POLITECNICO
MILANO 1863

2024-2025

KANs: KOLMOGOROV-ARNOLD NETWORKS

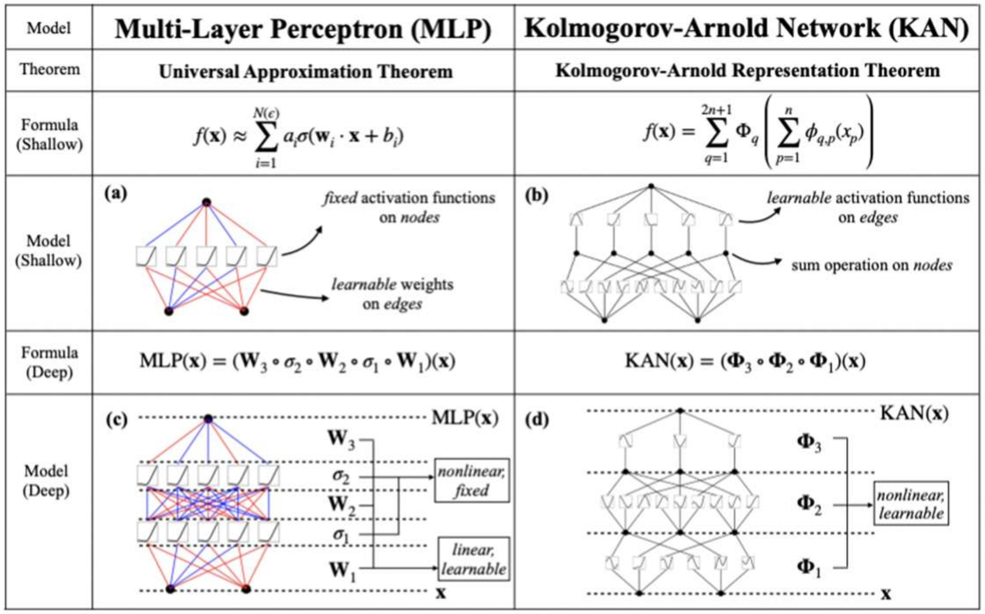
Project of Numerical Analysis for Machine Learning
Master’s degree in Computer Science Engineering

Pasqual Matteo Romilio - 10765765

Advisor Prof. Edie Miglio

Introduction to Kolmogorov-Arnold Networks (KANs)

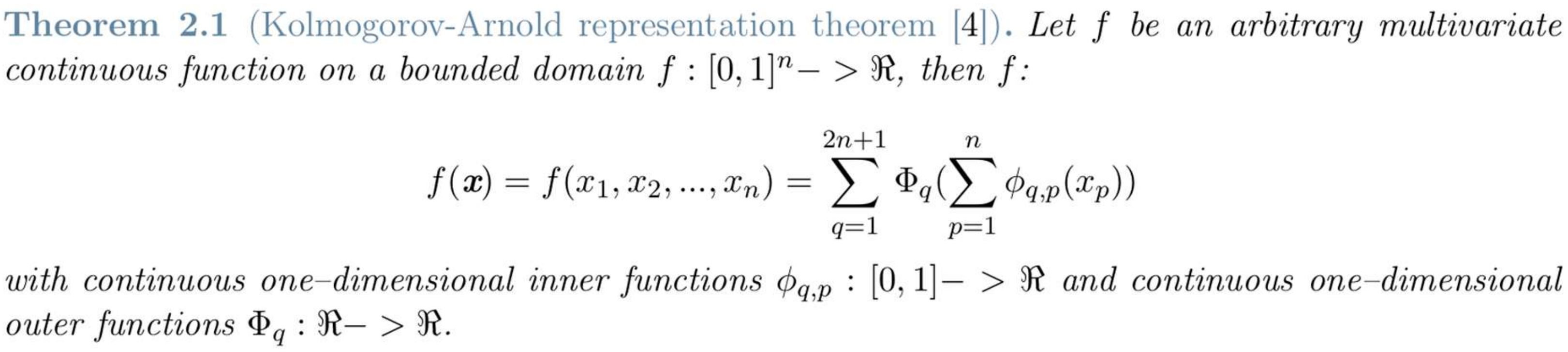
- KANs are inspired by the Kolmogorov-Arnold theorem.
- They provide a valid alternative to classical Multi-Layer Perceptrons (MLPs).
- Their key innovation lies in using learnable B-spline-based activation functions.
- Advantages: enhanced accuracy, improved interpretability, and greater scalability.



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*

Kolmogorov-Arnold Theorem

The theorem states that any multivariate continuous function on a bounded domain can be expressed as a finite composition of univariate continuous functions and additions.



Theorem 2.1 (Kolmogorov-Arnold representation theorem [4]). *Let f be an arbitrary multivariate continuous function on a bounded domain $f : [0,1]^n - > \Re$, then f:*

$$f(\boldsymbol{x}) = f(x_1, x_2, ..., x_n) = \sum_{q=1}^{2n+1} \Phi_q(\sum_{p=1}^{n} \phi_{q,p}(x_p))$$

with continuous one–dimensional inner functions $\phi_{q,p} : [0,1] - > \Re$ and continuous one–dimensional outer functions $\Phi_q : \Re - > \Re$.

- The advantage: Only a polynomial number of 1D functions must be learned.
- The disadvantage: These 1D functions can be non-smooth or fractal and challenging to learn directly, spline functions can approximate them effectively, achieving the desired accuracy.

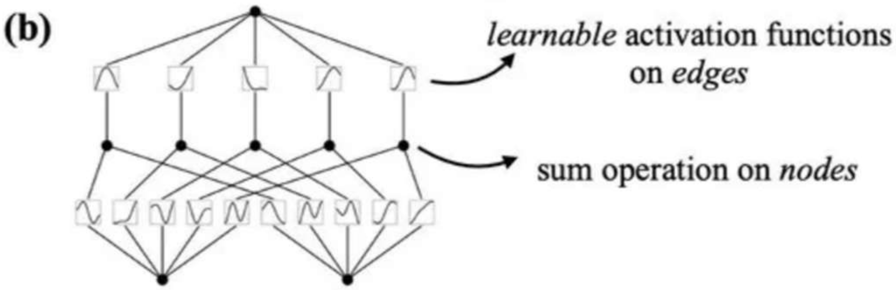
From Theorem to 2-Layers KAN

Starting from the theorem, we define the representation matrix that replaces the function’s summation with two matrices

$$f(\mathbf{x}) = \mathbf{\Phi}_{out} \times \mathbf{\Phi}_{in} \times \mathbf{x}$$

where:

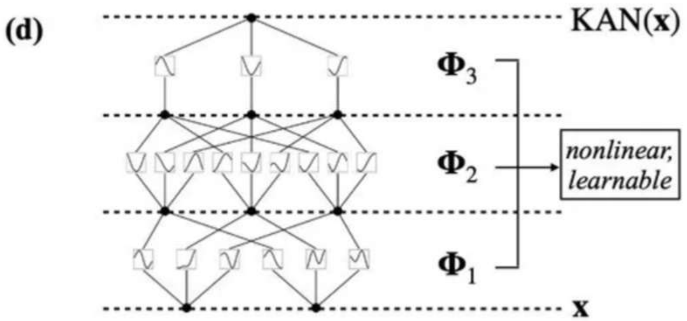
$$\mathbf{\Phi}_{in} = \begin{pmatrix} \phi_{1,1}(\cdot) & \cdots & \phi_{1,n}(\cdot) \\ \vdots & & \vdots \\ \phi_{2n+1,1}(\cdot) & \cdots & \phi_{2n+1,n}(\cdot) \end{pmatrix}, \quad \mathbf{\Phi}_{out} = (\Phi_1(\cdot) \ldots \Phi_{2n+1}(\cdot))$$



BUT Kolmogorov-Arnold networks are more expressive than their basic formulation from the Kolmogorov-Arnold representation theorem …

KANs Layer Structure

… in practice, we can design networks with different topologies by stacking layers from the input to the output.



A general KAN layer maps its input to the output through the matrix ϕ

$$\mathbf{x}_{l+1} = \mathbf{\Phi} \times \mathbf{x}_l \Rightarrow \mathbf{x}_{l+1} = \begin{pmatrix} \phi_{1,1}(\cdot) & \cdots & \phi_{1,n_{in}}(\cdot) \\ \vdots & & \vdots \\ \phi_{n_{out},1}(\cdot) & \cdots & \phi_{n_{out},n_{in}}(\cdot) \end{pmatrix} \mathbf{x}_l$$

Then the whole network will be represented by:

$$KAN(\mathbf{x}) = \mathbf{\Phi_{L-1}} \times \cdots \times \mathbf{\Phi_1} \times \mathbf{\Phi_0} \times \mathbf{x}$$

KANs Activation Functions

The most efficient technique involves leveraging spline functions. These functions are particularly effective for approximating the complexity and the non-linearities of the 1D functions associated with the KAN layers

Every activation function is defined as follows:

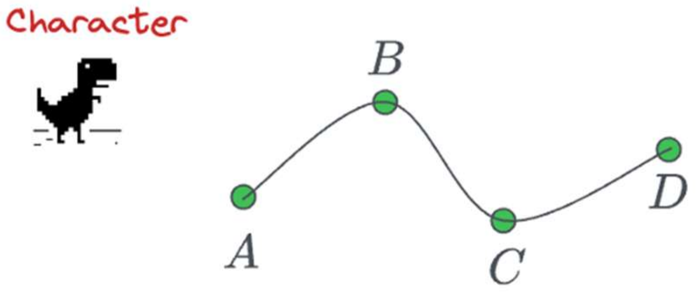
$$\phi(x) = w_b b(x) + w_s spline(x)$$

- W_b and W_s are learnable weights
- b(x) is the residual connection function (Bias function) defined by the silu(x) function. It is the counterpart of the bias in MLPs
- Spline(x) is the learnable function where the real power of KANs came from. In most cases is parametrized as a linear combination of B-splines functions

But what are B-splines …

B-splines (1): Naïve implementation

The problem we are going to solve is that the spline function should pass through some tag points that will be adjusted during the training phase.



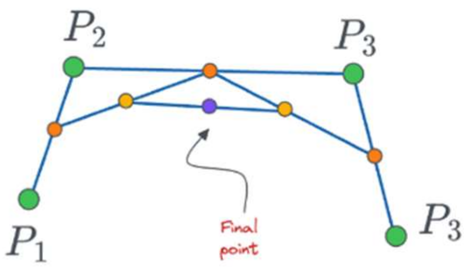
The naive way to implement it is to solve an n-dim linear system where we impose that the function passes through n points.

$$h(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0$$

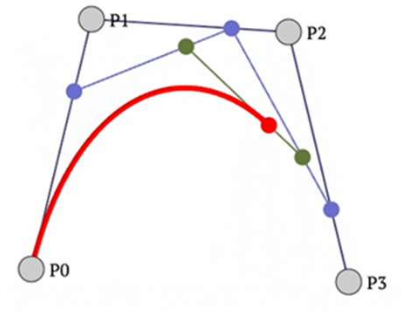
Solving such a system of linear equations will be computationally expensive and almost infeasible for large n such as in KANs.

B-splines (2): Bezier curves

The first idea is to use Bézier curves to solve the problem more smartly: the smooth curve passes near a set of control points without needing to solve a large system of equations



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



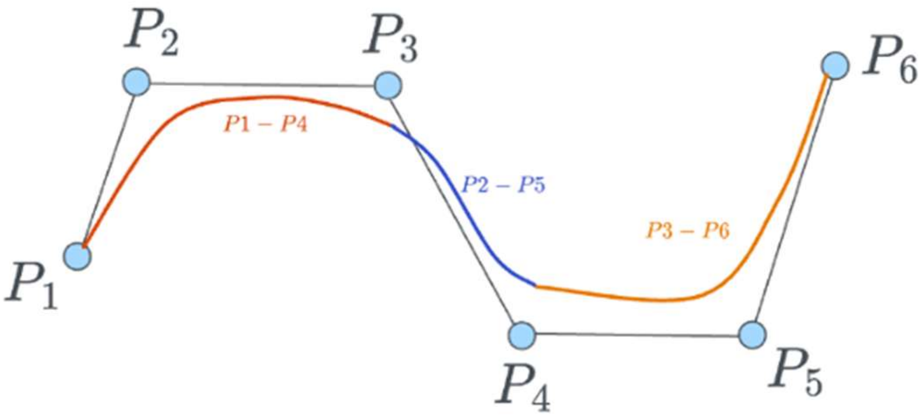
$$\mathbf{b}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i, \quad t \in [0,1]$$

However, the problem is still the same as before. Having N data points will result in a polynomial of degree N −1, which will be computationally expensive

B-splines (3): Real implementation

Finally, the intuition in B-splines is to use a series of lower-degree polynomial segments, which are connected smoothly.

In other words, instead of extending Bézier curves to tens of hundreds of data points, which leads to an equally high degree of the polynomial, we use multiple lower-degree polynomials and connect them to form a smooth curve



$$\mathbf{B_i}(x) = \sum_{i=0}^{n} P_i N_{i,k} \Rightarrow \mathbf{splines}(x) = \sum_{i=0}^{k} c_i B_i(x) = \sum_{i=0}^{k} c_i \sum_{i=0}^{n} P_i N_{i,k}$$

KANs Inizialization

KANs can be initialized and trained just like any other neural network in particular, for each activation function:

$$\phi(x) = w_b b(x) + w_s spline(x)$$

- We initialize the scaling factor for the spline function at 1: ws = 1.
- Each spline function is initialized with spline(x) ≈ 0. This is done by drawing B-spline coefficients ci~N(0,σ) with a small σ around 0.1. It's likely used to ensure symmetry and stability in early training stages.
- We initialize the scaling factor for the residual connection function with the Xavier initialization as in MLPs:

$$w_b \sim \mathcal{U}\left[-\sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}\right]$$

where n_in and n_out are specific for any layers

KANs Training

Specifically for Training:

- Forward propagation: we compute the outcome from the input:

$$KAN(\mathbf{x}) = \mathbf{\Phi_{L-1}} \times \cdots \times \mathbf{\Phi_1} \times \mathbf{\Phi_0} \times \mathbf{x}$$

- Backward propagation: We compute the loss using a standard loss function such as MSE, MAE, or cross-entropy, backpropagate the error, and then adjust the weights and spline tag points (all the parameters) using optimization methods such as Gradient Descent (GD), Stochastic Gradient Descent (SGD), Newton's Method (NM), BFGS, or others. The most common optimization method used is LBFGS

KANs Hyperparameters

The hyperparameters of a Kolmogorov-Arnold Network (KAN) are as follows:

- L: the depth of the KAN
- N ∈ $\{n_0, \ldots, n_l\}$: the width of each layer
- k: the number of B-splines used in the linear combination of each spline. (very small number usually 3)
- G: the number of control points for each B-spline

Given these hyperparameters L, N, G, and k, and considering N as the largest $n \in N$ the total number of parameters in a KAN can be expressed as

$$O(N^2 L(G+k)) \simeq O(N^2 LG)$$

In contrast, a multi-layer perceptron (MLP) with hyperparameters L, and N requires only

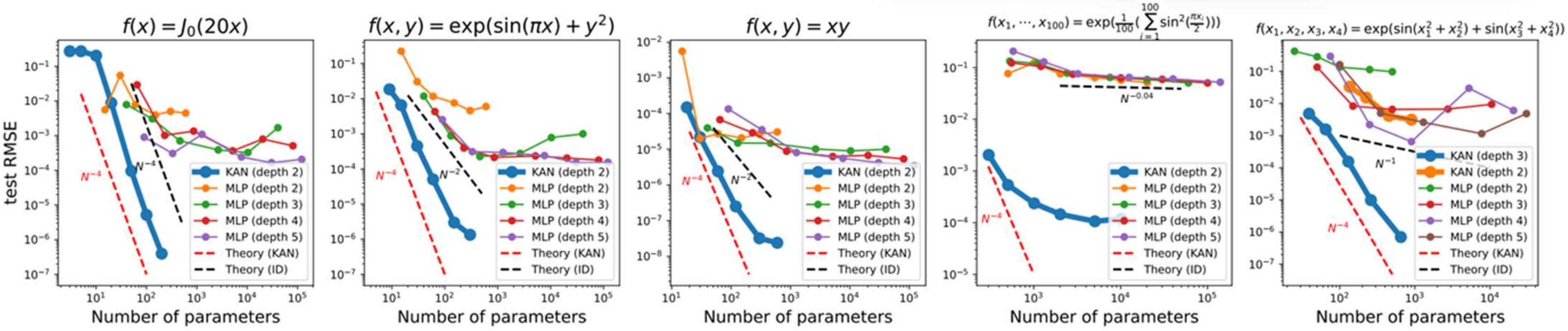
$$O(N^2 L)$$

Which appears to be more efficient than KAN. However, KANs typically require much smaller values of N compared to MLPs

KANs Accuracy

KANs consistently outperform MLPs, achieving significantly lower test loss across a range of parameters, and at much lower network depth (number of layers).

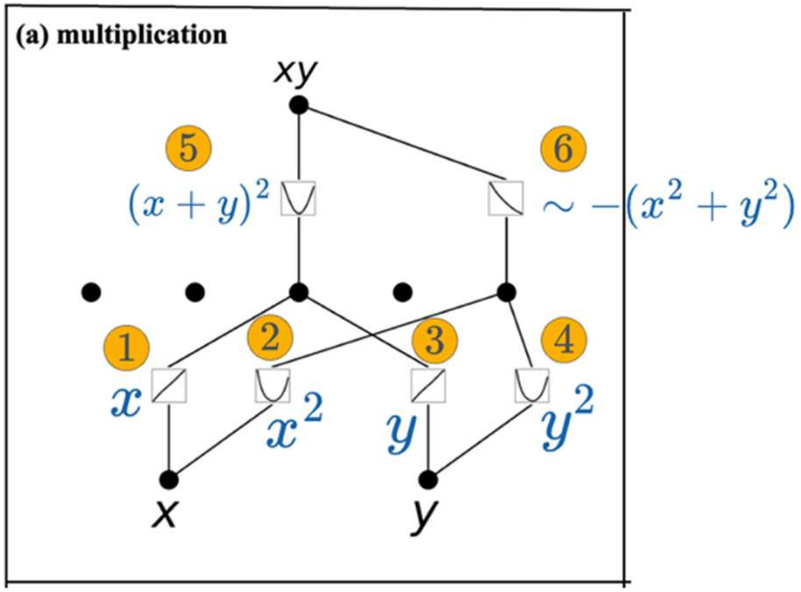
The performance of MLPs almost stagnates as the number of parameters increases.



KANs Interpretability

Kolmogorov-Arnold networks are interpretable since they learn univariate functions at all levels and it is relatively easy to determine the structure learned

Interpretation of $f(x, y) = xy$



Proof: $$\frac{1}{2}(x+y)^2 - \frac{1}{2}(x^2+y^2) = \frac{1}{2}x^2 + \frac{1}{2}y^2 + xy - \frac{1}{2}x^2 - \frac{1}{2}y^2 = xy$$

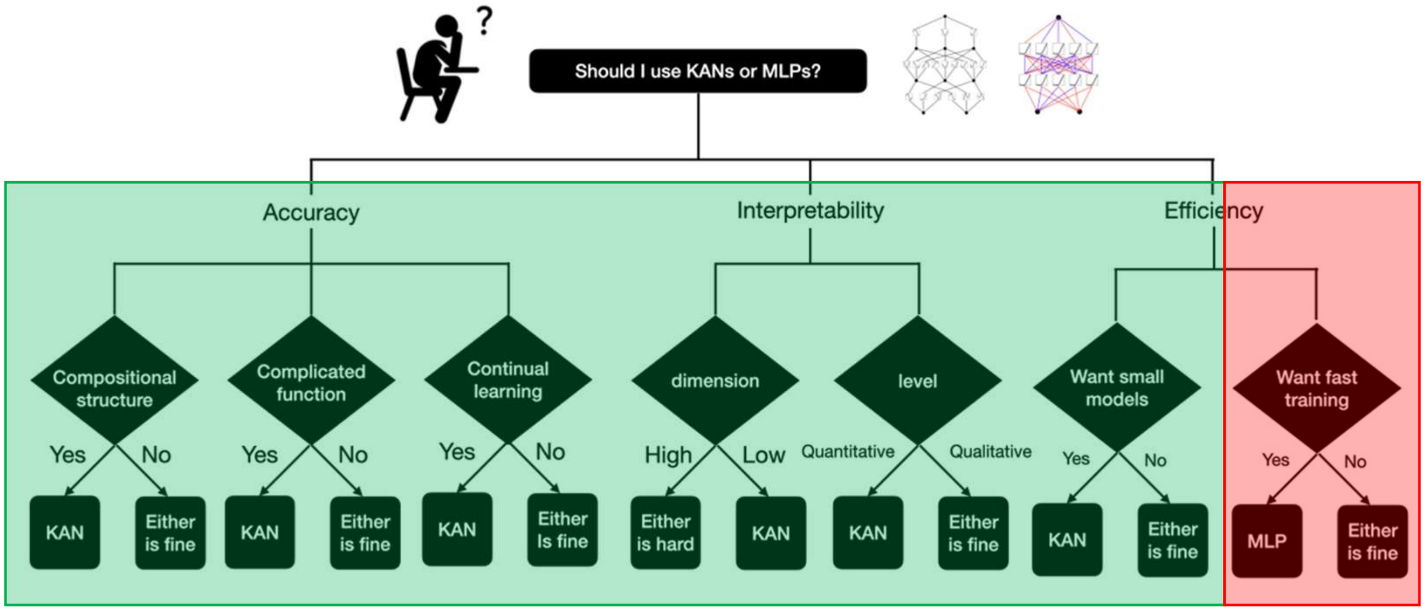
KANs Drawbacks

Currently, the biggest bottleneck of KANs lies in their slow training.

KANs are typically 10 times slower than MLPs, given the same number of parameters (though KANs require fewer parameters than MLPs to accomplish the same task).

This is primarily caused by the complex reshaping of spline(x), particularly on B-spline control points.

When should we use KANs?



Should I use KANs or MLPs?

Accuracy

Compositional structure: Yes → KAN; No → Either is fine

Complicated function: Yes → KAN; No → Either is fine

Continual learning: Yes → KAN; No → Either Is fine

Interpretability

dimension: High → Either is hard; Low → KAN

level: Quantitative → KAN; Qualitative → Either Is fine

Efficiency

Want small models: Yes → KAN; No → Either is fine

Want fast training: Yes → MLP; No → Either is fine

KANs or Both

MLPs

Advanced: Sparsification (1)

For MLPs, regularization, particularly L1 or L2 regularization of linear weights, is used to improve generalization and prevent overfitting, thereby improving predictions. KANs can adopt this high-level concept, but two modifications are necessary:

- Since there are no linear weights in KANs, linear weights are replaced by learnable activation functions. Therefore, the L1 norm must be redefined for these activation functions.

- Experimentally, we find that L1 regularization alone is insufficient for sparsifying KANs. Instead, additional entropy regularization encourages the model to use activation functions with lower complexity and fewer degrees of freedom.

Advanced: Sparsification (2)

We define the L1 norm of an activation function ϕ as its average magnitude over its Np inputs:

$$|\phi|_1 \equiv \frac{1}{N_p} \sum_{s=1}^{N_p} |\phi(x^{(s)})|$$

Then for a KAN layer Φ we define the L1 norm of Φ to be the sum of L1 norms of all activation functions: |Φ|1.

$$|\mathbf{\Phi}|_1 \equiv \sum_{i=1}^{n_{in}} \sum_{j=1}^{n_{out}} |\phi_{i,j}|_1$$

Advanced: Sparsification (3)

We define the regularization entropy of Φ to be S(Φ).

$$S(\mathbf{\Phi}) \equiv -\sum_{i=1}^{n_{in}}\sum_{j=1}^{n_{out}} \frac{|\phi_{i,j}|_1}{|\mathbf{\Phi}|_1} \log(\frac{|\phi_{i,j}|_1}{|\mathbf{\Phi}|_1})$$

The total training objective (l_total) is the prediction (l_pred) plus L1 and entropy regularization of all KAN layers where we define µ1, µ2 are relative magnitudes usually set to µ1 = µ2 = 1, and λ controls overall regularization magnitude

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

Advanced: Pruning

We may also want to prune the network to a smaller subnetwork by eliminating all the non-essential nodes. In matrix form, these nodes will be represented as 0.

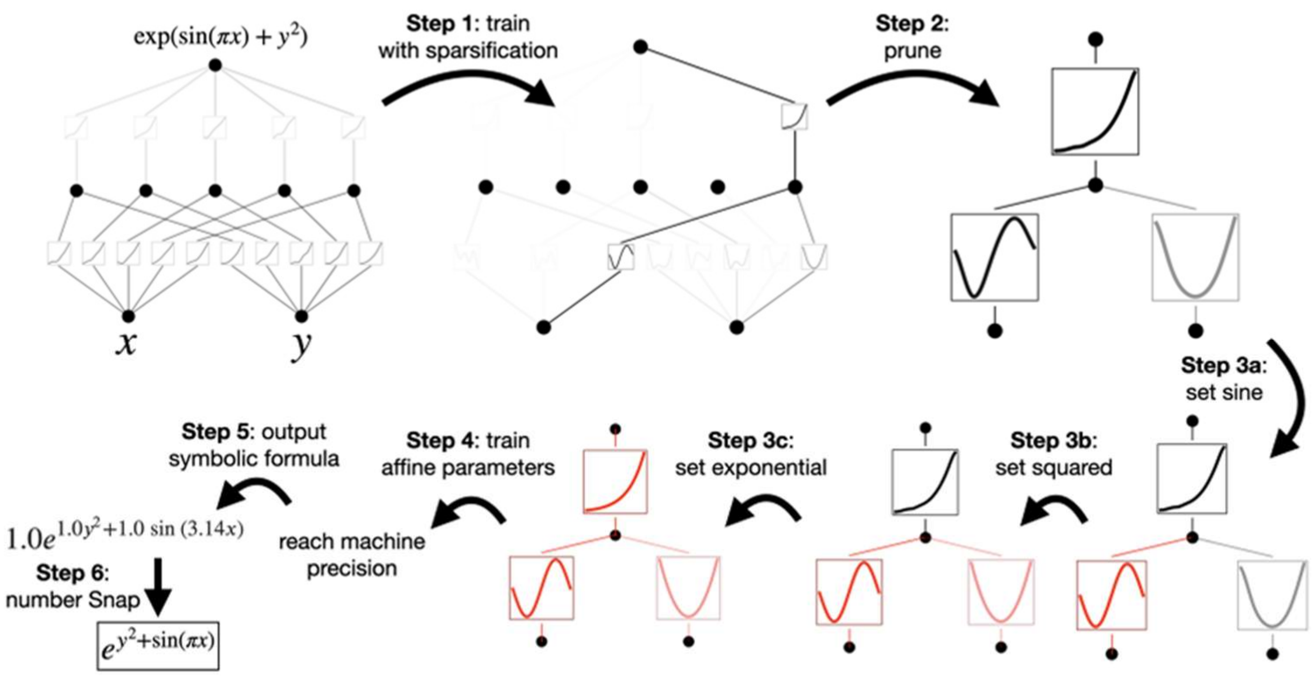
We define its incoming and outgoing scores as I_l,i and O_l,i.

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

and consider a node to be important if both incoming and outgoing scores are greater than a threshold hyperparameter θ = 10^−2 by default.

Advanced: Symbolification

In some cases, after pruning, we suspect that some activation functions are very similar to symbolic functions then symbolification provides an interface to set the symbolic ϕ to their specified symbolic form.

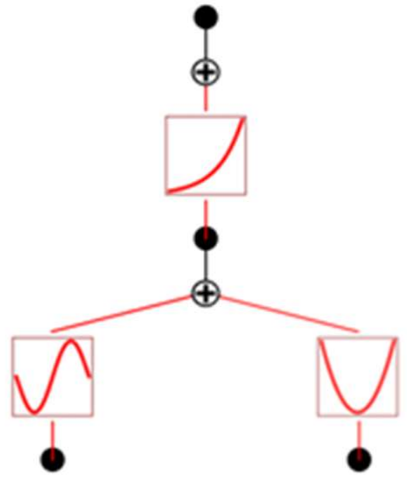

$\exp(\sin(\pi x) + y^2)$
x
y
Step 1: train with sparsification
Step 2: prune
Step 3a: set sine
Step 3b: set squared
Step 3c: set exponential
Step 4: train affine parameters
reach machine precision
Step 5: output symbolic formula
$1.0e^{1.0y^2+1.0\sin(3.14x)}$
Step 6: number Snap
$e^{y^2+\sin(\pi x)}$

Code: Symbolic Regression

NOTEBOOK 1:

The regression task involves training a Kolmogorov Arnold Network (KAN) to learn a non-linear function using symbolic regression to enhance interpretability.

$$f(x, y) = e^{\sin(\pi x)+y^2}$$



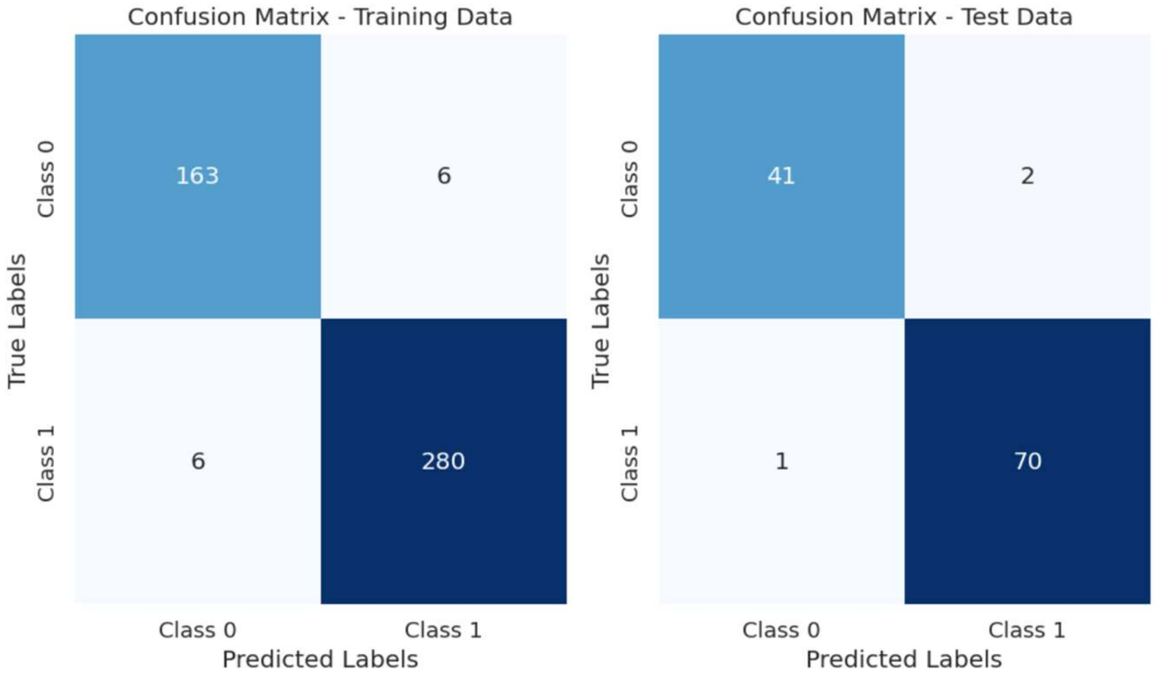
Cancer Analisys

NOTEBOOK 2:

The cancer analysis task involves training a Kolmogorov Arnold Networks (KAN) to predict cancer outcomes based on clinical and pathological data from the Wisconsin Breast Cancer Dataset from sklearn.

Accuracy:

- Training Accuracy: 0.974
- Test Accuracy: 0.973



Conclusions

KANs are not merely an alternative to MLPs but represent a paradigm shift toward more expressive, transparent, and efficient neural network architectures.

KANs in Practice:

- Outperform MLPs: Achieving lower test loss with reduced depth and complexity.
- Application: Superiority in non-linear regression, function approximation tasks, …

Challenges with KANs:

- Computational Overhead: Spline activation functions slow down the process.
- Optimization: Lack of optimization strategies for spline-based architectures.

Future of KANs:

- Improvement: Accelerated training methods and efficient spline representations.
- Broader Adoption: Scalable regularization techniques for various tasks.