Adaptive Basis Functions for Enhanced Kolmogorov-Arnold Neural Networks

ABSTRACT

Kolmogorov–Arnold Neural Networks (KANs) are powerful tools for approximating complex functions based on the Kolmogorov–Arnold theorem. However, they face several challenges, such as being inflexible due to fixed basis functions, struggling with high-dimensional data, and requiring significant computational resources. In this paper, we propose a new approach that introduces **adaptive basis functions** to the KAN framework. These basis functions can adjust dynamically during training, allowing the network to better fit complex patterns while reducing computational demands. Our experiments show that this method improves accuracy, handles noisy data more effectively, and generalizes better to new inputs. By addressing these key limitations, our work makes KANs more Practical and versatile for solving real-world problems, introducing Adaptive Basis Functions into the KAN resulted in smoother and faster convergence, reduced overfitting, and lower loss of approximately 7.59% over the noise-introduced sine curve.

Introduction

The ability of neural networks to approximate functions has long been an endeavor in the field of computer science. In the field of mathematical modeling, there exists a universal function approximation theorem that enables the replication of patterns and complex systems, as well as the creation of predictions based on these models. Multi-layer Perceptrons (MLPs) utilize the Universal Approximation Theorem. Introduced by Liu et al, Kolmogorov-Arnold Neural Networks emerged as powerful in the industry by utilizing the Kolmogorov-Arnold Representation theorem [1], which decomposes multivariate functions into the sum of singular univariate functions.

Where $\phi_{q,p}: [0,1] \to \mathbb{R}$ and $\Phi_q: \mathbb{R} \to \mathbb{R}$

This approach effectively tackles the curse of dimensionality, a phenomenon where the computational cost of expanding neural networks grows exponentially with the number of input dimensions. By decomposing a multivariate, continuous function into smaller functions, the corresponding high-dimensional relationship between variables decomposes into a structured combination of low-dimensional connections. This not only saves the storage cost of large parameter spaces but also improves interpretability and reduces overfitting risks.

Approximating real-valued functions that are too complicated or unmanageable for direct evaluation is a frequent problem in computational economics and numerical analysis. Applications where precise solutions are frequently impossible, such as asset pricing and dynamic optimization, frequently call for function approximation. The process of approximating a function f involves selecting a computationally feasible approximant $\hat{f}$, and two common problems emerge: interpolation and functional equation solving.

Interpolation refers to the process of constructing an approximant $\hat{f}$ that matches the known values or derivatives of the function f at specific data points. Modern interpolation techniques extend beyond simple table-based approximations, focusing on optimal data extraction and computational efficiency. In contrast, the functional equation problem entails finding a function f that satisfies a functional equation $Tf = g$, where T is an operator mapping functions to functions and g is known. This type of problem is frequently encountered in dynamic economic models, such as Bellman and Euler equations.

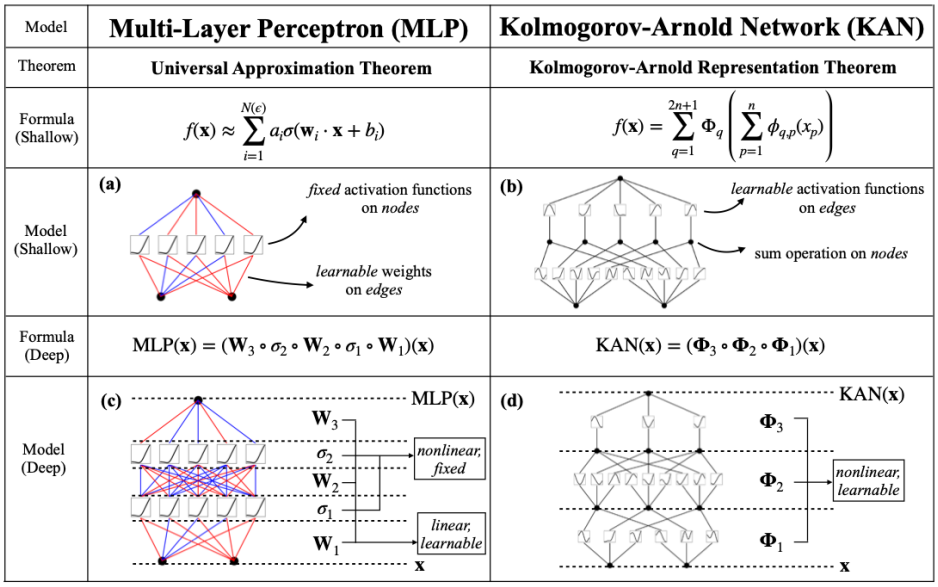

Figure 1. A table showing the contrast between typical MLP and KAN characteristics [1]

Function approximation methods like polynomial and spline interpolation, as well as more advanced techniques like collocation methods, are essential tools in addressing both of these problems. This discussion explores the principles of interpolation, the benefits of different polynomial bases, and the advantages of Chebyshev nodes, which have been shown to significantly improve the accuracy and stability of approximations in comparison to traditional methods of *Liu et al.* [1] and *Boston College* [4].

Traditional implementations of KANs, however, rely on static-based functions, which hampers their ability to capture non-linear relationships in real-world data. To address these limitations, this paper proposes integrating adaptive basis functions into KANs. By dynamically changing basis functions in the training process, KANs can recognize complex, non-linear relationships between the feature and target.

Adaptive Basis Functions (ABFs) solve the problem of inflexible basis functions by introducing learnable parameters. If a true function has irregular, non-linear relationships between the variables. Our introduction of Radial Basis Functions built upon the normal distribution dynamically fine-tunes to optimize the center and standard deviation based on the dataset and the true function.

Secondly, fixed basis functions can lead to consistent approximation error, leading to high bias if their form is mismatched to the target function. Because these functions are predetermined and static, they cannot adapt their parameters to fit the nonlinear complexities in datasets. This problem often forces the model to apply inappropriate biases, leading to systematic underfitting. For example, static polynomial basis functions will struggle with capturing periodic patterns even if the polynomial degree explodes, which can further exacerbate unstable training and poor generalization. Adaptive Basis Functions solve this problem by introducing trainable parameters into basis functions that can assist in capturing non-linear patterns, hence improving generalization and, as per the example, capturing periodic patterns.

This paper introduces Adaptive Basis Functions into KANs. The proposed approach not only improves approximation accuracy but also offers insights into the interplay between adaptability and efficiency in neural network design.

Related Works

Kolmogorov-Arnold Superposition and Representation Theorem

(KAST & KART)

Hilbert's 13th problem proposed that any continuous function on any finite interval could be represented as a finite sum of continuous functions or other forms, such as polynomials. The challenge was finding a **universal approximation** of any continuous function, as seen in Liu et al. [1] and [3]. Kolmogorov defined that on a compact interval, such as [0,1], continuous functions could be expressed on the intervals of two-variable functions.

$$f(x) = \sum_{i=1}^{n} \psi(x, y_i)$$

The original version of the KA representation theorem states that for any continuous function $f: [0,1]^d \to \mathbb{R}$, there exist univariate functions such that

$$f(x_1, \dots, x_d) = \sum_{q=0}^{2d+1} g_q\left(\sum_{q=0}^{d} \psi_{p,q}(x_p)\right)$$

This means that the $(2d+1)(2d)$ Univariate functions g_q and $\psi_{p,q}$ are enough for an exact representation of a d-variate function. Kolmogorov published the result in 1957, disproving the statement of Hilbert's 13th problem, that is concerned with the solution of algebraic equations. The earliest proposals in the literature introducing multiple layers in neural networks date back to the sixties, and the link between KA representation and multilayer neural networks occurred much later[2]. Kolmogorov and Arnold show that every continuous multivariate function can be represented as a superposition of continuous univariate functions and addition in a universal form and thus solve the problem positively. In Kolmogorov's representation, only one univariate function (the outer function) depends on it, and all the other univariate functions (inner functions) are independent of the multivariate function to be represented in [3].

KAN Architecture

KANs are specifically designed to follow the Kolmogorov-Arnold Representation Theorem (KART), which decomposes multivariate functions into the sum of univariate functions. This setup provides a way to where each layer has a matrix of 1D function parametrized as B-spline curves with trainable coefficients. The layers connect input and output through summation operations on nodes and edges, as seen in [1].

$$f(x) = f(x_1, \dots, x_d) = \sum_{p=0}^{2d+1} \Phi_q\left(\sum_{q=0}^{d} \psi_{p,q}(x_p)\right)$$

The Kolmogorov-Arnold Representation can be represented in matrix form

$$(\Phi_{\text{out}} \circ \Phi_{\text{in}})(x) = f(x)$$

Where:

$$\Phi_{\text{in}} = \begin{pmatrix} \psi_{1,1} & \cdots & \psi_{1,d} \\ \vdots & \ddots & \vdots \\ \psi_{2d+1,1} & \cdots & \psi_{2d+1,d} \end{pmatrix}, \Phi_{\text{out}} = (\Phi_1, \dots, \Phi_{2d+1})$$

We notice that Φ_{in} and Φ_{out} are special cases of the following function matrix Φ with n_{in} inputs and n_{out} outputs. As per Xiaoming [6], this is recognized as a Kolmogorov-Arnold Layer

$$\Phi = \begin{pmatrix} \psi_{1,1} & \cdots & \psi_{1,n_{in}} \\ \vdots & \ddots & \vdots \\ \psi_{n_{out},1} & \cdots & \psi_{n_{out},n_{in}} \end{pmatrix}$$

Deeper KANs are generated by stacking layers of such, each as $\Phi = \{\phi_{p,q}\}$ such that p and q are input and output dimensions, respectively. These layers process inputs through differentiable transformation, making them suitable for backpropagation. The shape of a KAN is represented by $[n_0, n_1, \ldots, n_L]$ where n_i represents the number of nodes in the i-th layer, as seen in [1]. The output $\hat{y}$ of a KAN model is, therefore, the composition of all layers inputting x.

$$(\Phi_1 \circ \Phi_2 \circ \ldots \circ \Phi_{\text{L}})(\mathbf{x}) = \text{KAN}(\mathbf{x})$$

Such that Φ_L represents the whole function 'set' of the L-th layer as per the KART, where the function set is $\psi_{p,q}$.

In contrast, MLPs are composed of linear layers W_L and non-linear activation functions represented as σ.

$$\text{MLP}(\mathbf{x}) = (W_1 \circ \sigma \ldots \circ W_L)(\mathbf{x})$$

KANs can be easily visualized by representing fully connected layers with 1D connections and a trainable spline curve at each end. Below is Xiaoming's demonstration of fitting a KAN over a noisy sine curve [6].

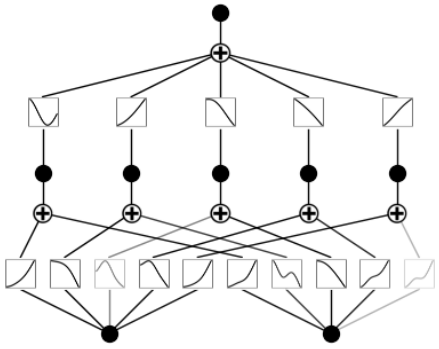

Figure 2. Representation of KAN Network under sparse regularization by Xiaoming [6]

The black circles represented by the + sign show a summation operation. Adhering to the KART, this showcases the summation of univariate functions denoted as $\psi_{p,q}$. This method is not trainable, as it simply aggregates all outputs and inputs the data into upcoming nodes.

KAN Optimization and Training

KAN introduces several optimization techniques to improve training. Residual activation functions combine B-spline approximations with a smooth residual component, enabling better learning dynamics. During training, spline grids dynamically adjust to maintain stability as activations change. The trainability of the network is further improved via parameter initialization, which includes techniques like Xavier initialization. Compared to MLPs, KANs may appear to require more parameters, but this expense is mitigated by their capacity to achieve high generalization with fewer nodes per layer. KANs are excellent in representing and interpreting high-dimensional functions; in some applications, they outperform traditional structures thanks to their special structure [1].

KANs share completely connected structures with MLPs. However, KANs place learnable activation functions on edges (also known as "weights"), whereas MLPs place fixed activation functions on nodes (also known as

"neurons") [1]. Activation functions are treated as weights such that they are learnable, making the functional relationship between nodes more adaptive and stable, as shown by Liu et al. [1] and Schmidt-Hieber [2].

As a result, KANs have no linear weight matrices at all; instead, each weight parameter is replaced by a learnable 1D function parametrized as a spline. KANs' nodes simply sum incoming signals without applying any non-linearities. One might worry that KANs are hopelessly expensive since each MLP's weight parameter becomes KAN's spline function. Fortunately, KANs usually allow much smaller computation graphs than MLPs, as seen in [1].

Adaptive Basis Function

As per Adams in [5], in the context of machine learning, basis functions are mathematical transformations applied to input data, enabling linear models to approximate more complex relationships. These transformations map input features into a higher-dimensional space, allowing regression and classification models to capture nonlinear patterns. Basis functions can include simple polynomials, Fourier series that represent periodic structures, radial basis functions (RBFs) that capture localized changes, or piecewise linear transformations. They play a crucial role in making data linearly separable or better suited for predictive modeling by effectively altering the representation of input features, as shown by Liu et al. [1]. These basis functions play a role in making data and patterns linearly separable, which may fail on real-world datasets.

Adaptive Basis Functions (ABFs) are functions used in mathematical modeling to adapt to data. In contrast to fixed basis functions, such as splines, ABFs change based on the problem at hand. ABFs are not constrained to a predefined form. Instead, their structure evolves during the training phase to fit the true function. This adaptivity allows ABFs to capture complex, nonstationary patterns more effectively than fixed transformations, reducing bias that arises from model misspecification. By enabling the basis functions themselves to be learned alongside other model parameters, ABFs provide a flexible mechanism for function approximation that is especially valuable in domains where prior assumptions about data smoothness, locality, or periodicity may be inadequate or unknown.

Examples of basis functions include simple polynomials, Fourier series, radial basis functions (RBFs), or piecewise linear transformations. Each type of basis function introduces a unique way to represent input features, offering flexibility in addressing a variety of modeling challenges. For instance, polynomial basis functions are often used to capture trends in data with curved patterns, while RBFs excel in localized transformations that emphasize certain regions of the input space. Each type of basis function introduces a distinct approach to representing input features, offering different trade-offs in terms of smoothness, locality, and computational complexity. For instance, polynomial basis functions can efficiently capture broad, curved relationships but may suffer from instability in higher degrees (Runge's phenomenon), while RBFs excel in modeling localized variations and are robust to outliers in distant regions of the input space. This diverse toolkit enables practitioners to tailor their models to the specific structure of the learning problem and to balance interpretability with approximation power.

The role of basis functions is particularly critical in enhancing the separability of data. By altering the representation of input features, basis functions enable machine learning algorithms to model relationships that would otherwise be difficult or impossible to capture using linear transformations alone. This principle underpins the effectiveness of kernel-based methods like Support Vector Machines (SVMs) and Gaussian Process Regression, where basis functions are implicitly defined through kernel functions 1.

ABFs use differentiable methods to process inputs, which can be set up for training through backpropagation. ABFs are represented as a set of parameters like the b-spline, which sets parameters as control points, slopes, and degrees of polynomials. ABFs can have location and shape parameters similar to Gaussian Functions with standard deviation (σ).

Methodology

This experiment aims to investigate the performance of two types of KANS-one with a fixed basis function and another with an adaptive basis function. Both will be targeted for effectiveness in approximation and accuracy using mean-squared error over a sine function in the range from [0,10] and will use the Adam optimizer.

Both KANs have a control architecture with an input layer, a basis function, and a linear output layer.

Control Experiment

The control experiment will use the Gaussian Radial Basis Function (GRBF). The GRBF is represented as such:

$$\phi_i(r) = e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}$$

Where σ is the parameter that controls the width of the curve. The GRBF is a smooth bell shape and is commonly used for interpolation and kernel methods. This function is a default basis function for a KAN. This basis function is a universal function approximator; a linear combination of Gaussian functions can arbitrarily approximate any continuous function given parameters. GRBFs are smooth and differentiable, making them suitable for convergence as a default for many KANs.

The GBRF has fixed mean and standard deviation parameters, which are not refined during the training process. The input is passed through the GBRF and outputs the weighted sum of the basis function outputs.

Adaptive KAN

In the Adaptive KAN, the centers and width are dynamically adjusted in backpropagation. The Adaptive Gaussian Radial Basis Functions (aGRBFs). These parameters are refined to adapt over time, which enhances function approximation and detection of non-linear relationships.

Dataset

The dataset comprises of a sine wave data using Gaussian noising. The input values range from $[0,10]$, and the corresponding y-values are outputted from the function $y = \sin(x) + \varepsilon$ such that ε is Gaussian noising to simulate real-world datasets. The data is divided into training, validation, and test sets.

Training

The model is trained using Mean Squared Error over the Adam optimizer. Both experiments are trained over 100 epochs on a batch size of 32. For the control experiment, the GRBFs' centers and standard deviations are randomly initialized, and the second experiment's aGRBFs' are randomly initialized and then optimized during the training process.

Results

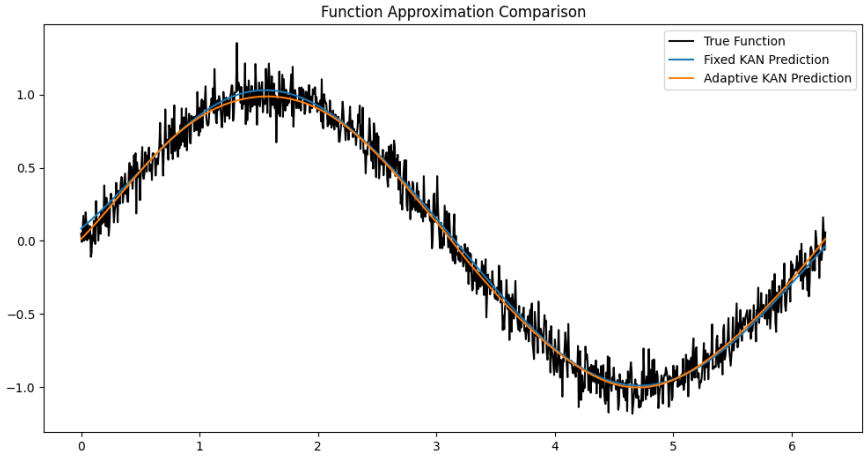

Figure 3. Function approximation between Fixed KAN and Adaptive KAN

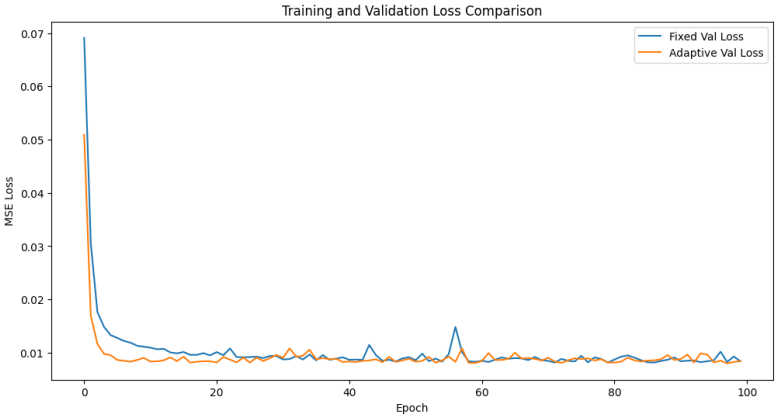

Figure 4. Validation loss between both models over 100 epochs

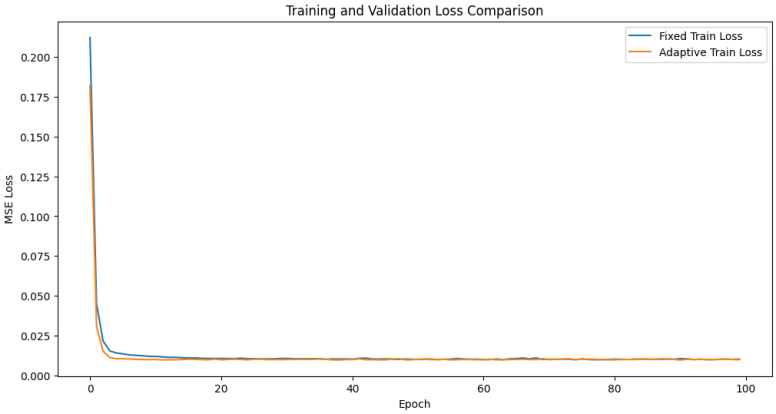

Figure 5. Training loss between both models over 100 epochs

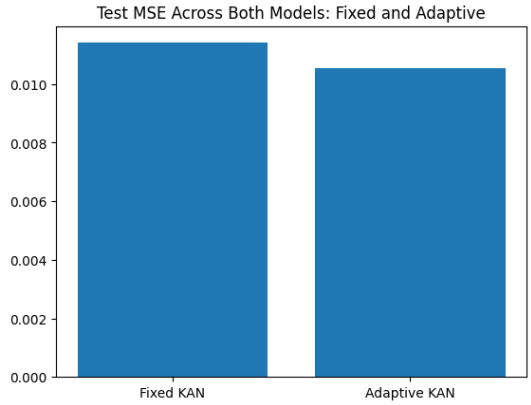

Figure 6. Difference in Test MSE across both models

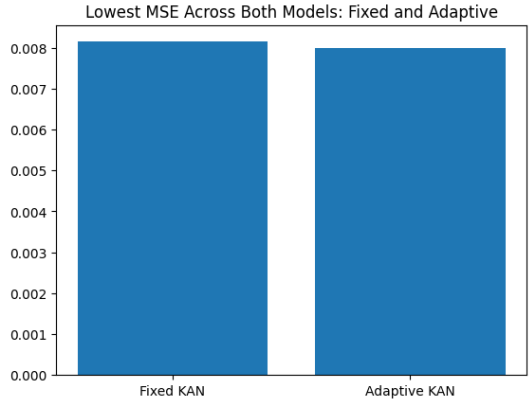

Figure 7. Difference in Lowest MSE across both models

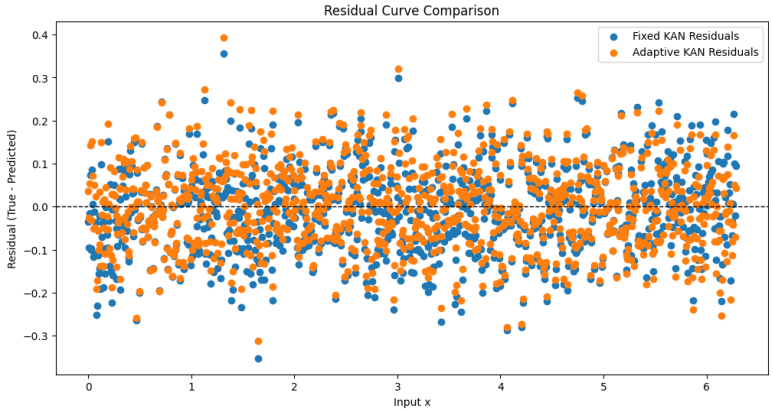

Figure 8. Residual Plot of Fixed and Adaptive

Discussion

As seen in Figures 6 and 7, it is evident that utilizing Adaptive Basis Function resulted in an approximately 7.59% drop in Test MSE and approximately 2% in the lowest MSE between the models. This is due to the larger parameter space that allowed the Adaptive KAN to generalize to a lower MSE and capture more complex non-linear relationships. The increased flexibility of the Adaptive Basis Function enables the model to adjust the shape of the kernel functions during training, effectively fitting intricate patterns.

The power of Adaptive Basis Function is showcased in Figures 4 & 5, where, although it is shown that Adaptive KAN and Fixed KAN converge to the same loss during training, the Adaptive KAN performance is considerably better during validation in comparison to the Fixed KAN. Additionally, Adaptive KAN validation performance contains a lower magnitude of abrupt fluctuations compared to the Fixed KAN. For example, around epoch 60, the validation of the Fixed KAN jumps rather than continuing the same path.

Limitations

More parameters can lead to a higher risk of overfitting.

Future Works

Conclusion

Kolmogorov–Arnold Neural Networks (KANs) enhanced with Adaptive Basis Functions demonstrate superior performance compared to regular KANs. The incorporation of adaptive basis functions leads to smoother and faster learning processes, which in turn improve generalization. These improvements highlight the potential of adaptive techniques in optimizing KANs for more efficient and effective applications. Future studies could explore further refinements to the adaptive basis function approach, as well as its applicability across different domains, to fully realize its potential in enhancing neural network performance.

Citations

[1] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T.Y Hou, M. Tegmark, "*KAN: Kolmogorov–Arnold Networks* " vol. arXiv:2404.19756, 2024

[2] J. Schmidt-Hieber, "*The Kolmogorov-Arnold representation theorem revisited*" vol. arXiv: 2007.15884, 2020

[3] Liu et al., "*Kolmogorov superposition theorem and its applications"* 2015

[4] Boston College, "*Function Approximation"*

[5] R.P. Adams, "*Features and Basis Functions"* Princeton University, Dept. of Computer Science, 2018

[6] Xiaoming et al, "pykan" GitHub https://github.com/KindXiaoming/pykan/blob/master/hellokan.ipynb