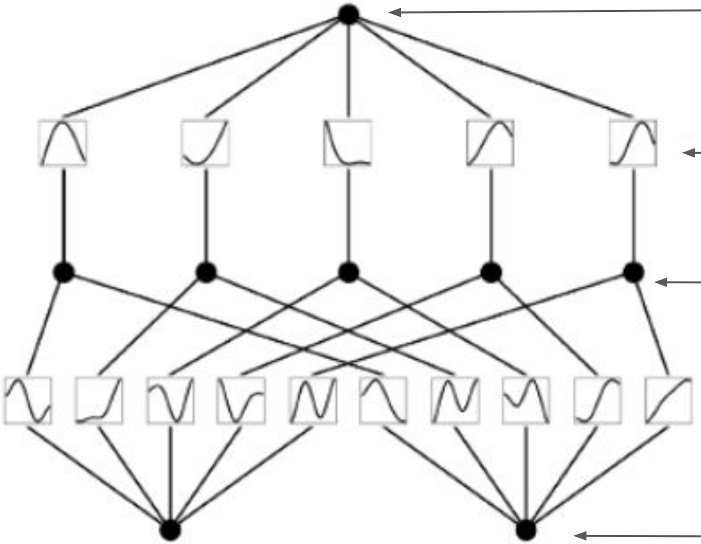
- Great! Learning any continuous multivariate function only requires us to learn a few univariate functions.
- But these univariate functions, although continuous, are often *awful* to work with (e.g. non-smooth, fractal etc.).
- So let's try: 1) stacking 'layers' of the above form;

KAN Diagrams

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

$$\begin{aligned} f &: [0,1]^n \to \mathbb{R} \\ \Phi_q &: \mathbb{R} \to \mathbb{R} \\ \phi_{q,p} &: [0,1] \to \mathbb{R} \end{aligned}$$

$n = 2$



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

$\Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$

$\sum_{p=1}^{n} \phi_{q,p}(x_p)$

$\phi_{q,p}(x_p)$

x_p

Limitations

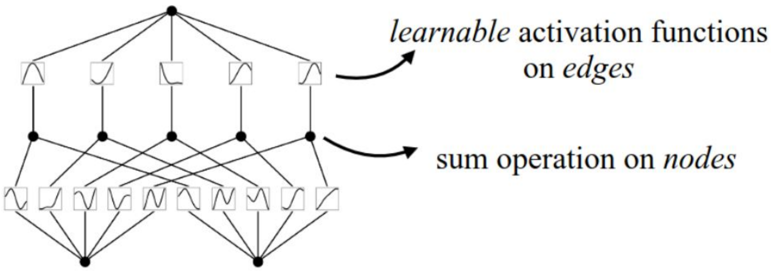
$$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$$

$$\begin{aligned} f &: [0,1]^n \to \mathbb{R} \\ \Phi_q &: \mathbb{R} \to \mathbb{R} \\ \phi_{q,p} &: [0,1] \to \mathbb{R} \end{aligned}$$

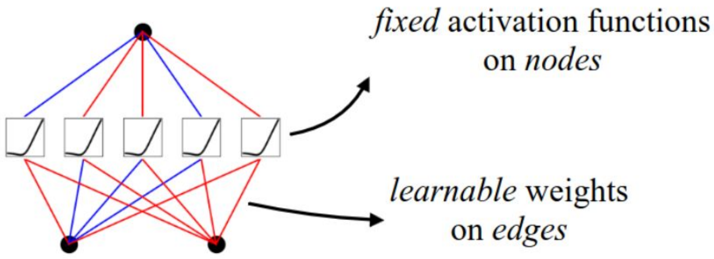
- Great! Learning any continuous multivariate function only requires us to learn a few univariate functions.
- But these univariate functions, although continuous, are often *awful* to work with (e.g. non-smooth, fractal etc.).
- So let's try: 1) stacking 'layers' of the above form; and 2) ignoring the n, 2n+1 requirement for the number of 'nodes' per layer

KAN Diagrams vs. MLP Diagrams

KAN
[2,5,1]



MLP



KANs are like MLPs but:

1. with learnable activation functions on each weight
2. with nodes only performing addition (no ‘global’ activation functions e.g. ReLU)

Section 2: Defining a KAN

Defining a KAN layer

- A KAN layer with n_{in}-dimensional inputs and n_{out}-dimensional outputs can be defined as a matrix of 1D functions

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

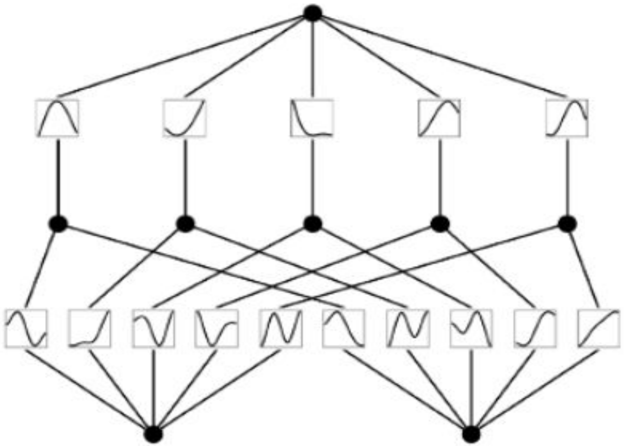
  where each function $\phi_{q,p}$ has trainable parameters (we'll use splines).

- Note that Kolmogorov-Arnold representation comes corresponds to two layers:
  - An inner layer with

    $$n_{\text{in}} = n \text{ and } n_{\text{out}} = 2n + 1$$

  - And an outer layer with

    $$n_{\text{in}} = 2n + 1 \text{ and } n_{\text{out}} = 1$$



Composing KAN layers

- Given the shape of a KAN (as an integer array), $[n_0, n_1, \cdots, n_L]$, write the activation in the (l, i)-neuron as $x_{l,i}$, and the activation function joining (l, i) and $(l + 1, j)$ as $\phi_{l,j,i}$.

Composing KAN layers

- Given the shape of a KAN (as an integer array), $[n_0, n_1, \cdots, n_L]$, write the activation in the (l, i)-neuron as $x_{l,i}$, and the activation function joining (l, i) and $(l+1, j)$ as $\phi_{l,j,i}$.
- Then the activation at $(l+1, j)$ is $x_{l+1,j} = \sum_{i=1}^{n_l} \tilde{x}_{l,j,i} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}),$

Composing KAN layers

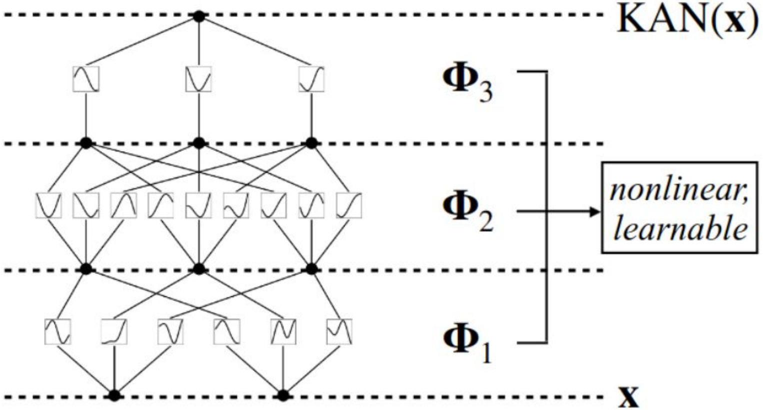
- Given the shape of a KAN (as an integer array), $[n_0, n_1, \cdots, n_L]$, write the activation in the (l, i)-neuron as $x_{l,i}$, and the activation function joining (l, i) and $(l+1, j)$ as $\phi_{l,j,i}$.
- Then the activation at $(l+1, j)$ is $x_{l+1,j} = \sum_{i=1}^{n_l} \tilde{x}_{l,j,i} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}),$
- In matrix form: $\mathbf{x}_{l+1} = \underbrace{\begin{pmatrix} \phi_{l,1,1}(\cdot) & \phi_{l,1,2}(\cdot) & \cdots & \phi_{l,1,n_l}(\cdot) \\ \phi_{l,2,1}(\cdot) & \phi_{l,2,2}(\cdot) & \cdots & \phi_{l,2,n_l}(\cdot) \\ \vdots & \vdots & & \vdots \\ \phi_{l,n_{l+1},1}(\cdot) & \phi_{l,n_{l+1},2}(\cdot) & \cdots & \phi_{l,n_{l+1},n_l}(\cdot) \end{pmatrix}}_{\mathbf{\Phi}_l} \mathbf{x}_l,$

Composing KAN layers

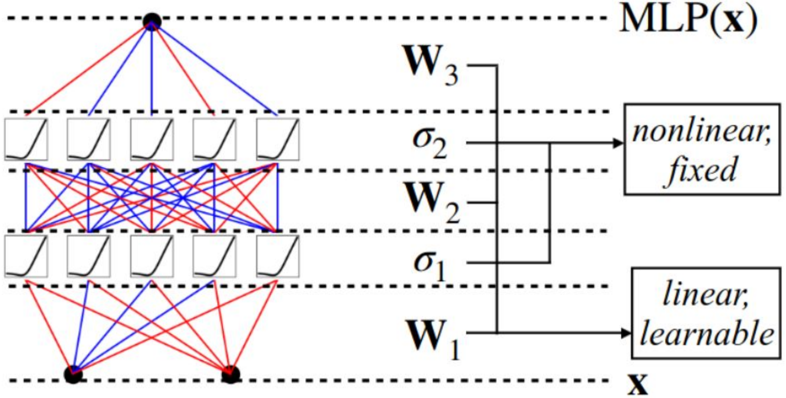
- Given the shape of a KAN (as an integer array), $[n_0, n_1, \cdots, n_L]$, write the activation in the (l, i)-neuron as $x_{l,i}$, and the activation function joining (l, i) and $(l+1, j)$ as $\phi_{l,j,i}$.
- Then the activation at $(l+1, j)$ is $x_{l+1,j} = \sum_{i=1}^{n_l} \tilde{x}_{l,j,i} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}),$
- In matrix form: $$\mathbf{x}_{l+1} = \underbrace{\begin{pmatrix} \phi_{l,1,1}(\cdot) & \phi_{l,1,2}(\cdot) & \cdots & \phi_{l,1,n_l}(\cdot) \\ \phi_{l,2,1}(\cdot) & \phi_{l,2,2}(\cdot) & \cdots & \phi_{l,2,n_l}(\cdot) \\ \vdots & \vdots & & \vdots \\ \phi_{l,n_{l+1},1}(\cdot) & \phi_{l,n_{l+1},2}(\cdot) & \cdots & \phi_{l,n_{l+1},n_l}(\cdot) \end{pmatrix}}_{\mathbf{\Phi}_l} \mathbf{x}_l,$$
- Finally, $\mathrm{KAN}(\mathbf{x}) = (\mathbf{\Phi}_{L-1} \circ \mathbf{\Phi}_{L-2} \circ \cdots \circ \mathbf{\Phi}_1 \circ \mathbf{\Phi}_0)\mathbf{x}$

KAN vs MLP (deep)

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



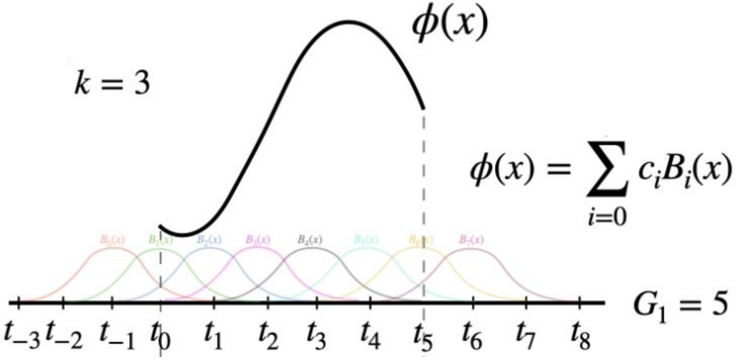
$$\mathrm{MLP}(\mathbf{x}) = (\mathbf{W}_{L-1} \circ \sigma \circ \mathbf{W}_{L-2} \circ \sigma \circ \cdots \circ \mathbf{W}_1 \circ \sigma \circ \mathbf{W}_0)\mathbf{x}$$



What type of functions $\phi_{l,j,i}$ are we learning?

- Splines!
  - *A spline function of order k is a piecewise polynomial function of degree k-1.*

- To simplify things, we'll learn coefficients for B-splines of order k, which:
  - form a basis for all splines of order k
  - can be found deterministically (though not all that cheaply) given k and a grid of size G over which to define our function's 'pieces'

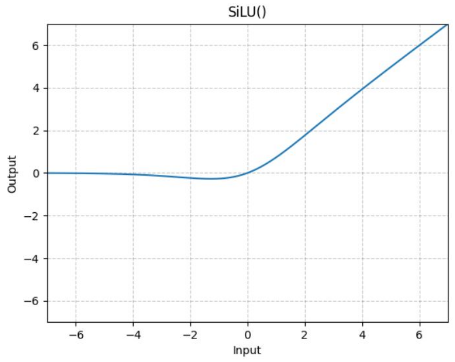


Implementation details

$$\phi(x) = w\,(b(x) + \mathrm{spline}(x))$$

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

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



- Splines initialised ($c_i \sim \mathcal{N}(0, \sigma^2)$) with a small grid, G_1 =3, but every 100 or so iterations, we'll make the grid finer, say, G_2=5 , and choose new coefficients via least squares on the outputs of each spline:

$$\{c'_j\} = \underset{\{c'_j\}}{\mathrm{argmin}} \underset{x \sim p(x)}{\mathbb{E}} \left(\sum_{j=0}^{G_2+k-1} c'_j B'_j(x) - \sum_{i=0}^{G_1+k-1} c_i B_i(x) \right)^2$$

KANs are parameter efficient!

- A KAN with:
  - Depth L;
  - Constant width N; using
  - Splines of order k, defined on grids of size G

KANs are parameter efficient!

- A KAN with:
  - Depth L;
  - Constant width N; using
  - Splines of order k, defined on grids of size G
- Parameters: $O(N^2L(G+k)) \sim O(N^2LG)$

KANs are parameter efficient!

- A KAN with:
  - Depth L;
  - Constant width N; using
  - Splines of order k, defined on grids of size G
- Parameters: $O(N^2L(G+k)) \sim O(N^2LG)$
- (cf. MLPs: $O(N^2L)$)

KANs are parameter efficient!

- A KAN with:
  - Depth L;
  - Constant width N; using
  - Splines of order k, defined on grids of size G
- Parameters: $O(N^2L(G+k)) \sim O(N^2LG)$
- (cf. MLPs: $O(N^2L)$)
- But KANs seem to use their parameters more efficiently than MLPs

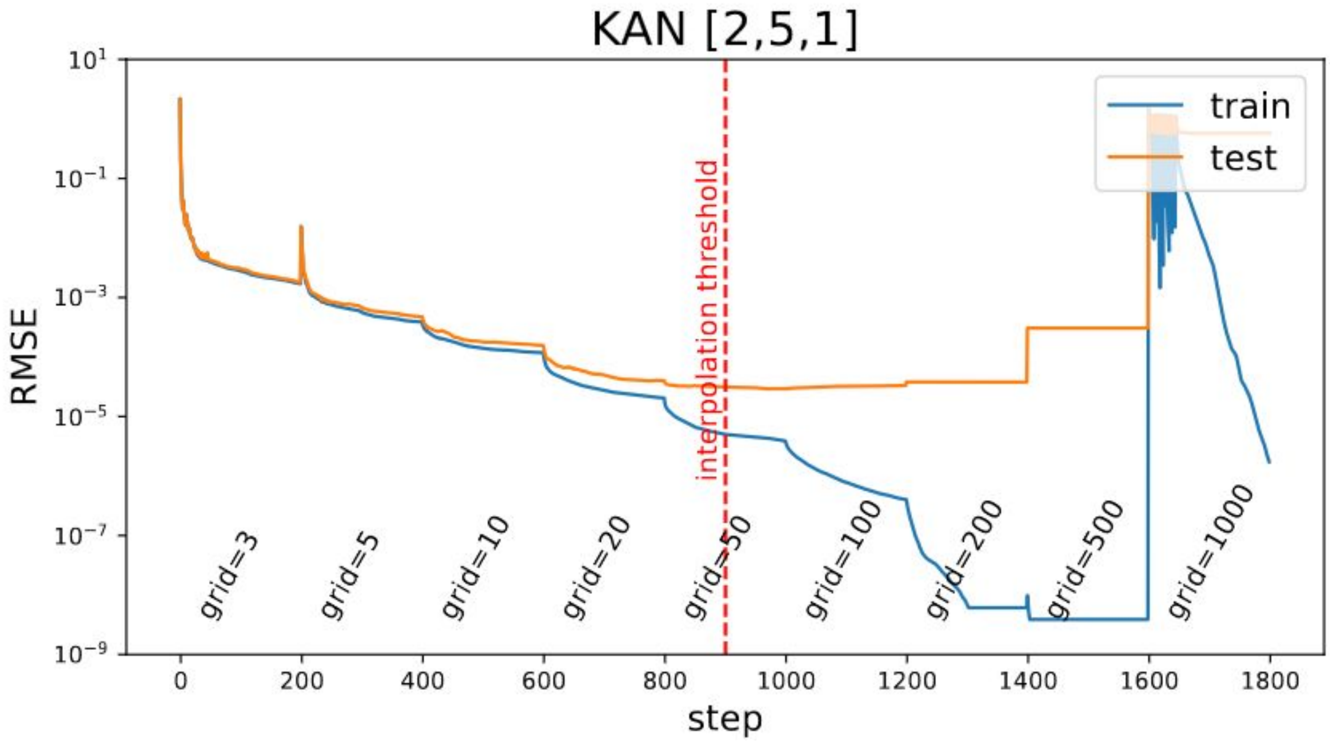
KANs are parameter efficient!

- A KAN with:
  - Depth L;
  - Constant width N; using
  - Splines of order k, defined on grids of size G
- Parameters: $O(N^2L(G+k)) \sim O(N^2LG)$
- (cf. MLPs: $O(N^2L)$)
- But KANs seem to use their parameters more efficiently than MLPs
  - Theoretical scaling exponents for test RMSE loss: $l \propto (\#\text{params})^{-\alpha}$

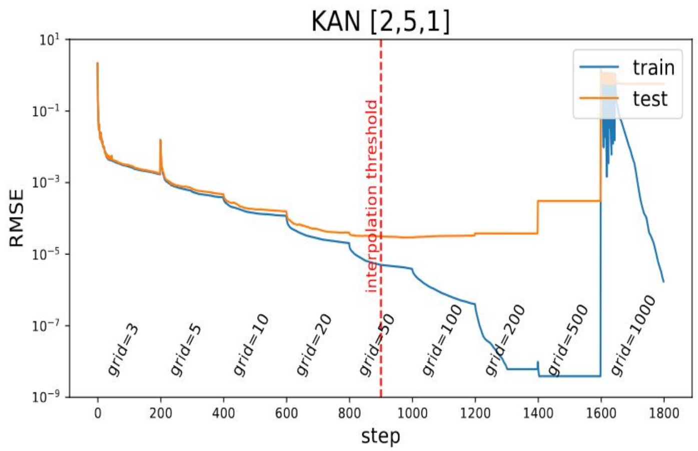
KANs are parameter efficient!

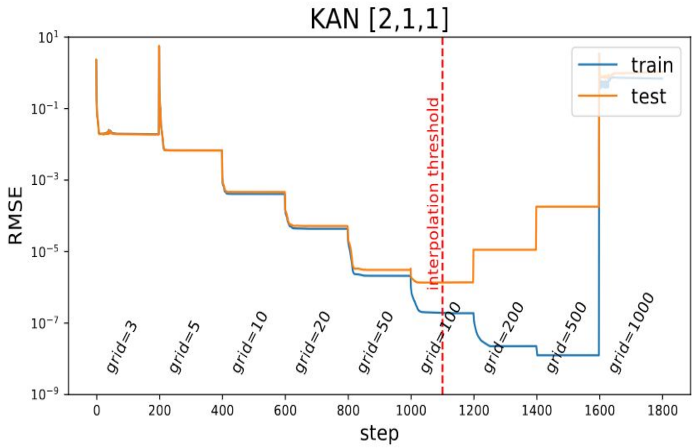
- A KAN with:
  - Depth L;
  - Constant width N; using
  - Splines of order k, defined on grids of size G
- Parameters: $O(N^2L(G+k)) \sim O(N^2LG)$
- (cf. MLPs: $O(N^2L)$)
- But KANs seem to use their parameters more efficiently than MLPs
  - Theoretical scaling exponents for test RMSE loss: $l \propto (\#\text{params})^{-\alpha}$
  - KANs are predicted (and empirically appear) to have $\alpha = k+1$ i.e. performance scales very well with parameter count.

Toy Problem: $f(x, y) = \exp(\sin(\pi x) + y^2)$



Toy Problem: $f(x,y) = \exp(\sin(\pi x) + y^2)$





Smaller KANs may generalize better.
How can we simplify large KANs semi- or fully-automatically?

Re: Smaller KANs may generalise better

- Essentially we want a notion of regularisation in KANs:

Re: Smaller KANs may generalise better

- Essentially we want a notion of regularisation in KANs:
    1. L1 norms (average activation over N_p inputs)

$$|\phi|_1 \equiv \frac{1}{N_p} \sum_{s=1}^{N_p} \left|\phi(x^{(s)})\right| \qquad\qquad |\mathbf{\Phi}|_1 \equiv \sum_{i=1}^{n_{\text{in}}} \sum_{j=1}^{n_{\text{out}}} |\phi_{i,j}|_1$$

Re: Smaller KANs may generalise better

- Essentially we want a notion of regularisation in KANs:
    1. L1 norms (average activation over N_p inputs) :

$$|\phi|_1 \equiv \frac{1}{N_p} \sum_{s=1}^{N_p} \left| \phi(x^{(s)}) \right| \qquad\qquad |\mathbf{\Phi}|_1 \equiv \sum_{i=1}^{n_{\mathrm{in}}} \sum_{j=1}^{n_{\mathrm{out}}} |\phi_{i,j}|_1$$

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

Re: Smaller KANs may generalise better

- Essentially we want a notion of regularisation in KANs:
  1. L1 norms (average activation over N_p inputs) :

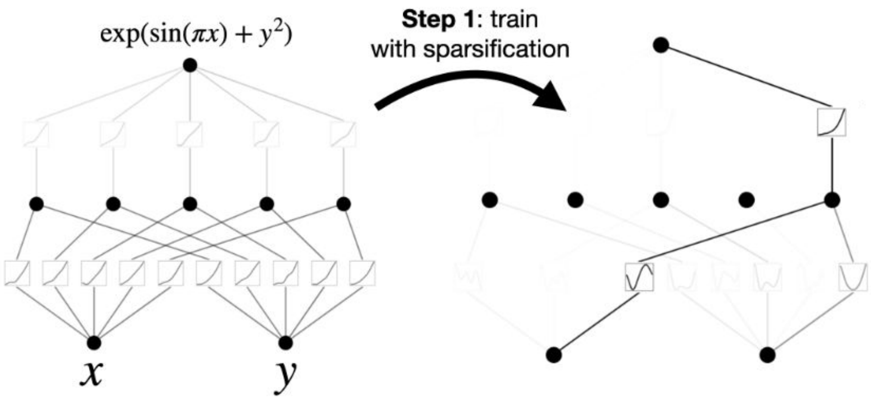
$$|\phi|_1 \equiv \frac{1}{N_p} \sum_{s=1}^{N_p} \left| \phi(x^{(s)}) \right| \qquad |\mathbf{\Phi}|_1 \equiv \sum_{i=1}^{n_{\rm in}} \sum_{j=1}^{n_{\rm out}} |\phi_{i,j}|_1$$

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

- Regularised KAN loss (usually $\mu_1 = \mu_2 = 1$)

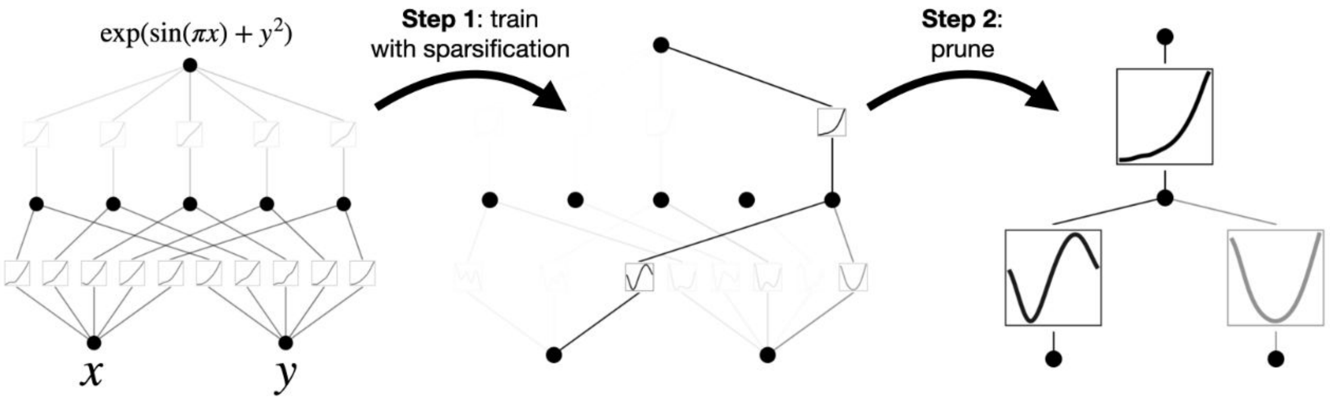
$$\ell_{\rm total} = \ell_{\rm pred} + \lambda \left(\mu_1 \sum_{l=0}^{L-1} |\mathbf{\Phi}_l|_1 + \mu_2 \sum_{l=0}^{L-1} S(\mathbf{\Phi}_l) \right)$$

Pruning KANs



Set transparency of each activation $\phi_{l,i,j}$ proportional to $\tanh(\beta A_{l,i,j})$ where $\beta = 3$

Pruning KANs

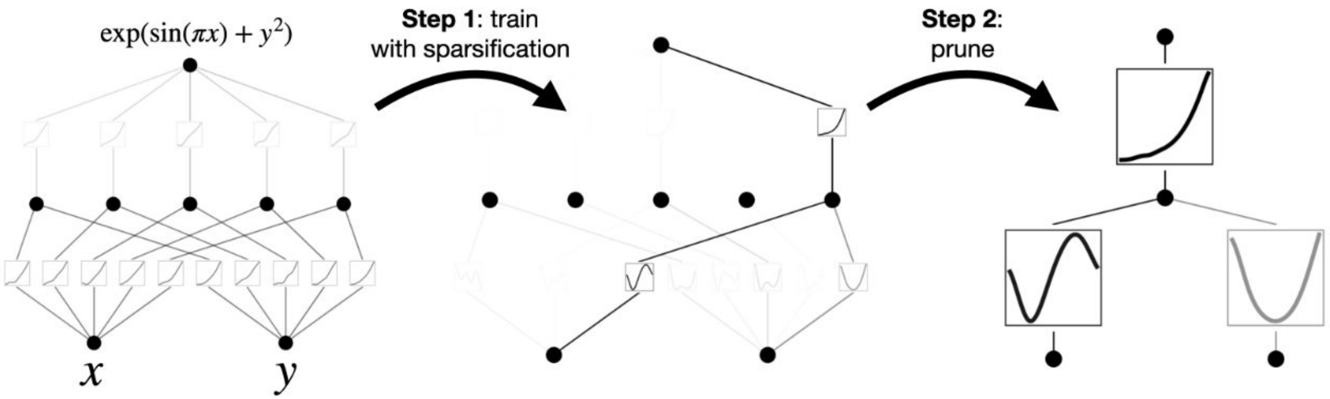


- Define incoming and outgoing score of neuron i in layer l:

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

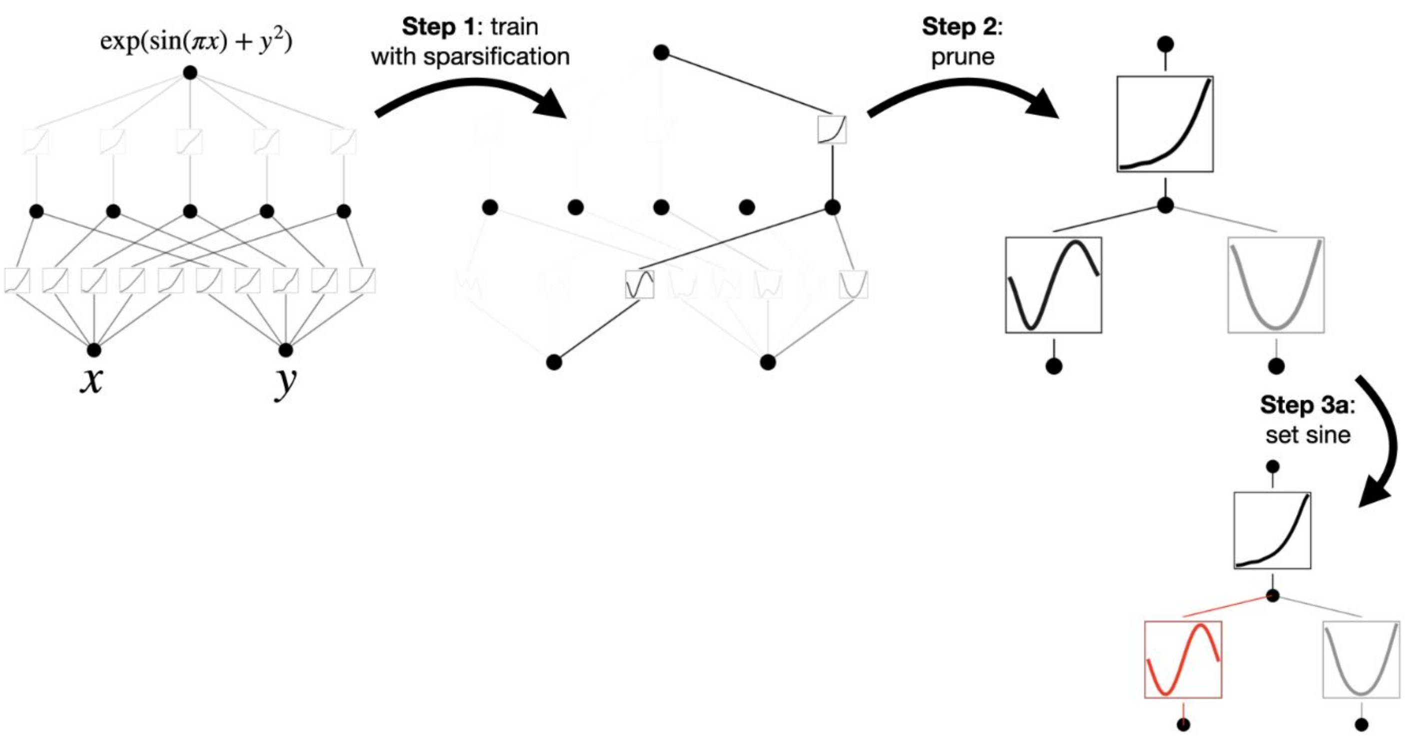
- Remove any neurons whose incoming or outgoing score is lower than some threshold, e.g. $\theta = 10^{-2}$

Symbolic Regression with KANs

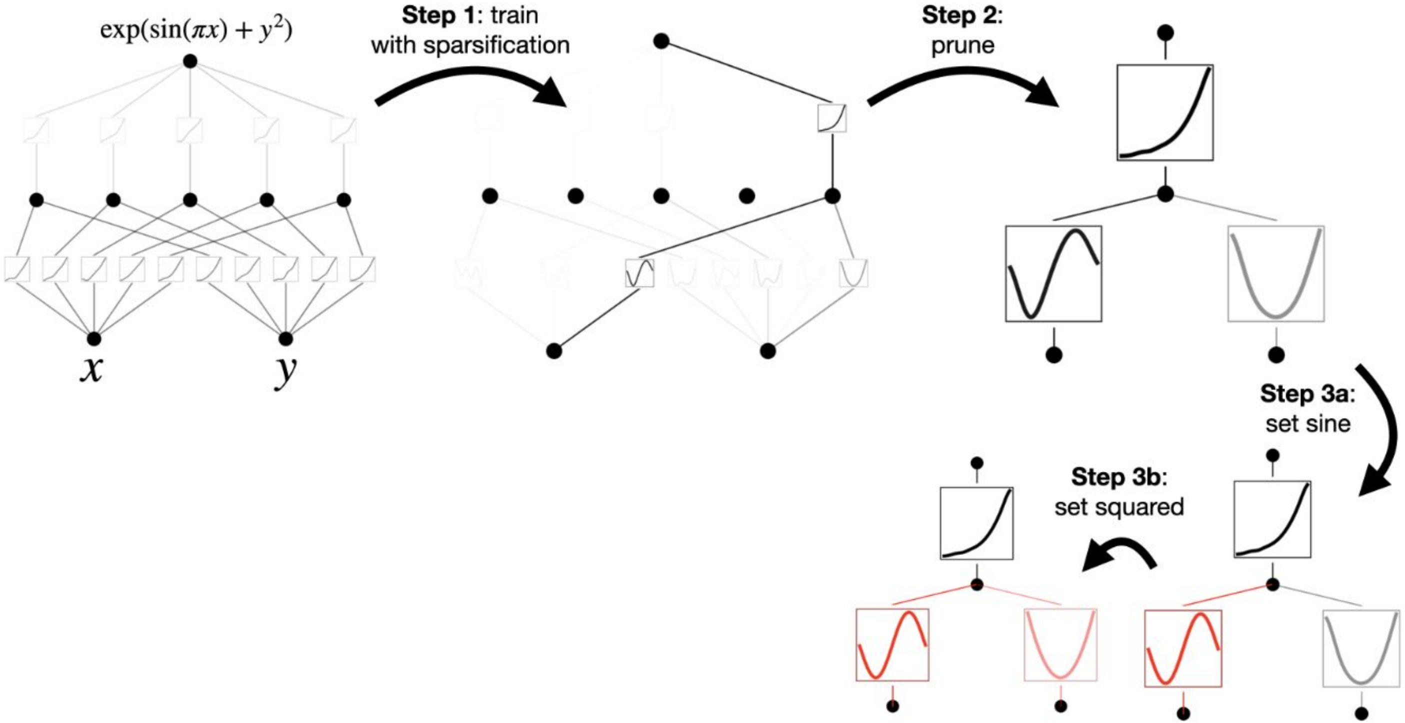


- If you recognise the shape of an activation function, set it explicitly (rather than just a spline approximation)
  - (Can be detected automatically…sometimes)

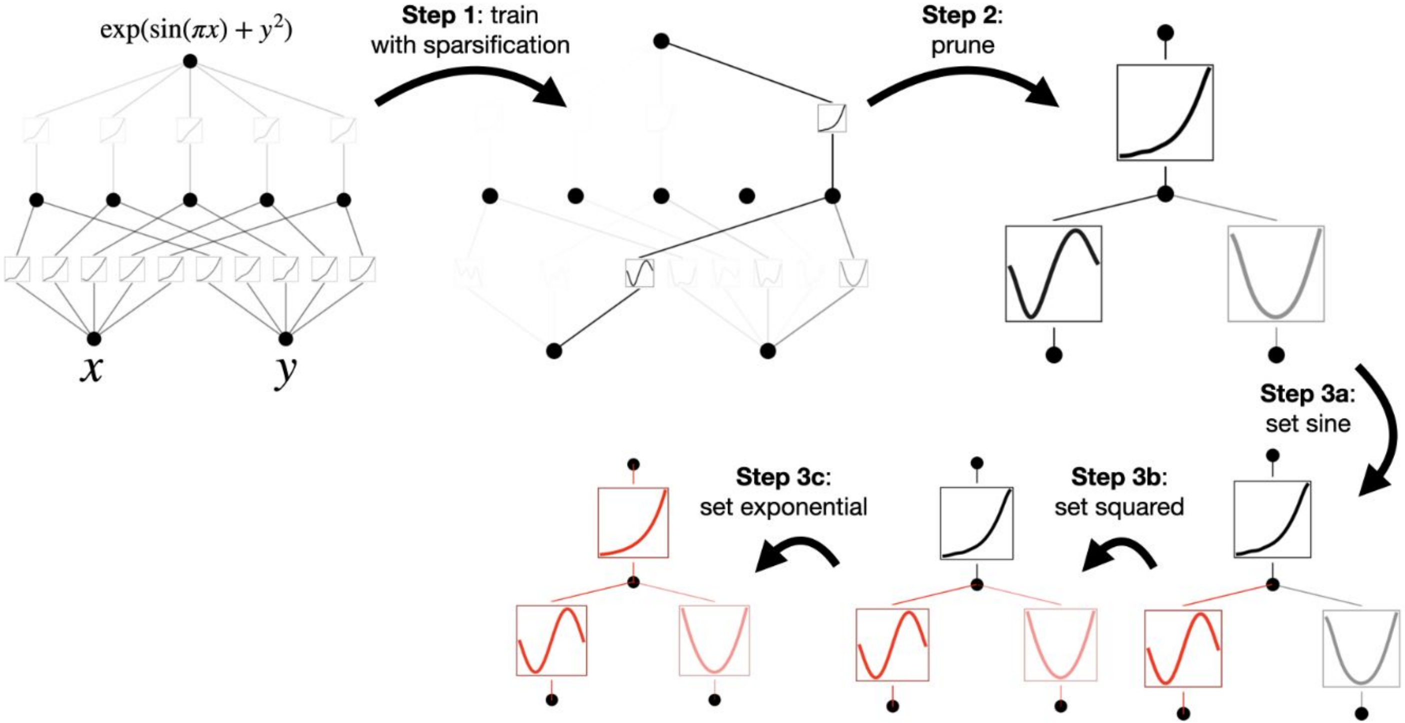
Symbolic Regression with KANs



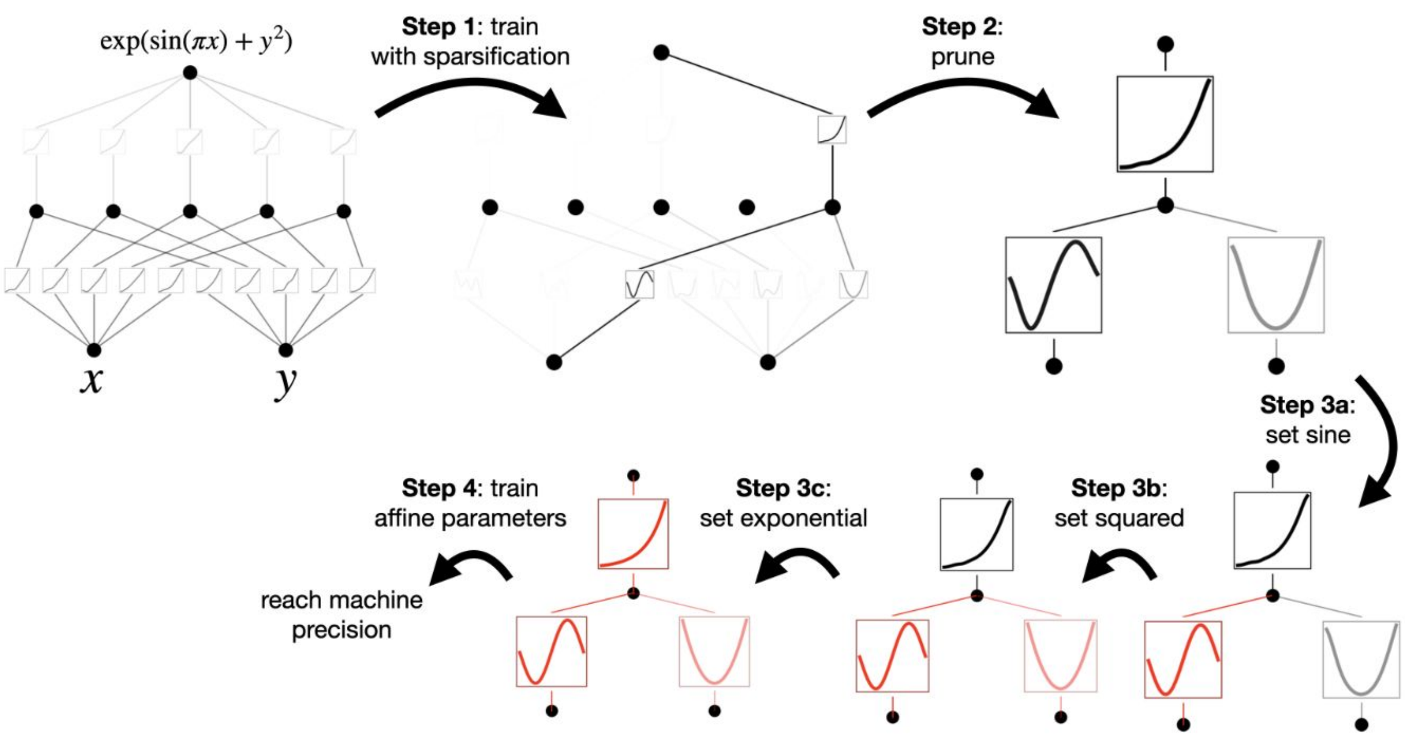
Symbolic Regression with KANs



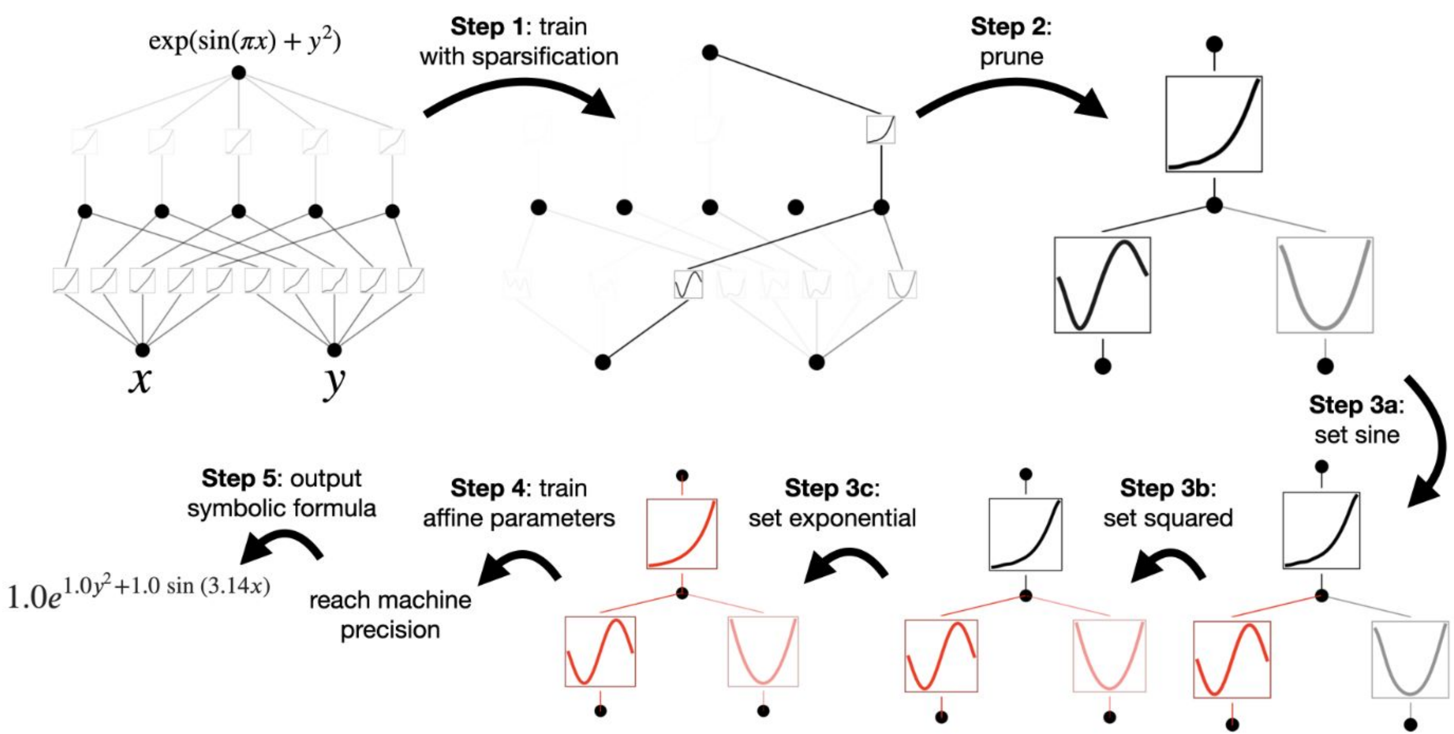
Symbolic Regression with KANs



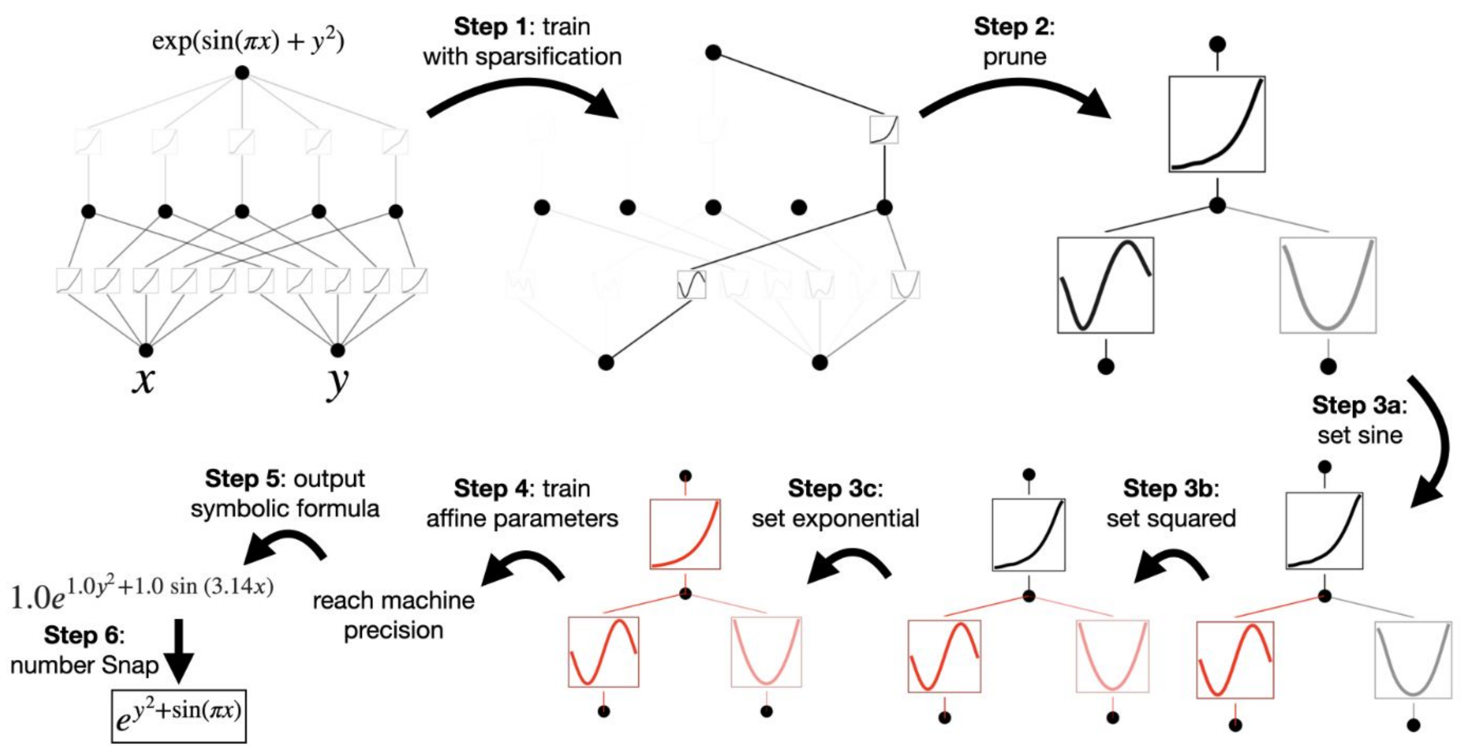
Symbolic Regression with KANs



Symbolic Regression with KANs



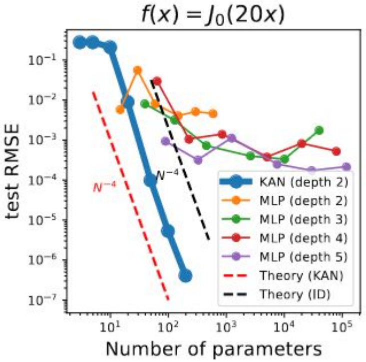
Symbolic Regression with KANs

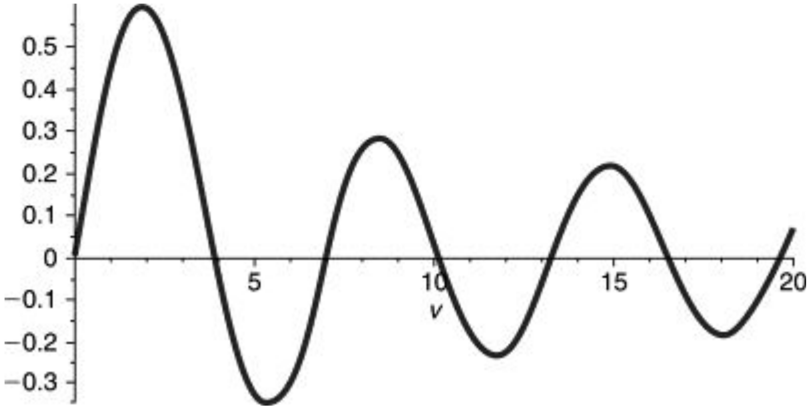


Symbolic regression (SR) without having to work directly in symbol-space.

Section 3: KANs are Accurate

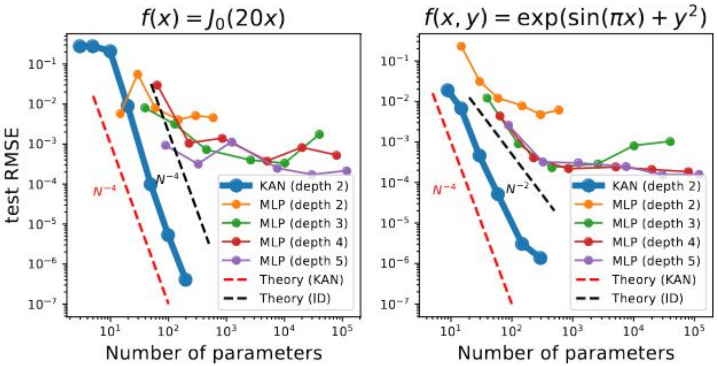
Using KANs with the expected ‘optimal’ shapes (knowing the function beforehand)





- 1D Bessel function
- [1,1] KAN (i.e. a spline)

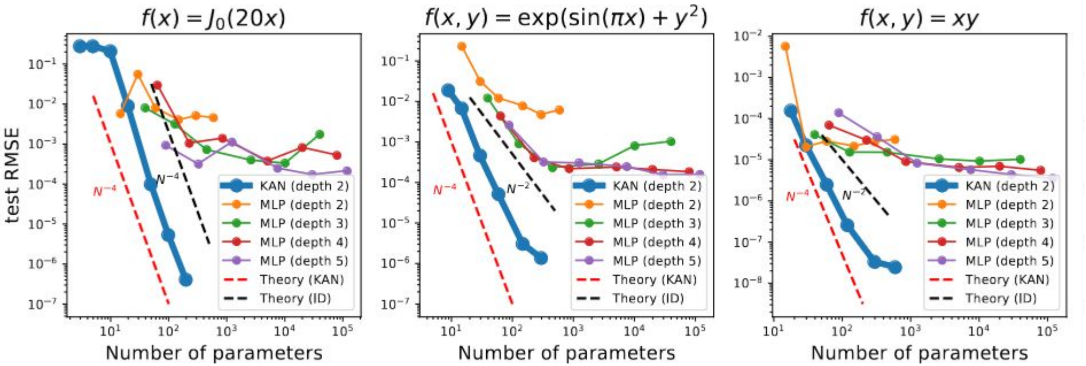
Using KANs with the expected 'optimal' shapes (knowing the function beforehand)



- 1D Bessel function
- [1,1] KAN (i.e. a spline)

- 2D function with two 'layers'
- [2,1,1] KAN

Using KANs with the expected ‘optimal’ shapes (knowing the function beforehand)

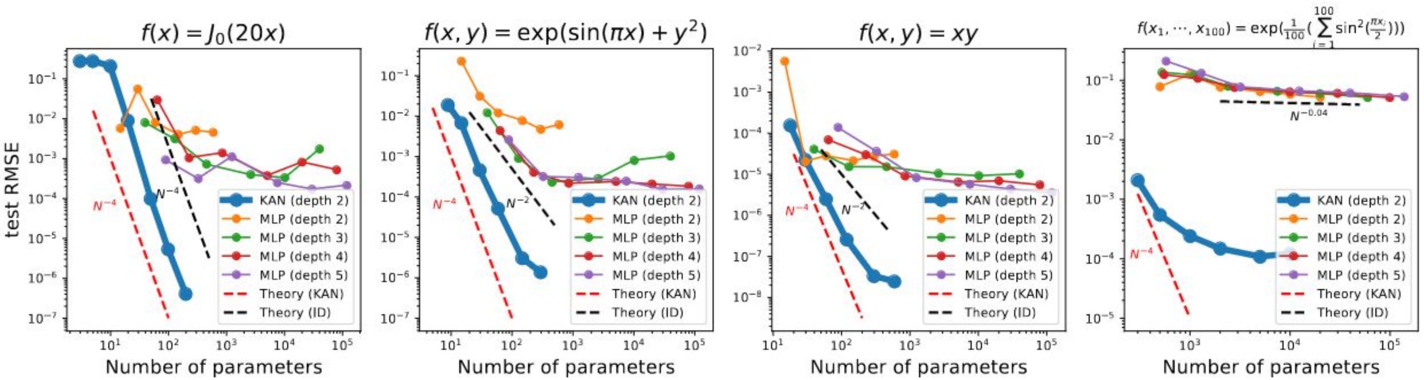


- 1D Bessel function
- [1,1] KAN (i.e. a spline)

- 2D function with two ‘layers’
- [2,1,1] KAN

- 2D function, can use
$$2xy = (x+y)^2 - (x^2 + y^2)$$
- [2,5,1] KAN

Using KANs with the expected ‘optimal’ shapes (knowing the function beforehand)



- 1D Bessel function
- [1,1] KAN (i.e. a spline)

- 2D function with two ‘layers’
- [2,1,1] KAN

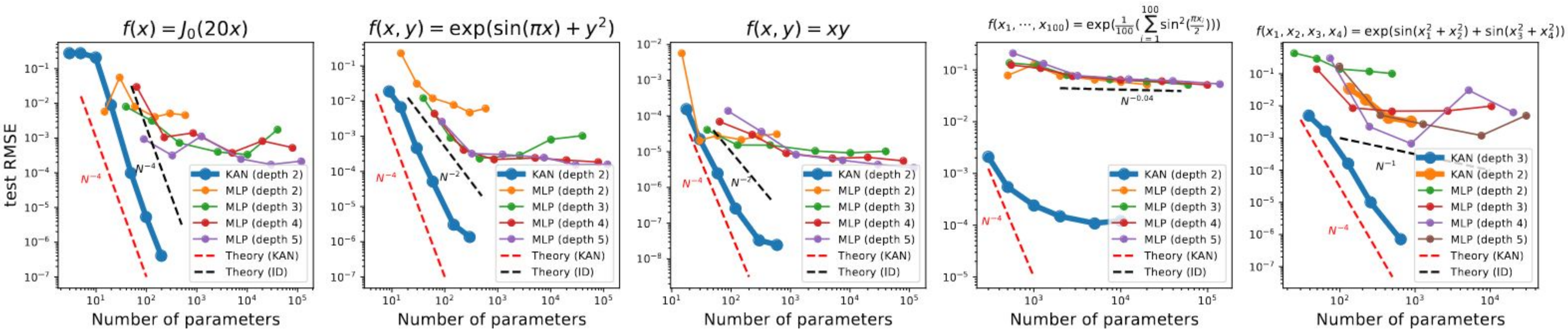
- 2D function, can use

$$2xy = (x+y)^2 - (x^2 + y^2)$$

- [2,5,1] KAN

- 100D function
- [100,1,1] KAN

Using KANs with the expected ‘optimal’ shapes (knowing the function beforehand)



- 1D Bessel function
- [1,1] KAN (i.e. a spline)

- 2D function with two ‘layers’
- [2,1,1] KAN

- 2D function, can use $2xy = (x+y)^2 - (x^2+y^2)$
- [2,5,1] KAN

- 100D function
- [100,1,1] KAN

- 4D function
- [4,4,2,1] KAN

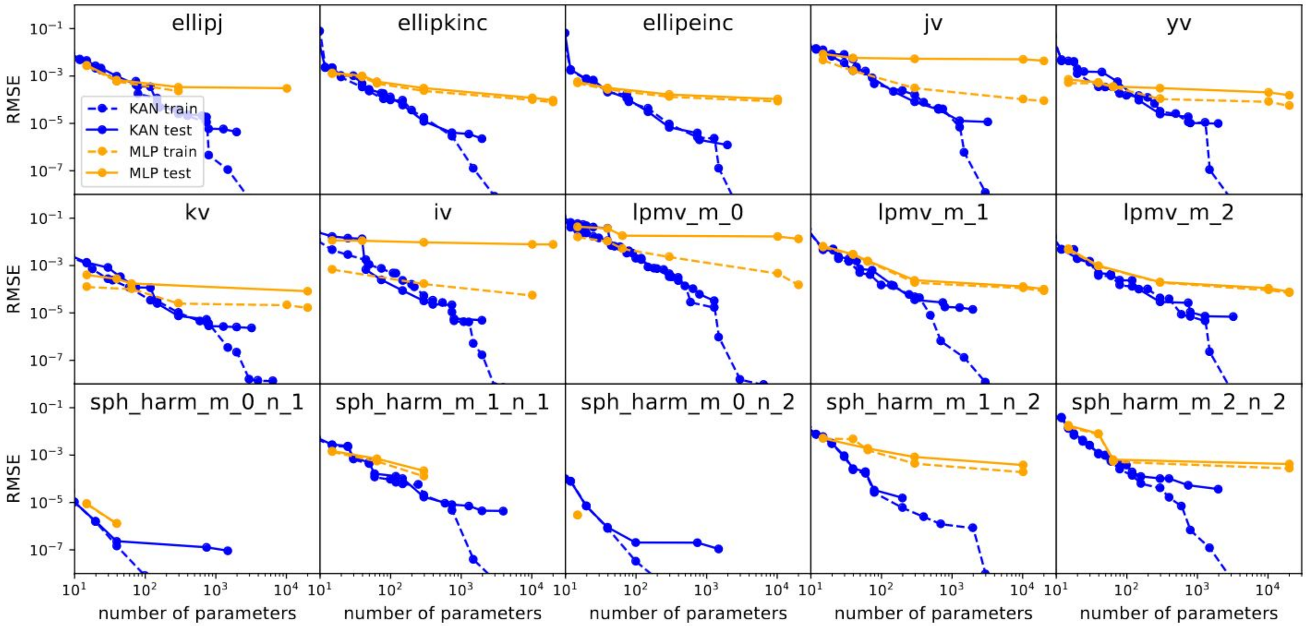
Using KANs where there are no easy expected 'optimal' shapes

- Sweep over a bunch of KAN shapes, with 0-6 middle layers of width 5, with and without pruning

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	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	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	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	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	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	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	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 $(m = 0)$	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 $(m = 1)$	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 $(m = 2)$	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 $(m = 0, n = 1)$	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 $(m = 1, n = 1)$	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 $(m = 0, n = 2)$	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 $(m = 1, n = 2)$	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 $(m = 2, n = 2)$	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}

Using KANs where there are no easy expected 'optimal' shapes

- Pareto frontiers: no other fit is both simpler and more accurate



Using KANs when we *think* we have an idea for the optimal KAN shape

- We learn that KANs can be more (parameter) efficient than we thought!

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}
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	$\frac{3}{4\pi\epsilon}\frac{p_dz}{r^5}\sqrt{x^2 + y^2}$	$\frac{1}{4\pi}c\sqrt{a^2 + b^2}$	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	$n_0(1 + \frac{p_dE_f\cos\theta}{k_bT})$	$n_0(1 + a\cos\theta)$	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	$\frac{n_0}{\exp(\frac{\mu_mB}{k_bT})+\exp(-\frac{\mu_mB}{k_bT})}$	$\frac{n_0}{\exp(a)+\exp(-a)}$	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	$\frac{\mu_mB}{k_bT} + \frac{\mu_m\alpha M}{\epsilon c^2k_bT}$	$a + \alpha b$	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	$\frac{p_dE_f}{h}\frac{\sin^2((\omega-\omega_0)t/2)}{((\omega-\omega_0)t/2)^2}$	$a\frac{\sin^2(\frac{b-c}{2})}{(\frac{b-c}{2})^2}$	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}

Using KANs when we *think* we have an idea for the optimal KAN shape

- Take the relativistic velocity addition formula

$$f(u, v) = (u + v)/(1 + uv)$$

Using KANs when we *think* we have an idea for the optimal KAN shape

- Take the relativistic velocity addition formula

$$f(u, v) = (u + v)/(1 + uv)$$

- Expected shape: [2,2,2,2,2,1]

Using KANs when we *think* we have an idea for the optimal KAN shape

- Take the relativistic velocity addition formula

$$f(u,v) = (u+v)/(1+uv)$$

- Expected shape: [2,2,2,2,2,1]
    - 2 layers for multiplication of uv (leveraging the identity $2xy = (x+y)^2 - (x^2+y^2)$)

Using KANs when we *think* we have an idea for the optimal KAN shape

- Take the relativistic velocity addition formula

$$f(u, v) = (u + v)/(1 + uv)$$

- Expected shape: [2,2,2,2,2,1]
    - 2 layers for multiplication of uv (leveraging the identity $2xy = (x+y)^2 - (x^2 + y^2)$)
    - 1 layer to invert 1+uv

Using KANs when we *think* we have an idea for the optimal KAN shape

- Take the relativistic velocity addition formula

$$f(u, v) = (u + v)/(1 + uv)$$

- Expected shape: [2,2,2,2,2,1]
  - 2 layers for multiplication of uv (leveraging the identity $2xy = (x + y)^2 - (x^2 + y^2)$)
  - 1 layer to invert 1+uv
  - 2 layers to multiply u+v and 1/(1+uv)

Using KANs when we *think* we have an idea for the optimal KAN shape

- Take the relativistic velocity addition formula

$$f(u,v) = (u+v)/(1+uv)$$

- Expected shape: [2,2,2,2,2,1]
  - 2 layers for multiplication of uv (leveraging the identity $2xy = (x+y)^2 - (x^2 + y^2)$)
  - 1 layer to invert 1+uv
  - 2 layers to multiply u+v and 1/(1+uv)

- Pruned shape: [2,2,1]
  - Used the ‘rapidity trick’: $\frac{u+v}{1+uv} = \tanh(\text{arctanh } u + \text{arctanh } v)$

Using KANs to solve PDEs

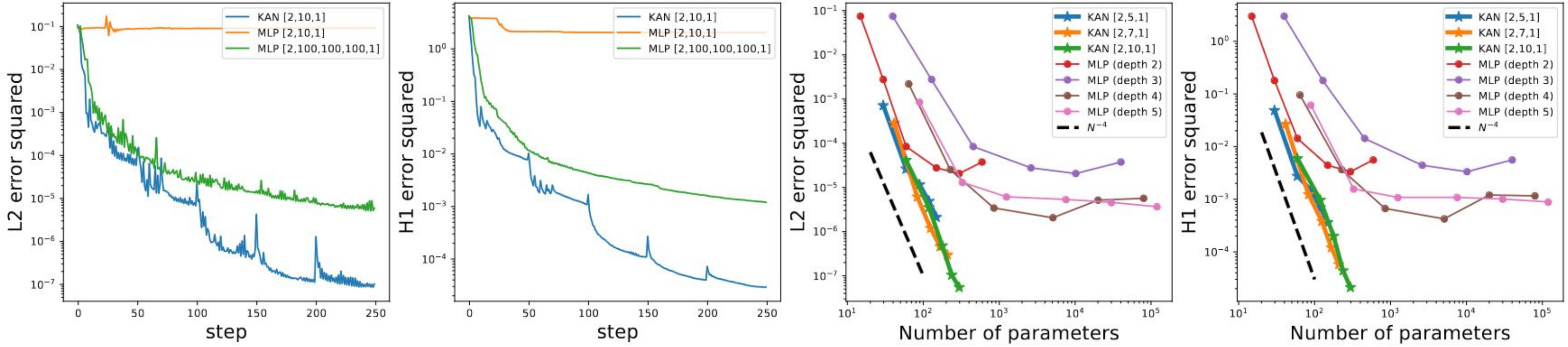


Figure 3.3: The PDE example. We plot L2 squared and H1 squared losses between the predicted solution and ground truth solution. First and second: training dynamics of losses. Third and fourth: scaling laws of losses against the number of parameters. KANs converge faster, achieve lower losses, and have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

Using KANs for Continual Learning

- Grids used in splines are naturally pretty robust against catastrophic forgetting

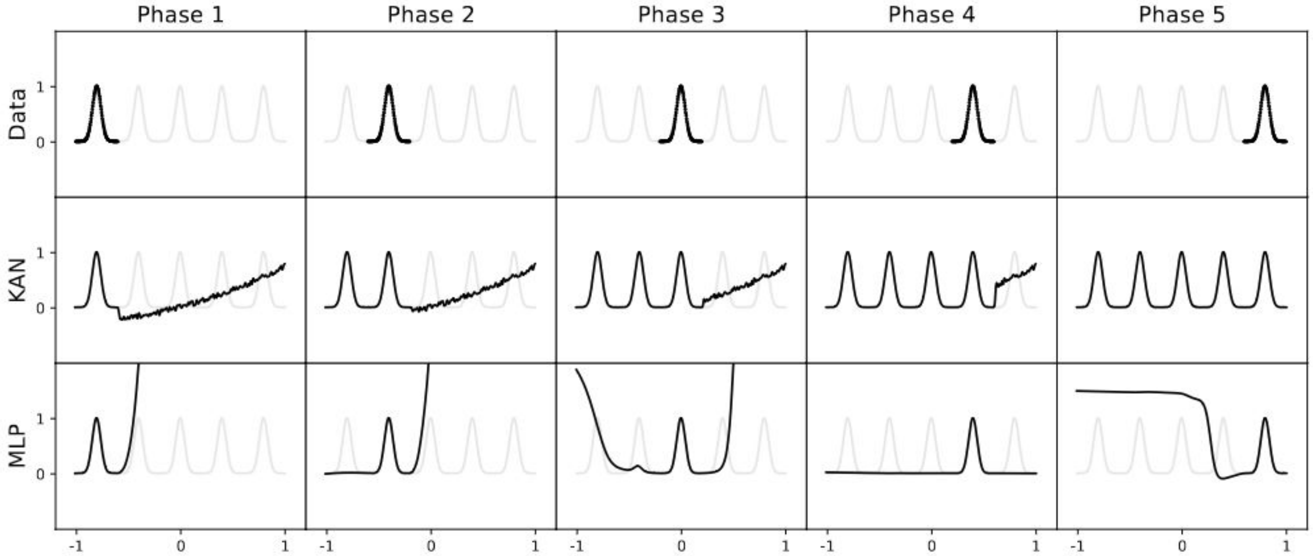
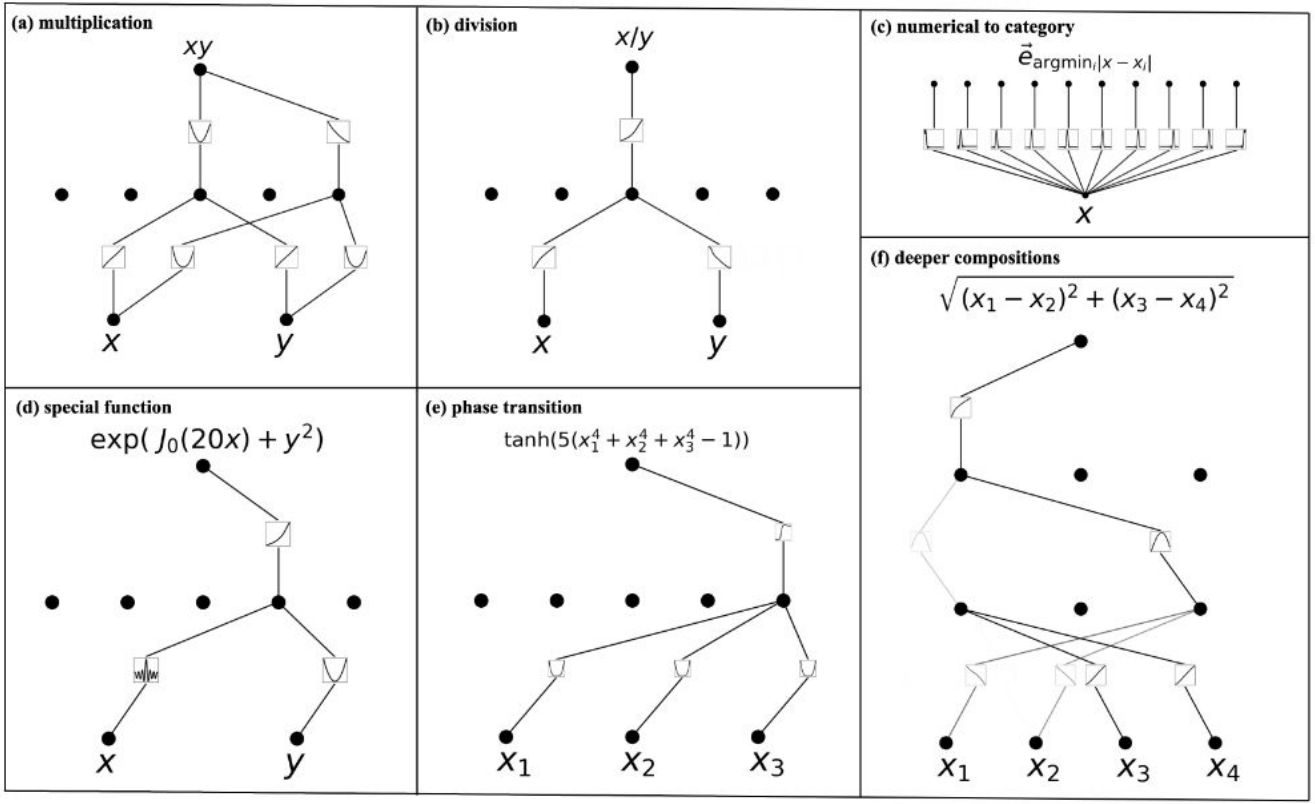


Figure 3.4: A toy continual learning problem. The dataset is a 1D regression task with 5 Gaussian peaks (top row). Data around each peak is presented sequentially (instead of all at once) to KANs and MLPs. KANs (middle row) can perfectly avoid catastrophic forgetting, while MLPs (bottom row) display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

Section 4: KANs are Interpretable

Interpreting Known Symbolic Functions



(a) $2xy = (x+y)^2 - (x^2+y^2)$

(b) $x/y = \exp(\log x - \log y)$

(c) learns approx. dirac deltas/spikes

(task in (c) was classifying from [0,1] by leading decimal digit.)

KANs for Unsupervised Learning

KANs for Unsupervised Learning

- Unsupervised learning: given dataset of features $\{(x_1^{(i)}, \ldots, x_d^{(i)})_i\}_{i=1}^n$ find a non-zero $f : \mathbb{R}^d \to \mathbb{R}$ such that

$$f(x_1, x_2, \cdots, x_d) \approx 0$$

KANs for Unsupervised Learning

- Unsupervised learning: given dataset of features $\{(x_1^{(i)}, \ldots, x_d^{(i)})_i\}_{i=1}^n$ find a non-zero $f : \mathbb{R}^d \to \mathbb{R}$ such that

$$f(x_1, x_2, \cdots, x_d) \approx 0$$

- Add some ‘negative’ samples to your dataset (e.g. by corrupting ‘positive’/true samples through shuffling/permutation/noise/etc.)

KANs for Unsupervised Learning

- Unsupervised learning: given dataset of features $\{(x_1^{(i)}, \ldots, x_d^{(i)})_i\}_{i=1}^n$ find a non-zero $f : \mathbb{R}^d \to \mathbb{R}$ such that

$$f(x_1, x_2, \cdots, x_d) \approx 0$$

- Add some 'negative' samples to your dataset (e.g. by corrupting 'positive'/true samples through shuffling/permutation/noise/etc.)

- Fix the final layer of your KAN to be Dirac-delta-like (Gaussian with small scale) and see what structures come out when you do pruning.

KANs for Unsupervised Learning

- Unsupervised learning: given dataset of features $\{(x_1^{(i)}, \ldots, x_d^{(i)})_i\}_{i=1}^n$ find a non-zero $f : \mathbb{R}^d \to \mathbb{R}$ such that

$$f(x_1, x_2, \cdots, x_d) \approx 0$$

- Add some 'negative' samples to your dataset (e.g. by corrupting 'positive'/true samples through shuffling/permutation/noise/etc.)
- Fix the final layer of your KAN to be Dirac-delta-like (Gaussian with small scale) and see what structures come out when you do pruning.
- Repeat KAN training with lots of different random seeds and each time you might get a different set of structures appearing

KANs for Unsupervised Learning

- Unsupervised learning: given dataset of features $\{(x_1^{(i)}, \ldots, x_d^{(i)})_i\}_{i=1}^n$ find a non-zero $f : \mathbb{R}^d \to \mathbb{R}$ such that

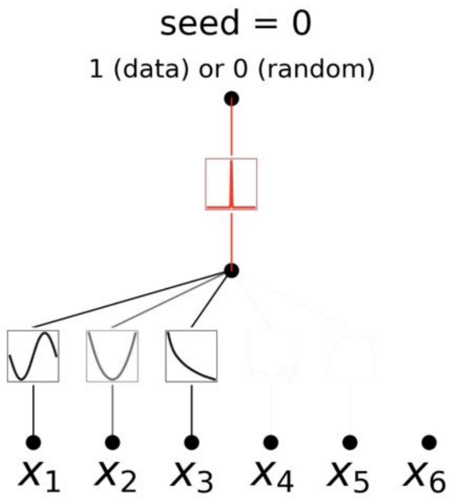
$$f(x_1, x_2, \cdots, x_d) \approx 0$$

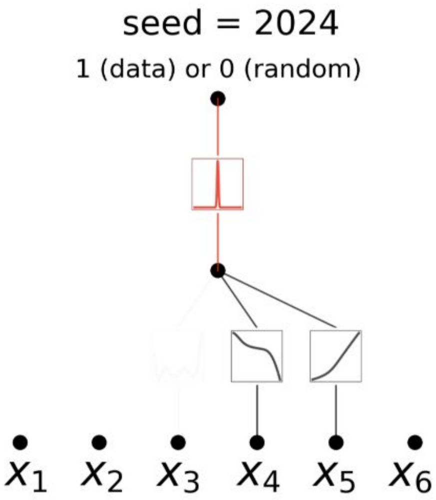
- Add some ‘negative’ samples to your dataset (e.g. by corrupting ‘positive’/true samples through shuffling/permutation/noise/etc.)
- Fix the final layer of your KAN to be Dirac-delta-like (Gaussian with small scale) and see what structures come out when you do pruning.
- Repeat KAN training with lots of different random seeds and each time you might get a different set of structures appearing
    - "in the future we would like to investigate a more systematic and more controlled way to discover a complete set of relations"

KANs for Unsupervised Learning: Toy example

6D dataset, with dependencies:

$x_3 = \exp(\sin(x_1) + x_2^2)$ $\quad$ $x_5 = x_4^3$ $\quad$ x_6 independent of other variables





Application to Knot Theory

Article

Advancing mathematics by guiding human intuition with AI

https://doi.org/10.1038/s41586-021-04086-x

Received: 10 July 2021

Alex Davies[1] ✉, Petar Veličković[1], Lars Buesing[1], Sam Blackwell[1], Daniel Zheng[1], Nenad Tomašev[1], Richard Tanburn[1], Peter Battaglia[1], Charles Blundell[1], András Juhász[2], Marc Lackenby[2], Geordie Williamson[3], Demis Hassabis[1] & Pushmeet Kohli[1] ✉

Application to Knot Theory

Article

Advancing mathematics by guiding human intuition with AI

https://doi.org/10.1038/s41586-021-04086-x

Received: 10 July 2021

Alex Davies[1]✉, Petar Veličković[1], Lars Buesing[1], Sam Blackwell[1], Daniel Zheng[1], Nenad Tomašev[1], Richard Tanburn[1], Peter Battaglia[1], Charles Blundell[1], András Juhász[2], Marc Lackenby[2], Geordie Williamson[3], Demis Hassabis[1] & Pushmeet Kohli[1]✉

- DeepMind paper: found link between certain variables about knots, which mathematicians refined to prove a new theorem.

Application to Knot Theory

Article

Advancing mathematics by guiding human intuition with AI

https://doi.org/10.1038/s41586-021-04086-x

Received: 10 July 2021

Alex Davies[1✉], Petar Veličković[1], Lars Buesing[1], Sam Blackwell[1], Daniel Zheng[1], Nenad Tomašev[1], Richard Tanburn[1], Peter Battaglia[1], Charles Blundell[1], András Juhász[2], Marc Lackenby[2], Geordie Williamson[3], Demis Hassabis[1] & Pushmeet Kohli[1✉]

- DeepMind paper: found link between certain variables about knots, which mathematicians refined to prove a new theorem.
  1. Signature σ is mostly dependent on meridinal distance μ (real μ_r, imag μ_i) and longitudinal distance λ

Application to Knot Theory

Article

Advancing mathematics by guiding human intuition with AI

https://doi.org/10.1038/s41586-021-04086-x

Received: 10 July 2021

Alex Davies[1✉], Petar Veličković[1], Lars Buesing[1], Sam Blackwell[1], Daniel Zheng[1], Nenad Tomašev[1], Richard Tanburn[1], Peter Battaglia[1], Charles Blundell[1], András Juhász[2], Marc Lackenby[2], Geordie Williamson[3], Demis Hassabis[1] & Pushmeet Kohli[1✉]

- DeepMind paper: found link between certain variables about knots, which mathematicians refined to prove a new theorem.

1. Signature σ is mostly dependent on meridinal distance μ (real μ_r, imag μ_i) and longitudinal distance λ
2. Found a bound for $|2\sigma - \text{slope}|$ where $\text{slope} \equiv \text{Re}(\frac{\lambda}{\mu}) = \frac{\lambda\mu_r}{\mu_r^2+\mu_i^2}$

Application to Knot Theory: Supervised Learning

- Given 17 input features and use knot signatures (even numbers) as output/targets for a [17,1,14] KAN

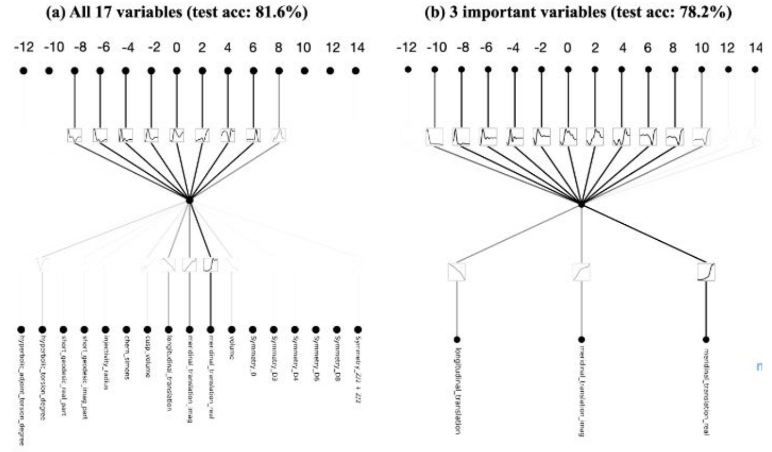


Figure 4.3: Knot dataset, supervised mode. With KANs, we rediscover Deepmind's results that signature is mainly dependent on meridinal translation (real and imaginary parts).

Application to Knot Theory: Supervised Learning

- Given 17 input features and use knot signatures (even numbers) as output/targets for a [17,1,14] KAN

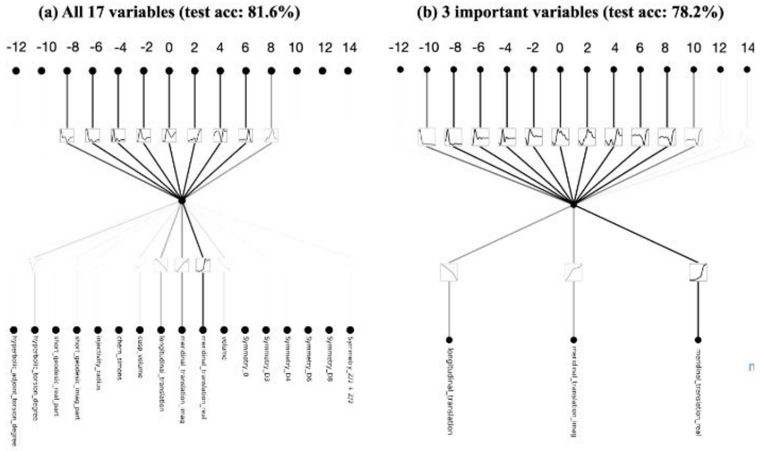


Figure 4.3: Knot dataset, supervised mode. With KANs, we rediscover Deepmind's results that signature is mainly dependent on meridinal translation (real and imaginary parts).

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 4: KANs can achieve better accuracy than MLPs with much fewer parameters in the signature classification problem.

Application to Knot Theory: Supervised Learning

- Given 17 input features and use knot signatures (even numbers) as output/targets for a [17,1,14] KAN

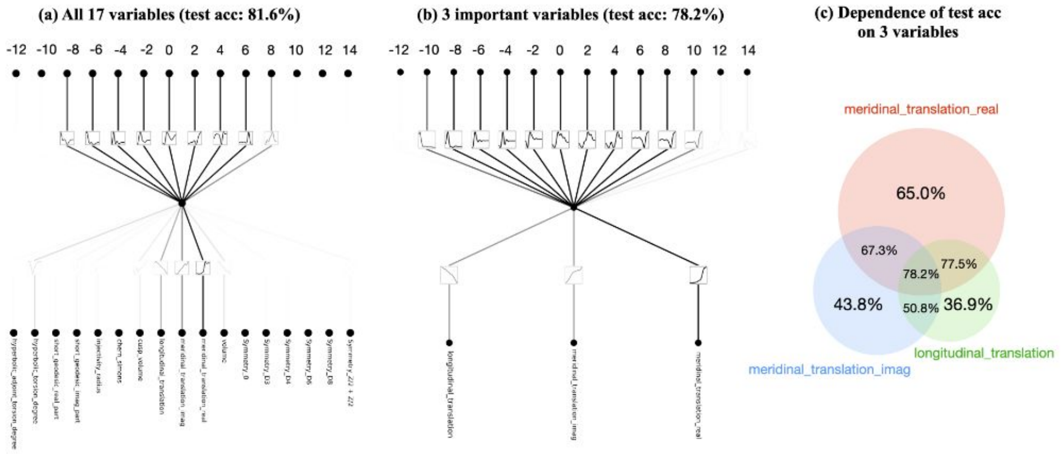


Figure 4.3: Knot dataset, supervised mode. With KANs, we rediscover Deepmind's results that signature is mainly dependent on meridinal translation (real and imaginary parts).

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 4: KANs can achieve better accuracy than MLPs with much fewer parameters in the signature classification problem.

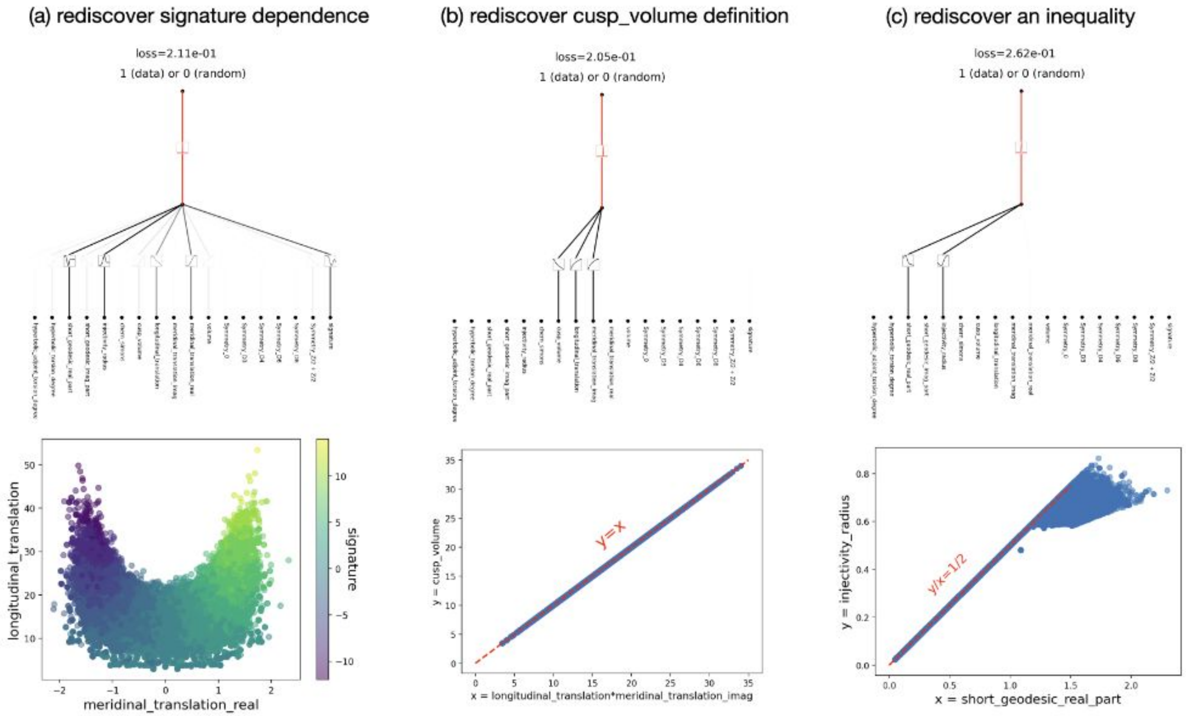
Application to Knot Theory: Supervised Learning

- They even found a (potentially) interesting relationship: you can still get 77.8% test accuracy with only 2 of the variables DeepMind used

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	$-0.09+1.04\exp(-9.59(-0.62\sin(0.61\mu_r+7.26))-0.32\tan(0.03\lambda-6.59)+1-0.11e^{-1.77(0.31-\mu_i)^2)^2}-1.09e^{-7.6(0.65(1-0.01\lambda)^3}+0.27\mathrm{atan}(0.53\mu_i-0.6)+0.09+\exp(-2.58(1-0.36\mu_r)^2))$	$[3,2,1]$ 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

Application to Knot Theory: Unsupervised Learning

- Take 17 input features and knot signatures together
- Create negative samples by shuffling features in dataset



Application to Physics (that I don’t really understand)

- But the KANs do well and they walk through how a researcher might use a KAN to discover a model
  - (the latter which might be worth walking through in this talk...)

Drawbacks

- “KANs are usually 10x slower than MLPs, given the same number of parameters”
  - Batching is more complicated because of variety of activation functions
    - (Though you can restrict some neurons to use the same activation)
  - B-spline computation isn’t incredibly fast

- Very small scale architectures (for now)

Conclusion

- Accurate
- Good scaling with #params (so far)
- Very interpretable
  - Useful in other scientific fields, but not necessarily shown to be good for deep learning
- Don't have to use splines:
  - Radial basis functions or other local kernels supposedly work well (according to a guy on reddit)
- "Kansformers" - replace MLP in transformer by KANs
  - Working "okay" (or some other non-committal word) according to the lead author on twitter

Appendix

Parameter efficiency: External vs Internal Degrees of Freedom

- External dofs (large-scale structure):
    - MLPs: connections between nodes
    - KANs: connections between nodes

- Internal dofs (small-scale structure):
    - KANs: splines
    - MLPs: linear transformations (and ‘global’ activation functions)

KAN approximation theory

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*

$$\|f - (\mathbf{\Phi}^G_{L-1} \circ \mathbf{\Phi}^G_{L-2} \circ \cdots \circ \mathbf{\Phi}^G_1 \circ \mathbf{\Phi}^G_0)\mathbf{x}\|_{C^m} \le CG^{-k-1+m}\,. \tag{2.15}$$

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

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

- "beats the curse of dimensionality!"
- This error bound doesn't depend on input dimension, (...except for the multiplicative constant)

PDE Problem

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

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

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.