KANs and DeepOKANs

Brief Introduction

MAS

IISc, Bengaluru

August 26, 2024

1. Introduction to Data Fitting
2. Introduction to KAN
3. KAT
4. Results

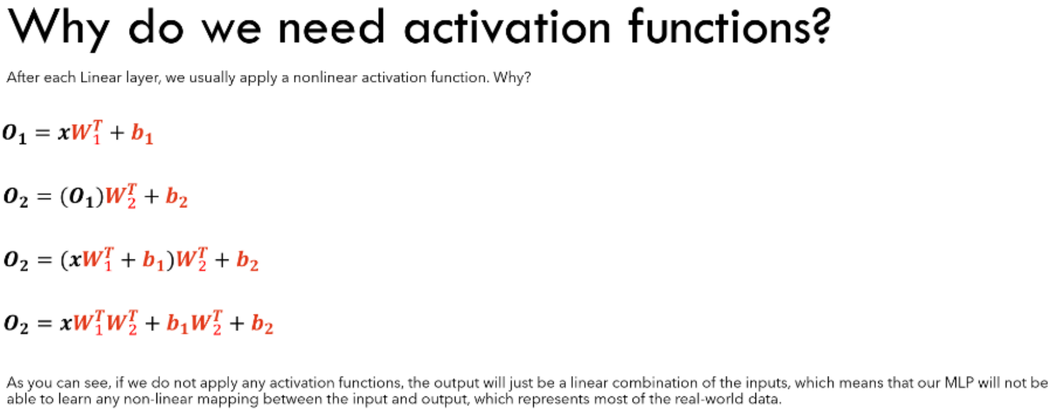
1. Introduction to Data Fitting

2. Introduction to KAN

3. KAT

4. Results

Representing Smooth Functions



Why do we need activation functions?

After each Linear layer, we usually apply a nonlinear activation function. Why?

$$O_1 = xW_1^T + b_1$$

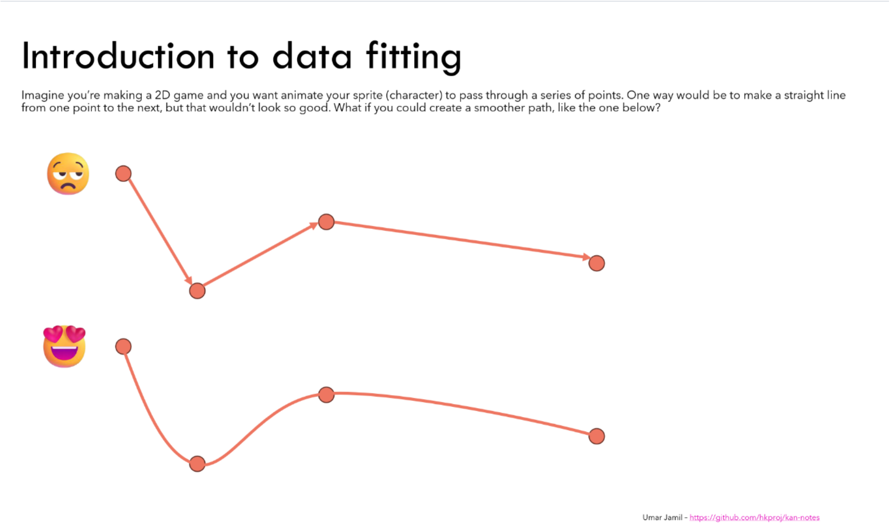
$$O_2 = (O_1)W_2^T + b_2$$

$$O_2 = (xW_1^T + b_1)W_2^T + b_2$$

$$O_2 = xW_1^TW_2^T + b_1W_2^T + b_2$$

As you can see, if we do not apply any activation functions, the output will just be a linear combination of the inputs, which means that our MLP will not be able to learn any non-linear mapping between the input and output, which represents most of the real-world data.

Representing Smooth Functions

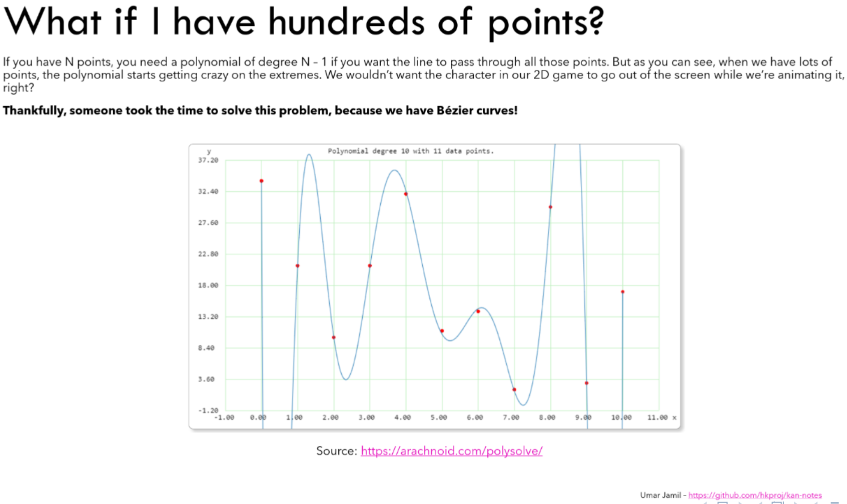


Introduction to data fitting

Imagine you're making a 2D game and you want animate your sprite (character) to pass through a series of points. One way would be to make a straight line from one point to the next, but that wouldn't look so good. What if you could create a smoother path, like the one below?

Umar Jamil - https://github.com/hkproj/kan-notes

Representing Smooth Functions



What if I have hundreds of points?

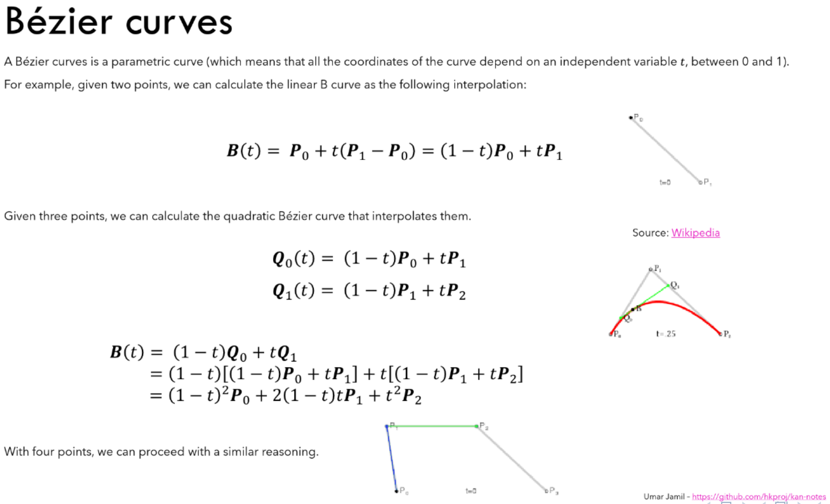
If you have N points, you need a polynomial of degree N - 1 if you want the line to pass through all those points. But as you can see, when we have lots of points, the polynomial starts getting crazy on the extremes. We wouldn't want the character in our 2D game to go out of the screen while we're animating it, right?

Thankfully, someone took the time to solve this problem, because we have Bézier curves!

Source: https://arachnoid.com/polysolve/

Umar Jamil - https://github.com/hkproj/kan-notes

Representing Smooth Functions



Bézier curves

A Bézier curves is a parametric curve (which means that all the coordinates of the curve depend on an independent variable t, between 0 and 1).

For example, given two points, we can calculate the linear B curve as the following interpolation:

$$\boldsymbol{B}(t) = \boldsymbol{P}_0 + t(\boldsymbol{P}_1 - \boldsymbol{P}_0) = (1-t)\boldsymbol{P}_0 + t\boldsymbol{P}_1$$

Given three points, we can calculate the quadratic Bézier curve that interpolates them.

Source: Wikipedia

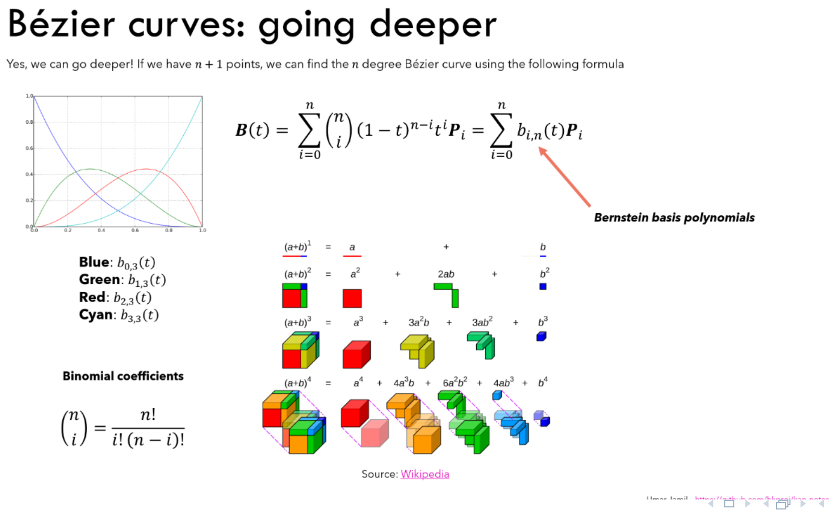
$$\boldsymbol{Q}_0(t) = (1-t)\boldsymbol{P}_0 + t\boldsymbol{P}_1$$
$$\boldsymbol{Q}_1(t) = (1-t)\boldsymbol{P}_1 + t\boldsymbol{P}_2$$

$$\begin{aligned}\boldsymbol{B}(t) &= (1-t)\boldsymbol{Q}_0 + t\boldsymbol{Q}_1 \\ &= (1-t)[(1-t)\boldsymbol{P}_0 + t\boldsymbol{P}_1] + t[(1-t)\boldsymbol{P}_1 + t\boldsymbol{P}_2] \\ &= (1-t)^2\boldsymbol{P}_0 + 2(1-t)t\boldsymbol{P}_1 + t^2\boldsymbol{P}_2\end{aligned}$$

With four points, we can proceed with a similar reasoning.

Umar Jamil - https://github.com/hkproj/kan-notes

Representing Smooth Functions



Bézier curves: going deeper

Yes, we can go deeper! If we have $n+1$ points, we can find the n degree Bézier curve using the following formula

$$\boldsymbol{B}(t)=\sum_{i=0}^{n}\binom{n}{i}(1-t)^{n-i}t^{i}\boldsymbol{P}_i=\sum_{i=0}^{n}b_{i,n}(t)\boldsymbol{P}_i$$

Bernstein basis polynomials

Blue: $b_{0,3}(t)$
Green: $b_{1,3}(t)$
Red: $b_{2,3}(t)$
Cyan: $b_{3,3}(t)$

Binomial coefficients

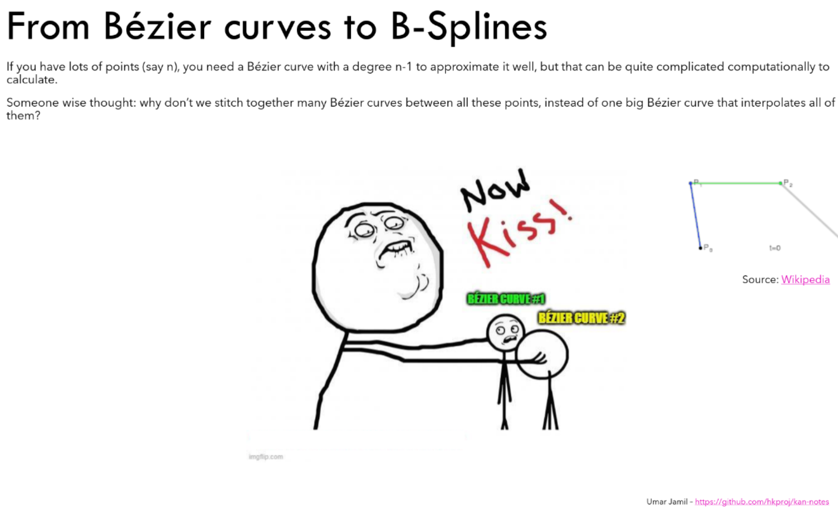
$$\binom{n}{i}=\frac{n!}{i!\,(n-i)!}$$

Source: Wikipedia

Umar Jamil https://github.com/hkproj/kan-notes

0.0
0.2
0.4
0.6
0.8
1.0
(a+b)^1 = a + b
(a+b)^2 = a^2 + 2ab + b^2
(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3
(a+b)^4 = a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4

Representing Smooth Functions



From Bézier curves to B-Splines

If you have lots of points (say n), you need a Bézier curve with a degree n-1 to approximate it well, but that can be quite complicated computationally to calculate.

Someone wise thought: why don't we stitch together many Bézier curves between all these points, instead of one big Bézier curve that interpolates all of them?

Source: Wikipedia

Umar Jamil - https://github.com/hkproj/kan-notes

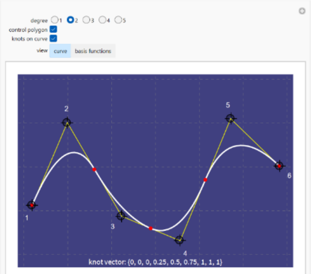
Representing Smooth Functions

B-splines in detail

A k-degree B-Spline curve that is defined by n control points, will consist of $n - k$ Bézier curves.
For example, if we want to use a quadratic Bézier curve and we have 6 points, we need $6 - 2 = 4$ Bézier curves.

In this case we have n=6 and k=2

B-Spline Curve with Knots



degree ○1 ●2 ○3 ○4 ○5
control polygon ☑
knots on curve ☑
view curve basis functions

knot vector: {0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1}

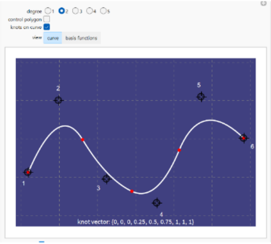
Source: Wolfram Alpha

Umar Jamil - https://github.com/hkproj/kan-notes

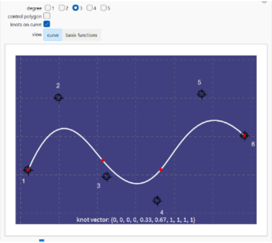
Representing Smooth Functions

B-splines in detail

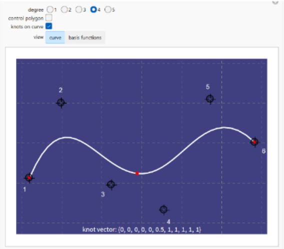
The degree of our B-Spline also tells what kind of continuity we get.


degree ○1 ●2 ○3 ○4 ○5
control polygon □
knots on curve ☑
view curve basis functions

knot vector: {0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1}


degree ○1 ○2 ●3 ○4 ○5
control polygon □
knots on curve ☑
view curve basis functions

knot vector: {0, 0, 0, 0, 0.33, 0.67, 1, 1, 1, 1}


degree ○1 ○2 ○3 ●4 ○5
control polygon □
knots on curve ☑
view curve basis functions

knot vector: {0, 0, 0, 0, 0, 0.5, 1, 1, 1, 1, 1}

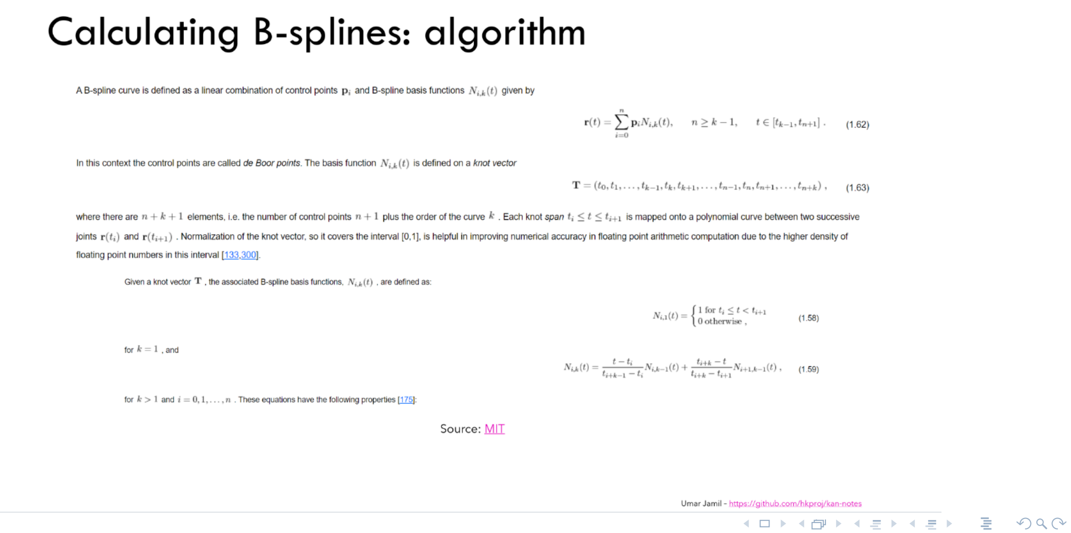
1.4.1 B-splines

An order k B-spline is formed by joining several pieces of polynomials of degree $k-1$ with at most C^{k-2} continuity at the breakpoints.

Source: MIT

Umar Jamil - https://github.com/hkproj/kan-notes

Representing Smooth Functions



Calculating B-splines: algorithm

A B-spline curve is defined as a linear combination of control points $\mathbf{p}_i$ and B-spline basis functions $N_{i,k}(t)$ given by

$$\mathbf{r}(t) = \sum_{i=0}^{n} \mathbf{p}_i N_{i,k}(t), \quad n \geq k-1, \quad t \in [t_{k-1}, t_{n+1}] . \tag{1.62}$$

In this context the control points are called *de Boor points*. The basis function $N_{i,k}(t)$ is defined on a *knot vector*

$$\mathbf{T} = (t_0, t_1, \ldots, t_{k-1}, t_k, t_{k+1}, \ldots, t_{n-1}, t_n, t_{n+1}, \ldots, t_{n+k}) , \tag{1.63}$$

where there are $n+k+1$ elements, i.e. the number of control points $n+1$ plus the order of the curve k . Each knot *span* $t_i \leq t \leq t_{i+1}$ is mapped onto a polynomial curve between two successive joints $\mathbf{r}(t_i)$ and $\mathbf{r}(t_{i+1})$. Normalization of the knot vector, so it covers the interval [0,1], is helpful in improving numerical accuracy in floating point arithmetic computation due to the higher density of floating point numbers in this interval [133,300].

Given a knot vector $\mathbf{T}$, the associated B-spline basis functions, $N_{i,k}(t)$, are defined as:

$$N_{i,1}(t) = \begin{cases} 1 \text{ for } t_i \leq t < t_{i+1} \\ 0 \text{ otherwise} , \end{cases} \tag{1.58}$$

for $k = 1$, and

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t) , \tag{1.59}$$

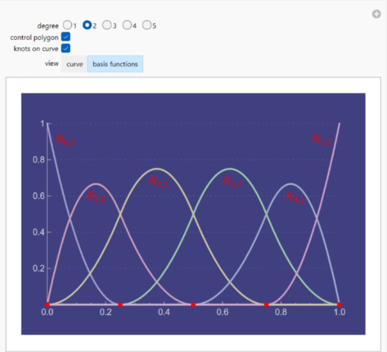
for $k > 1$ and $i = 0, 1, \ldots, n$. These equations have the following properties [175]:

Source: MIT

Umar Jamil - https://github.com/hkproj/kan-notes

Representing Smooth Functions

B-Splines: basis functions



degree ○1 ●2 ○3 ○4 ○5

control polygon ☑

knots on curve ☑

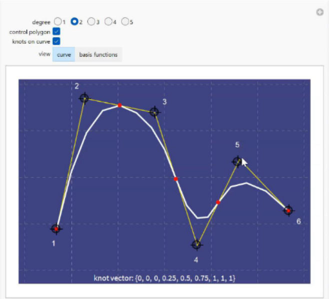
view curve [✓ basis functions]

Umar Jamil - https://github.com/hkproj/kan-notes

Representing Smooth Functions

B-splines: local control

Moving a control point only changes the curve locally (in the proximity of the control point), leaving the adjacent Bezier curves unchanged!



degree ○1 ●2 ○3 ○4 ○5

control polygon ☑

knots on curve ☑

view curve basis functions

knot vector: {0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1}

Umar Jamil - https://github.com/hkproj/kan-notes

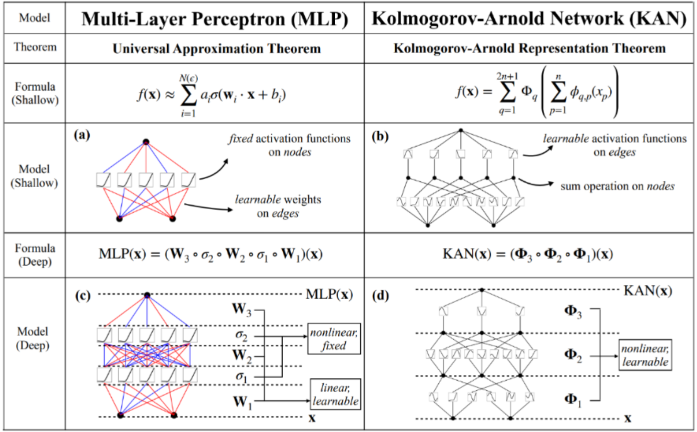
1. Introduction to Data Fitting

2. Introduction to KAN

3. KAT

4. Results

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) *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*

Why KAN and not MLPs?

- **MLPs & Nonlinear Approximation**: MLPs are widely used for nonlinear function approximation due to their expressive power (Universal Approximation Theorem), but they have drawbacks: low interpretability, less accuracy in low dimensions, and the curse of dimensionality (COD) in high dimensions.

- **Introduction of KANs**: The paper introduces Kolmogorov-Arnold Networks (KANs), inspired by the Kolmogorov-Arnold representation theorem, as an alternative to MLPs.

- **Previous Work**: Earlier research focused on depth-2, width-(2n + 1) networks, showing potential in breaking the COD empirically and theoretically, particularly with compositional function structures.

- **New Approach**: This paper proposes deeper and wider network architectures, leveraging backpropagation for training, to enhance performance beyond the traditional depth-2, width-(2n + 1) representation.

KANs are a combo of MLPs and splines

- **Splines**: Accurate for low dimensions, easy to adjust locally and able to switch between different resolutions.(resolutions are the number of knots in the spline) But suffer from COD in high dimensions, unable to learn compositional structures

- **MLPs**: suffer less from COD thanks to their feature learning, but are less accurate than splines in low dimensions, because of their inability to optimize univariate functions.

To learn a function accurately, a model should not only learn the compositional structure (external degrees of freedom), but should also approximate well the univariate functions (internal degrees of freedom).

Solution: KANs = MLPs(learns external dofs) + splines(learns internal dofs)

Brief Idea of Advantages of KANs

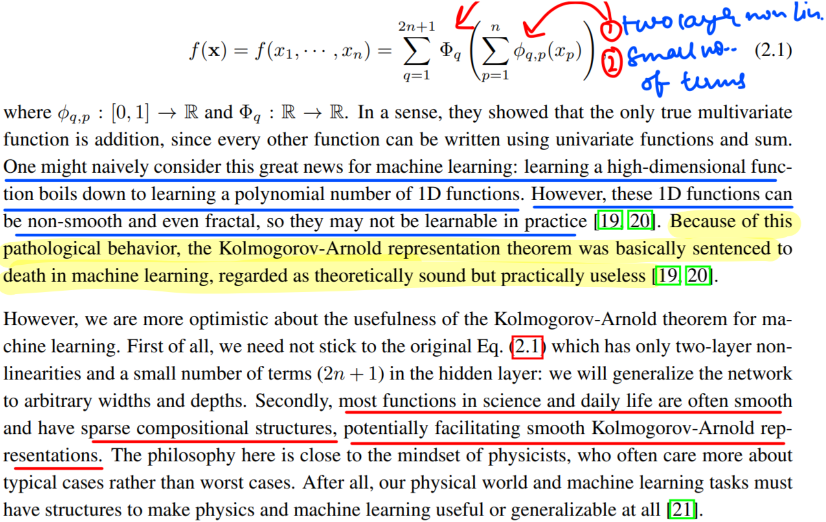
- KANs can lead to accuracy and interpretability improvement over MLPs
- KANs can be made more accurate with grid extension
- KANs can beat the curse of dimensionality when there is a compositional structure in data, achieving better scaling laws than MLPs.
- PDE solutions can be approximated with KANs (Something we might want to research on)
- Two examples from mathematics (knot theory) and physics (Anderson localization) to demonstrate that KANs can be helpful “collaborators” for scientists to (re)discover math and phys- ical laws. **No Background hence skipped**

1. Introduction to Data Fitting
2. Introduction to KAN
3. **KAT**
4. Results

Kolmogorov-Arnold Represenation Theorem

Kolmogorov-Arnold Representation Theorem states 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. More specifically, for a smooth $f : [0,1]^n \to \mathbb{R}$,

KAT



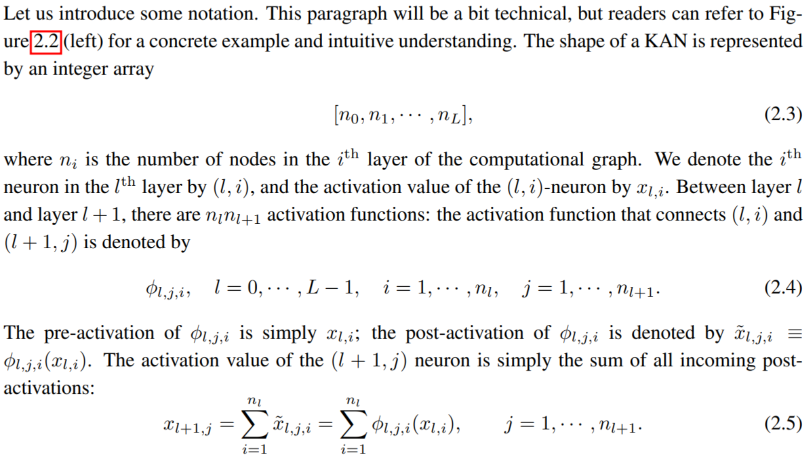
$$f(\mathbf{x}) = f(x_1,\cdots,x_n) = \sum_{q=1}^{2n+1} \Phi_q\left(\sum_{p=1}^{n}\phi_{q,p}(x_p)\right) \tag{2.1}$$

① two layer non lin.
② small no. of terms

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

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

Architecture



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

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

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

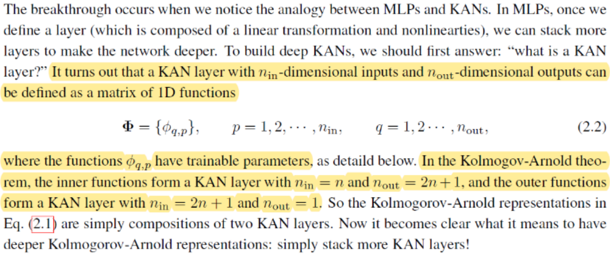
$$\phi_{l,j,i}, \quad l = 0, \cdots, L-1, \quad i = 1, \cdots, n_l, \quad j = 1, \cdots, n_{l+1}. \tag{2.4}$$

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

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

Architecture

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



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

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

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

Layer 2
5 input features, 1 output features
total of 5 functions to "learn"

Layer 1
2 input features, 5 output features
total of 10 functions to "learn"

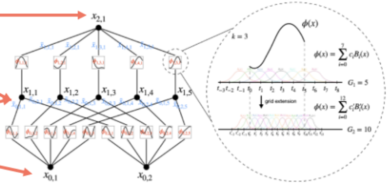
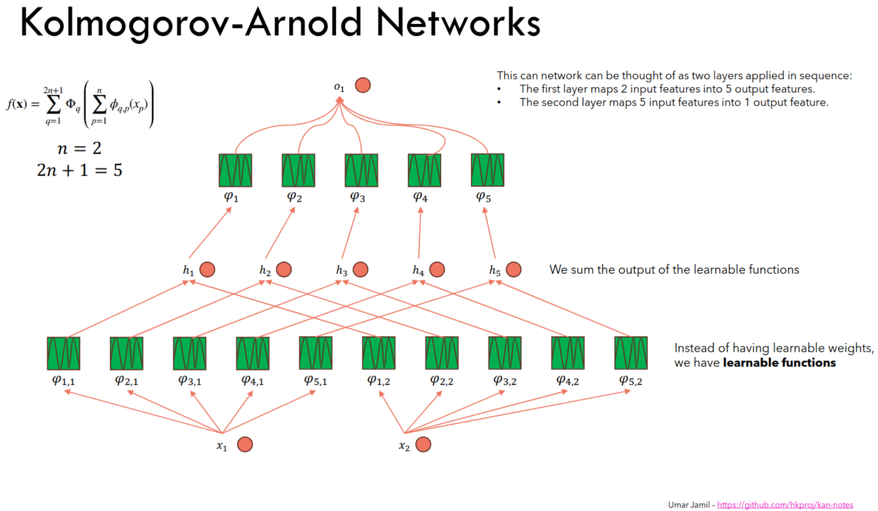


Figure 2.2: Left: Notations of activations that flow through the network. Right: an activation function is parameterized as a B-spline, which allows switching between coarse-grained and fine-grained grids.

Example KAN



Kolmogorov-Arnold Networks

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

$$n = 2$$
$$2n + 1 = 5$$

This can network can be thought of as two layers applied in sequence:

- The first layer maps 2 input features into 5 output features.
- The second layer maps 5 input features into 1 output feature.

We sum the output of the learnable functions

Instead of having learnable weights, we have **learnable functions**

o_1
φ_1 φ_2 φ_3 φ_4 φ_5
h_1 h_2 h_3 h_4 h_5
$\varphi_{1,1}$ $\varphi_{2,1}$ $\varphi_{3,1}$ $\varphi_{4,1}$ $\varphi_{5,1}$ $\varphi_{1,2}$ $\varphi_{2,2}$ $\varphi_{3,2}$ $\varphi_{4,2}$ $\varphi_{5,2}$
x_1 x_2

Umar Jamil - https://github.com/hkproj/kan-notes

Example KAN

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

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

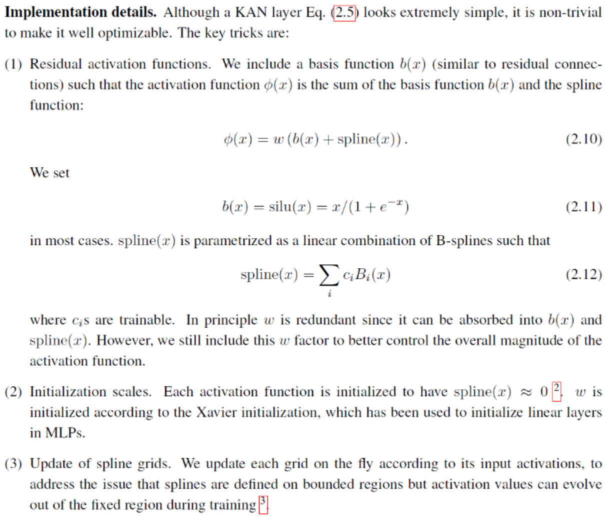
which is quite cumbersome. In contrast, our abstraction of KAN layers and their visualizations are cleaner and intuitive. The original Kolmogorov-Arnold representation Eq. (2.1) corresponds to a 2-Layer KAN with shape $[n, 2n+1, 1]$. Notice that all the operations are differentiable, so we can train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

Ax+b

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

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

Implementation



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

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

$$\phi(x) = w\left(b(x) + \text{spline}(x)\right). \tag{2.10}$$

We set

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

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

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

where c_is are trainable. In principle w is redundant since it can be absorbed into $b(x)$ and $\text{spline}(x)$. However, we still include this w factor to better control the overall magnitude of the activation function.

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

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

Implementation

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

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

Parameter Count

For a network of:

- depth L
- all layers of equal width N
- each spline of order k on a grid of size $G+1$ i.e. G intervals

The total number of parameters is: $O(N^2L(G+k))$ $O(N^2L(G))$.

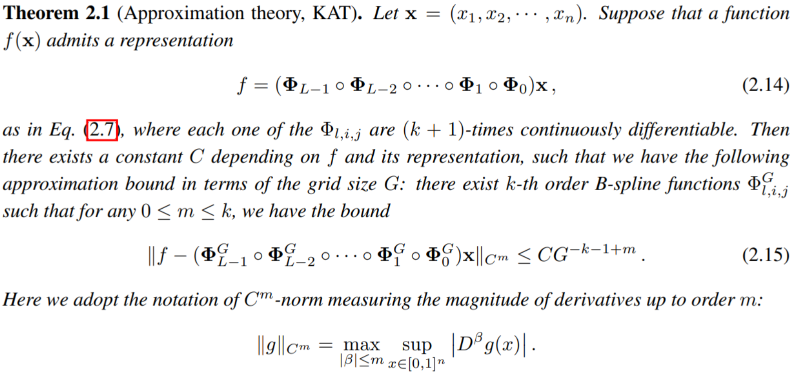
In contrast, MLP has $O(N^2L)$ parameters.

Well good news is N is pretty low for KANs compared to MLPs.

For 1D problems, $N = L = 1$ i.e. nothing but a spline approximation.

For higher dimensions, generalization behaviour of KANs is stated in the theorem below.

Approximation Theory, KAT



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

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

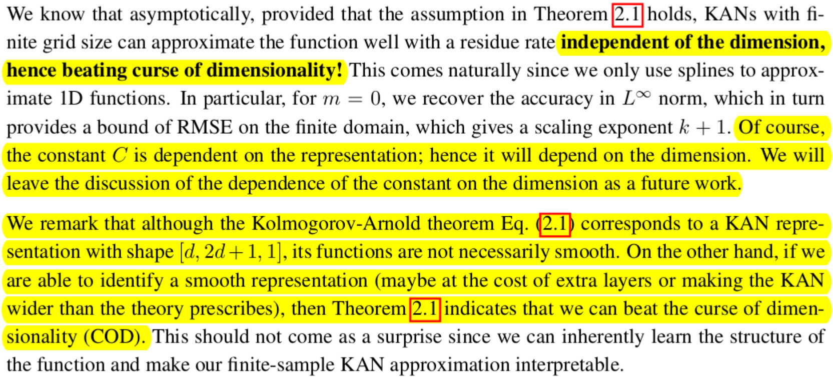
as in Eq. (2.7), *where each one of the* $\Phi_{l,i,j}$ *are* $(k+1)$*-times continuously differentiable. Then there exists a constant* C *depending on* f *and its representation, such that we have the following approximation bound in terms of the grid size* G*: there exist* k*-th order B-spline functions*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

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

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

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

Inferences from Approximation Theory



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

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

Neural Scaling Laws

Neural Scaling Laws: The test loss ℓ decreases with more model parameters N following the law $\ell \propto N^{-\alpha}$, where α is the scaling exponent. A larger α indicates better improvement by scaling the model.

Theories for Predicting α: Sharma & Kaplan:Suggest that α is related to the intrinsic dimensionality d of the input manifold.

- **Standard Approximation Theory:** For a piecewise polynomial of order k, $\alpha = \frac{k+1}{d}$. This suffers from the curse of dimensionality.

Alternative Approaches:

- **Michaud et al.:** Found $\alpha = \frac{k+1}{d^*}$, with $d^* = 2$ being the maximum arity. They considered only unary and binary functions.
- **Poggio et al.:** Showed that for function classes W_m with continuous derivatives up to order m, $\alpha = \frac{m}{2}$.

Neural Scaling Laws

Approaches for (KANs):

- **Proposed Approach:** Decomposes high-dimensional functions into several 1D functions, resulting in $\alpha = k+1$, with $k = 3$ for cubic splines, yielding $\alpha = 4$.
- **Empirical Validation:** Section 3.1 confirms $\alpha = 4$ for KANs, while traditional MLPs have shown slower bounds, like $\alpha = 1$.

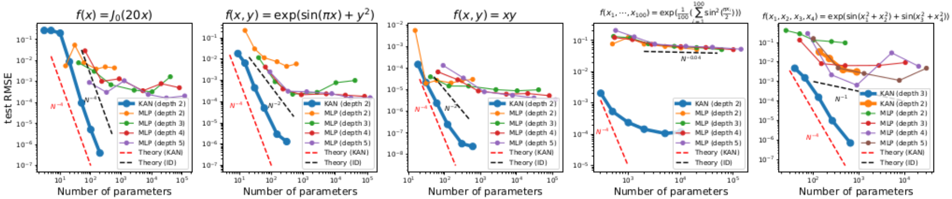


Figure 3.1: Compare KANs to MLPs on five toy examples. KANs can almost saturate the fastest scaling law predicted by our theory ($\alpha = 4$), while MLPs scales slowly and plateau quickly.

KAT vs UAT

Universal Approximation Theorem (UAT):

- Justifies the power of fully-connected neural networks, stating that a two-layer network with $k > N(\epsilon)$ neurons can approximate any function within an error ϵ.
- However, UAT does not provide a bound on how $N(\epsilon)$ scales with ϵ, leading to COD.
- In some cases, N grows exponentially with the dimensionality d.

Kolmogorov-Arnold Theorem (KAT) vs UAT:

- KAT leverages the intrinsic low-dimensional representation of functions, unlike MLPs under UAT.
- KAT allows for quantifying approximation errors in compositional spaces, offering advantages over UAT.

- Generalization error bounds for finite samples in a similar space have been studied, showing that MLPs with ReLU activations face difficulties in overcoming the COD.
- Nonlinear n-widths theory suggests that MLPs with ReLU activations can achieve a tight rate, but cannot beat the COD in general function spaces like Sobolev or Besov spaces.

Implications of KAT:

- KAT motivates the use of compositional structures for functions commonly encountered in practice and science, helping to overcome the COD.
- This allows the use of smaller architectures in practice, as general nonlinear activation functions are learned.
- In contrast, ReLU MLPs may require a depth of at least $\log n$ to achieve the desired rate, where n is the number of samples.
- KANs are shown to be better aligned with symbolic functions compared to MLPs.

Grid Extension

Fine-Graining in KANs: KANs can achieve arbitrarily accurate approximations by refining the grid, a process called "fine-graining." This is a key advantage over MLPs, which lack the concept of grid refinement.

Efficiency: Unlike MLPs, where increasing width and depth can improve performance but is computationally expensive, KANs can be efficiently enhanced by simply refining the spline grids without retraining from scratch.

How to do Grid Extension

We begin with a coarse grid approximation of a 1D function $f(x)$ defined on the interval $[a, b]$. The grid has G_1 intervals, with grid points $\{t_0 = a, t_1, t_2, \ldots, t_{G_1} = b\}$, and B-spline basis functions of order k, denoted by $B_i(x)$.

The function $f(x)$ on the coarse grid is approximated as:

$$f_{\text{coarse}}(x) = \sum_{i=0}^{G_1+k-1} c_i B_i(x)$$

We then extend this to a finer grid with G_2 intervals, introducing additional grid points $\{t_{G_1+1}, t_{G_1+2}, \ldots, t_{G_2+k}\}$. The new approximation on the fine grid is:

$$f_{\text{fine}}(x) = \sum_{j=0}^{G_2+k-1} c'_j B'_j(x)$$

Note that the i^{th} B-spline i.e. $B_i(x)$ is non zero only in the interval $[t_{-k+i}, t_{i+1}]$.

How to do Grid Extension

The new coefficients $\{c'_j\}$ are determined by minimizing the distance:

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

This can be implemented using the least squares algorithm.

Staircase like loss curves

We use a toy example $f(x,y) = \exp(\sin(\pi x) + y^2)$ to demonstrate the effect of grid extension. In Figure 2.3 (top left), we show the train and test RMSE for a $[2,5,1]$ KAN. The number of grid points starts at 3, increases to a higher value every 200 L-BFGS steps, ending up with 1000 grid points.

It is clear that every time fine graining happens, the training loss drops faster than before (except for the finest grid with 1000 points, where optimization ceases to work probably due to bad loss landscapes). However, the test losses first go down, then go up, displaying a U-shape due to the bias-variance tradeoff (underfitting vs. overfitting).

We conjecture that the optimal test loss is achieved at the interpolation threshold when the number of parameters matches the number of data points. Since our training samples are 1000 and the total parameters of a $[2,5,1]$ KAN is $15G$ (where G is the number of grid intervals), we expect the interpolation threshold to be $G = \frac{1000}{15} \approx 67$, which roughly agrees with our experimentally observed value $G \sim 50$.

Staircase like loss curves

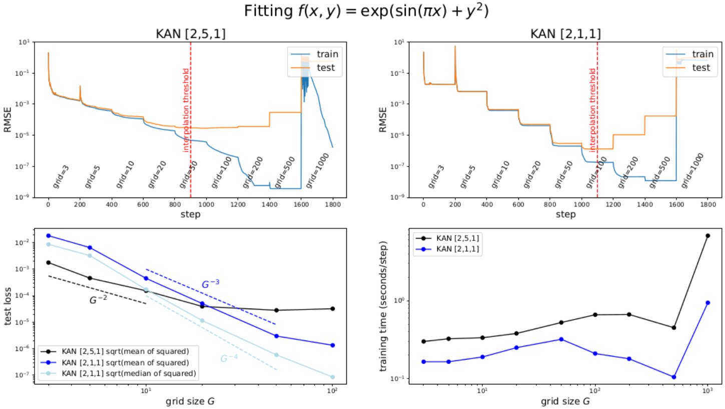


Figure 2.3: We can make KANs more accurate by grid extension (fine-graining spline grids). Top left (right): training dynamics of a $[2, 5, 1]$ ($[2, 1, 1]$) KAN. Both models display staircases in their loss curves, i.e., loss suddently drops then plateaus after grid extension. Bottom left: test RMSE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

1. Introduction to Data Fitting
2. Introduction to KAN
3. KAT
4. **Results**

- different themes
- different themes
- different themes
- different themes

End

MAS