

KANs and DeepOKANs

Brief Introduction

MAS

IISc, Bengaluru

August 25, 2024



- ① Introduction to Data Fitting
- ② Introduction to KAN
- ③ Methods
- ④ Results

- 1 Introduction to Data Fitting
- 2 Introduction to KAN
- 3 Methods
- 4 Results

Representing Smooth Functions

Why do we need activation functions?

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

$$o_1 = xW_1^T + b_1$$

$$o_2 = (o_1)W_2^T + b_2$$

$$o_2 = (xW_1^T + b_1)W_2^T + b_2$$

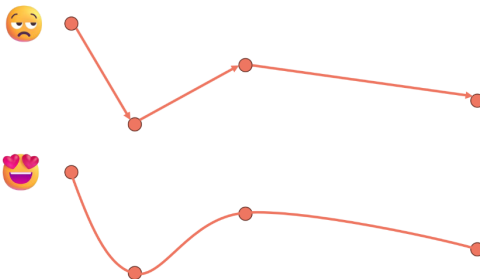
$$o_2 = xW_1^TW_2^T + b_1W_2^T + b_2$$

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?



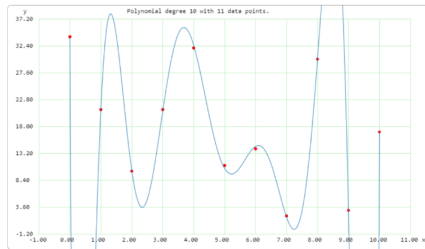
Umar Jamil - <https://github.com/hkproj/kan-notes>

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 curve 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:

$$B(t) = P_0 + t(P_1 - P_0) = (1 - t)P_0 + tP_1$$



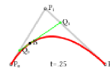
Given three points, we can calculate the quadratic Bézier curve that interpolates them.

$$Q_0(t) = (1 - t)P_0 + tP_1$$

$$Q_1(t) = (1 - t)P_1 + tP_2$$

$$\begin{aligned} B(t) &= (1 - t)Q_0 + tQ_1 \\ &= (1 - t)[(1 - t)P_0 + tP_1] + t[(1 - t)P_1 + tP_2] \\ &= (1 - t)^2P_0 + 2(1 - t)tP_1 + t^2P_2 \end{aligned}$$

Source: [Wikipedia](#)



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



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

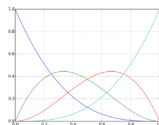
Representing Smooth Functions

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

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

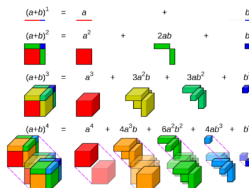
Bernstein basis polynomials



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](#)

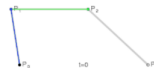
Figure from: [Bernstein Polynomials and Bézier Curves](#)

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](https://en.wikipedia.org/wiki/B%C3%A9zier_curve)

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

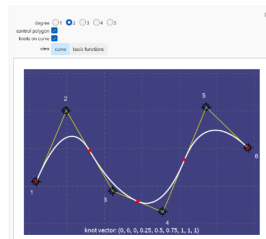
Representing Smooth Functions

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



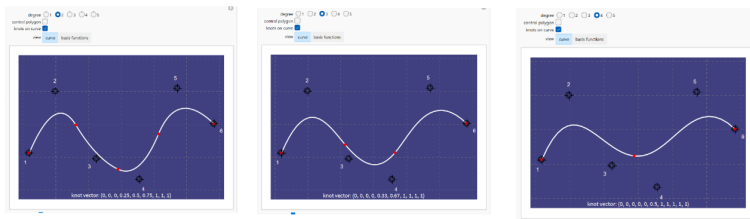
Source: [Wolfram Alpha](#)

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.



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

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

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

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

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 [133,309].

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

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

for $k=1$, and

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

for $k>1$ and $i=0,1,\dots,n$. These equations have the following properties [172]:

Source: [MIT](#)

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

Representing Smooth Functions

B-Splines: basis functions

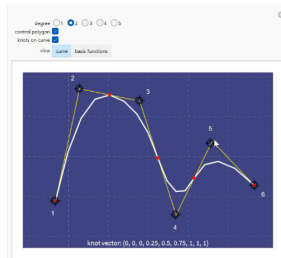


Umar Jamil - <https://github.com/fikriproj/kan-notes>

Representing Smooth Functions

B-splines: local control

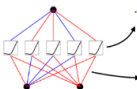

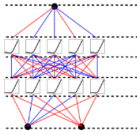
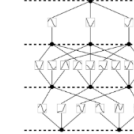
Moving a control point only changes the curve locally (in the proximity of the control point), leaving the adjacent Bezier curves unchanged!



Umar Jamil - <https://github.com/hikproy/kan-notes>

- 1 Introduction to Data Fitting
- 2 Introduction to KAN**
- 3 Methods
- 4 Results

MLP vs KAN

Model	Multi-Layer Perceptron (MLP)	Kolmogorov-Arnold Network (KAN)
Theorem	Universal Approximation Theorem	Kolmogorov-Arnold Representation Theorem
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(e)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^n \phi_{q,p}(x_p) \right)$
Model (Shallow)	(a)  fixed activation functions on nodes learnable weights on edges	(b)  learnable activation functions on edges sum operation on nodes
Formula (Deep)	$\text{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\text{KAN}(\mathbf{x}) = (\Phi_3 \circ \Phi_2 \circ \Phi_1)(\mathbf{x})$
Model (Deep)	(c)  MLP(x) W ₃ sigma ₂ W ₂ sigma ₁ W ₁ x nonlinear, fixed linear, learnable	(d)  KAN(x) Phi ₃ Phi ₂ Phi ₁ x nonlinear, learnable

Title

- different themes which are usable in practice

Figures



- 1 Introduction to Data Fitting
- 2 Introduction to KAN
- 3 Methods
- 4 Results**

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

End

MAS