

# KAN: Kolmogorov-Arnold Networks

Şeyma GÜLŞEN AKKUŞ

Applied Data Science Master's Program

ADS 521 - Numerical Methods Final Project

Doç. Dr. Yıldırım AKBAL

December 25, 2024

# Table of Contents

- 1 Introduction to Kolmogorov-Arnold Networks (KANs)
- 2 Kolmogorov-Arnold Representation Theorem
- 3 Architecture of KANs
- 4 Experimental Results
- 5 Conclusion
- 6 References

# What are Kolmogorov-Arnold Networks (KANs)?

Kolmogorov-Arnold Networks (KAN) takes role for promising alternative to Multi-Layer Perceptrons and its architecture inspired by the Kolmogorov-Arnold representation Theorem.

## Multi-Layer Perceptron

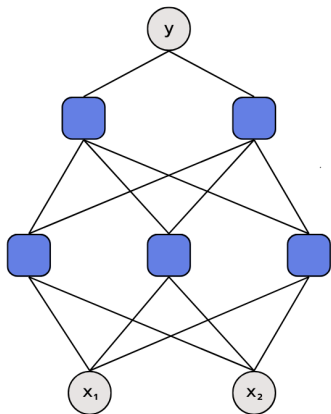
- Fixed activation functions on nodes.
- Learnable weights on edges.
- Activation functions are globally applied across all neurons.
- Limited flexibility and interpretability for complex tasks.

## Kolmogorov-Arnold Network

- Learnable activation functions on edges.
- Sum operation on nodes.
- Every edge can have a unique activation function.
- Greater flexibility and accuracy in modeling complex functions.

# Architecture of KANs and MLPs

**MLP** (Multi Layer Perceptron)



**KAN** (Kolmogorov-Arnold Network)

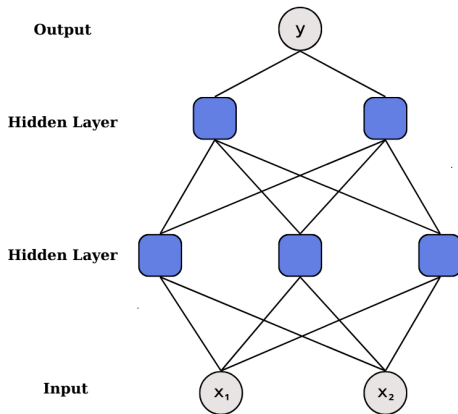
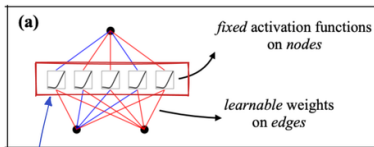


Figure: KANs vs MLPs

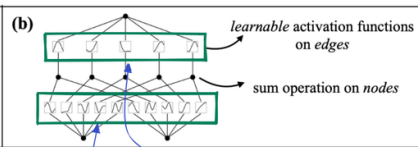
"KANs are the models that have MLP structures on the outside and splines on the inside."

Neural Network



All neurons have  
a fixed activation  
function

KAN



Every edge has  
a different  
activation  
function

Figure: Computational Graphs for Shallow KAN and MLP Model

# Kolmogorov-Arnold Representation Theorem

The works of Vladimir Arnold and Andrey Kolmogorov established the following:

## Theorem (Kolmogorov-Arnold)

*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. More specifically:*

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

where:

$$\phi_{q,p} : [0, 1] \rightarrow \mathbb{R}, \quad \Phi_q : \mathbb{R} \rightarrow \mathbb{R}.$$

- The theorem essentially states that any multivariate continuous function can be reduced to a combination of single-variable continuous functions ( $\phi_{q,p}$ ) and an addition operation.
- This representation eliminates the complexity of directly handling multivariable functions by breaking them into simpler, one-dimensional functions.

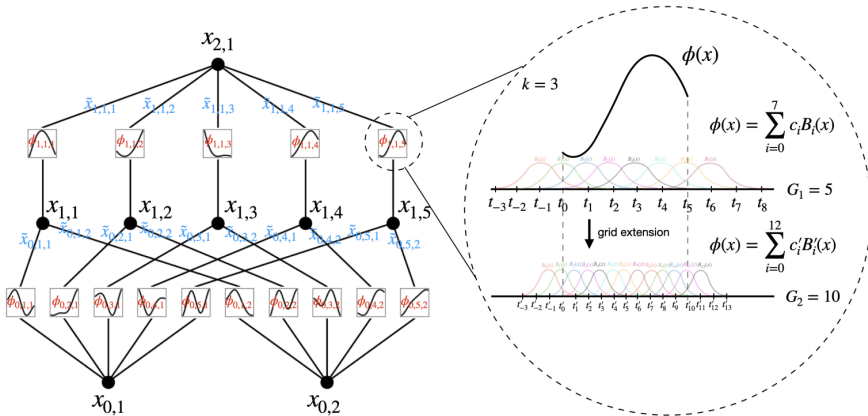


# Architecture of KANs

In a supervised learning problem with input-output pairs  $(x_i, y_i)$ , our goal is to find a function  $f$  such that:

$$y_i \approx f(x_i)$$

Kolmogorov-Arnold Theorem implies that we are done if we can find the appropriate univariate functions  $\Phi_q$  and  $\phi_{q,p}$ . These univariate functions  $\phi_{q,p}$  can be parametrized as B-spline curves, with learnable coefficients of local B-spline basis functions. This gives us a prototype of KAN, whose computation graph is illustrated as follows with the input dimension  $n = 2$ :



**Figure:** The activations that flow through the network and the visualization an activation function which is parametrized as B-spline.

# Definition of B-Spline Basis Function

A B-spline of order  $p + 1$  is a collection of piecewise polynomial functions  $B_{i,p}(t)$  of degree  $p$  in a variable  $t$ . The points where the polynomial pieces meet are called  $[t_0, t_1, t_2, \dots, t_m]$  where the knot sequence is in non-decreasing order.

For a given sequence of knots, there exists a unique spline  $B_{i,p}(t)$ , up to a scaling factor, defined as:

$$B_{i,p}(t) = \begin{cases} \text{non-zero,} & \text{if } t_i \leq t < t_{i+p+1}, \\ 0, & \text{otherwise.} \end{cases}$$

B-splines can be constructed using the Cox-de Boor recursion formula. We start with the B-splines of degree  $p=0$ .

$$B_{i,0}(t) = \begin{cases} 1, & \text{if } t_i \leq t < t_{i+1}, \\ 0, & \text{otherwise.} \end{cases}$$

Then, the higher-degree B-splines are defined by recursion:

$$B_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} B_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} B_{i+1,p-1}(t).$$

# What is a KAN Layer?

In traditional MLPs, we stack more layers to build deeper networks. So, in KANs, we need to first define:

"What is a KAN layer?"

KAN layer with  $n_{\text{in}}$ -dimensional inputs and  $n_{\text{out}}$ -dimensional outputs can be defined as a matrix of 1D functions:

$$\Phi = \{\phi_{q,p}\}, \quad p = 1, 2, \dots, n_{\text{in}}, \quad q = 1, 2, \dots, n_{\text{out}}$$

So, a Kolmogorov-Arnold representation is simply a composition of two KAN layers.

To make KANs deeper: We stack more KAN layers!

# Activation Value in KAN Layers

The activation value of the  $(l + 1, j)^{\text{th}}$  neuron is the sum of all incoming post-activations:

$$x_{l+1,j} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}), \quad j = 1, \dots, n_{l+1}$$

where  $i$  corresponds to the  $i^{\text{th}}$  neuron.  $l$  corresponds to the  $l^{\text{th}}$  layer.  
In matrix form:

$$x_{l+1} = \underbrace{\begin{pmatrix} \phi_{l,1,1}(\cdot) & \phi_{l,1,2}(\cdot) & \cdots & \phi_{l,1,n_l}(\cdot) \\ \phi_{l,2,1}(\cdot) & \phi_{l,2,2}(\cdot) & \cdots & \phi_{l,2,n_l}(\cdot) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{l,n_{l+1},1}(\cdot) & \phi_{l,n_{l+1},2}(\cdot) & \cdots & \phi_{l,n_{l+1},n_l}(\cdot) \end{pmatrix}}_{\Phi_l} x_l$$

where:  $\Phi_l$  is the function matrix corresponding to the  $l^{\text{th}}$  KAN layer.

# General KAN Network Structure

A general KAN network is a composition of  $L$  layers. Given an input vector:

$$x_0 \in \mathbb{R}^{n_0}$$

The output of the KAN network can be expressed as:

$$KAN(x) = (\Phi_{L-1} \circ \Phi_{L-2} \circ \cdots \circ \Phi_1 \circ \Phi_0)x$$

where  $\Phi_l$  represents the function matrix at layer  $l$ . Each  $\Phi_l$  applies trainable univariate functions to transform input activations.

## Note

The original Kolmogorov-Arnold representation corresponds to a 2-layer KAN with the shape:  $[n, 2n + 1, 1]$

# Architecture of Neural Networks

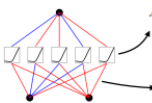
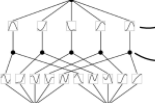
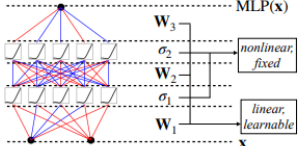
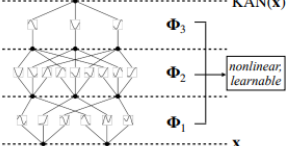
Model	Multi-Layer Perceptron (MLP)	Kolmogorov-Arnold Network (KAN)
Theorem	Universal Approximation Theorem	Kolmogorov-Arnold Representation Theorem
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(e)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(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

Figure: Multi-Layer Perceptrons vs. Kolmogorov-Arnold Networks



# Implementation Details of KAN Layers (1/2)

While implementing KAN layers to optimize them, follow these steps:

## 1) Activation Function

The activation function should combine a basis function and a spline function:

$$\varphi(x) = w_b b(x) + w_s \text{spline}(x)$$

where:

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

The spline function is parametrized as a linear combination of B-splines:

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

## 2) Initialize scales as follows:

- The spline contribution is initialized near zero by:

$$w_s = 1, \text{spline}(x) \approx 0^2$$

- The basis contribution  $w_b$  follows Xavier Initialization to ensure stable gradients.

## 3) Adaptive Spline Grids

Use adaptive spline grids. This prevents the activation values from exceeding predefined spline regions.

These optimizations make KAN layers scalable, stable, and well-suited for deep architectures.

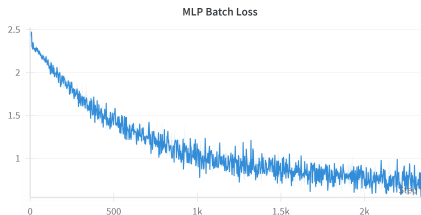
# Experimental Results

The experiment compared the performance of KANs and MLPs on the MNIST dataset. Evaluation metrics included training loss and test accuracy. The MNIST dataset consists of handwritten digits from 0 to 9.

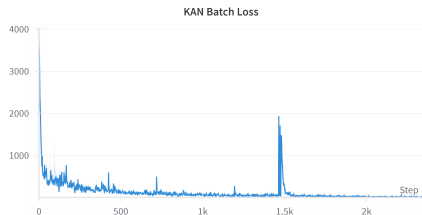
The images were preprocessed and normalized before feeding them into both models. The SimpleKAN model was implemented with spline-based activation functions, while the SimpleMLP model used traditional fixed activation functions “Relu”. Additionally, a dropout layer is introduced on both models to prevent overfitting.

- Dataset: MNIST (28x28 grayscale images)
- Models: SimpleKAN and SimpleMLP with AdamW Optimizer
- Frameworks: JAX, Flax and TensorFlow
  - JAX: Used for numerical computation and automatic differentiation.
  - Flax: A neural network library built on JAX, used to define and train both the SimpleKAN and SimpleMLP models.
  - TensorFlow: Utilized primarily for dataset management and preprocessing.
- Optimization Algorithm: AdamW with adjusted learning rate and '0.0001' weight decay
- Learning Rates:
  - KAN: Scheduled learning rate starting at 3e-5
  - MLP: Fixed learning rate of 1e-4
- Epochs: 4
- Batch Size: 128

# Batch Loss Comparison



MLP Batch Loss



KAN Batch Loss

# Epoch Loss Comparison

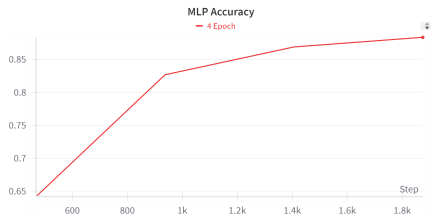


MLP Loss per Epoch

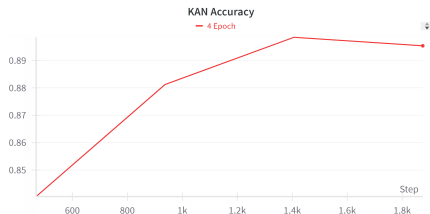


KAN Loss per Epoch

# Epoch Accuracy Comparison



MLP Accuracy per Epoch



KAN Accuracy per Epoch

# Conclusion

The experimental results show that KANs learn faster and achieve higher accuracy than MLPs. However, they are more sensitive to learning rate adjustments, which sometimes cause instability or spikes in batch loss. While KANs converge faster, they require careful hyperparameter tuning to avoid issues like sudden spikes or slight accuracy drops.

As a result, Kolmogorov-Arnold Networks present a promising evolution in neural network design by leveraging learnable edge-based activation functions inspired by the Kolmogorov-Arnold Representation Theorem.



# References



Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., Hou, T. Y., & Tegmark, M. (2024). *KAN: Kolmogorov–Arnold Networks*. Massachusetts Institute of Technology, California Institute of Technology, Northeastern University, NSF Institute for Artificial Intelligence and Fundamental Interactions.



Wikipedia Contributors. (2024). *De Boor's Algorithm*.



Daily Dose of Data Science. (2024). *A Beginner-Friendly Introduction to Kolmogorov-Arnold Networks (KAN)*.



Srigas. (2024). *JAX-KAN Implementation Repository*.