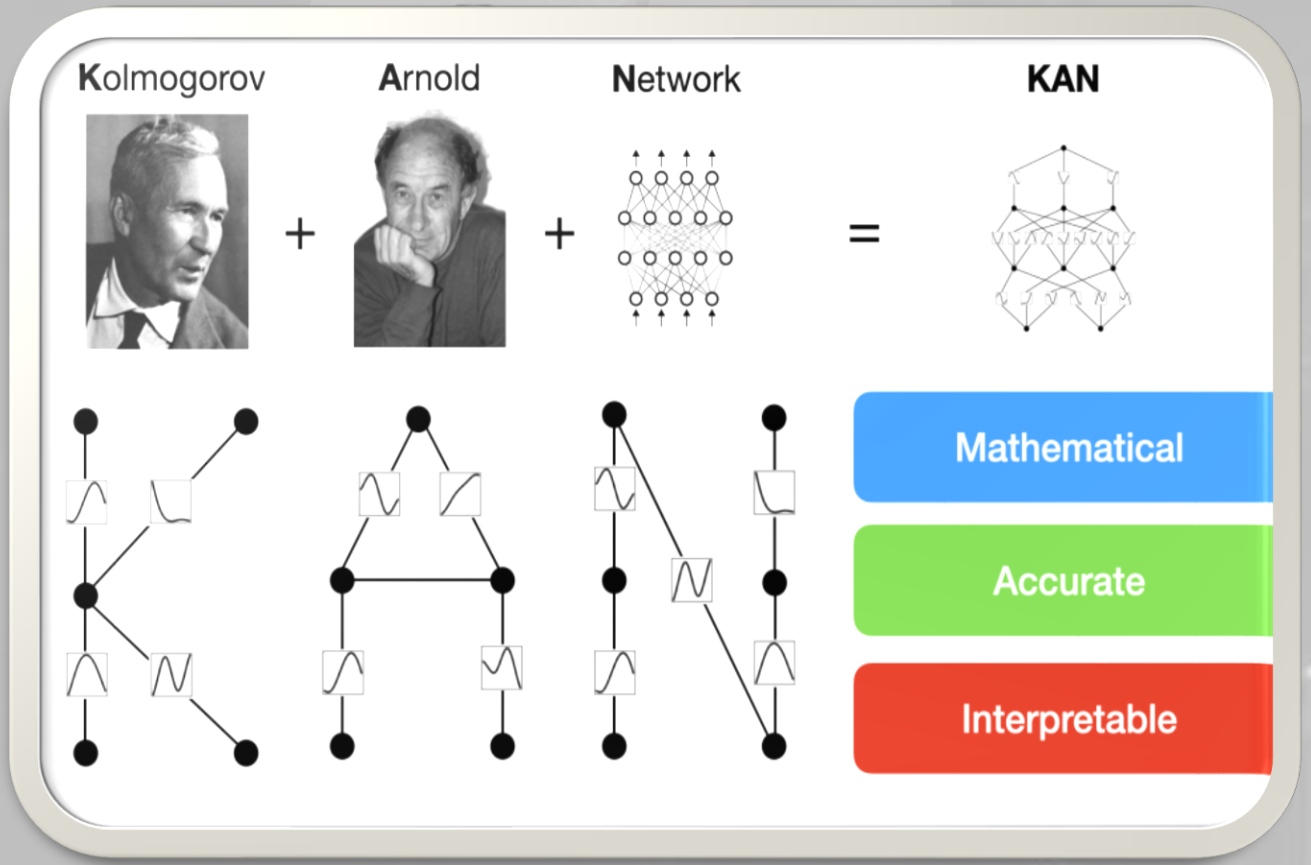


Kolmogorov–Arnold Networks

Author (s): Ziming Liu et al.

- Massachusetts Institute of Technology
- California Institute of Technology
- Northeastern University
- The NSF Institute for Artificial Intelligence and Fundamental Interactions

Computer Science > Machine Learning

[Submitted on 30 Apr 2024 (v1), last revised 16 Jun 2024 (this version, v4)]

KAN: Kolmogorov-Arnold Networks

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, Max Tegmark

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.

Comments: 48 pages, 20 figures. Codes are available at this https URL

Subjects: **Machine Learning (cs.LG)**; Disordered Systems and Neural Networks (cond-mat.dis-nn); Artificial Intelligence (cs.AI); Machine Learning (stat.ML)

Cite as: arXiv:2404.19756 **[cs.LG]**

(or arXiv:2404.19756v4 **[cs.LG]** for this version)

https://doi.org/10.48550/arXiv.2404.19756

Submission history

From: Ziming Liu [view email]

[v1] Tue, 30 Apr 2024 17:58:29 UTC (15,986 KB)

[v2] Thu, 2 May 2024 16:18:21 UTC (15,986 KB)

[v3] Fri, 24 May 2024 22:30:07 UTC (15,991 KB)

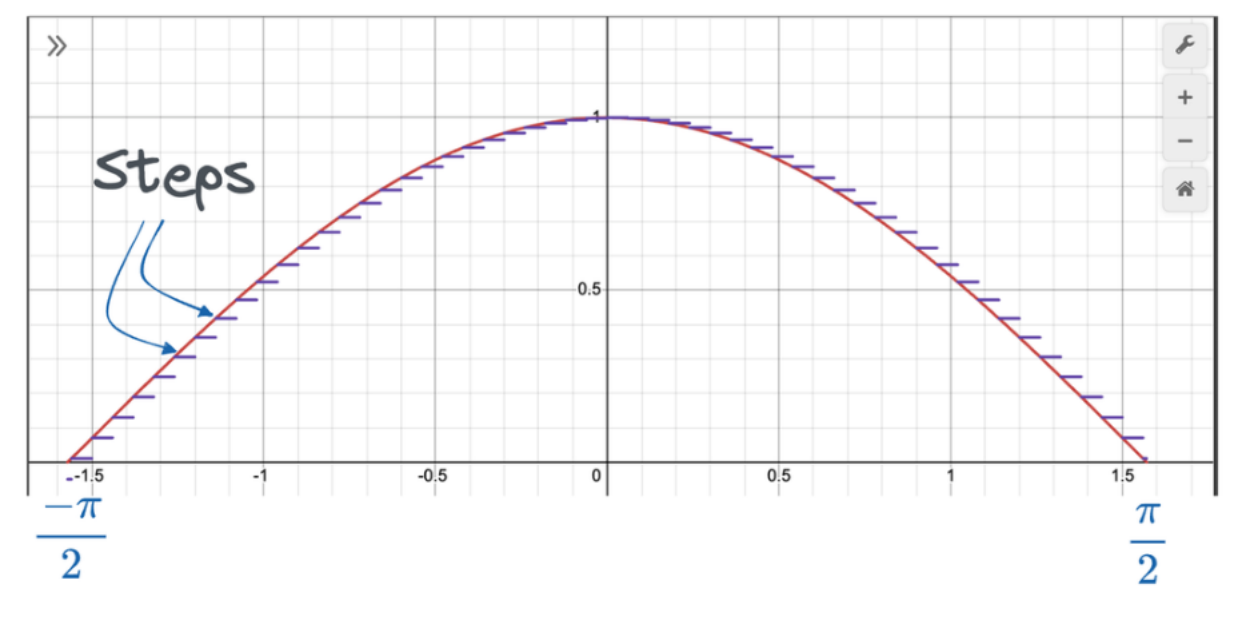
[v4] Sun, 16 Jun 2024 13:34:56 UTC (17,266 KB)

Universal Approximation Theorem

Under certain conditions, a specific type of neural network architecture called MLP can approximate any continuous function to an arbitrary degree of accuracy.

Mathematically speaking, for any continuous function f and $\epsilon > 0$, there always exists a neural network $\hat{f}$ such that:

$$|f(x) - \hat{f}(x)| < \epsilon$$



Kolmogorov-Arnold 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 single variable and the binary operation of addition.

- these 1D functions can be non-smooth and even fractal
- sentenced to death in machine learning, regarded as theoretically sound but practically useless !!!

for a smooth $f : [0,1]^n \to \mathbb{R}$,

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

where $\phi_{q,p} : [0,1] \to \mathbb{R}$ and $\Phi_q : \mathbb{R} \to \mathbb{R}$.

Multivariate continuous function

Composition of univariate functions

$$\begin{aligned} F(x_1, x_2, x_3, \cdots, x_n) = {} & \psi_1(\phi_{11}(x_1) + \phi_{21}(x_2) + \cdots + \phi_{n1}(x_n)) \\ & + \psi_2(\phi_{12}(x_1) + \phi_{22}(x_2) + \cdots + \phi_{n2}(x_n)) \\ & \vdots \\ & + \psi_m(\phi_{1m}(x_1) + \phi_{2m}(x_2) + \cdots + \phi_{nm}(x_n)) \end{aligned}$$

Multivariate continuous function

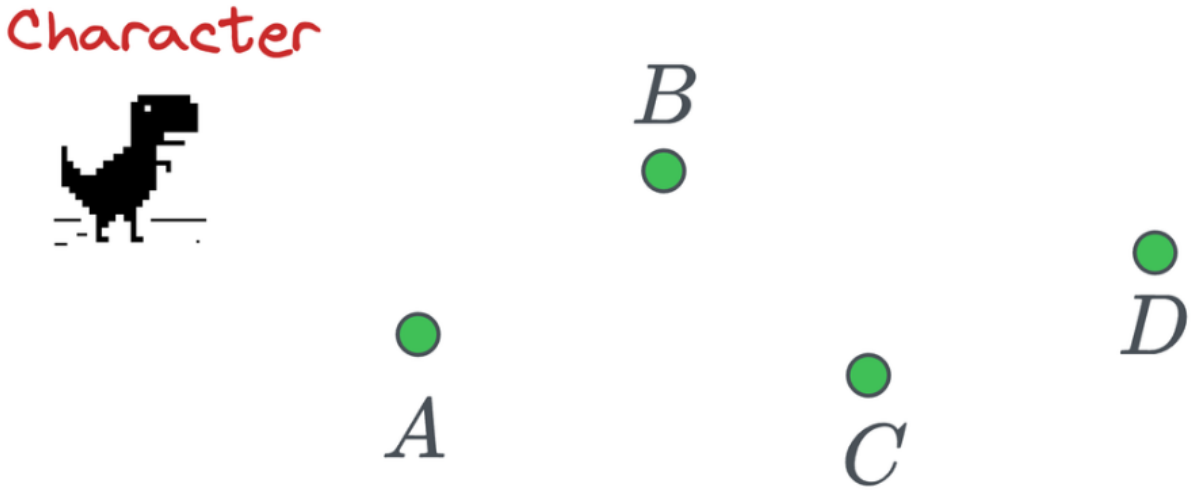
$$F(x, y) = xy$$

Composition of univariate functions

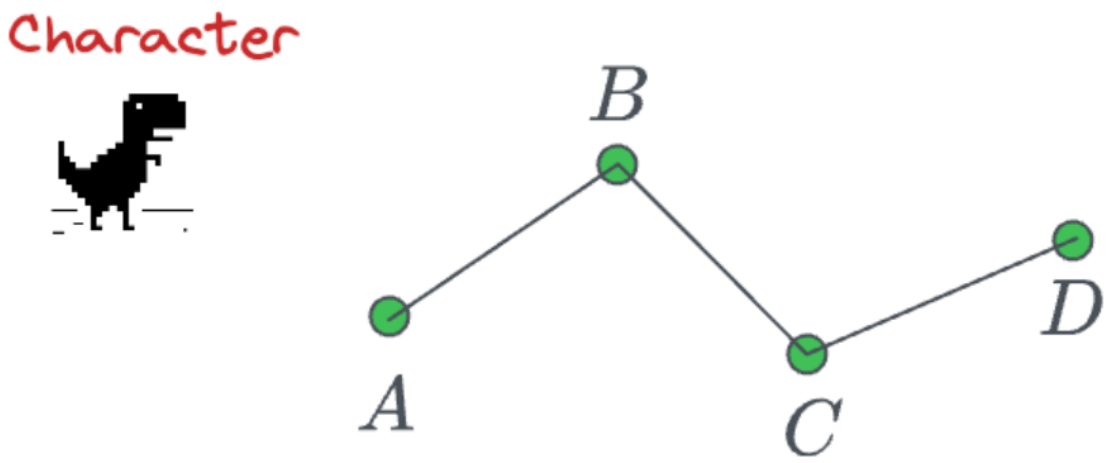
$$F(x, y) = \underset{\psi}{\boxed{exp}}\left(\underset{\phi_x}{\boxed{log(x)}} + \underset{\phi_y}{\boxed{log(y)}}\right)$$

Bezier curves

Imagine a gaming character that must pass through the following four points:



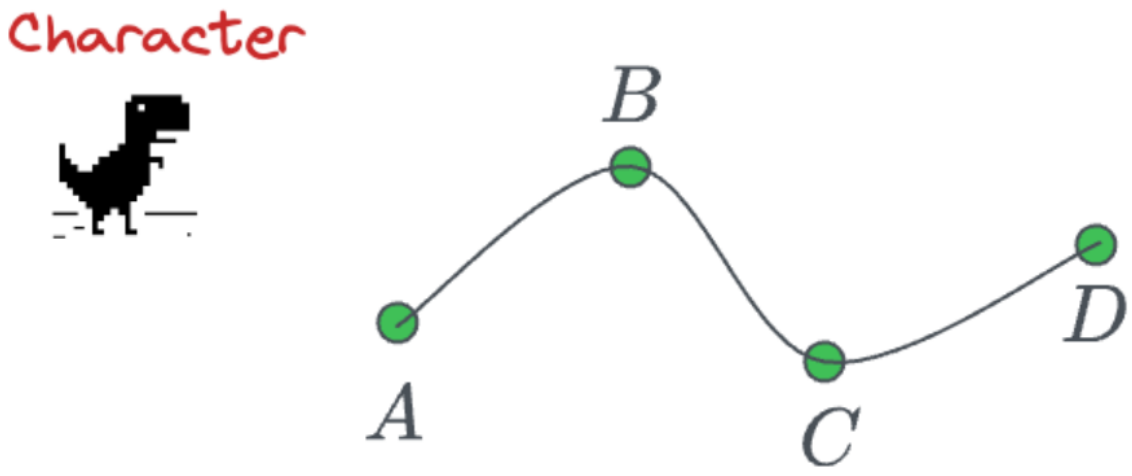
The most obvious way is to traverse them as follows:



But this movement does not appear natural, does it?

Bezier curves

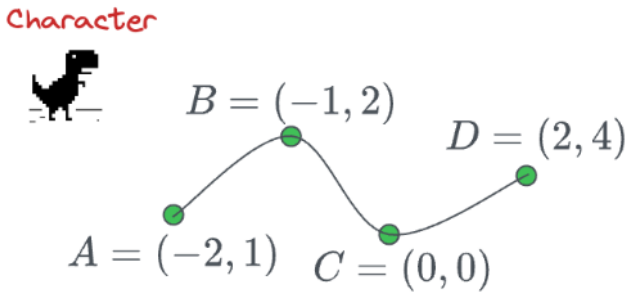
In computer graphics, we would desire a smooth traversal that looks something like this:



One way to do this is by estimating the coefficients of a higher-degree polynomial by solving a system of linear equations:

$$f(x) = ax^3 + bx^2 + cx + d$$

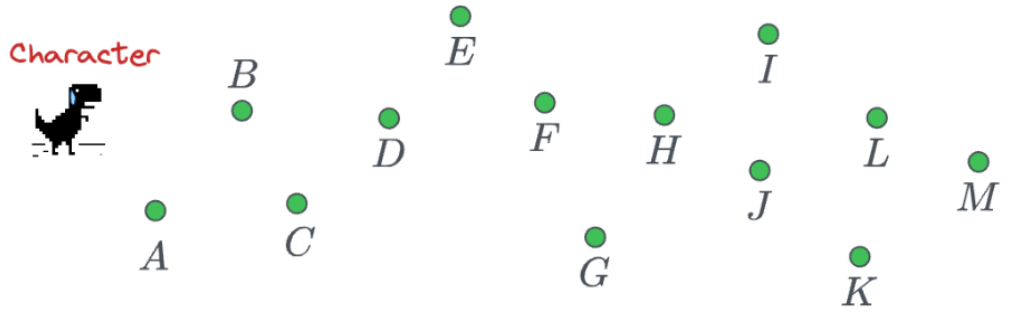
More specifically, we know the location of points (A, B, C, D), so we can substitute them into the above function and determine the values of the coefficients (a, b, c, d).



Solve these equations

$$A \to 1 = a(-2)^3 + b(-2)^2 + c(-2) + d$$
$$B \to 2 = a(-1)^3 + b(-1)^2 + c(-1) + d$$
$$C \to 0 = a(0)^3 + b(0)^2 + c(0) + d$$
$$D \to 4 = a(2)^3 + b(2)^2 + c(2) + d$$

However, what if we have hundreds of data points?



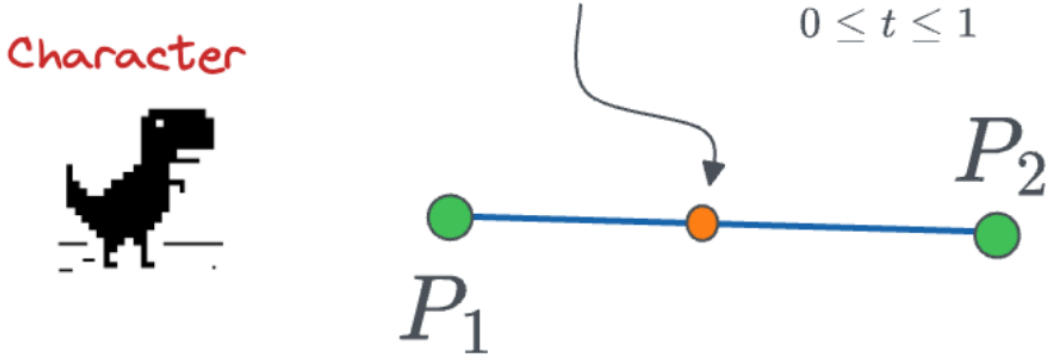
Bezier curves

👉 Points P_1 and P_2 are called control points.

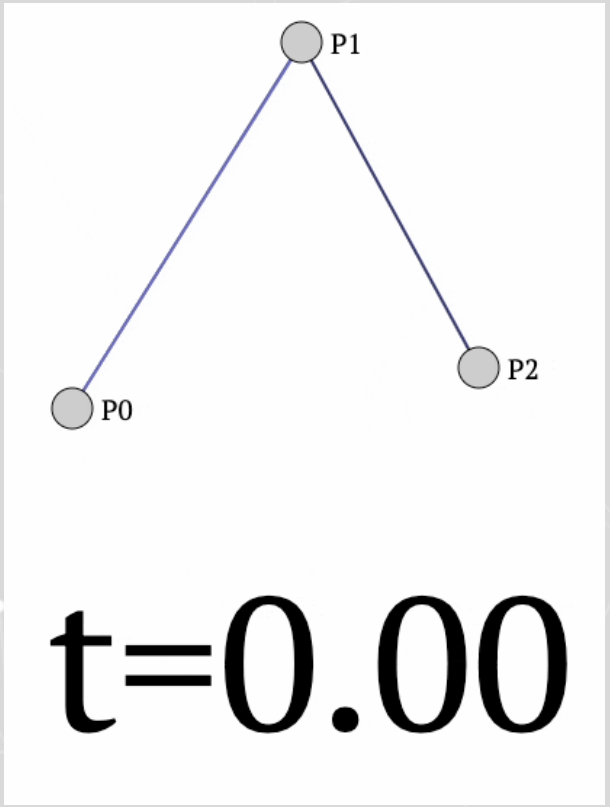
We can determine the curve as follows:

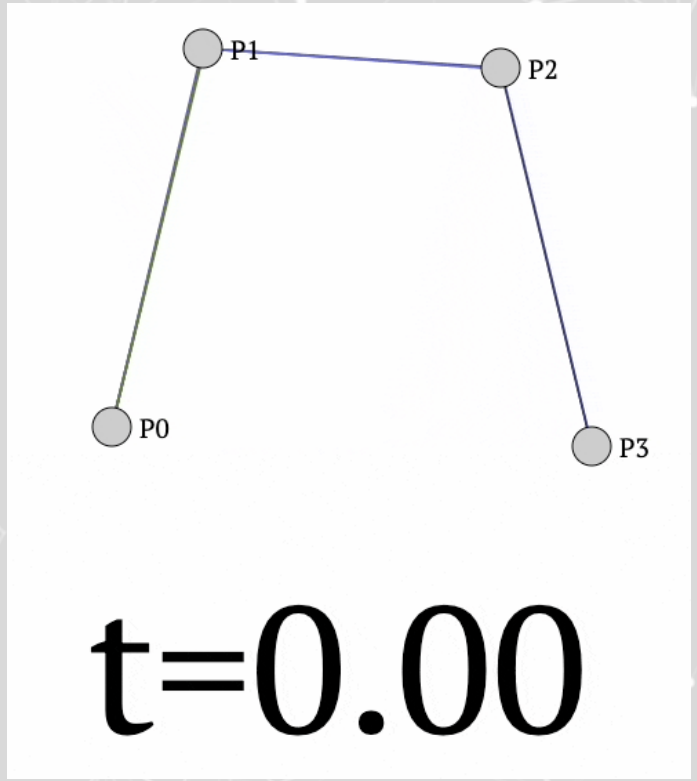
$$P = (1-t) \cdot P_1 + t \cdot P_2$$

$$0 \leq t \leq 1$$



- When $t = 0$, the position P will be P_1.
- When $t = 1$, the position P will be P_2.
- We can vary the parameter t from [0,1] and obtain the trajectory.





Bezier curves

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i}\, t^i\, P_i \;;\; \binom{n}{i} = \frac{n!}{i!(n-i)!}$$

Binomial coefficient

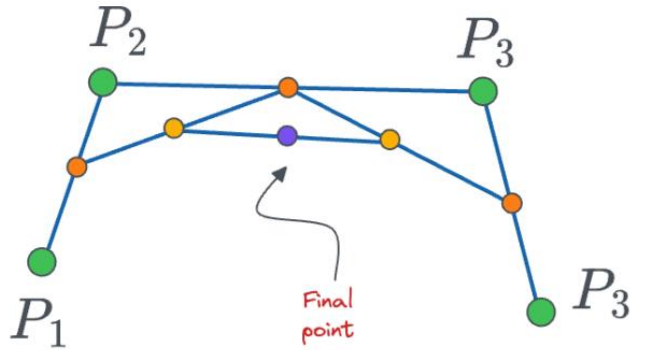
$$= \sum_{i=0}^{n} c_{i,n}(t)\, P_i$$

In the above formula, the $c_{i,n}$ is a function of t, and at every time stamp, it denotes the contribution of point P_i to the final curve.

In the above plot, at every timestamp t, the plot denotes the contribution of every point to the final curve. For instance:

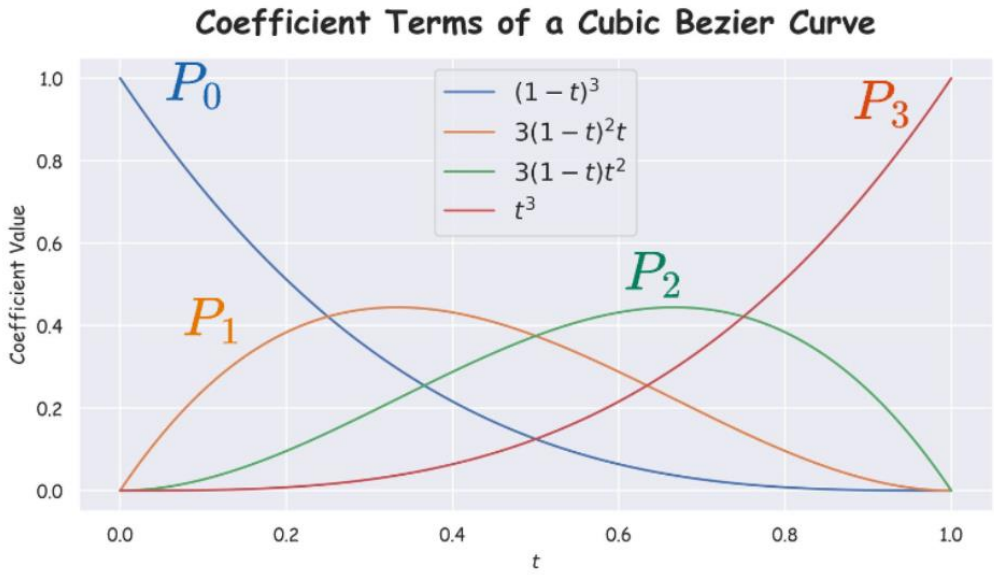
- At $t = 0$, except P_0, coefficients of all points are zero, which means the curve starts from P_0.
- From $t \in (0.25, 0.5)$, the coefficient of point P_1 is max, which means that the curve is closest to P_1 in that duration.
- From $t \in (0.5, 0.75)$, the coefficient of point P_2 is max, which means that the curve is closest to P_2 in that duration.
- At $t = 1$, except P_3, coefficients of all points are zero, which means the curve ends at P_3.

Let's go back to the case where we had 4 points:



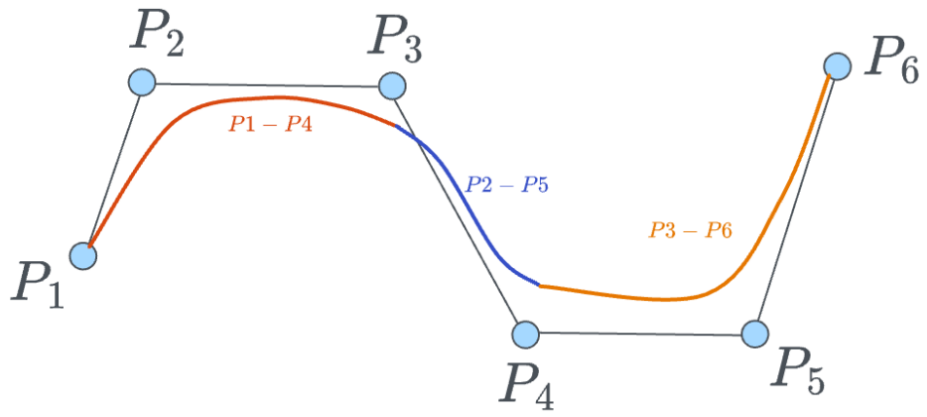
$$B(t) = \boxed{(1-t)^3} P_1 + \boxed{3(1-t)^2 t} P_2 + \boxed{3(1-t)t^2} P_3 + \boxed{t^3} P_4,\; t \in [0,1]$$

If we plot the coefficient terms (marked in yellow boxes above), we get the following plot:



(Basis/B) Splines

Mathematical functions that's used to connect data points smoothly by connecting simpler polynomial functions (basis functions).



Note: In this diagram, the individual Bezier curves don't appear to be connected that well, but in reality, the final curve is smooth.

When we have n control pints (6 in the diagram above), and we create k degree polynomial Bezier curves, we get $(n - k)$ Bezier curves in the final Bsplines.

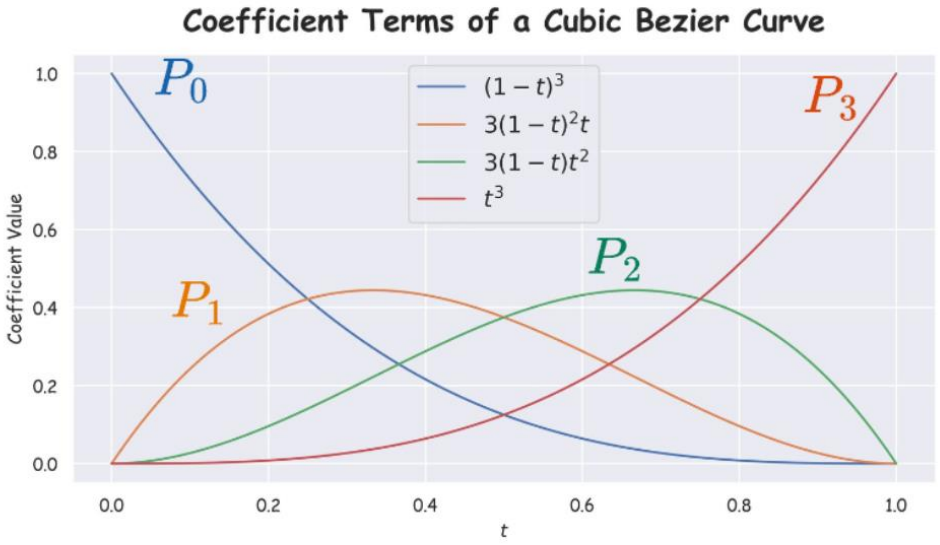
In a gist, the core idea is to ensure certain continuity conditions at the points where the curves meet.

$$S(t) = \sum_{i=0}^{n} P_i \cdot N_{i,k}$$

control points

Basis function

- P_i: Control points that define the shape of the curve.
- $N_{i,k}(t)$: B-spline basis functions of degree k associated with each control point P_i, and they are similar to what we saw earlier in the case of Bezier curves and are **fixed**.



The basis functions of a 3-degree Bezier curve

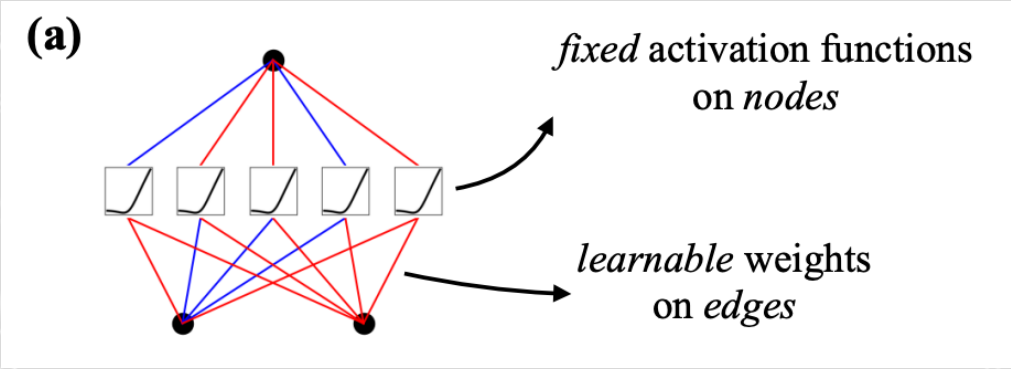
MLPs

Fully-connected feedforward neural networks

Expressive power guaranteed by the **universal approximation theorem**

Significant drawbacks

- Efficiency: In transformers for example, MLPs consume almost all non-embedding parameters
- Interpretability: typically less interpretable (relative to attention layers) without post-analysis tools



Splines vs MLPs

Splines:

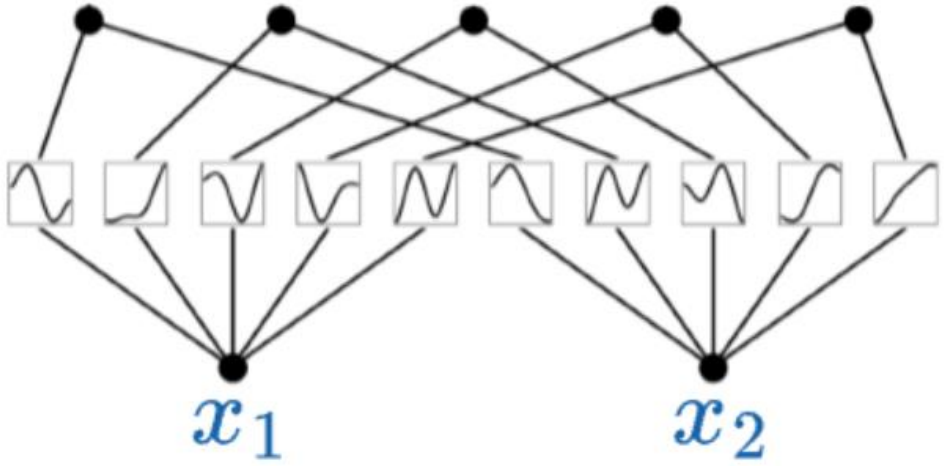
- Strengths:
  - Accurate for low-dimensional functions (functions with few variables).
  - Easy to adjust locally (fine-tune specific parts of the function).
  - Flexible resolution (can represent functions with varying levels of detail).
- Weakness:
  - Curse of dimensionality (become very complex and inefficient for high-dimensional functions - functions with many variables).

MLPs:

- Strength:
  - Less susceptible to the curse of dimensionality due to their ability to learn features (identify patterns within the data).
- Weakness:
  - Less accurate than splines in low dimensions because they can't perfectly optimize simple functions involving single variables.

- KANs combine the strengths of both approaches

One layer of KAN



ϕ^1

ϕ^1 denotes the transformation in the first layer

$$\phi^1 = \begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} \\ \phi_{21} & \phi_{12} & \cdots & \phi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{m1} & \phi_{m2} & \cdots & \phi_{mn} \end{bmatrix}$$

Output of first layer

$$z^1 = \underbrace{\begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} \\ \phi_{21} & \phi_{12} & \cdots & \phi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{m1} & \phi_{m2} & \cdots & \phi_{mn} \end{bmatrix}}_{\text{Transformation matrix } \phi^1} * \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_{\text{Input } X}$$

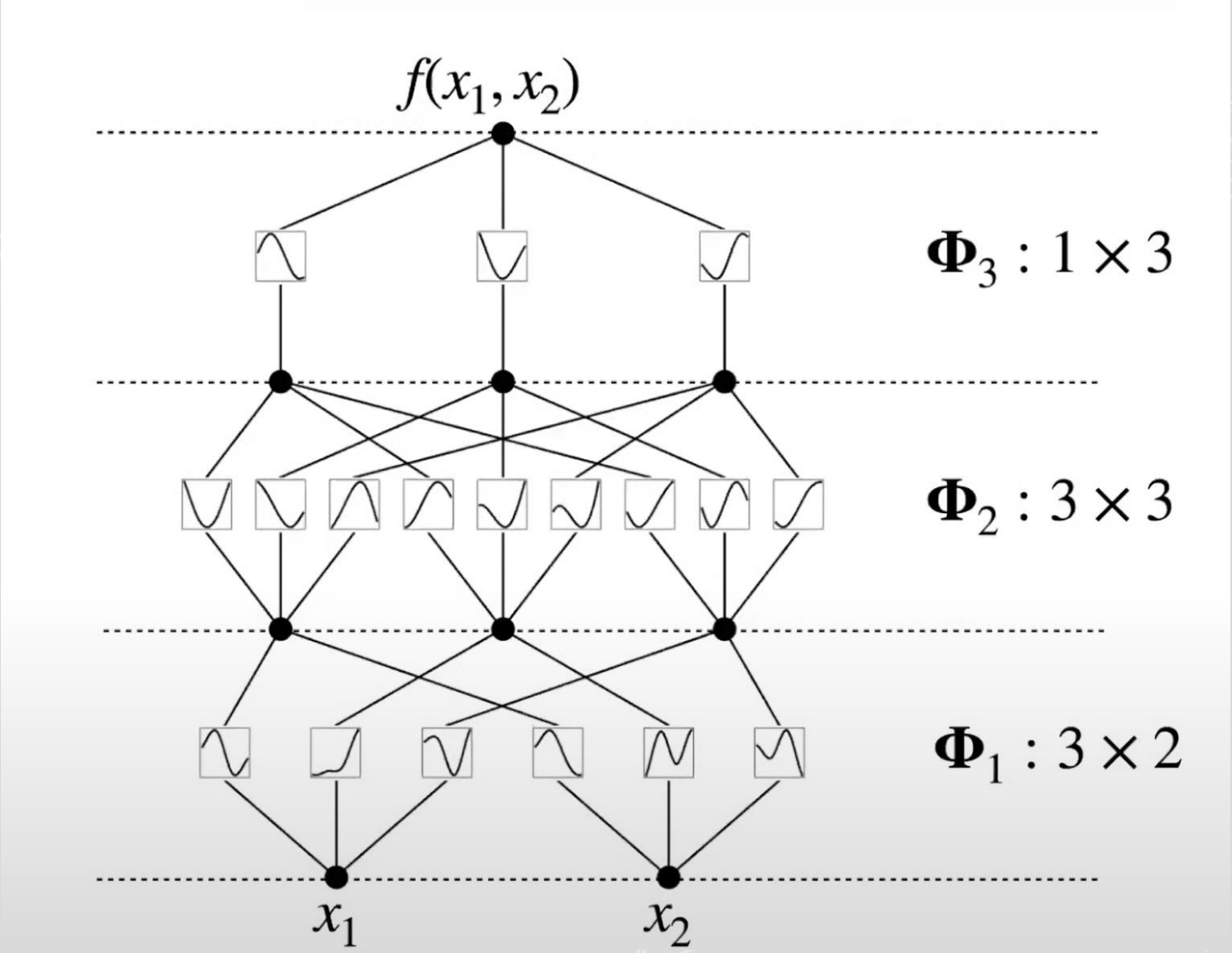
$$z^1 = \begin{bmatrix} \overbrace{\phi_{11}(x_1)}^{\text{Function applied to } x_1} + \phi_{12}(x_2) + \cdots + \overbrace{\phi_{1n}(x_n)}^{\text{Function applied to } x_n} \\ \phi_{21}(x_1) + \phi_{22}(x_2) + \cdots + \phi_{2n}(x_n) \\ \vdots \\ \phi_{m1}(x_1) + \phi_{m2}(x_2) + \cdots + \phi_{mn}(x_n) \end{bmatrix}$$

What is a layer of KAN?

- n denotes the number of inputs.
- m denotes the number of output nodes in that layer.

Deeper KAN

- Layers can be stacked
- Scalability proof



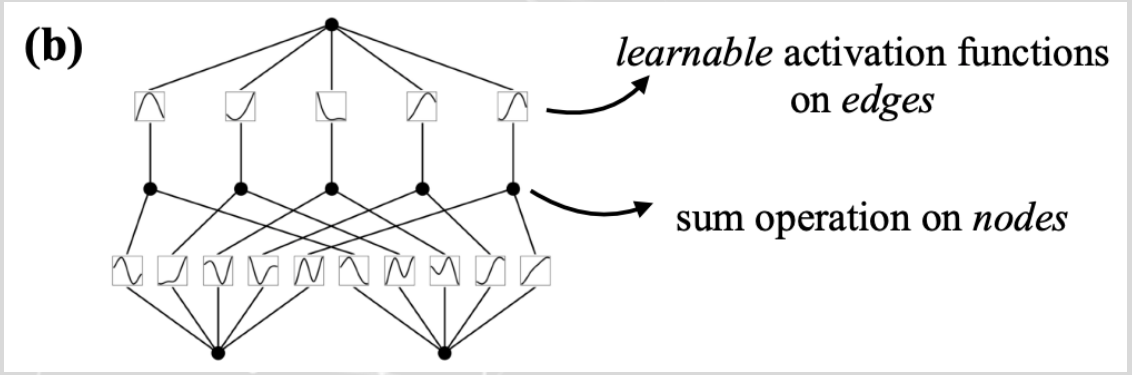
KAN architecture

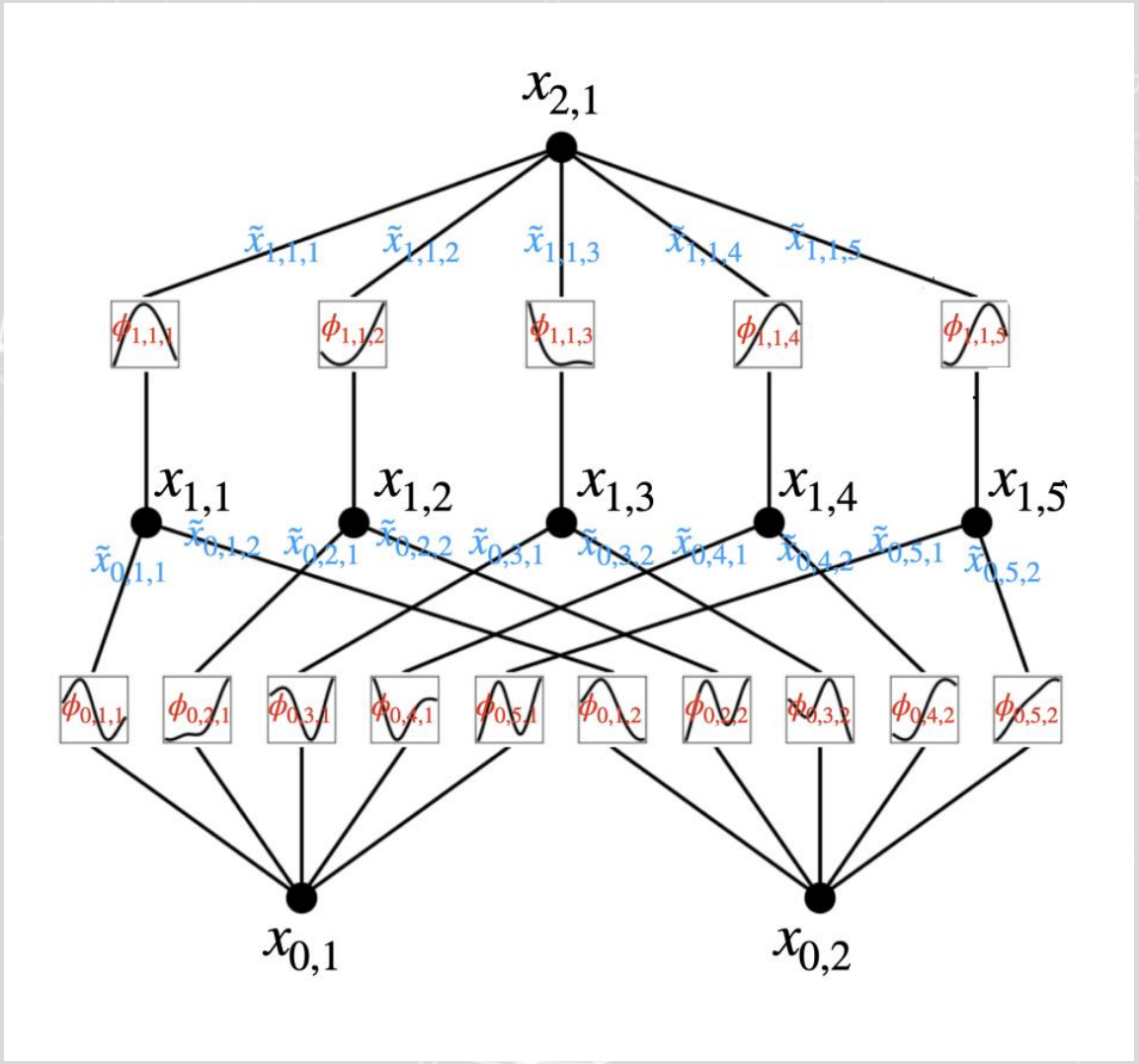
Thus, the entire KAN network can be condensed into one formula as follows:

$$KAN(x) = \phi^{L}\left(\phi^{L-1}\left(\ \dots\ \left(\phi^{2}\left(\phi^{1}(x)\right)\right)\right)\right)$$

Where:

- x denotes the input vector
- ϕ^{k} denotes the function transformation matrix of layer k.
- $KAN(x)$ is the output of the KAN network.





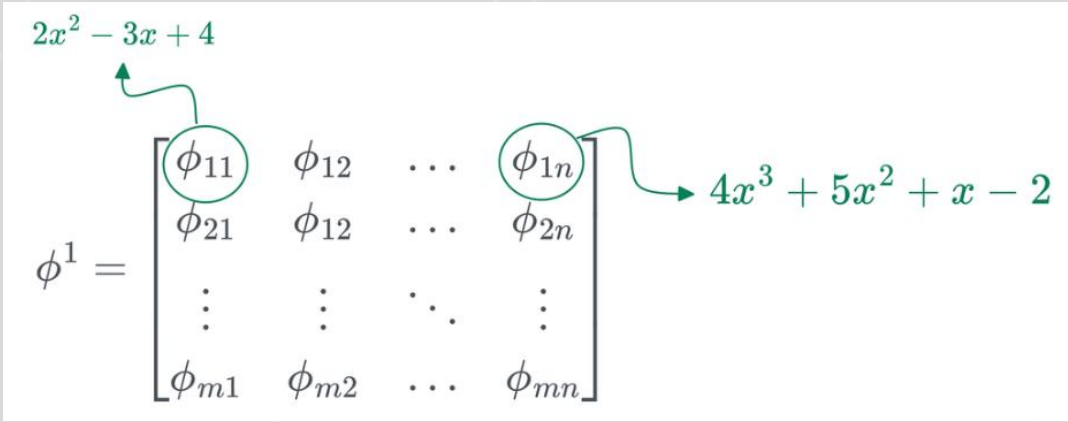
KAN formula vs MLP formula

$$KAN(x) = \phi^{L}\left(\phi^{L-1}\left(\ \dots\ \left(\phi^{2}\left(\phi^{1}(x)\right)\right)\right)\right)$$

$$NN(x) = \theta^{L}\left(\sigma(\theta^{L-1}\left(\ \dots\ \left(\sigma(\theta^{2}\left(\sigma(\theta^{1}(x))\right)\right)\right)\right)\right)$$

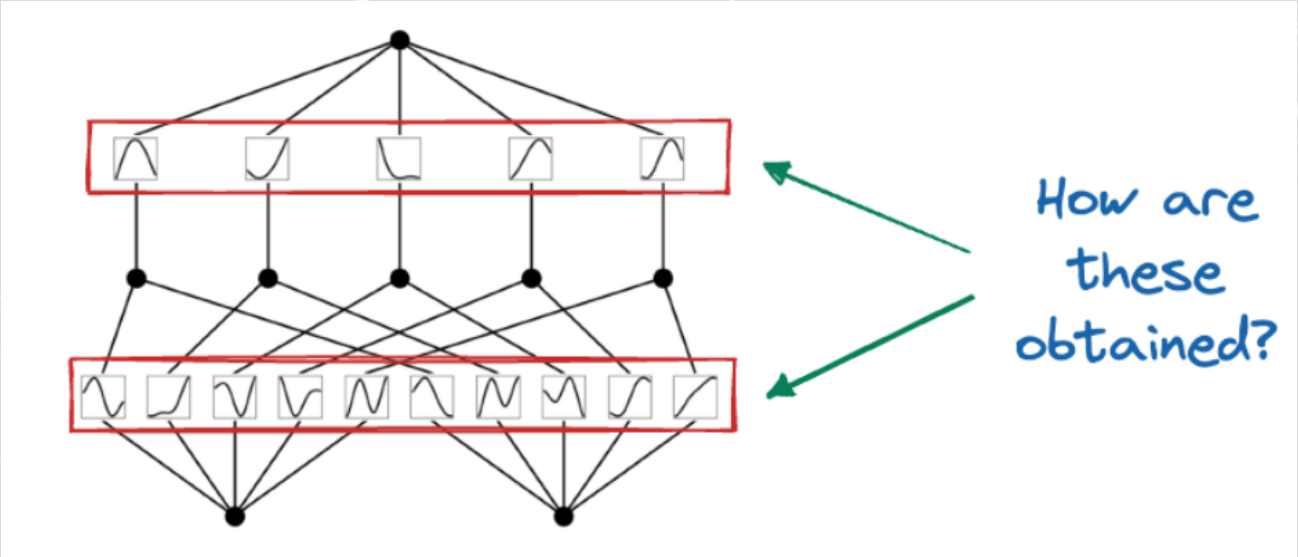
The only difference is that the parameters θ^i are linear transformations, and σ denotes the activation function used for non-linearity, and it is the same activation function across all layers.

In the case of KANs, the matrices ϕ^k themselves are non-linear transformation matrices, and each univariate function can be quite different.



KAN architecture

- Parametrize each 1D function as a B-spline curve
- Coefficients as Learnable Parameters



$$\phi^1 = \begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} \\ \phi_{21} & \phi_{12} & \cdots & \phi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{m1} & \phi_{m2} & \cdots & \phi_{mn} \end{bmatrix}$$

KAN architecture

Let's make the positions of control points learnable in the activation function so that the model is free to learn any arbitrary shape activation function that fits the data best.

By adjusting the positions of the control points during training, the KAN model can dynamically shape the activation functions that best fit the data.

- Start with the initial positions for the control points, just like we do with weights.
- During the training process, update the positions of these control points through backpropagation, similar to how weights in a neural network are updated.
- Use an optimization algorithm (e.g., gradient descent) to adjust the control points so that the model can minimize the loss function.

KAN architecture

Mathematically speaking, in KANs, every activation function $\phi(x)$ is defined as follows:

$$\phi(x) = w(\, b(x) \;+\; spline(x)\,)$$

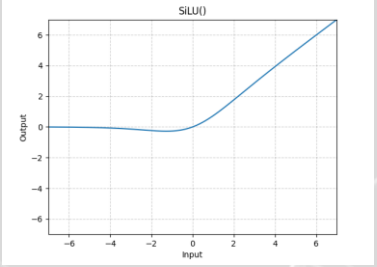
Within this:

learnable fixed learnable

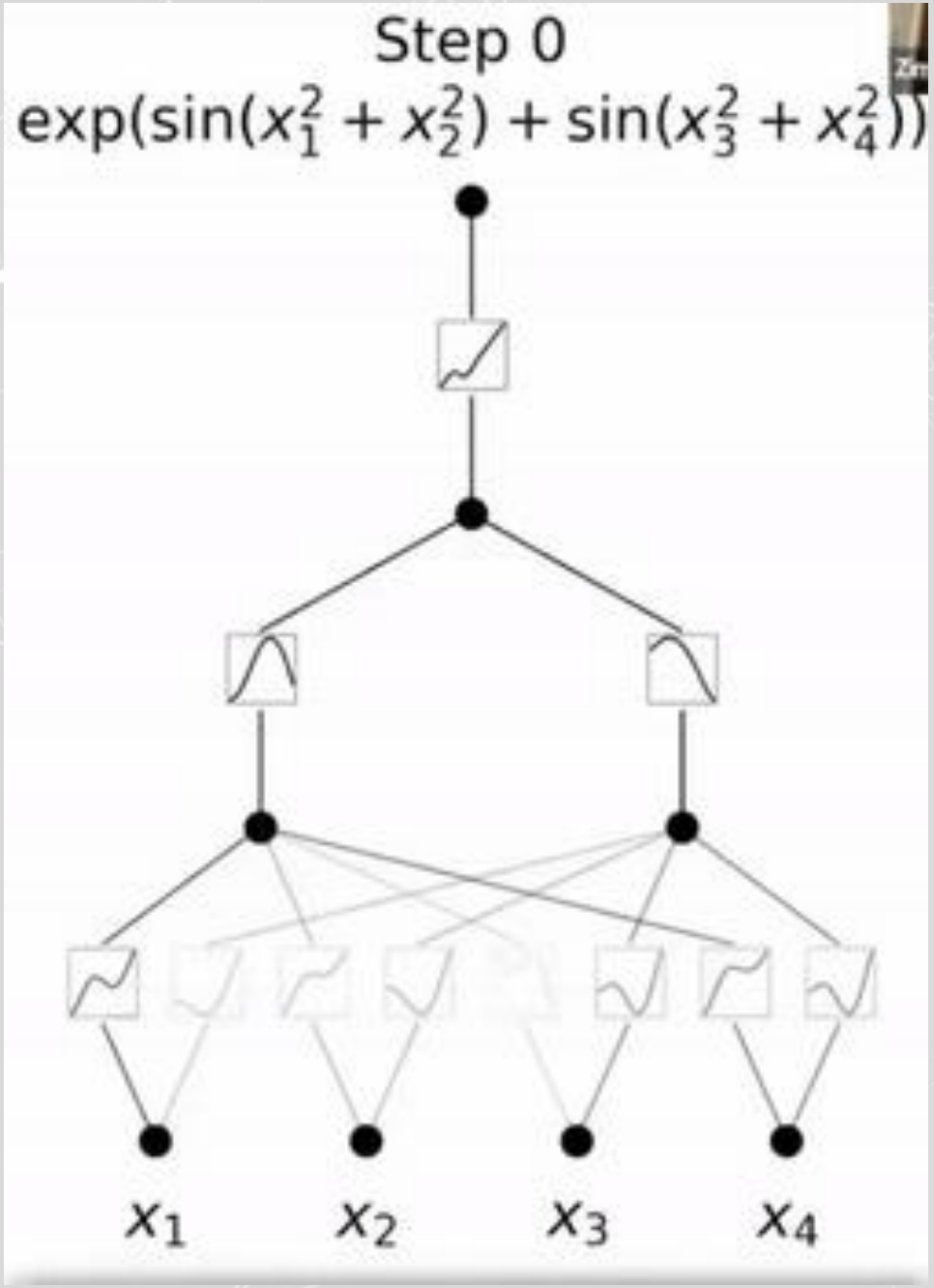
$$\phi(x) = w\,(b(x) + spline(x))$$

$$b(x) = \frac{x}{1+e^{-x}} \qquad spline(x) = \sum_i c_i\, B_i(x)$$

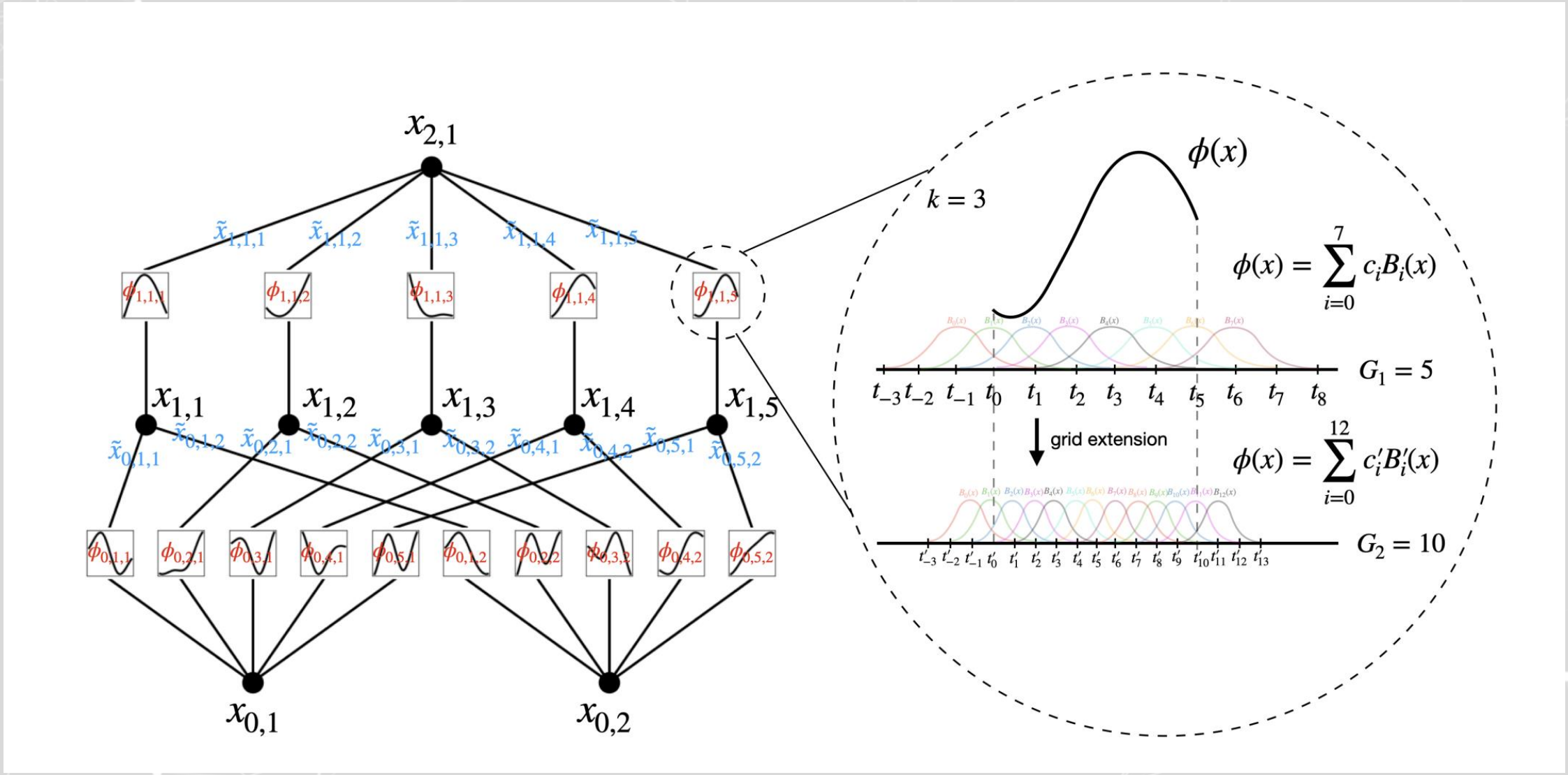
learnable



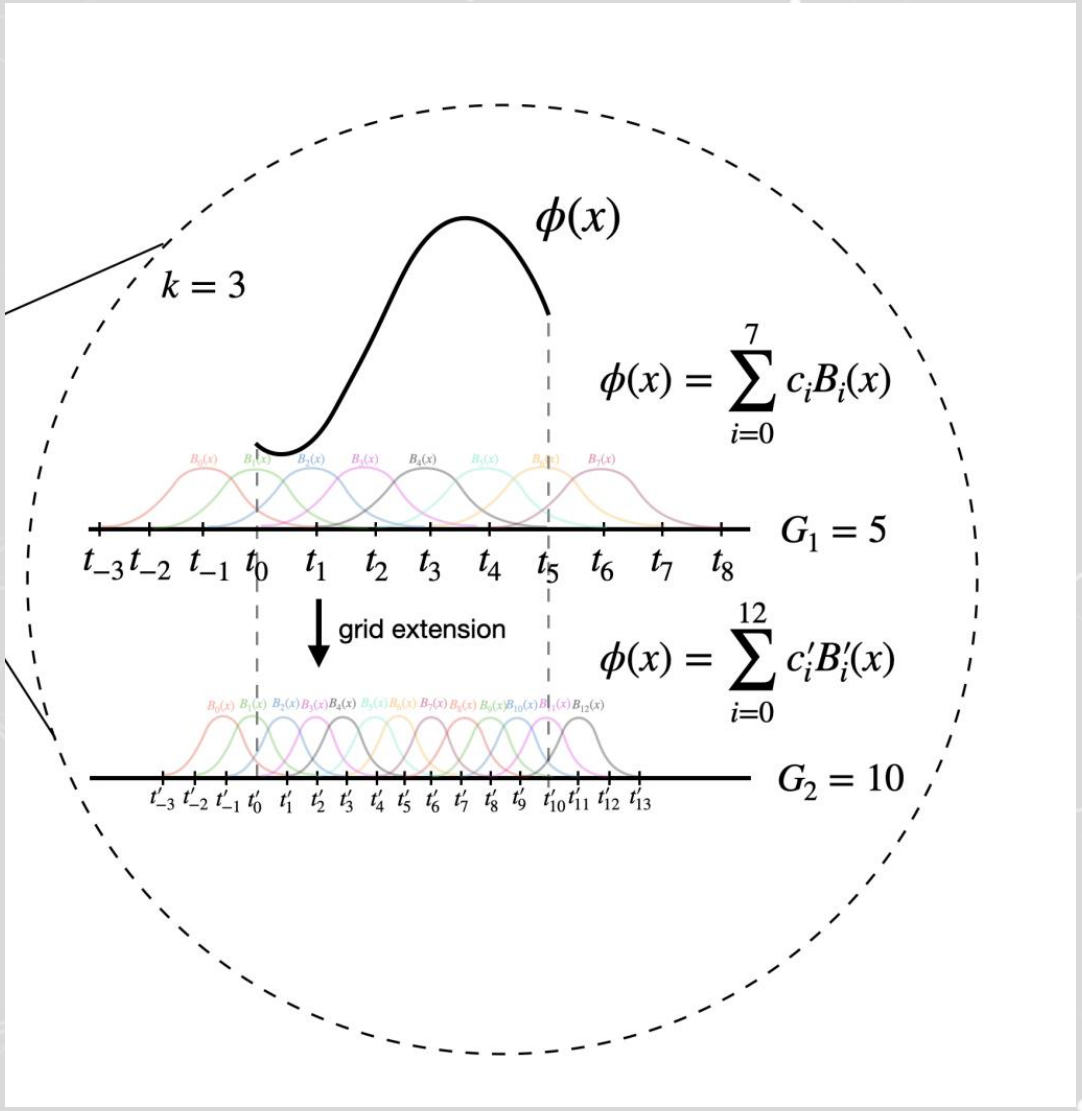
- The computation involves a basis function $b(x)$ (similar to residual connections).
- $spline(x)$ is learnable, specifically the parameters c_i, which denotes the position of the control points.
- There's another parameter w. The authors say that, in principle, w is redundant since it can be absorbed into $b(x)$ and $spline(x)$. Yet, they still included this w factor to better control the overall magnitude of the activation function.



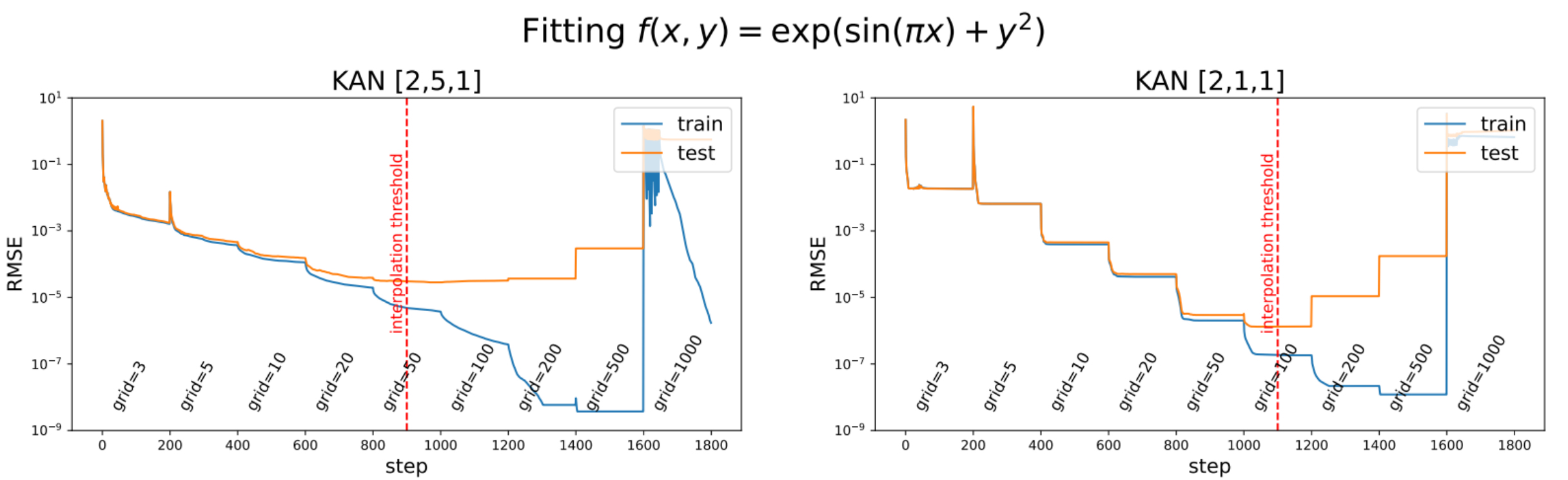
KAN architecture



Grid Extension

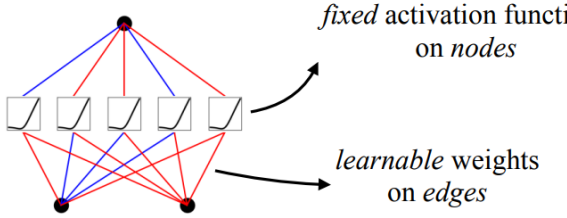
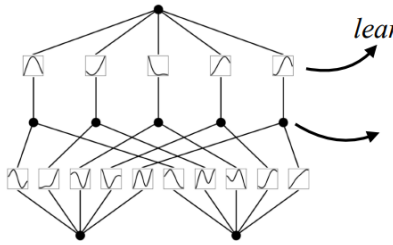
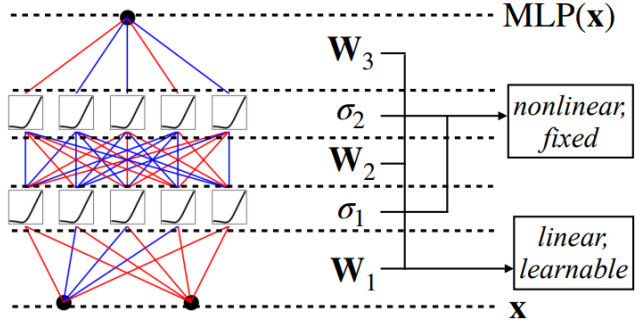
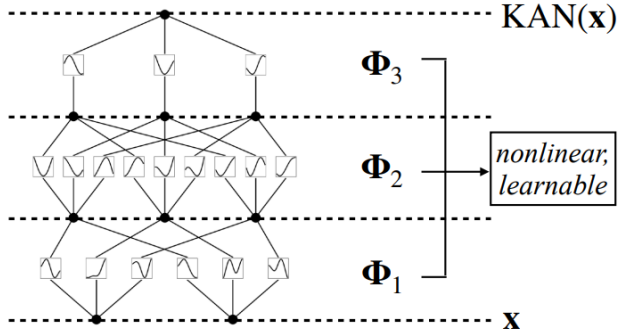


Grid Extension



$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},$$

MLP vs KAN

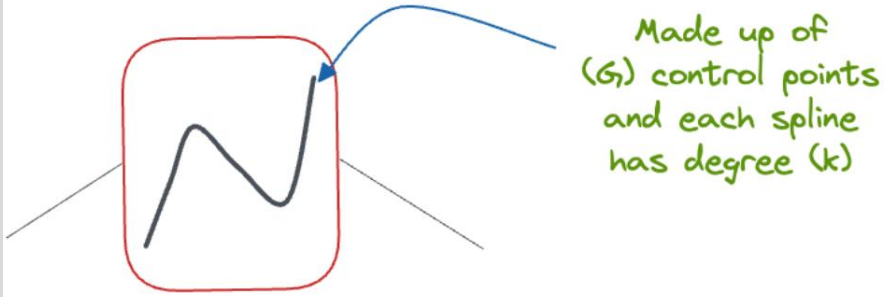
Model	Multi-Layer Perceptron (MLP)	Kolmogorov-Arnold Network (KAN)
Theorem	**Universal Approximation Theorem**	**Kolmogorov-Arnold Representation Theorem**
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1}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

MLP vs KAN

MLP	KAN
Universal approximation theorem	**Kolmogorov-Arnold representation theorem**
Less interpretable	Interpretable
More parameters	Less parameters
Place fixed activation functions on nodes (“neurons”)	Place learnable activation functions on edges (“weights”)
Have linear weight matrices	Each weight parameter is replaced by a learnable 1D function parametrized as a spline
Nodes sum incoming signals with applying non-linearities.	Nodes sum incoming signals without applying any non-linearities.

MLP vs KAN: Parameter count

- The number of edges from one layer to another is the same in both cases – N^2.
- While the edges of MLP just hold one weight, the edges of KAN hold more parameters because of splines.

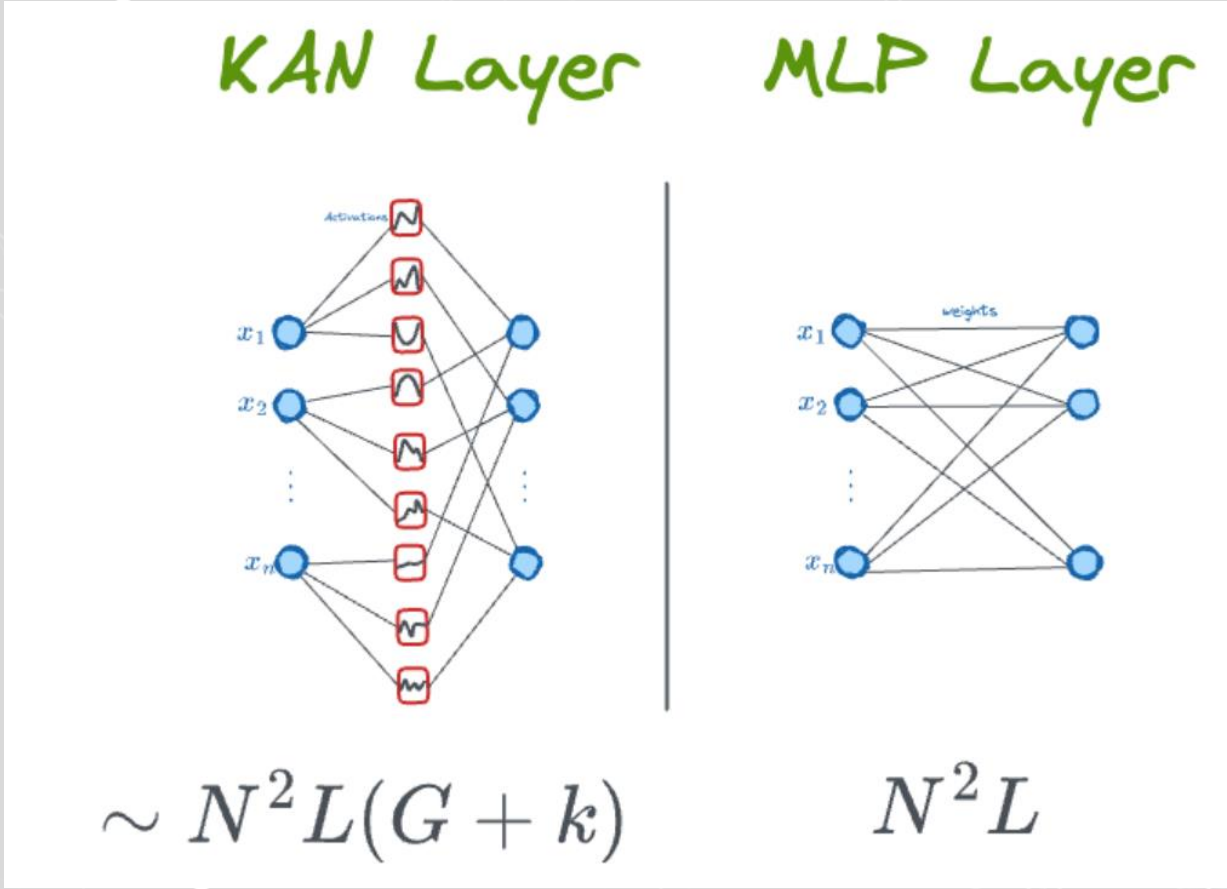


  - For a B-spline with G control points and degree k, the number of basis functions are $G + k - 1$. In KAN, as each basis function is associated with one parameter, the number of parameters is also the same – $G + k - 1$.

$$spline(x) = \sum_i c_i \, B_i(x)$$

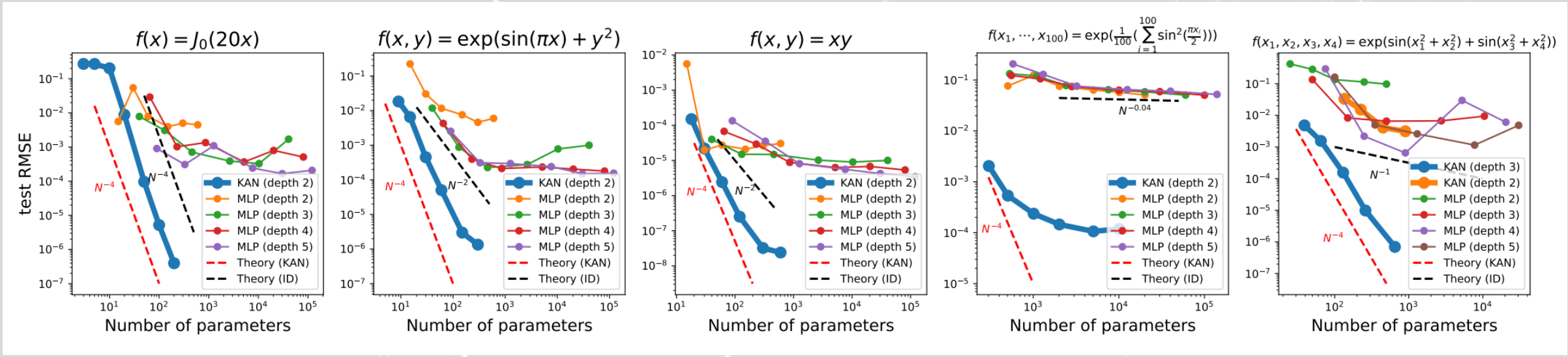
parameter

  - So, every edge in a KAN holds $G + k - 1$ parameters.



So why they say it is better?
Because it is more expressive and can solve the same problem using less parameters or layers

Results


$f(x) = J_0(20x)$
$f(x, y) = \exp(\sin(\pi x) + y^2)$
$f(x, y) = xy$
$f(x_1, \cdots, x_{100}) = \exp(\frac{1}{100}(\sum_{i=1}^{100} \sin^2(\frac{\pi x_i}{2})))$
$f(x_1, x_2, x_3, x_4) = \exp(\sin(x_1^2 + x_2^2) + \sin(x_3^2 + x_4^2))$
test RMSE
Number of parameters
KAN (depth 2)
KAN (depth 3)
MLP (depth 2)
MLP (depth 3)
MLP (depth 4)
MLP (depth 5)
Theory (KAN)
Theory (ID)
N^{-4}
N^{-2}
N^{-1}
$N^{-0.04}$

Results

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}

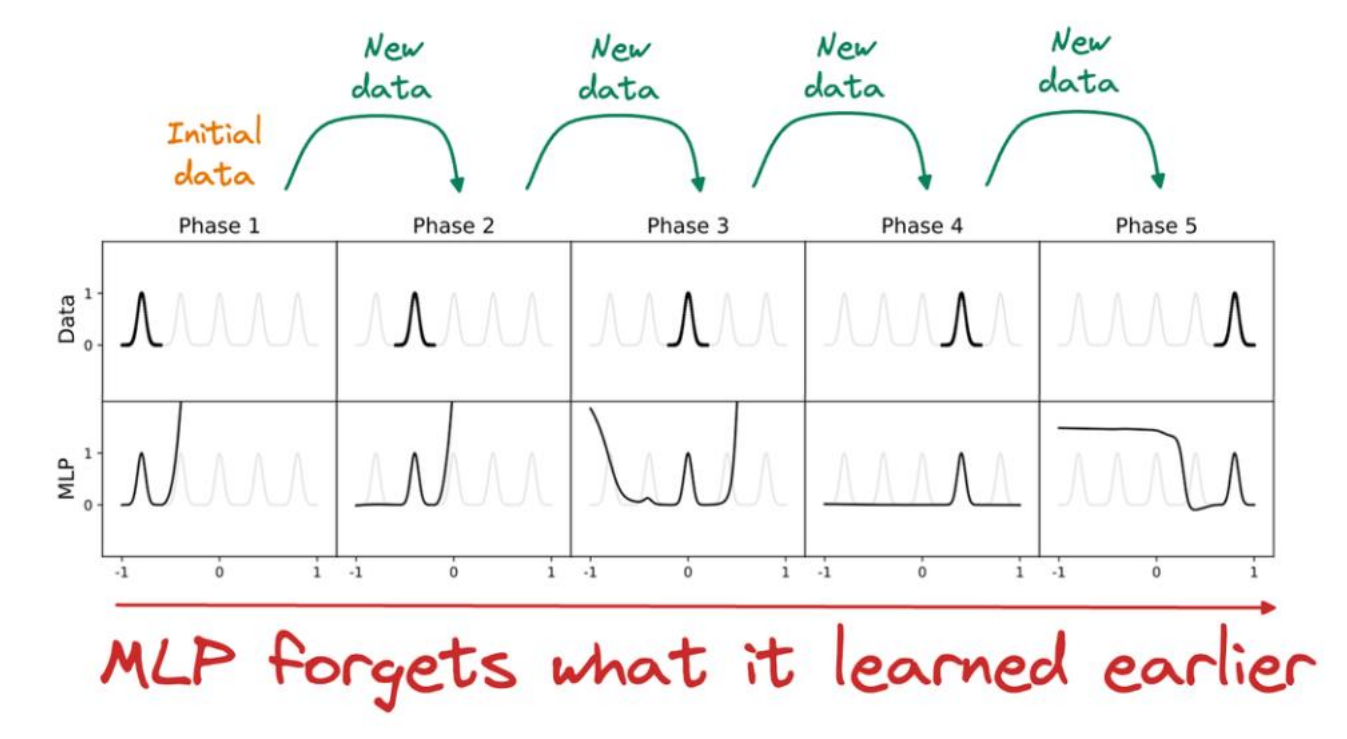
Table 1: Special functions

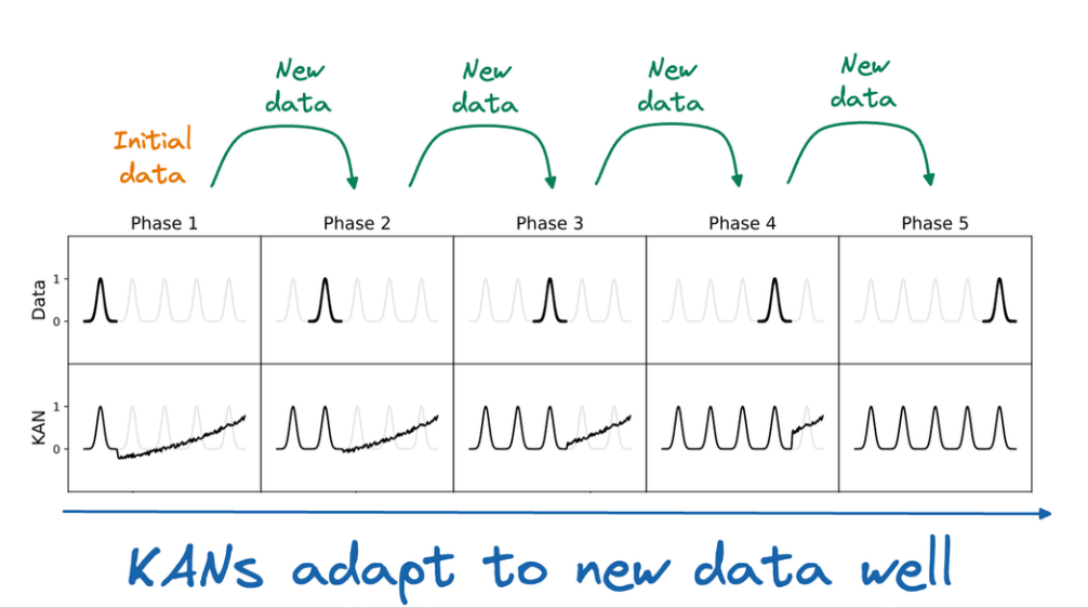
Results

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}

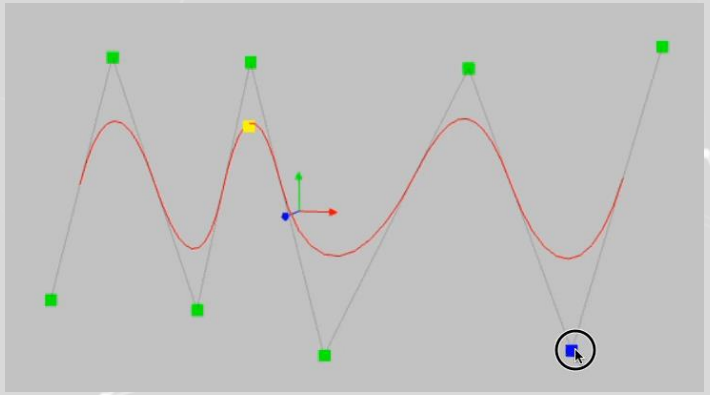
Table 2: Feynman dataset

Continual Learning



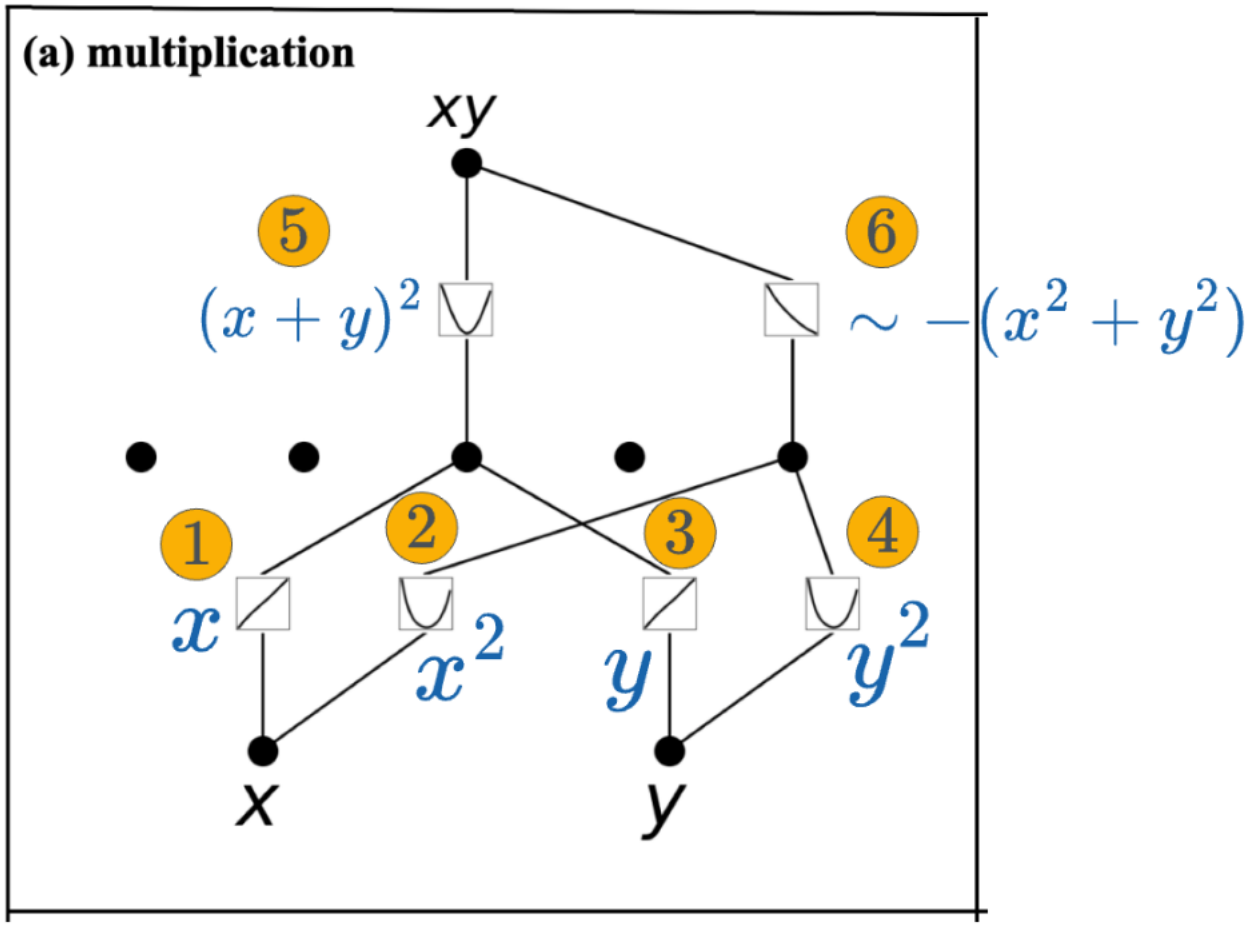


Why it does not forget?
Notion of locality-storage of knowledge



Interpretability

For instance, consider the KAN network below, which learns $f(x, y) = xy$.



Simplification

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,

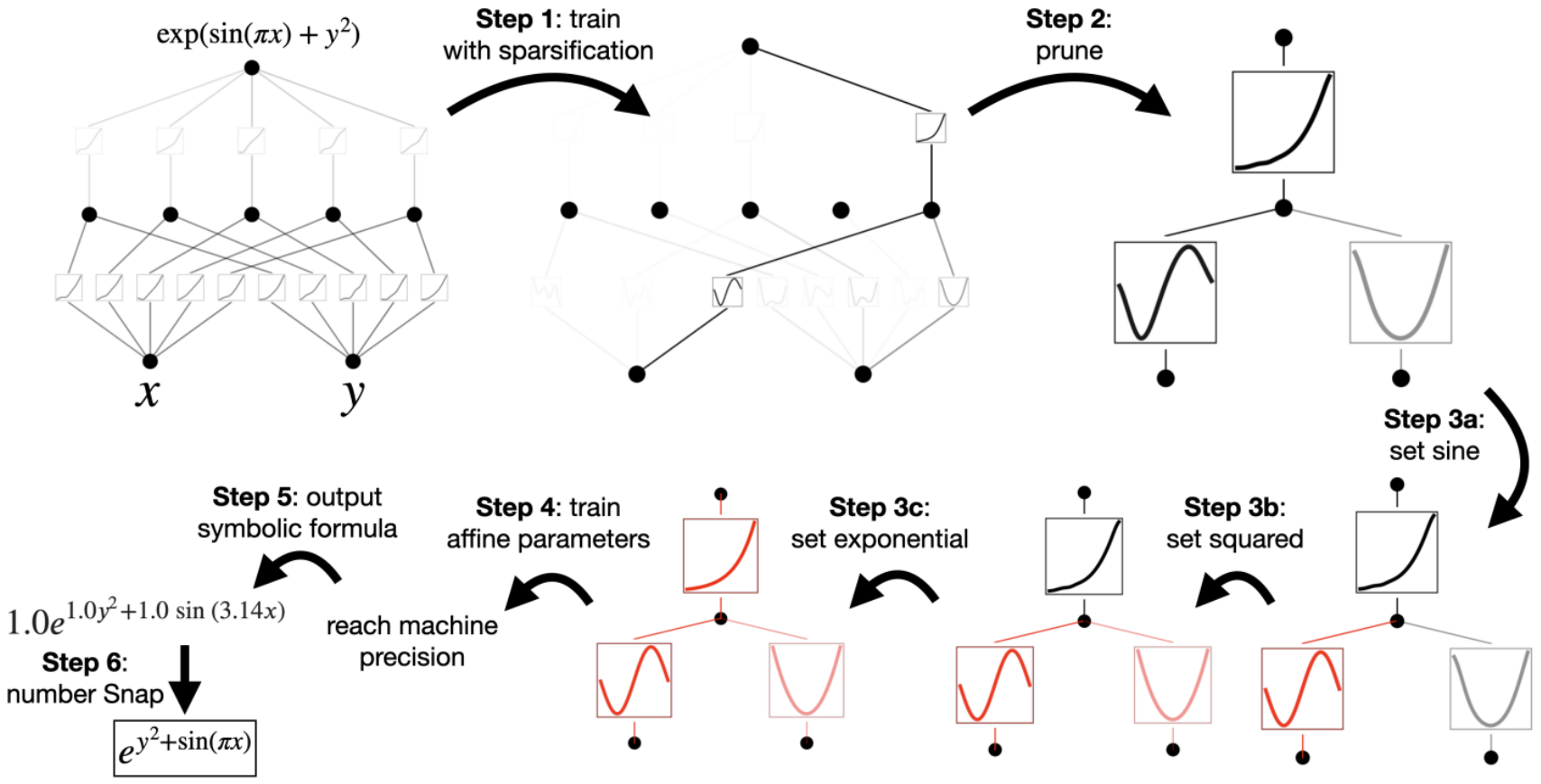
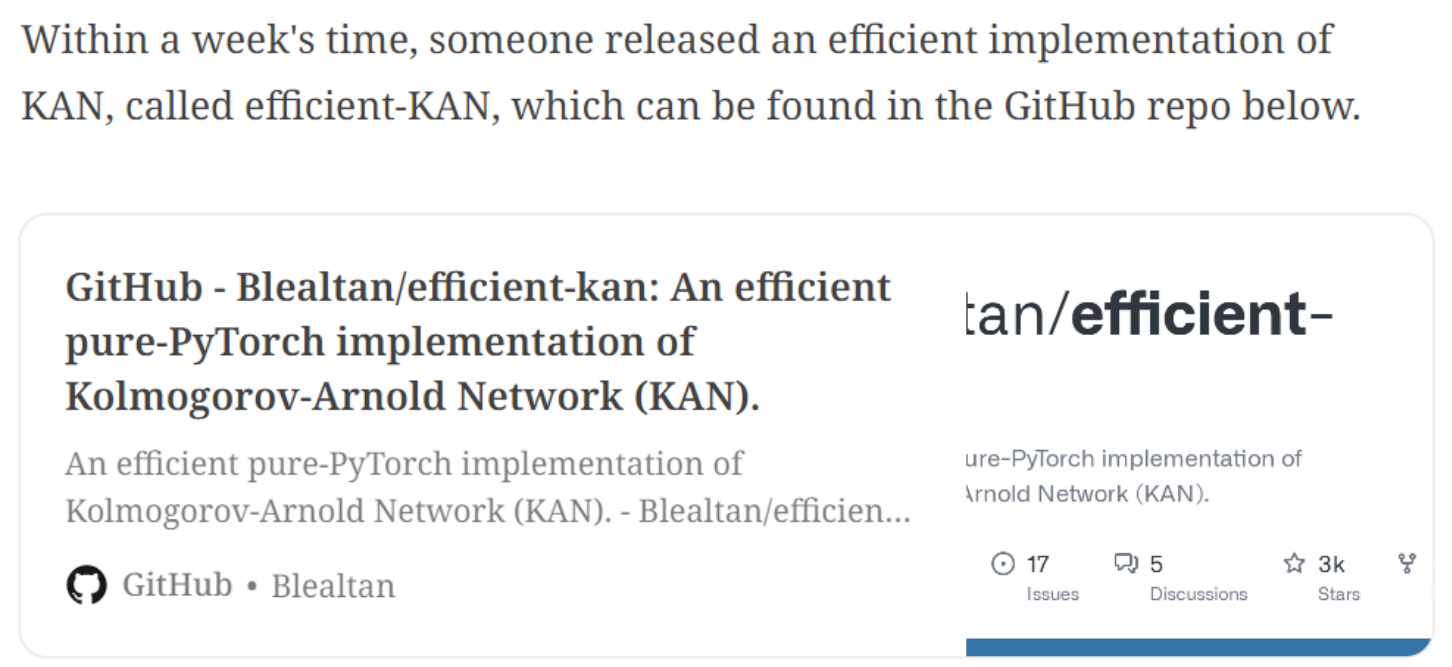


Figure 2.4: An example of how to do symbolic regression with KAN.

Run-time

They noticed that KANs are usually 10x slower than MLPs

The authors mention that they did not try much to optimize KANs' efficiency, and they treat bottlenecks more like an engineering problem, which can be improved in future iterations of KAN released by them or the community.



Conclusion

Scalability proof??

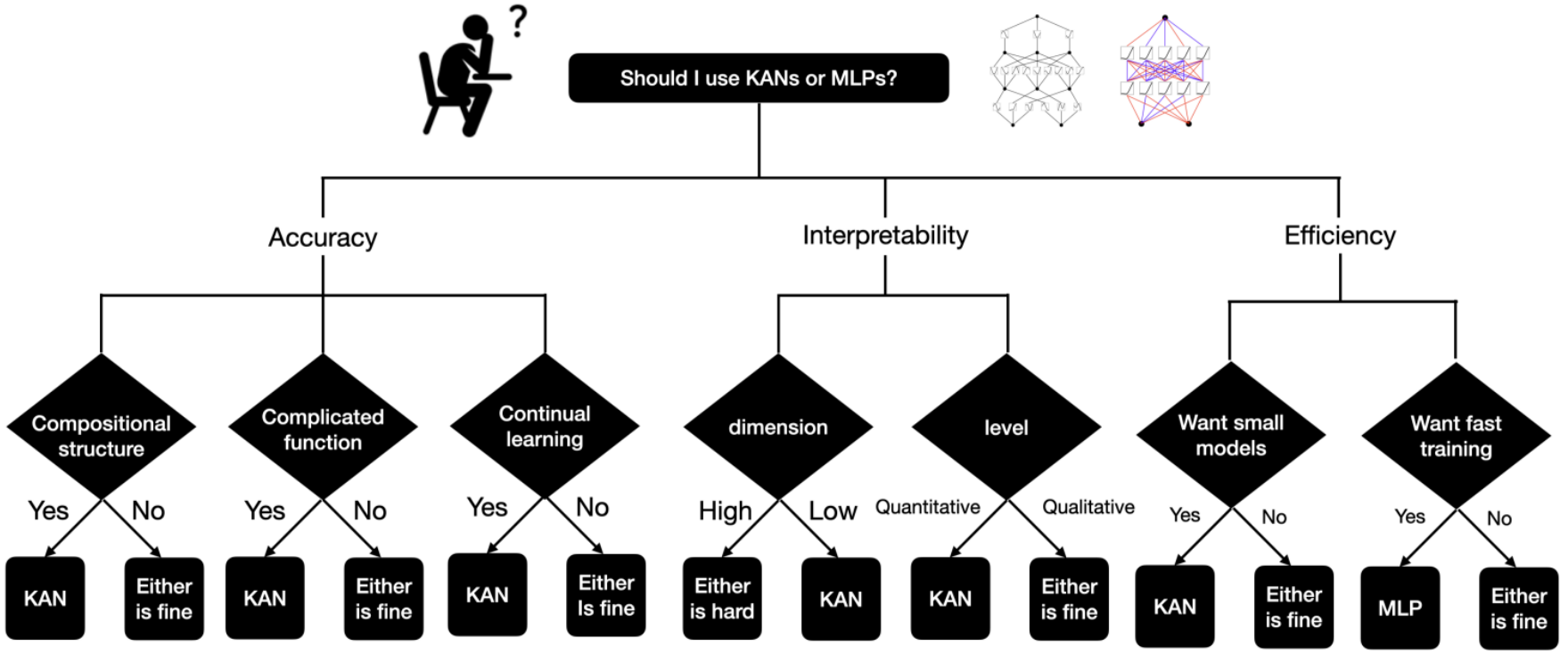


Figure 6.1: Should I use KANs or MLPs?


Thank You