CoDMAV

KEEPING UP WITH KANs

A paper presentation on the **K**olmogorov **A**rnold **N**etworks

By: Harini Anand

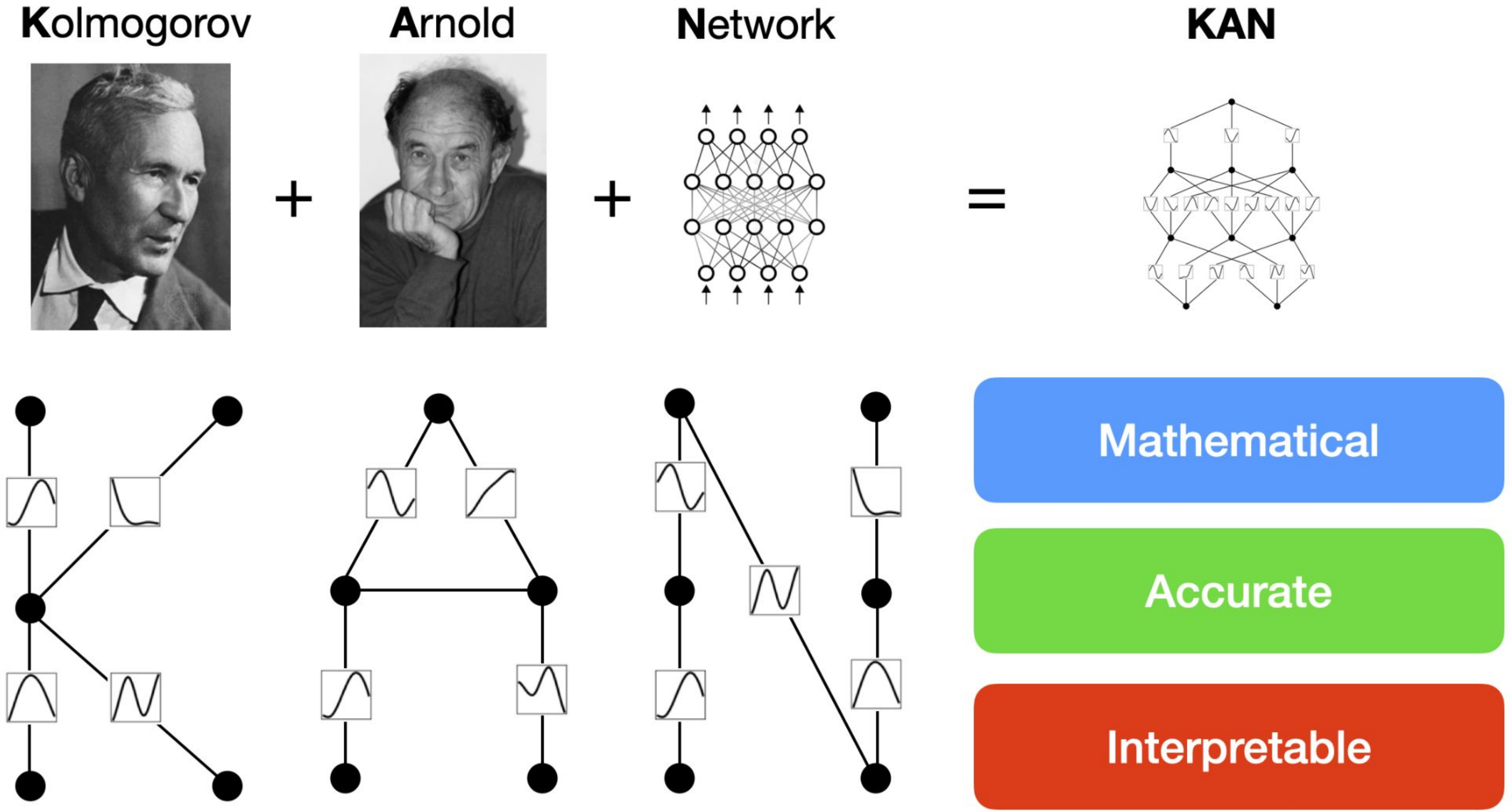
$whoami

- CSE Junior with a keen interest in **Computational Cognition.**
- Working at Niramai Health Analytix, a deep tech startup focusing on Breast Cancer, and is using ML frameworks as a research intern at the Indian Institute of Technology Hyderabad, to predict gene regulatory networks.
- Previously worked on building cognitive tools for reducing the onset of Dementia as a student entrepreneur.
- Strong advocate for representation in STEM and recognized as a High Impact APAC Ambassador for the Women In Data Science Community, an initiative by Stanford University & a Harvard WE Tech Fellow.

Reach out to me on : LinkedIn & X

Note: The source of the images/figures used in this article is the “KAN: Kolmogorov–Arnold Network” paper unless stated otherwise.

WHY IS IT CALLED KANs ?


Kolmogorov
+
Arnold
+
Network
=
KAN
Mathematical
Accurate
Interpretable

WHY DO WE NEED KANs ?

What if all your weights were functions?

IS IT KAN ENOUGH?

MLP < KAN (small AI tasks)

- Multi-layer perceptrons (MLPs), or fully-connected feedforward neural networks, are fundamental in deep learning, serving as default models for approximating nonlinear functions.
- Despite their importance affirmed by the universal approximation theorem, they possess drawbacks.

WHY KANs ?

- Designed as an alternative to Multi-Layer Perceptron (MLP) models, KAN employs learnable activation functions on edges and summation operations at nodes.
- This approach enhances the model's flexibility, allowing for more accurate data analysis while also providing scientists with a more understandable and collaborative architecture.
- KAN, particularly in data fitting and partial differential equation (PDE) solving applications, achieves better results than MLPs.

WHAT DRAWBACKS ?

aka; the starting research bottlenecks

- In applications like transformers, MLPs often monopolize parameters
- They lack interpretability compared to **attention** layers

BACKGROUND OF KANs

tiny bit math heavy

Background: Kolmogorov's Superposition Theorem

Kolmogorov’s Superposition Theorem (1957):

- **Statement: Kolmogorov–Arnold representation theorem** (or **superposition theorem**) states that every multivariate continuous function f:[0,1]$n \to R$ can be represented as a superposition of the two-argument addition of continuous functions of one variable.
- **Implication:** This theorem indicates that any complex function of multiple variables can be broken down into simpler functions of a single variable and sums, making it a powerful tool for function approximation

Background: Kolmogorov's Superposition Theorem

- **Kolmogorov's Theorem:** Any multivariate continuous function can be decomposed into continuous functions of one variable plus addition.
- **Arnold's Extension:** Practical framework for implementation.

KAN Math

$y = F(x_1, x_2, x_3, x_4 \ldots . x_n)$

$\hookrightarrow$ [multivariate function]

can represent ⇑ as UNIVARIATE COMPOSITION OF FUNCTIONS

HOW?

$\hookrightarrow \phi_1(x_1) \quad \phi_2(x_2)\ \phi_3(x_3) \ldots$

$\phi_n(x_n)$

each of them is univariate

Sum them up

↓

$$\phi_1(x_1) + \phi_2(x_2) + \phi_3(x_3)$$

$$= \sum_{q=1}^{n} \phi_q(x_q)$$

↙ Pass this through another function

$$\boxed{f}\left(\sum_{q=1}^{n} \phi_q(x_q)\right)$$

single composition

What does this mean in ML ctx?

Example

(multivariate) Taylor Swift $= F(\overset{①}{\text{Red}}, \overset{②}{\text{folklore}}, \overset{③}{\text{Midnights}})$

(univariate) Taylor Swift $= f(\phi_R(\text{Red}))$ ①

$+ f(\phi_f(\text{folklore}))$ ②

$+ f(\phi_m(\text{midnights}))$ ③

$$f(\mathbf{x}) = f(x_1, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right).$$

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

IS KAN the Latest SOTA?

- Kolmogorov-Arnold representation theorem is not new, so why has the practice of using it in machine learning not been studied before?
- It has been tinkered with before in several attempts
- *However, most work has stuck with the original depth-2 width-(2n + 1) representation, and did not have the chance to leverage more modern techniques (e.g., back propagation) to train the networks.*

WHAT IS KAN?

- Kolmogorov–Arnold Networks (KAN) are based on a fundamental result in mathematical analysis known as Kolmogorov's Superposition Theorem.
- This theorem, and its extension by Vladimir Arnold, provide a theoretical foundation for representing any continuous multivariate function as a sum of continuous univariate functions.

Just as a Taylor Swift album is composed of various songs (each with different themes and styles), a KAN represents a multivariate function as a sum of simpler functions.

MACHINE LEARNING
VERSION UPGRADE
BEFORE GTA 6 XD

KANs vs MLP

- KANs have faster scaling than MLPs. KANs have better accuracy than MLPs with fewer parameters.

WDYM?

KANs vs MLP

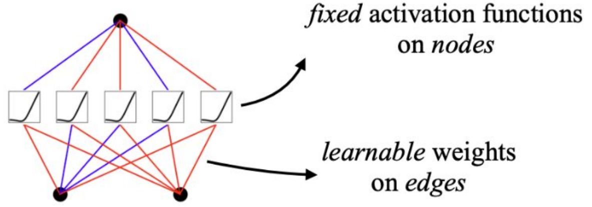
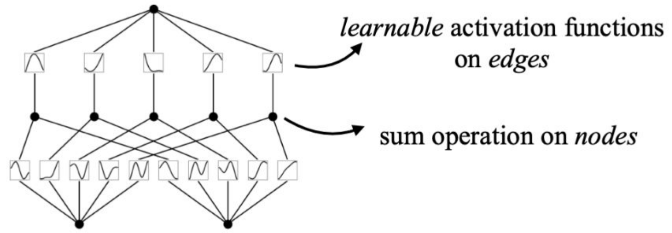
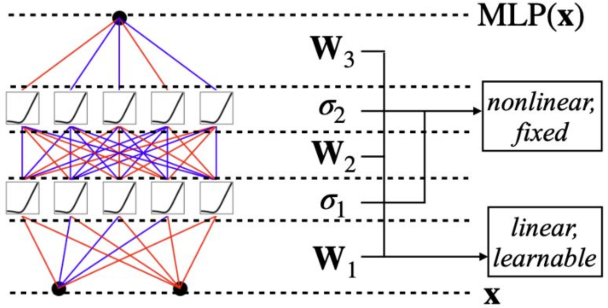
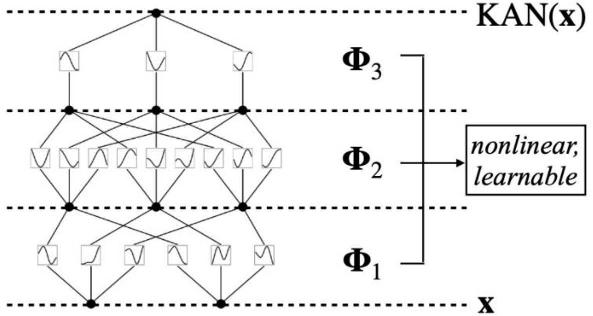
Network Structure of MLP:

- Activation Functions: Fixed activation functions (e.g. ReLU or sigmoid) are used and implemented at each node.
- Weights and Bias: Each edge has a fixed weight (w) and bias (b) value. These values are optimized during the learning process.
- Layers: MLPs usually have several hidden layers and a certain number of nodes in each layer.

KANs vs MLP

Network Structure of KAN:

- Activation Functions: Learnable activation functions (spline (piecewise polynomial) based) assigned to edges are used.
- Aggregate Operation: The interaction between nodes is expressed with learned functions and each node performs an aggregate operation. Nodes collect the results from splines.
- Layers: KANs have a multi-layered structure that includes learnable spline-based activation functions in each layer.

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*

KAN Concept

- This theorem allows for the decomposition of complex high-dimensional functions into compositions of simpler one-dimensional functions.
- By focusing on optimizing these one-dimensional functions rather than the entire multivariate space, KANs reduce the complexity and the number of parameters needed to achieve accurate modeling.
- Furthermore, Because of working with simpler one-dimensional functions, KANs can be simple and interpretable models.

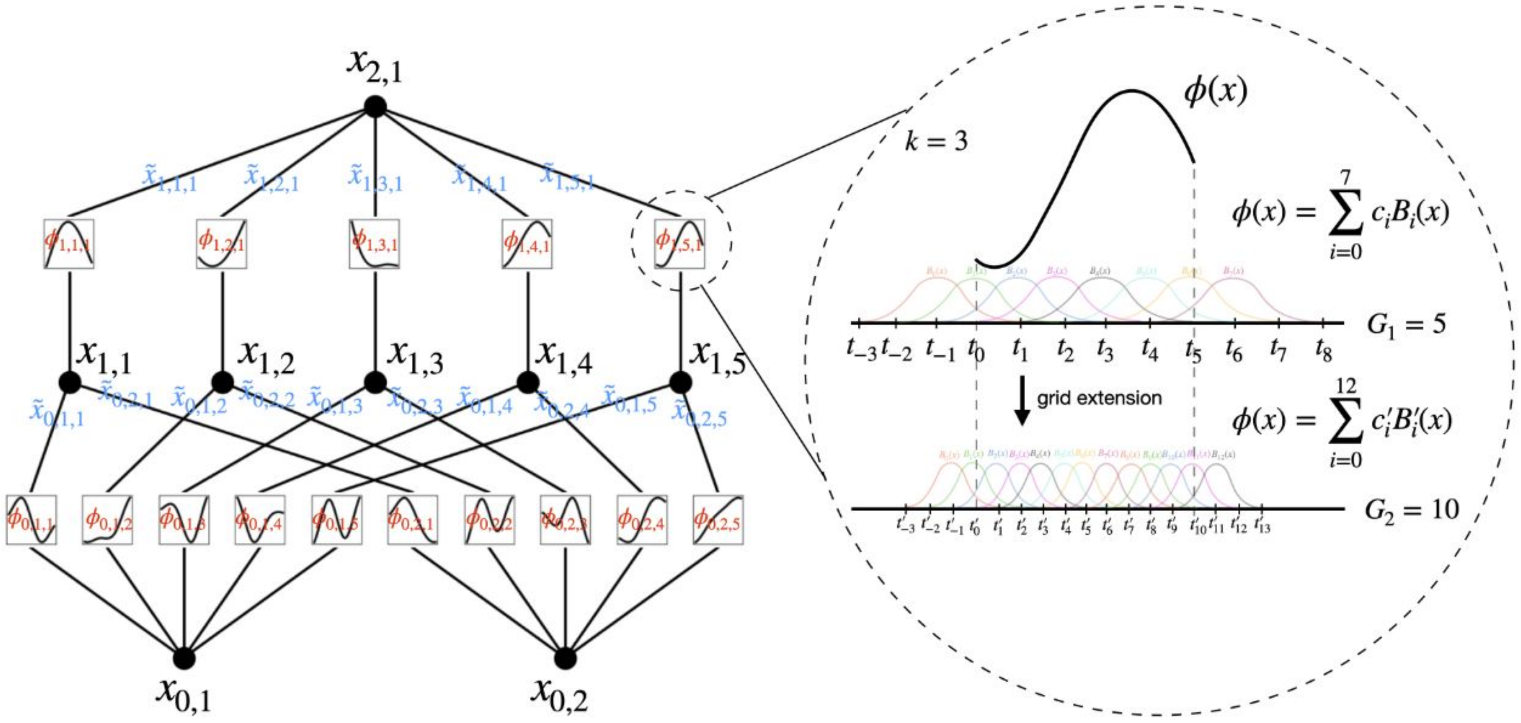
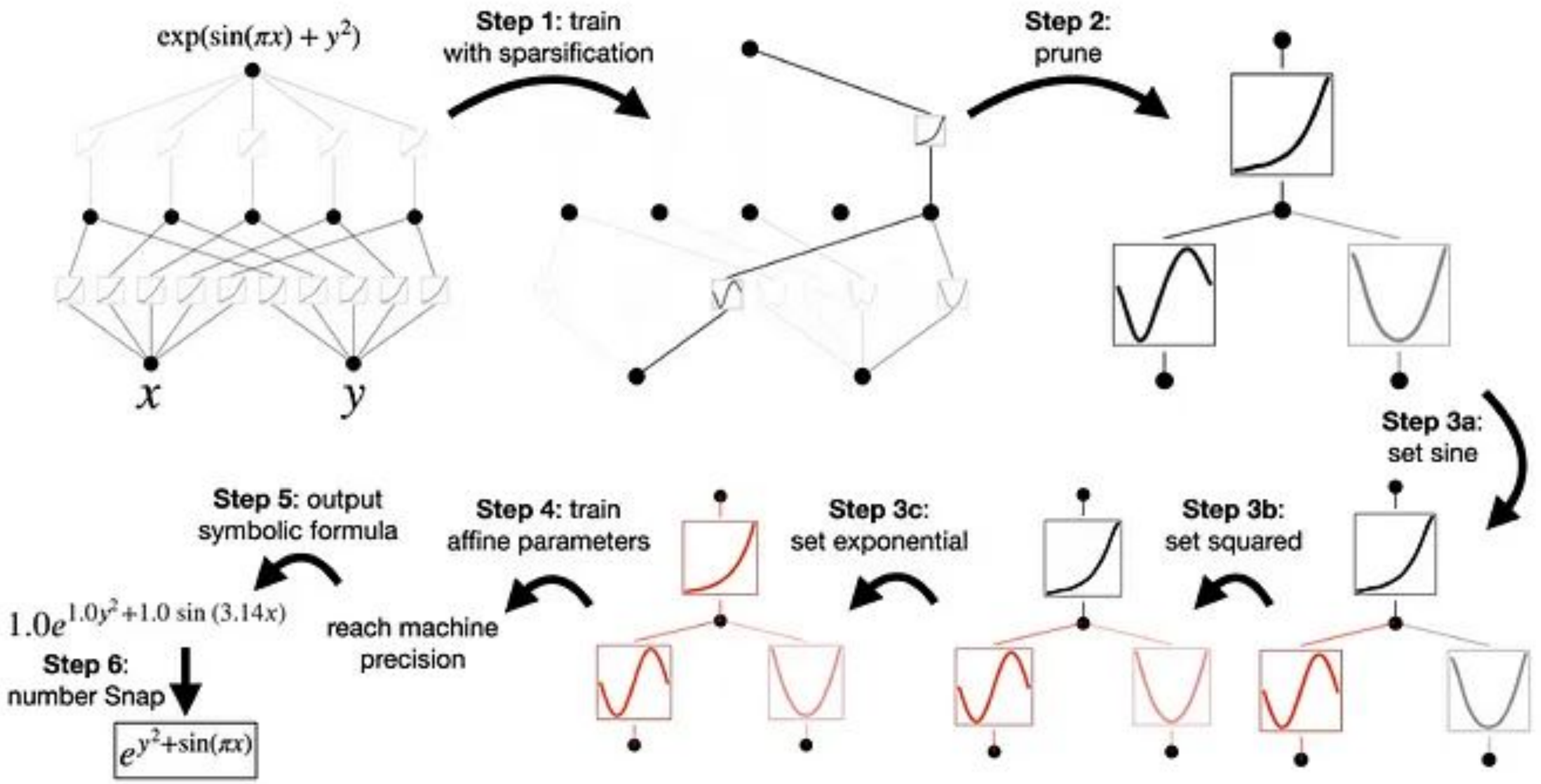

$x_{2,1}$
$\tilde{x}_{1,1,1}$ $\tilde{x}_{1,2,1}$ $\tilde{x}_{1,3,1}$ $\tilde{x}_{1,4,1}$ $\tilde{x}_{1,5,1}$
$\phi_{1,1,1}$ $\phi_{1,2,1}$ $\phi_{1,3,1}$ $\phi_{1,4,1}$ $\phi_{1,5,1}$
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$
$\tilde{x}_{0,1,1}$ $\tilde{x}_{0,2,1}$ $\tilde{x}_{0,1,2}$ $\tilde{x}_{0,2,2}$ $\tilde{x}_{0,1,3}$ $\tilde{x}_{0,2,3}$ $\tilde{x}_{0,1,4}$ $\tilde{x}_{0,2,4}$ $\tilde{x}_{0,1,5}$ $\tilde{x}_{0,2,5}$
$\phi_{0,1,1}$ $\phi_{0,1,2}$ $\phi_{0,1,3}$ $\phi_{0,1,4}$ $\phi_{0,1,5}$ $\phi_{0,2,1}$ $\phi_{0,2,2}$ $\phi_{0,2,3}$ $\phi_{0,2,4}$ $\phi_{0,2,5}$
$x_{0,1}$ $x_{0,2}$
$k = 3$
$\phi(x)$
$\phi(x) = \sum_{i=0}^{7} c_i B_i(x)$
$G_1 = 5$
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}

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.

KAN Architecture


$\exp(\sin(\pi x) + y^2)$
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
$e^{y^2+\sin(\pi x)}$

Pruning

- They try optimizing the network architecture by removing less important nodes or connections after the network has been trained.
- This process helps in reducing the complexity and size.
- Pruning focuses on identifying and eliminating those parts of the network that contribute minimally to the output.
- This makes the network lighter and potentially more **interpretable**

B Splines

- Splines are the backbone of KAN’s learning mechanism. They replace the traditional weight parameters typically found in neural networks.
- The flexibility of splines allows them to adaptively model complex relationships in the data by adjusting their shape to minimize approximation error, therefore, enhancing the network’s capability to learn subtle patterns from high-dimensional datasets.

B Splines

- Splines are a type of function that are defined by **piecewise polynomials** and are smoothly connected at certain points called **knots.**
- ***Basis function:*** *These are a set of simple functions that can be used to represent complex non-linear functions.*
- A **piecewise polynomial** is a function that uses different polynomial expressions over different intervals of its domain.

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

- *spline(x)* represents the spline function
- c*i* are the coefficients that are optimized during training, and $Bi(x)$ are the B-spline basis functions defined over a grid.
- The grid points define the intervals where each basis function Bi is active and significantly affects the **shape** and **smoothness** of the spline.
- Aka a **hyperparameter** that affects the accuracy of the network. More grids = **more control** & **precision**, also resulting in more parameters to learn.

TLDR;

- Input Layer: Receives raw data.
- Hidden Layers: Apply univariate functions.
- Output Layer: Aggregates and produces the final output

Training of KANs

- During training, the coefficients of these splines are optimized to minimize prediction error.
- Techniques such as gradient descent are often used for this.
- In each iteration, the spline parameters are updated to reduce prediction error, allowing the model to find the most suitable curves for the data.

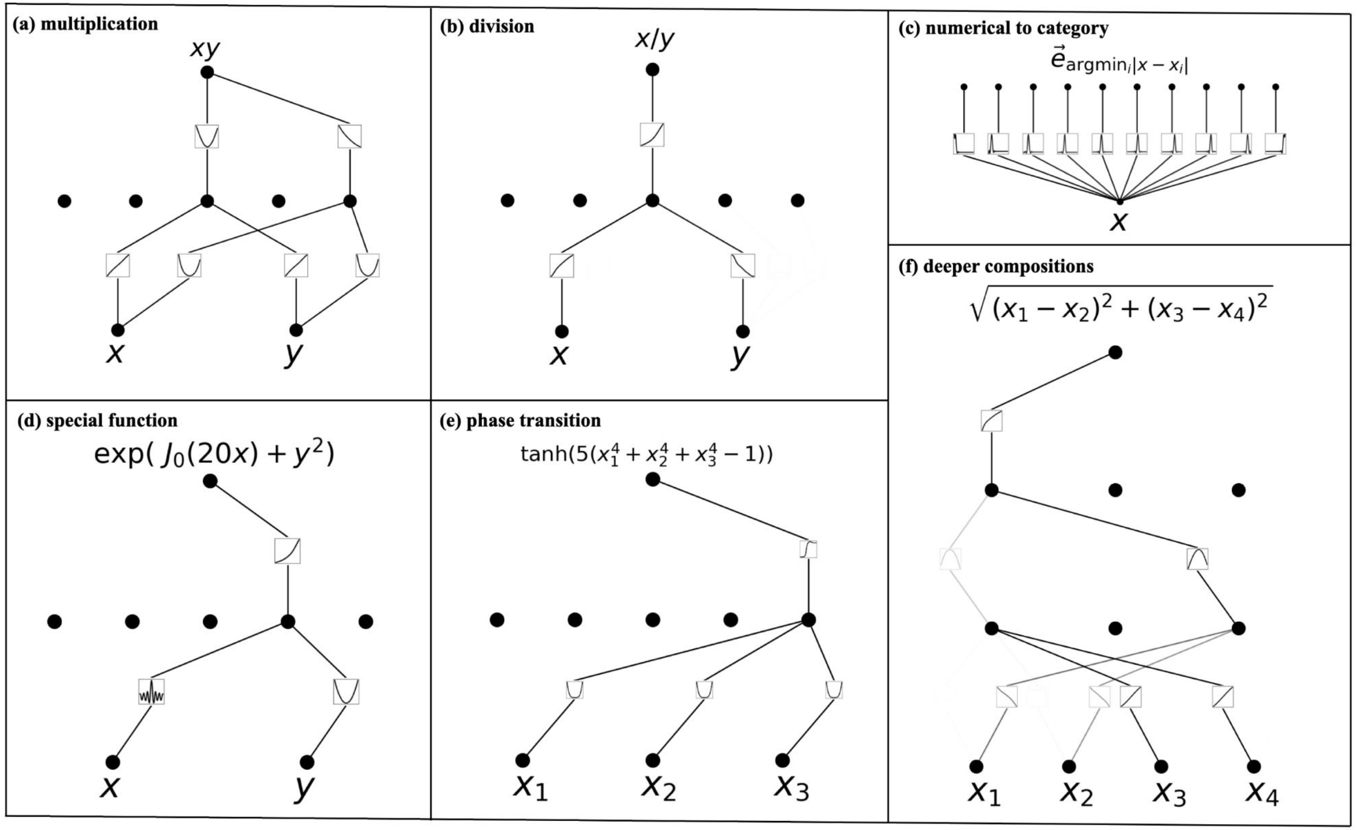
Advantages of KAN

- **Function approximation**: More efficient representation of complex functions.
- **Reduced complexity**: Fewer parameters in certain cases.
- **Interpretability**: Easier to understand the role of each component function.
- Better accuracy than MLP for various tasks, with fewer parameters. The paper argues that KAN has better scaling than MLP in terms of number of parameters needed for the same accuracy.
- Better interpretability by simplifying a KAN using sparsity regularization and pruning techniques.

Applications of KAN

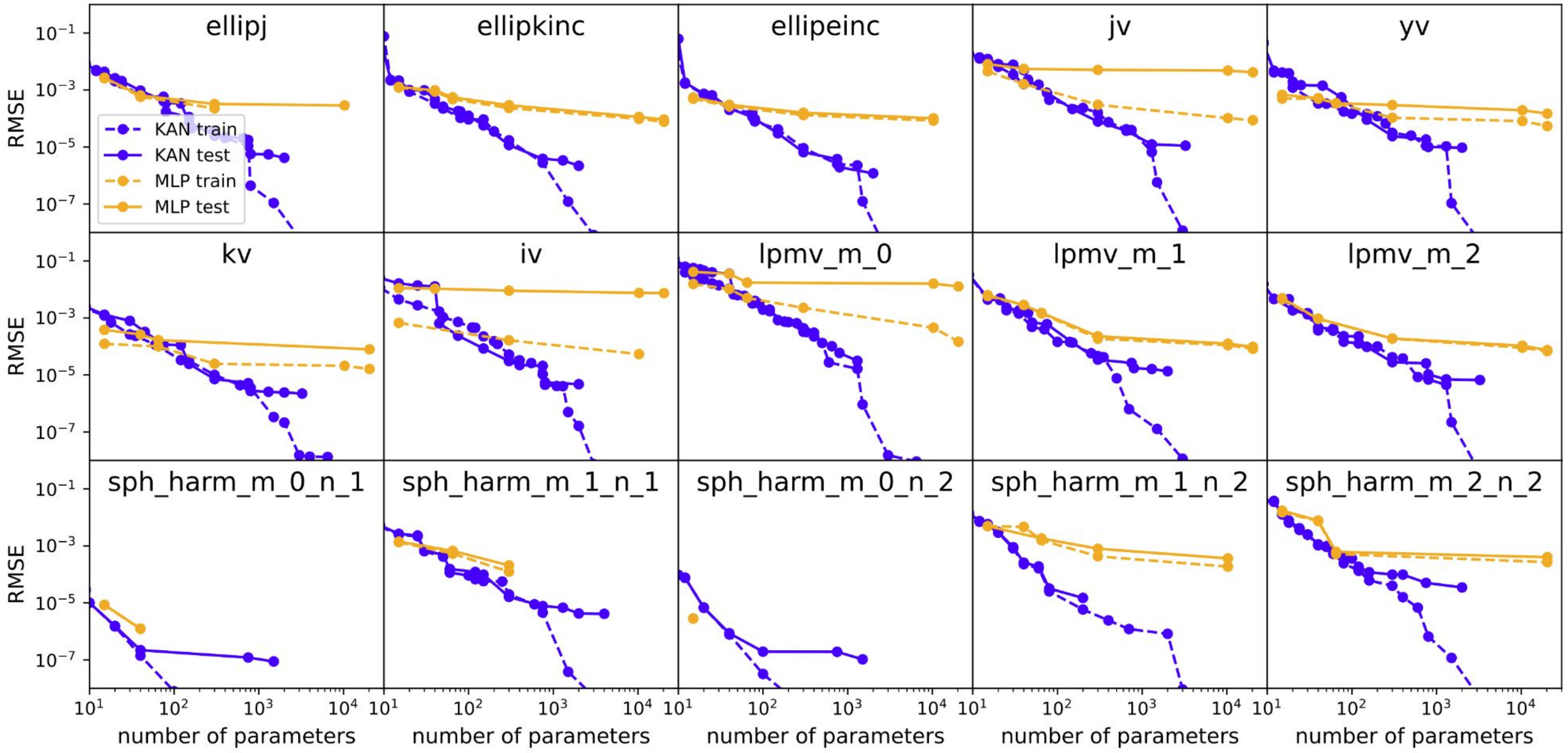
- Signal Processing: Efficiently handling complex signals.
- Data Compression: Compact representation of high-dimensional data.
- Control Systems: Designing controllers with specific functional requirements.
- Case Studies: Brief examples of practical applications.

Symbolic Formulas



Fitting Special Functions

result shows that in almost all of these functions, KANs achieve a **lower train/test loss** having the same number of parameters compared to MLPs



Continual Learning

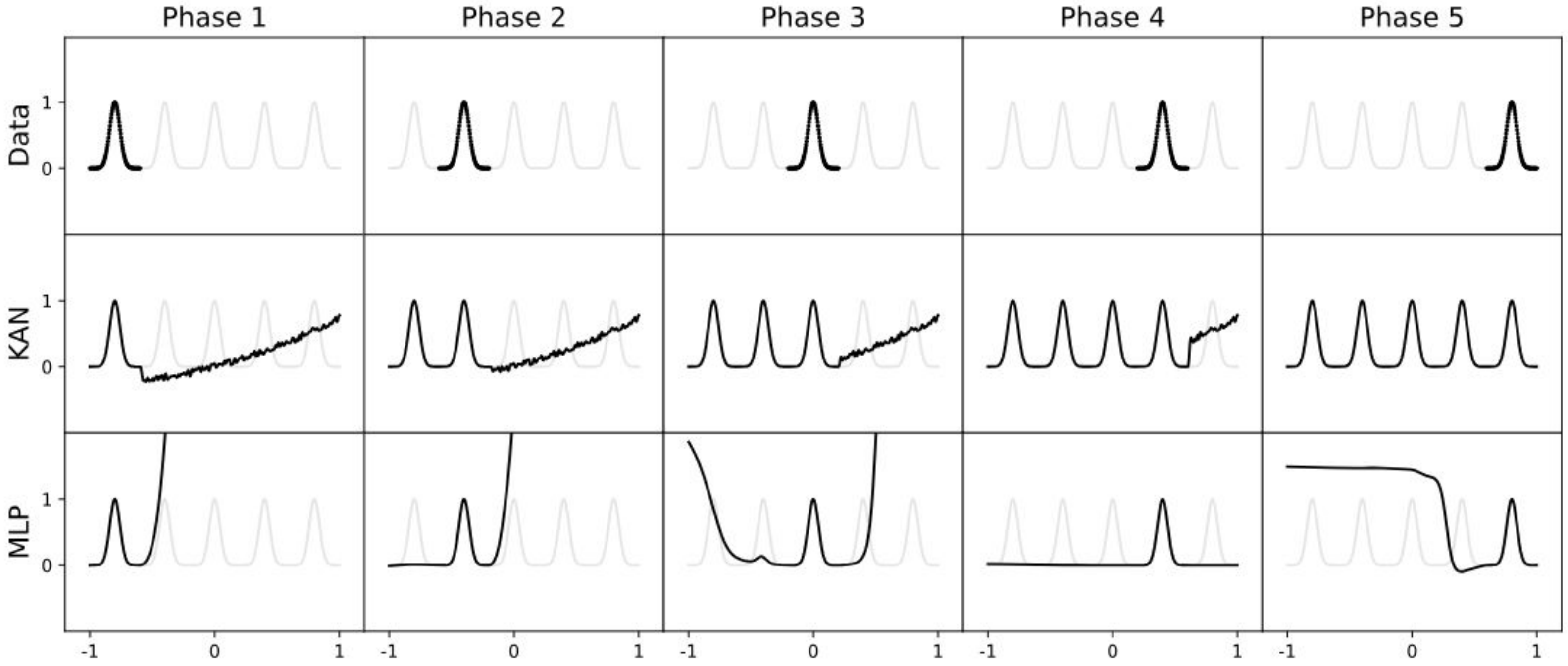


Figure 3.4: A toy continual learning problem. The dataset is a 1D regression task with 5 Gaussian peaks (top row). Data around each peak is presented sequentially (instead of all at once) to KANs and MLPs. KANs (middle row) can perfectly avoid catastrophic forgetting, while MLPs (bottom row) display severe catastrophic forgetting.

WHY CONTINUAL LEARNING

- Continual Learning is the quality of how networks can adapt to new information over time without forgetting previously learned knowledge.
- It is a major challenge in neural network training, particularly in avoiding the problem of **catastrophic forgetting**, where acquiring new knowledge leads to a rapid erosion of previously established information.

WHY CONTINUAL LEARNING

- KANs demonstrate an ability to retain learned information and adapt to new data without catastrophic forgetting, thanks to the **local nature** of spline functions.
- Unlike MLPs, which rely on global activations that might unintentionally affect distant parts of the model, KANs modify only a limited set of nearby spline coefficients with each new sample.

Challenges and Limitations

- **Computational complexity**: High computational cost for large networks.
- **Scalability:** Difficulties in scaling to very large datasets.
- **Comparison**: Performance trade-offs compared to other neural network models.

Practical Implementation

Unsupervised Learning Demo

KANVas Playground

Spline Exploration

Performance on Toy Datasets

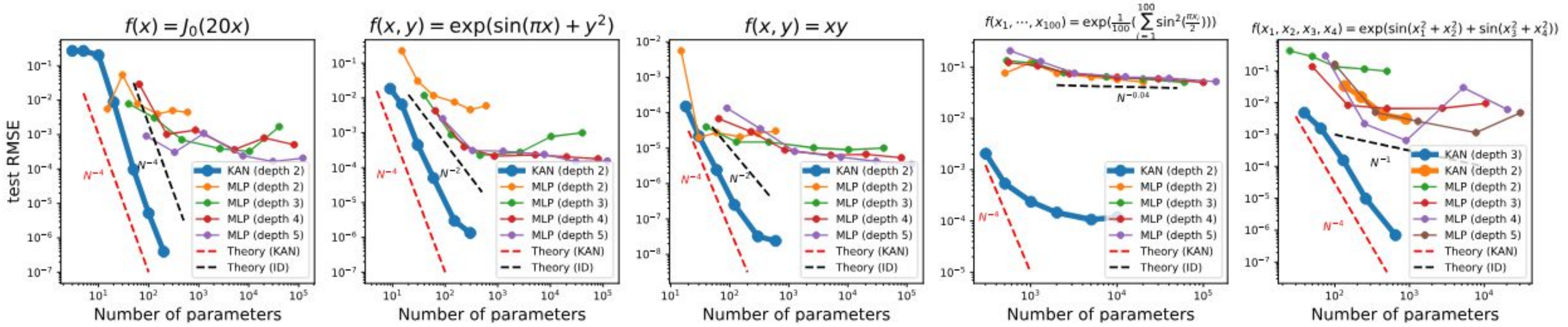


Figure 3.1: Compare KANs to MLPs on five toy examples. KANs can almost saturate the fastest scaling law predicted by our theory ($\alpha = 4$), while MLPs scales slowly and plateau quickly.

TAKEAWAY RESOURCES

- **Recommended readings: research papers, blog posts**
- **Useful tools and libraries: KAN Tutorials, TensorFlow, PyTorch**
- **Community and support: forums, GitHub repositories**
- **ML Twitter (now X) >>**
- **KAN Paper Preprint : https://arxiv.org/abs/2404.19756**

KAN YOU KEEP UP?

IF NOT, TIME FOR Q&A!!!