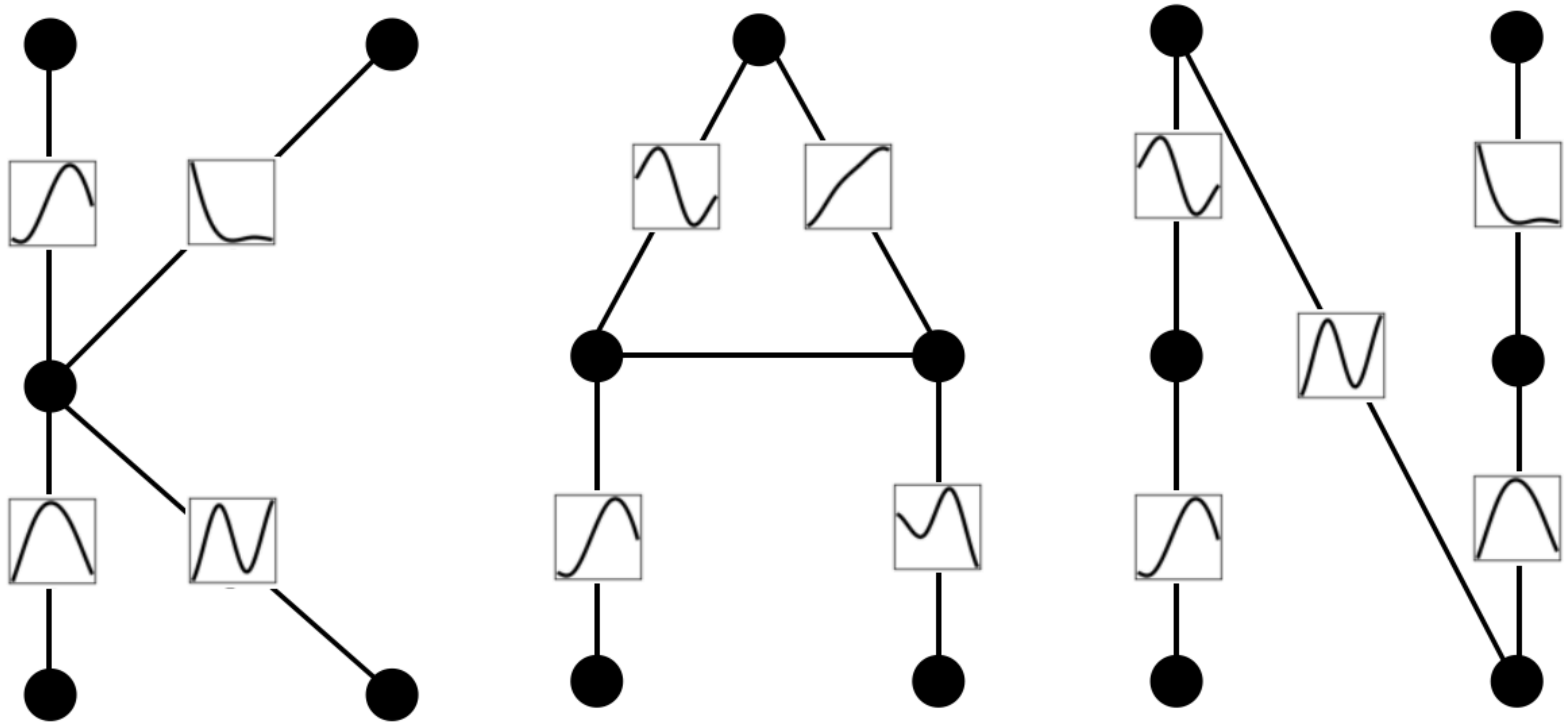
KAN: Kolmogorov-Arnold Networks



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

***PhD candidate, Caltech**

Based on arXiv: 2404.19756, 2408.10205, 2410.01803

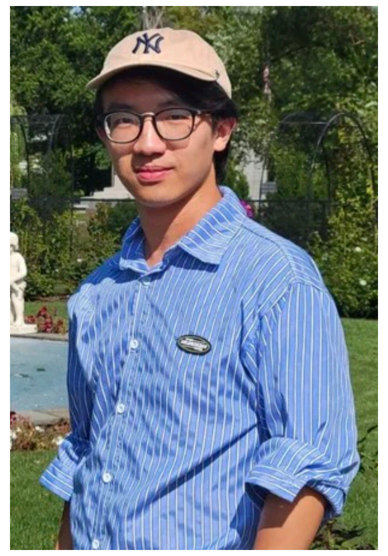
Computer Science > Machine Learning

[Submitted on 30 Apr 2024 (v1), last revised 2 May 2024 (this version, v2)]

KAN: Kolmogorov-Arnold Networks

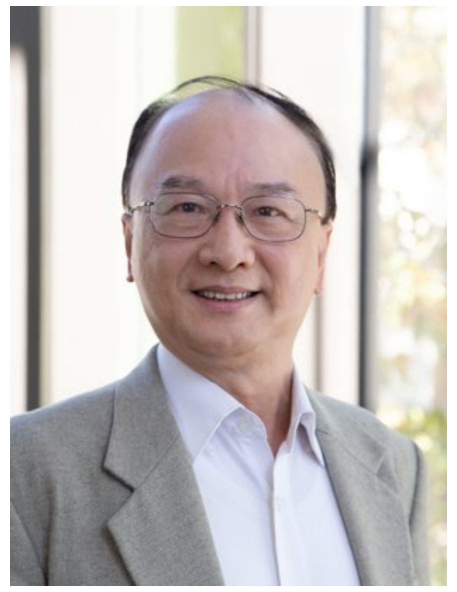
Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, Max Tegmark

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.



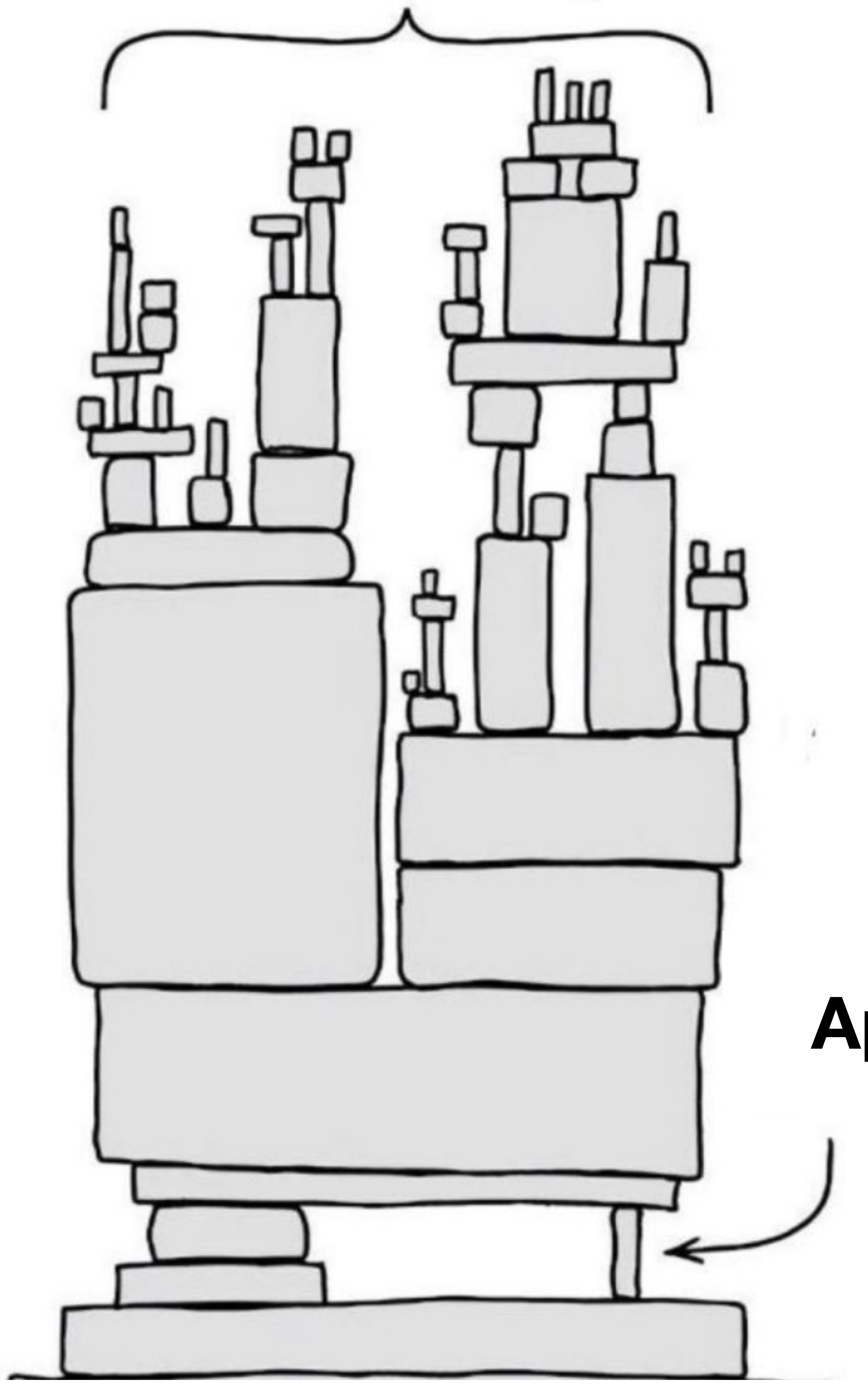






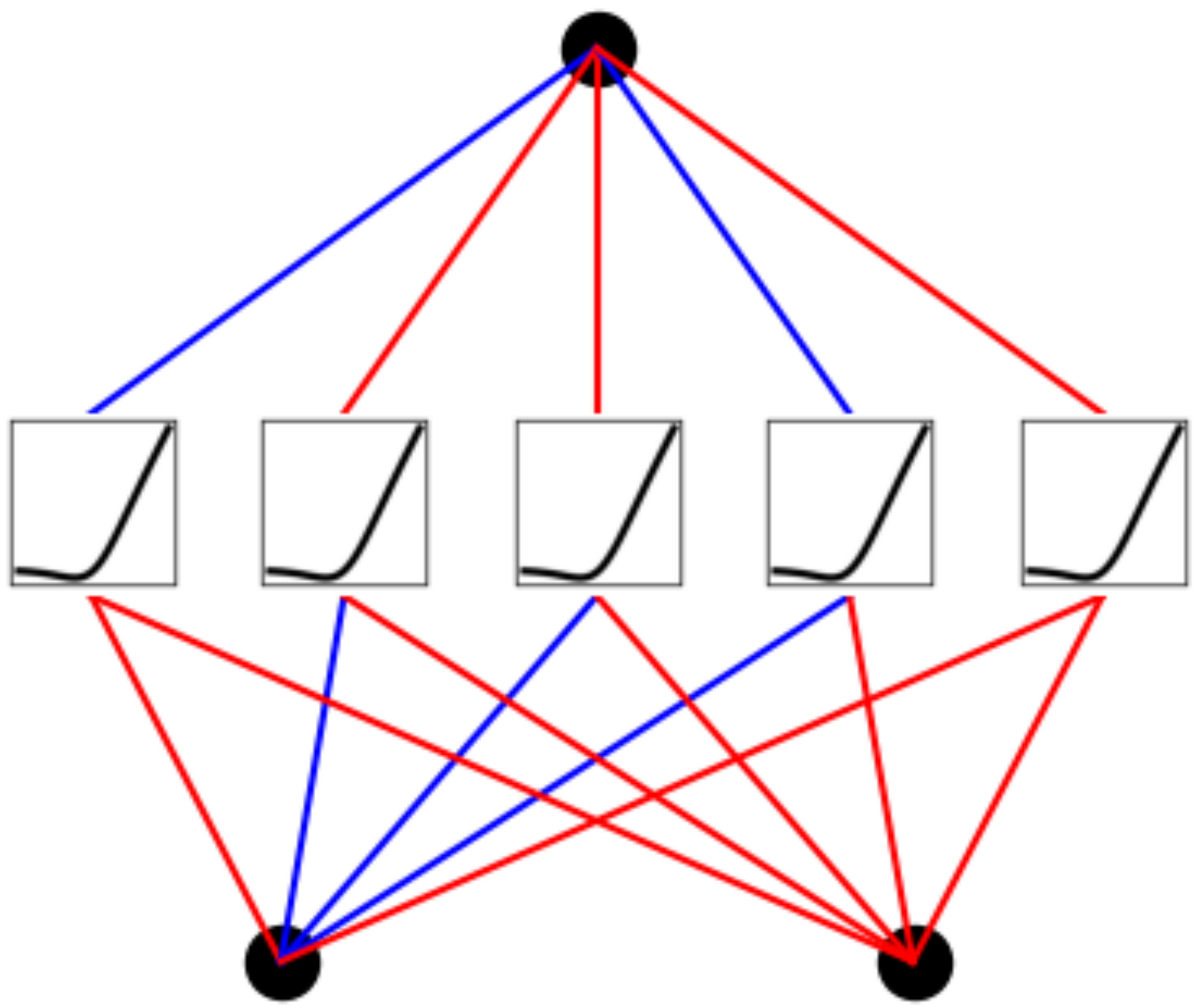

The foundation of AI

Today's AI



Universal Approximation Theorem

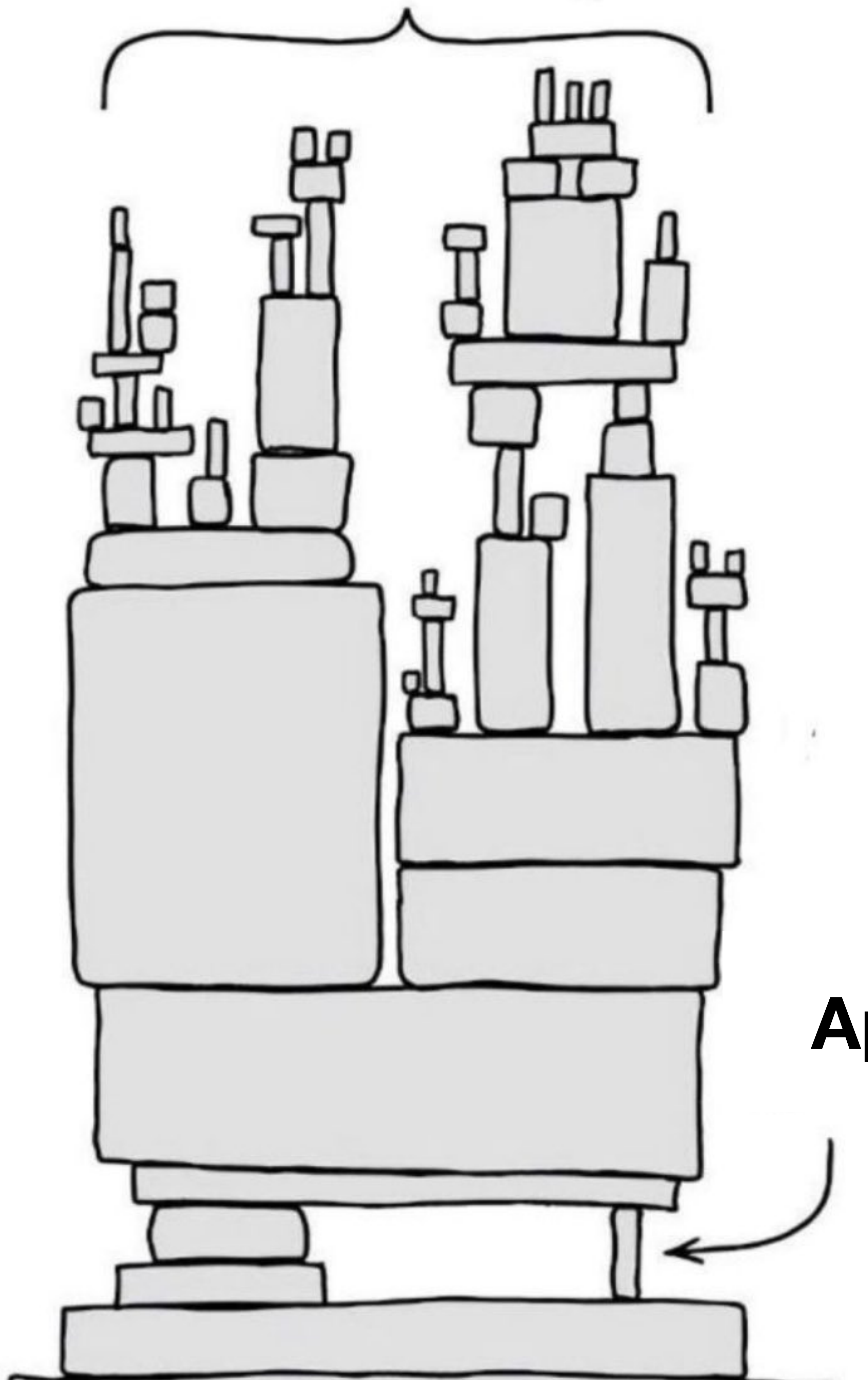
$$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$$



Multi-Layer Perceptron (MLP)

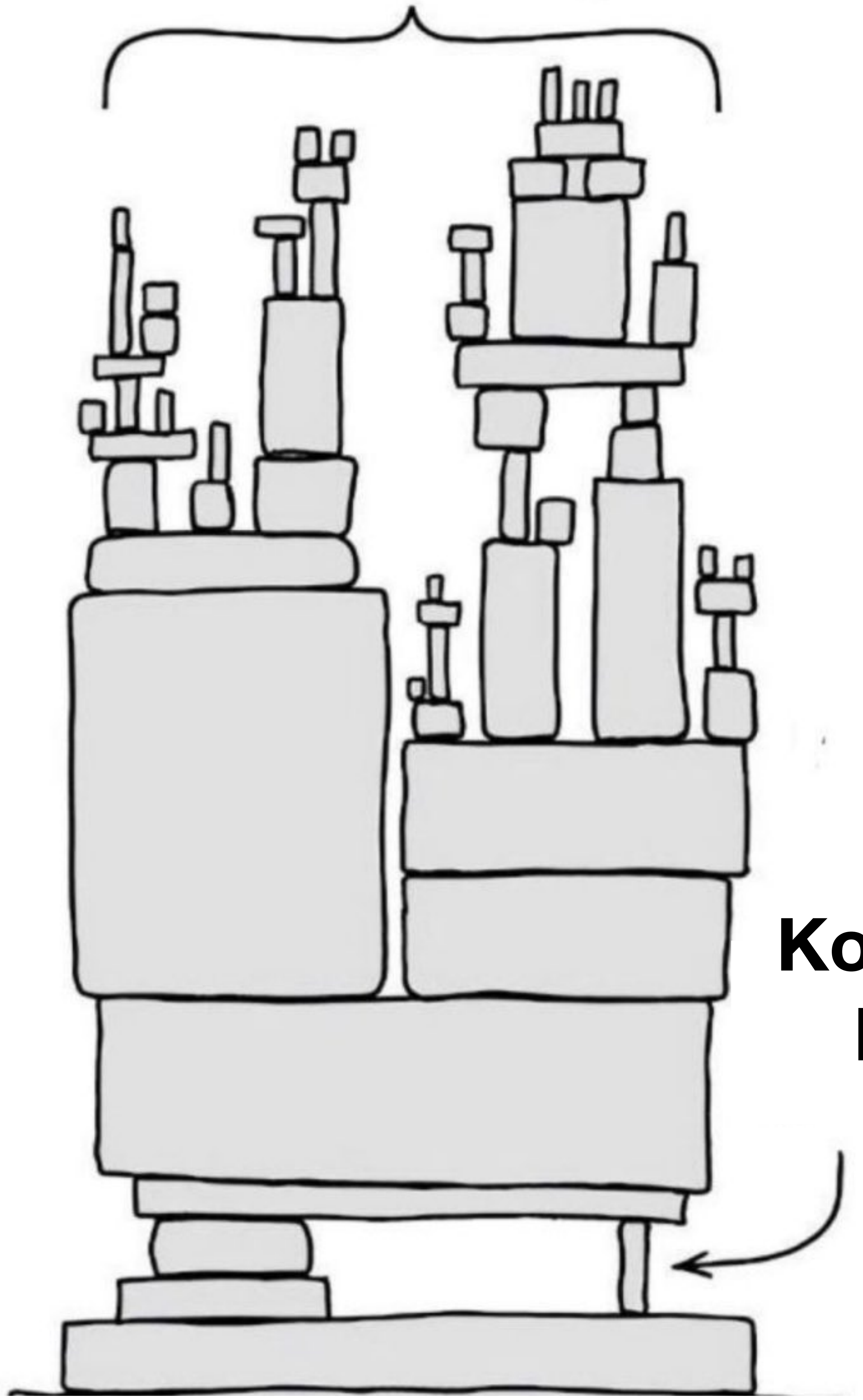
The foundation of AI

Today's AI



Universal Approximation Theorem

An Alternative to today's AI?



Kolmogorov-Arnold Representation Theorem

KA representation theorem

[From wikipedia]

If f is a multivariate continuous function, then f can be written as a finite composition of continuous functions of a single variable and the binary operation of addition.

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

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

In a sense, they showed that the **only true multivariate function is the sum**, since every other function can be written using **univariate functions and summing**.

Curse of dimensionality is gone!?

What's going wrong? Smoothness!

Journals & Magazines > Neural Computation > Volume: 1 Issue: 4

1989

Representation Properties of Networks: Kolmogorov's Theorem Is Irrelevant

Publisher: MIT Press

Cite This

PDF

Federico Girosi
Tomaso Poggio

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

Could be non-smooth!

A stable and usable *exact* representation of a function in terms of two or more layers network seems hopeless. In fact the result obtained by Kolmogorov can be considered as a "pathology" of the continuous functions: it fails to be true if the inner functions h_{pq} are required to be smooth, as it has been shown by Vitushkin (1954). The theorem, though mathematically surprising and beautiful, cannot be used by itself in any constructive way in the context of networks for learning. This conclu-

"Naive" optimism

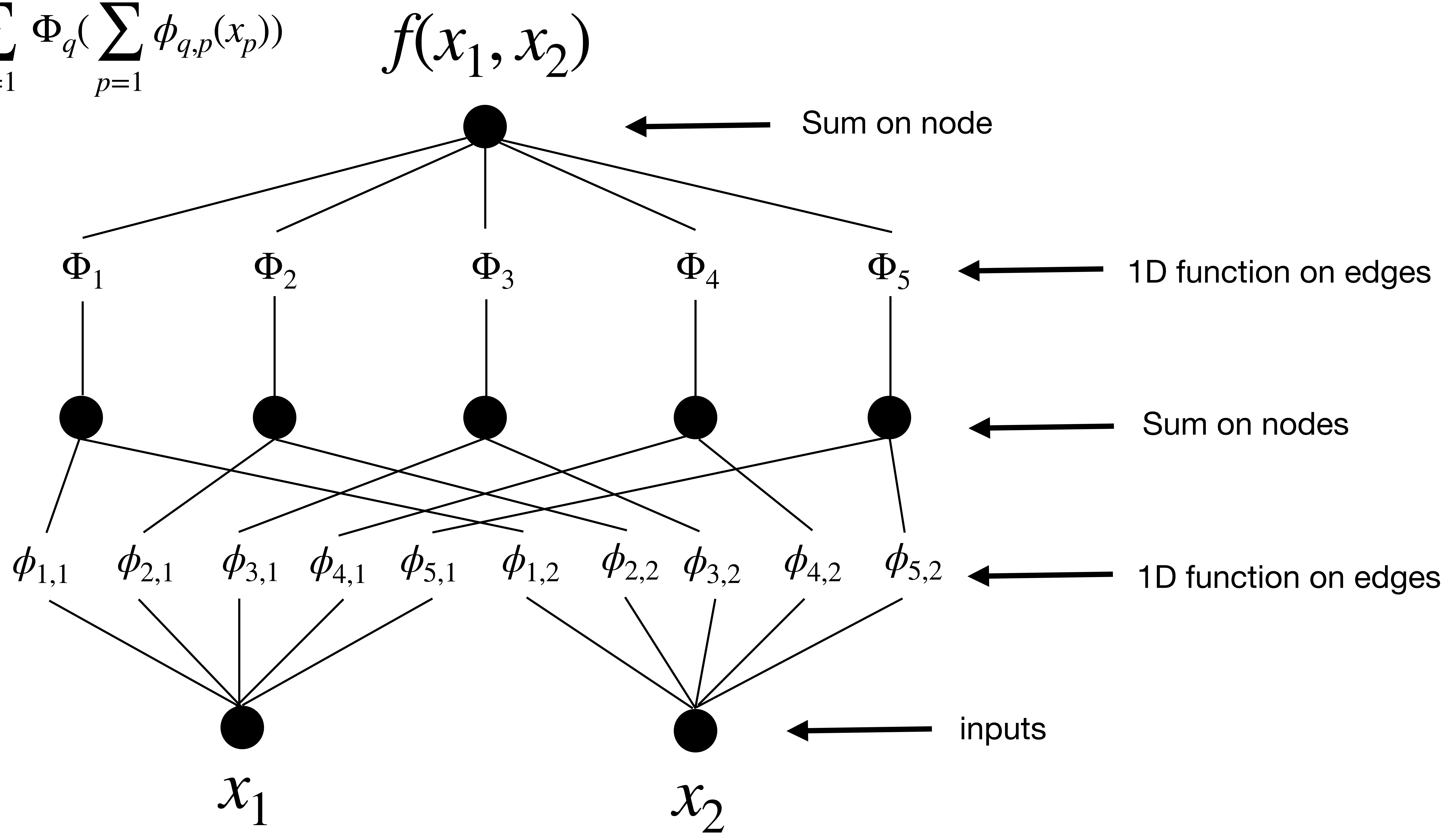
1. Empirically, sometimes we don't care about *exact* representations. *Approximate* ones may suffice.

2. A common wisdom of modern deep learning: go deeper! Even under the smooth constraint, deep nets can be more expressive.

3. The KAN-do spirit: let's just do it and see how it works! KANs may not work for the *worst* cases, but may work for *typical* cases.

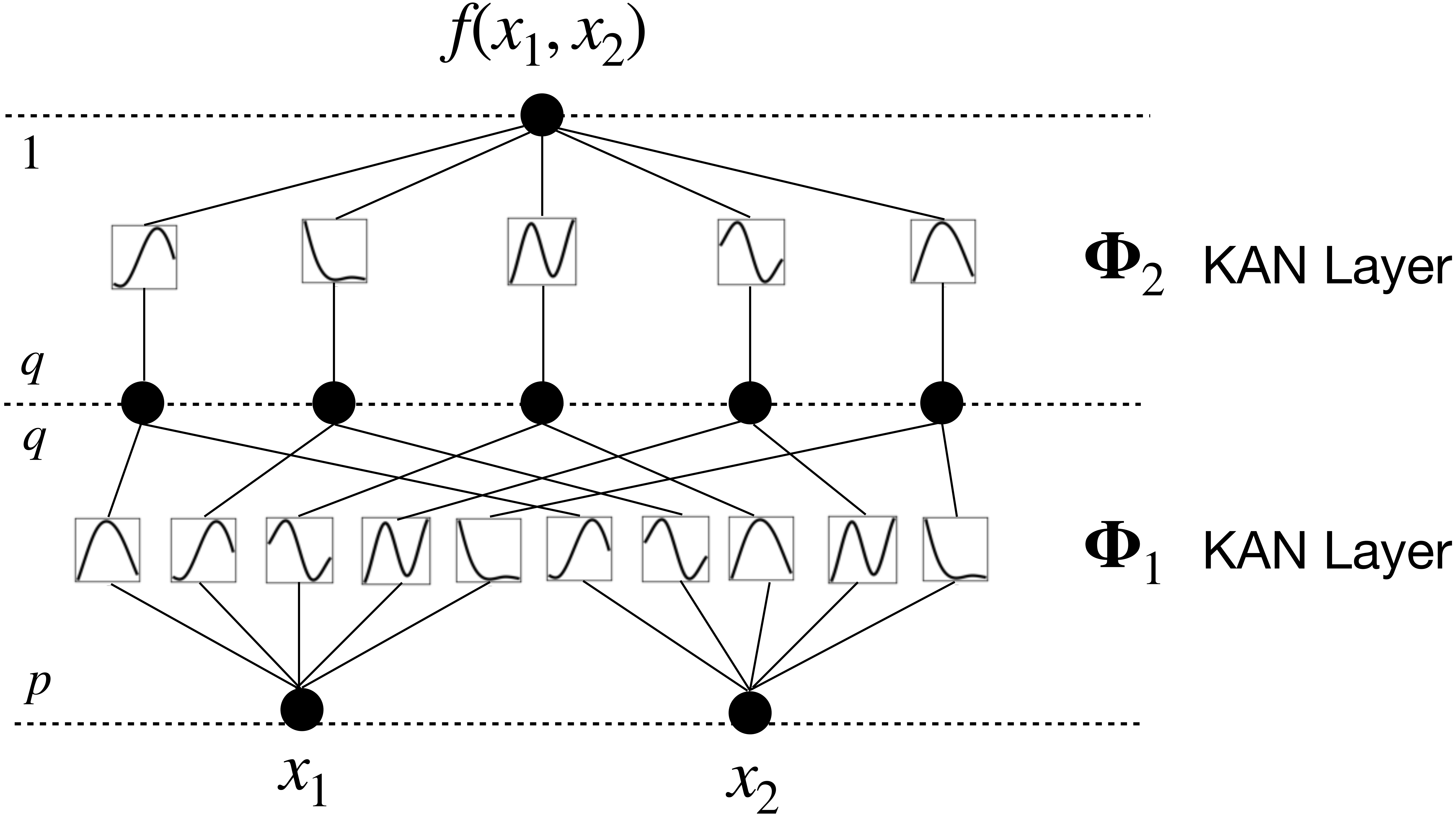
Intuitive picture of KART

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

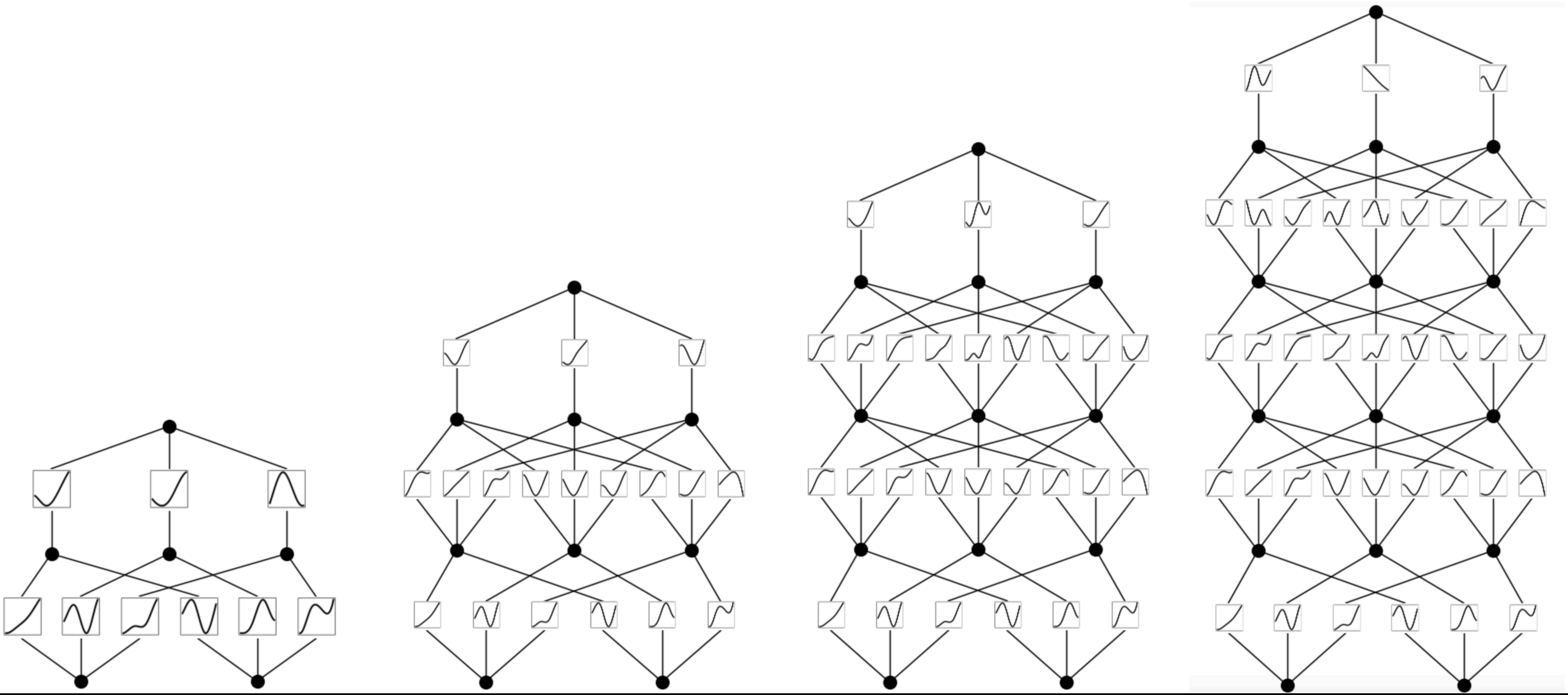


Defining KAN Layers

$$\sum \square_{1,q} \sum \square_{q,p}$$



Going deep: simply stack more layers!



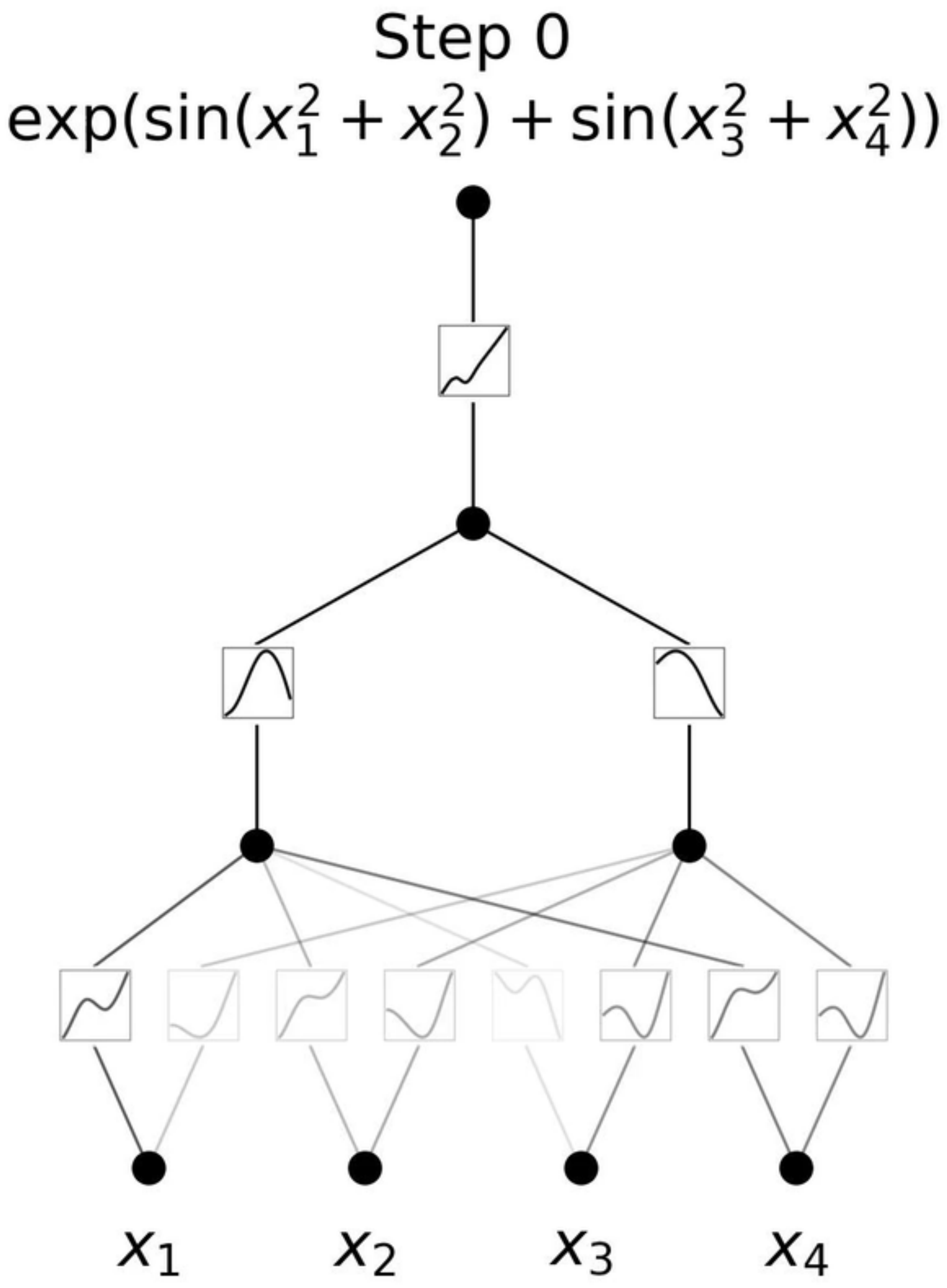
A concrete example

***We**:*
Functions may need more compositions, e.g.,

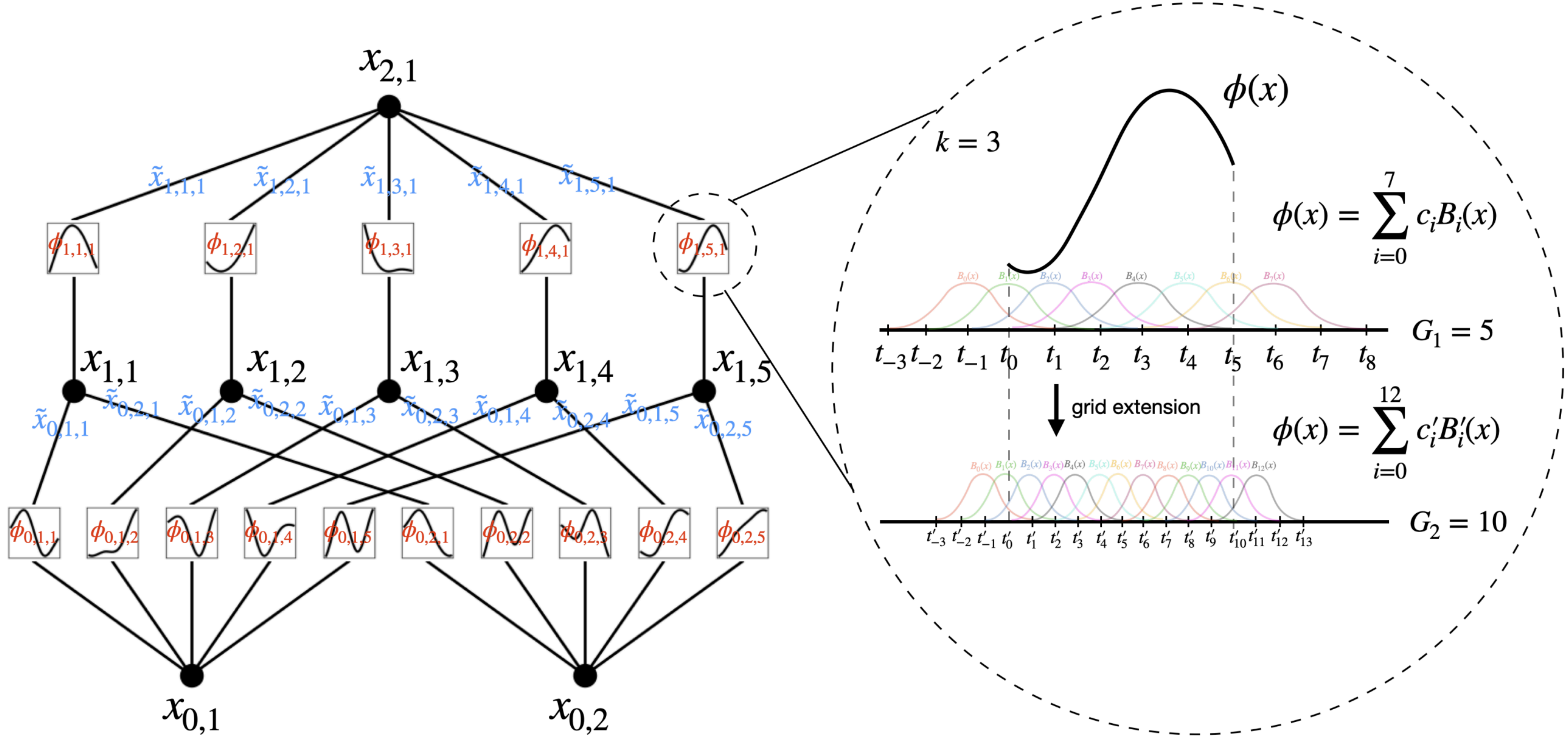
$$f(x_1, x_2, x_3, x_4) = \exp(\sin(x_1^2 + x_2^2) + \sin(x_3^2 + x_4^2))$$

needs three layers.

Giros & Poggio (1989):
For the original two-layer constructions, inner functions can be very pathological.

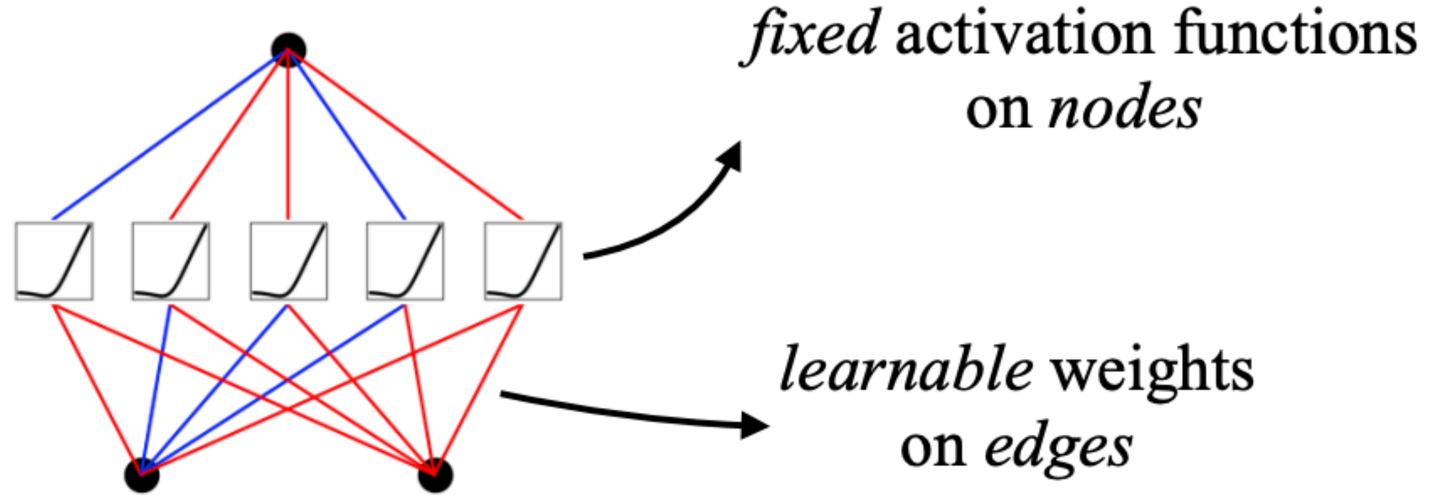
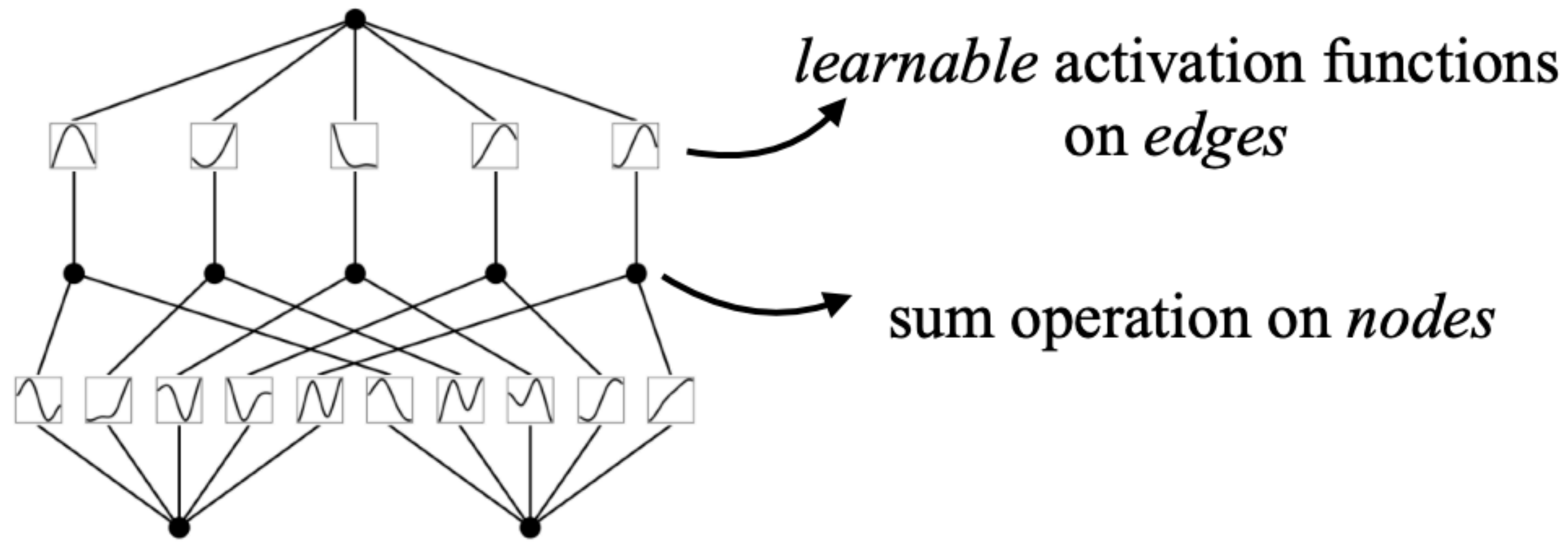
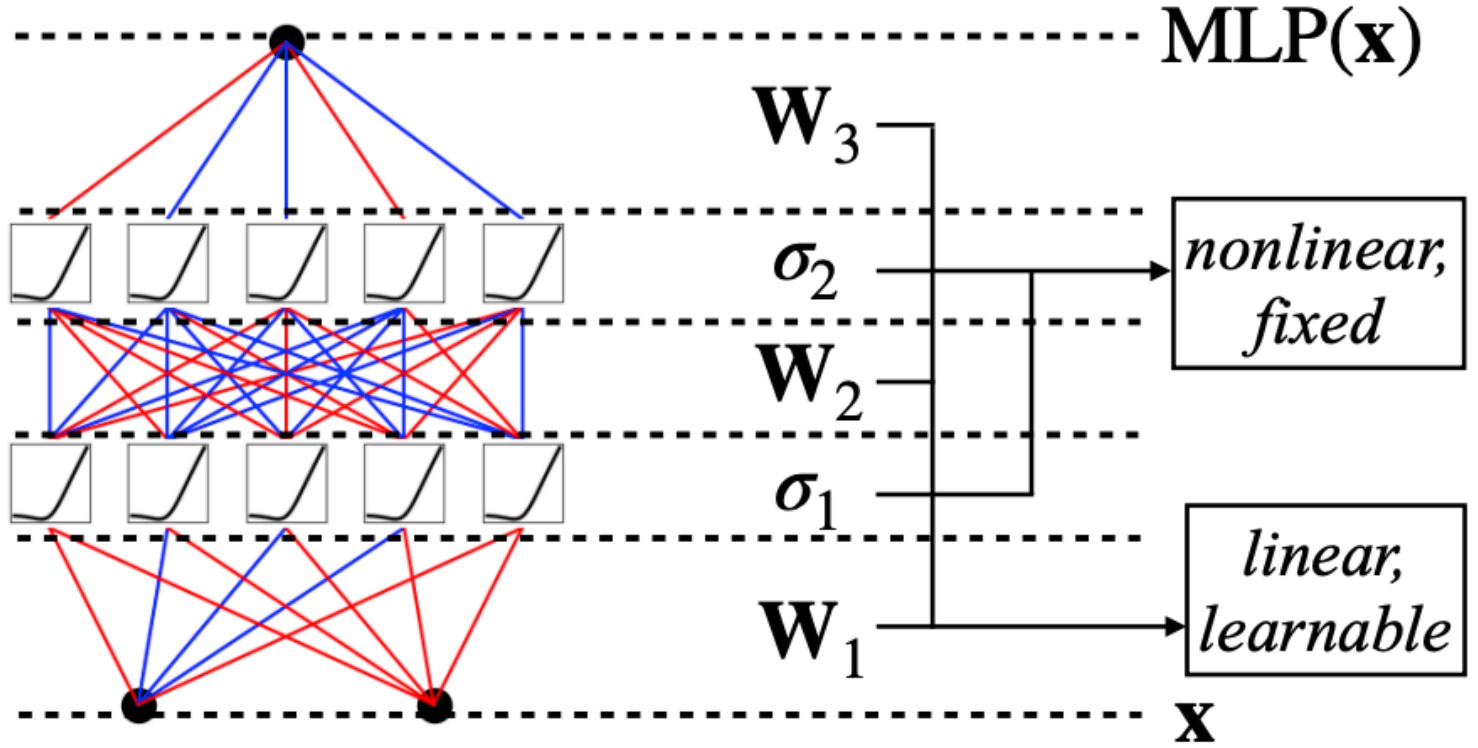
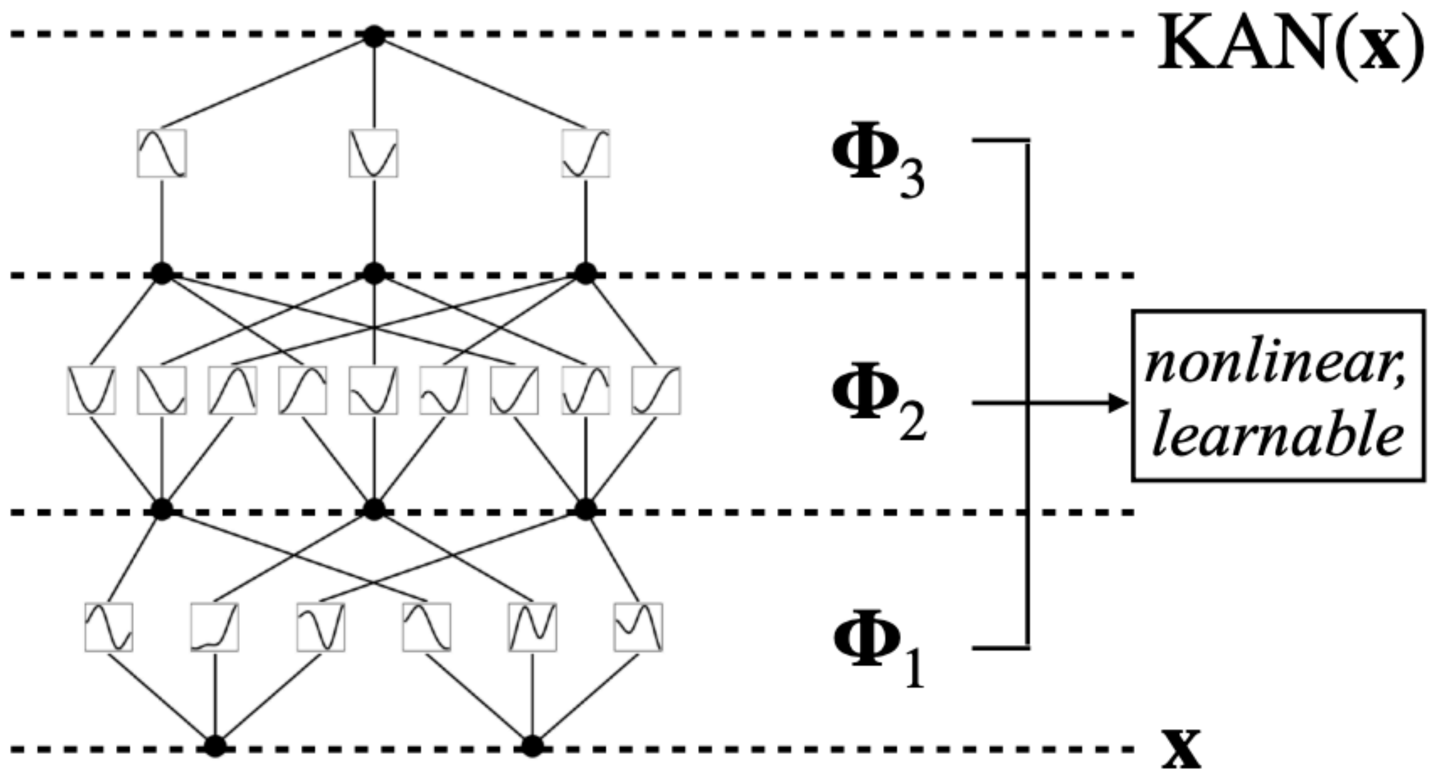


B-splines



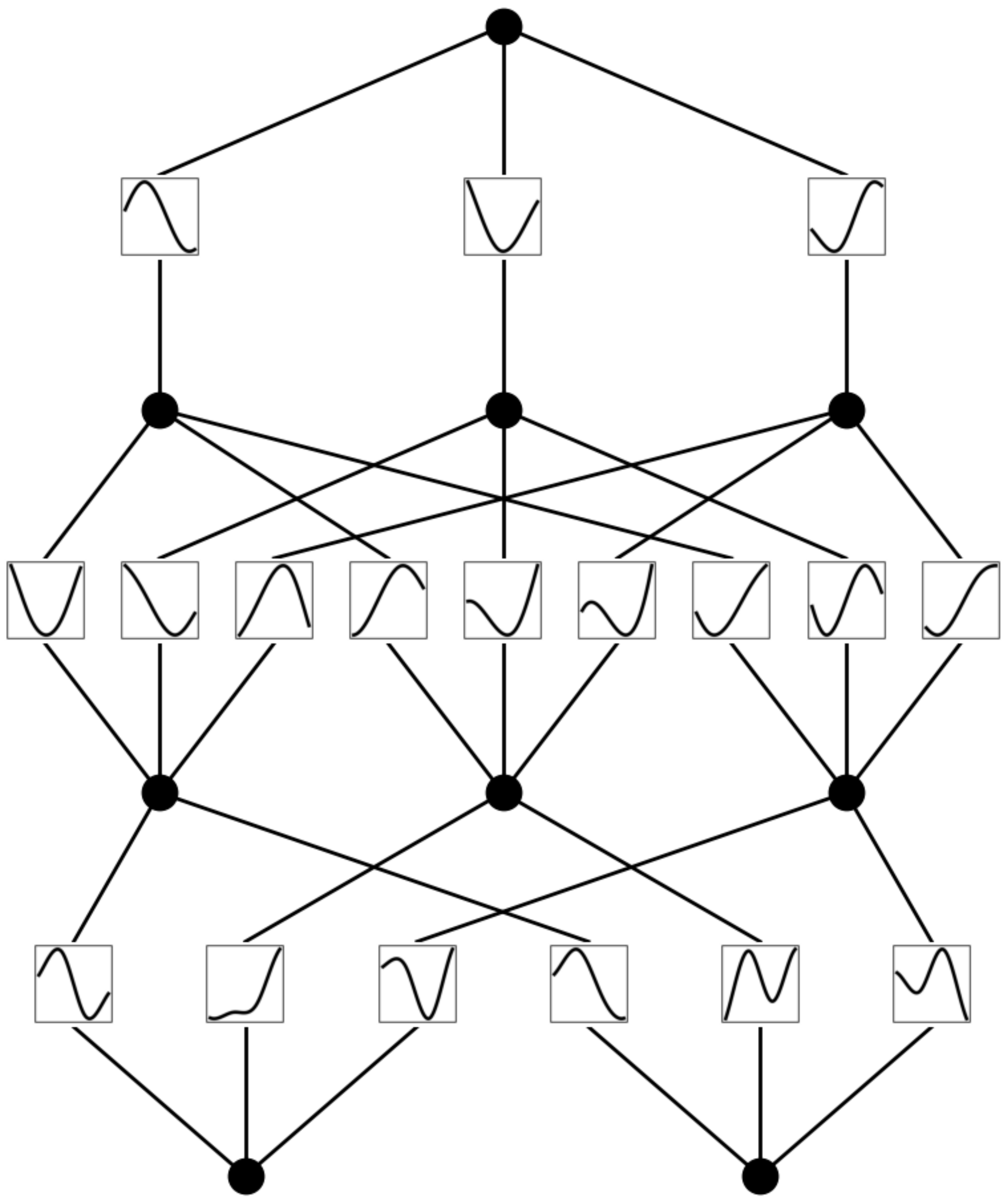
$x_{2,1}$
$\tilde{x}_{1,1,1}$ $\tilde{x}_{1,2,1}$ $\tilde{x}_{1,3,1}$ $\tilde{x}_{1,4,1}$ $\tilde{x}_{1,5,1}$
$\phi_{1,1,1}$ $\phi_{1,2,1}$ $\phi_{1,3,1}$ $\phi_{1,4,1}$ $\phi_{1,5,1}$
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$
$\tilde{x}_{0,1,1}$ $\tilde{x}_{0,2,1}$ $\tilde{x}_{0,1,2}$ $\tilde{x}_{0,2,2}$ $\tilde{x}_{0,1,3}$ $\tilde{x}_{0,2,3}$ $\tilde{x}_{0,1,4}$ $\tilde{x}_{0,2,4}$ $\tilde{x}_{0,1,5}$ $\tilde{x}_{0,2,5}$
$\phi_{0,1,1}$ $\phi_{0,1,2}$ $\phi_{0,1,3}$ $\phi_{0,1,4}$ $\phi_{0,1,5}$ $\phi_{0,2,1}$ $\phi_{0,2,2}$ $\phi_{0,2,3}$ $\phi_{0,2,4}$ $\phi_{0,2,5}$
$x_{0,1}$ $x_{0,2}$
$k = 3$
$\phi(x)$
$\phi(x) = \sum_{i=0}^{7} c_i B_i(x)$
$G_1 = 5$
t_{-3} t_{-2} t_{-1} t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8
grid extension
$\phi(x) = \sum_{i=0}^{12} c_i' B_i'(x)$
$G_2 = 10$
t'_{-3} t'_{-2} t'_{-1} t'_0 t'_1 t'_2 t'_3 t'_4 t'_5 t'_6 t'_7 t'_8 t'_9 t'_{10} t'_{11} t'_{12} t'_{13}

MLP & KAN are *dual*

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)** 

Scaling

$$\boxed{n - \text{dim function} = \text{poly}(n)\ 1 - \text{dim functions}}$$



Scaling

$$\ell \propto N^{-\alpha}$$

approximation error *number of parameters* *scaling exponent*

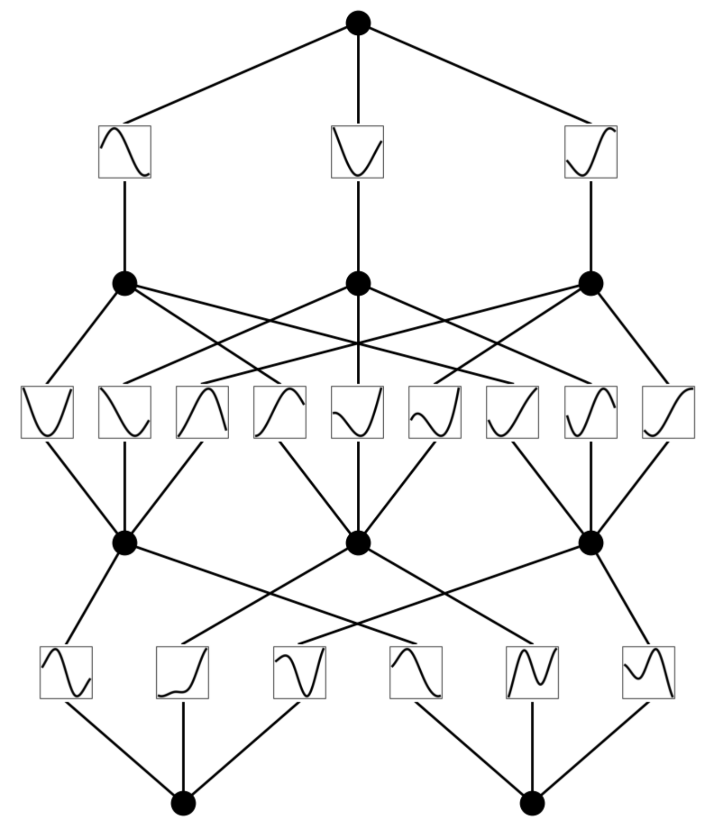
For a function (n dimensions), in general (order k splines on uniform grids):

$$\ell \propto N^{\boxed{-(k+1)/n}}$$

For a function (n dimensions) smoothly represented as a KAN*:

$$\ell \propto \mathrm{poly}(n) N^{\boxed{-(k+1)}}$$

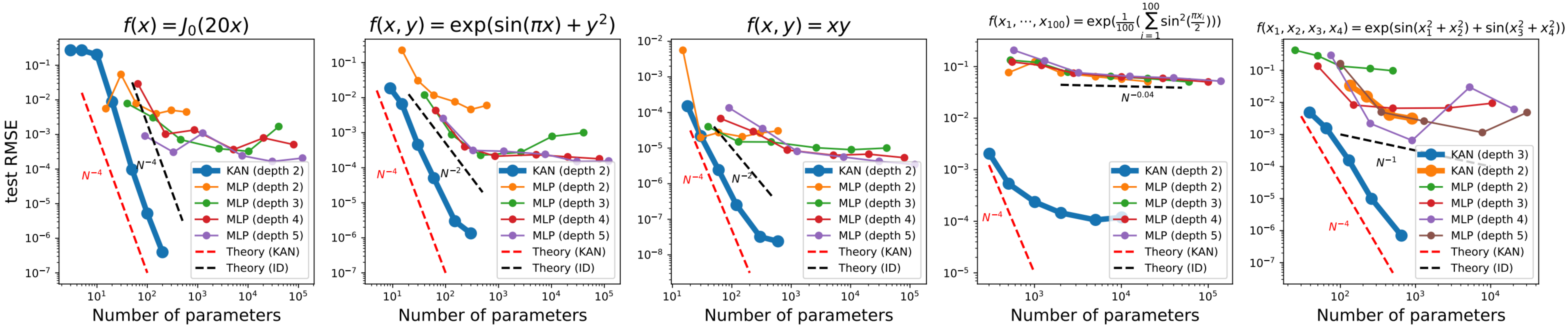
Which is equivalent to $n = 1$, because of KART.



*Informally (Lai-Shen 2021) such functions are dense in C[0,1]

Symbolic formulas

$$\ell \propto N^{-(k+1)}, k = 3$$



😃 Avoid curse of dimensionality (at least for these simples cases)!

🤔 But with caveats (the 1989 paper has made some valuable points):

(1) There might not exist *finite smooth* KA representations, even we now allow deeper ones.

(2) Even if they exist, our learning algorithms may not find these solutions.

Solving PDE

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.

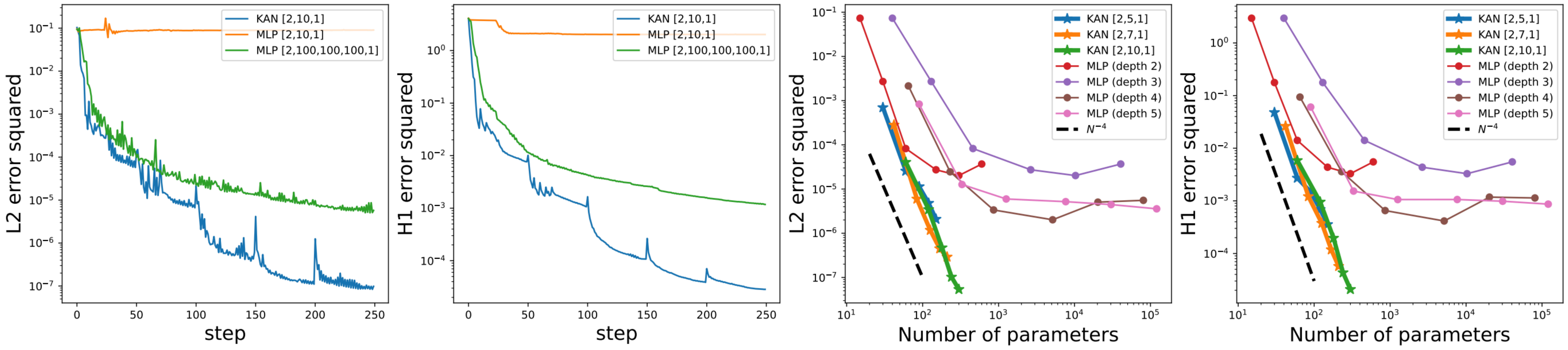


Image Fitting

KANs can scale and speed up

Problem	Model	PSNR / L2² Error	Training Time (s)
Image Fitting	KAN [2,128,128,128,128,1], G=[100,10,10,10,10]	45.76	1809
Image Fitting	MLP [2,404,404,404,404,1]	22.09	182
Image Fitting	SIREN 1 [2,128,128,128,128,1]	27.34	254
Image Fitting	SIREN 2 [2,404,404,404,404,1]	30.79	407
Image Fitting	MLP_RFF [2,404,404,404,404,1]	26.26	195
Allen-Cahn	KAN [2,5,5,1], G=5	3.4×10^{-3}	2801
Allen-Cahn	MLP [2,128,128,128,1]	1.5×10^{-1}	478
Allen-Cahn	MLP [2,128,128,128,1] (10x training)	3.9×10^{-4}	4766
Darcy Flow	KAN [2,10,1], G=20	3.9×10^{-4}	66
Darcy Flow	KAN [2,100,1], G=10	4.3×10^{-6}	107
Darcy Flow	KAN [2,10,10,10,10,1], G=5	8.5×10^{-5}	123
Darcy Flow	MLP [2,128,128,128,1]	3.0×10^{-5}	30
Darcy Flow	MLP [2,128,128,128,1] (10x training)	4.5×10^{-6}	277
Darcy Flow	MLP_RFF [2,128,128,128,1]	5.9×10^{-6}	31

A toy example: symbolic regression

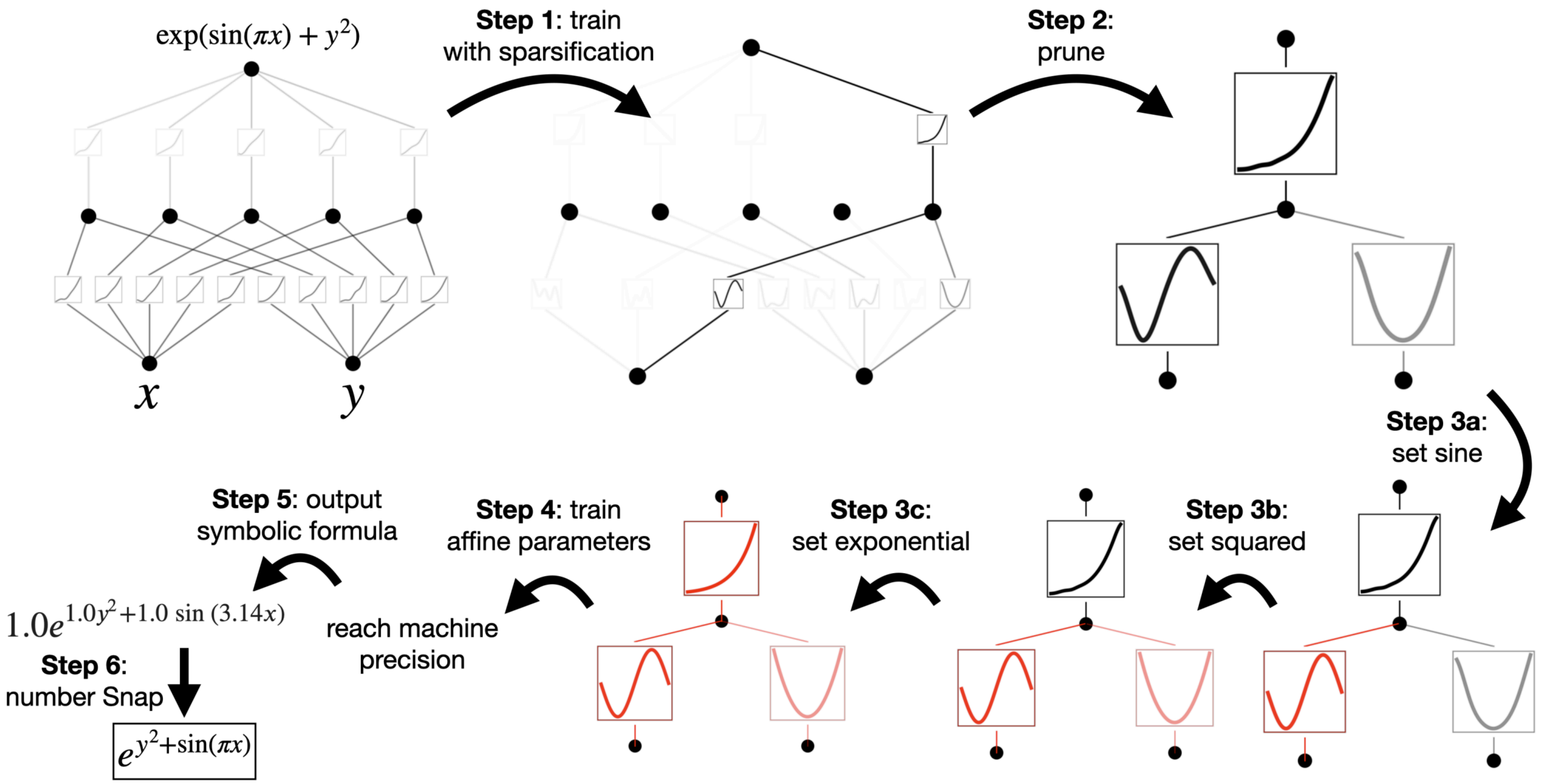


Figure 2.4: An example of how to do symbolic regression with KAN.

KAN 2.0: three levels of interp, both ways (Liu 2024)