Kolmogorov–Arnold Networks Research Report

Author: Pham Huy Chau Long
Date: July 15, 2024

Why Review This Paper?

I have been working with this network for a time, but there are some limitations that have not been resolved, preventing further progress with KAN.

Table of Content

Content

A. KAN and its main advantages

- Networks are inspired by the Kolmogorov-Arnold Representation Theorem, which describes how a multivariate continuous function can be represented as a superposition of functions of one variable. This theoretical backing suggests a different architectural approach where the activation functions themselves are subject to learning and adaptation, potentially allowing for more flexibility in modeling complex relationships.

A-1. Idea

When compare with MLPs, KANs have been developed based on Kolmogorov-Arnold Theorem (KAT). In contrast, MLPs is using Universal Approximation Theorem (UAT)

Kolmogorov-Arnold Theorem (KAT):

- states that every multivariate continuous function $f : [0, 1]^n \to \mathrm{R}$ can be represented as a superposition of the two-argument addition of continuous functions of one variable.

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

where $\varphi_{q,p} : [0, 1] \to \mathrm{R}$ are univariate continuous functions and $\Phi_q : \mathrm{R} \to \mathrm{R}$ are also continuous functions. This decomposition uses the binary operation of addition.

Universal Approximation Theorem (UAT):

- states that any function can be approximated by a wide enough 2-layer neural network.

$$f(x) = \sum_{q=1}^{\infty} w_q^{(2)} \sigma \left(\sum_{p=1}^{n} w_{q,p}^{(1)} x_p \right)$$

where $W_2 = [w_q^{(2)}]_{q=1}^{\infty}$ and $W_1 = [w_{q,p}^{(1)}]_{q=1}^{\infty p=1}$

I wrote the MLP in terms of summation instead of matrix multiplication to draw parallels between UAT and KAT. There are two main differences between UAT and KAT

Conclusion When Experiment:

- With KAT is about needing only $2n + 1$ units. But when experiment, I see that the KAN just use KAT to split out the original equation into univariate continuous functions. The constant $2n + 1$ does not necessary.
- With the large dimension of data, using $2n + 1$ hidden units could make the model too slow. Because the complexity of KANs will increase drastically faster than MLPs with the same size.
- With KAT, KAN could make it possible to interpret.

A-2. Architecture

- $MLP\ Model$:
  - Consist of layers of neurons where each neuron in one layer is connected to every neuron in the next layer through fixed, but learnable, weights. The activation functions in MLPs are typically static and not part of the learning process.
  - Use fixed activation functions (e.g., sigmoid, tanh, ReLU) which are chosen prior to training and do not change.
- $KAN\ Model$:

- KANs may use a flexible architecture where the activation functions are not predetermined and learnable during training. This can include using functions like splines as part of the network's learnable parameters, enabling the network to adapt more complex functional mappings.

- In paper, author proposed the KAN Layer:

$$\phi(x) = w(\text{silu}(x) + \text{spline}(x))$$

$$\text{spline}(x) = \sum_{i=1}^{G} c_i B_i(x).$$

⟹ In KANs, the activation functions are not predetermined but are learned during the training process, which introduces flexibility in modeling complex data patterns.

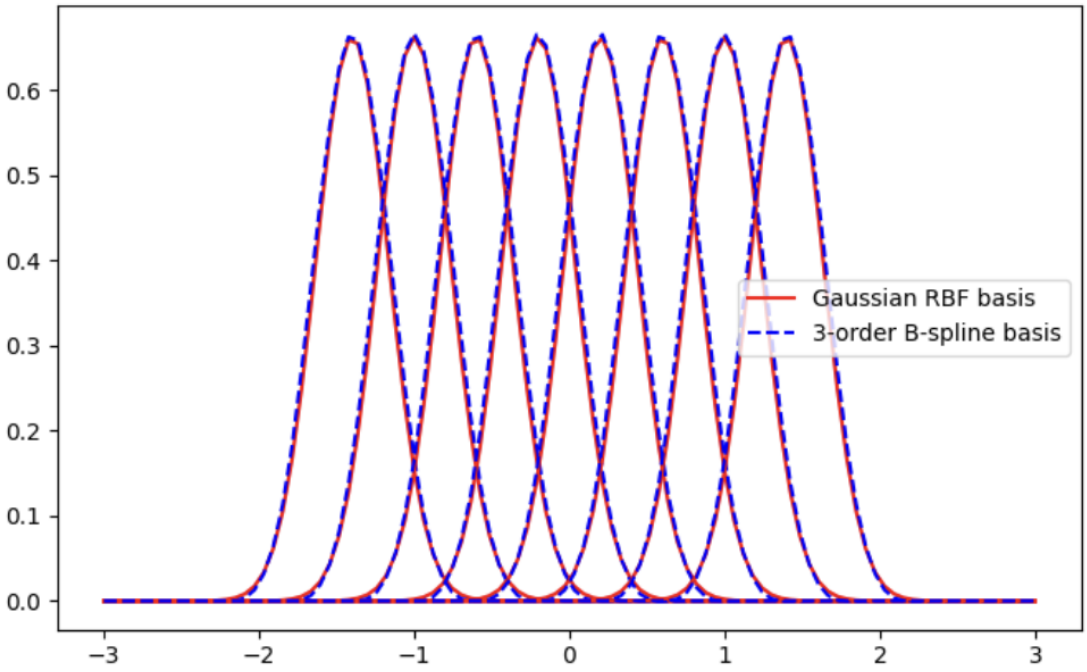
⟹ However, this flexibility comes at a cost. Unlike static, predetermined activation functions in MLPs that allow for optimized batch tensor operations and are highly efficient on parallel computation hardware like GPUs, the dynamic nature of KANs' activation functions prevents such optimizations. The trade-off between the enhanced modeling capabilities and computational efficiency is significant, especially in scenarios where processing speed is critical.

B. Paper Improvement

B-1. FastKAN

- This paper proposes replacing `B-Spline` with `Gaussian Radial Basis Functions` because these functions can be calculated more efficiently than B-Splines. This could lead to a more optimized model complexity while achieving similar approximation results.

$$b_i(u) = \exp\left(-\left(\frac{u - u_i}{h}\right)^2\right)$$



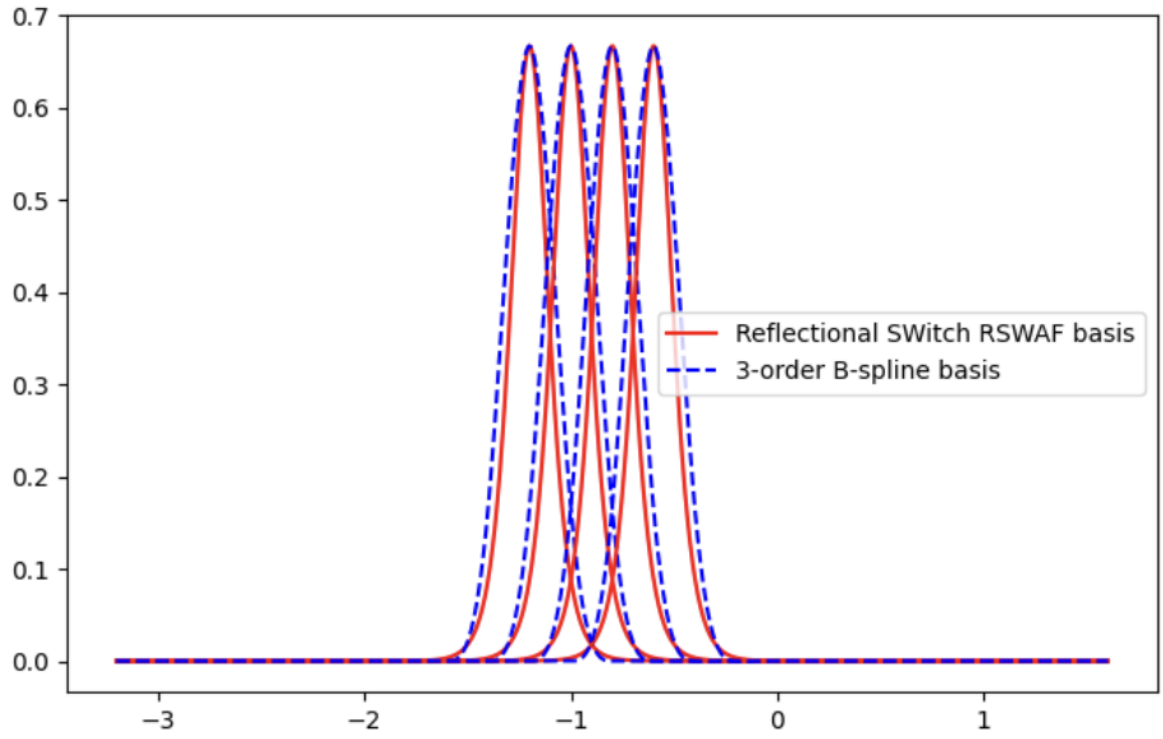
- Using LayerNorm to scale inputs to the range of spline grids, so there is no need to adjust the grids. This also reduce computational time by skip grid adapting step.

⟹ Then FastKAN is 3.33x in forward speed

B-2. FasterKAN

- The improvement for `FastKAN`, this paper using `Reflectional SWitch Activation Function` instead of `Gaussian Radial Basis Functions`

$$b_i(u) = 1 - (\tanh(\frac{u - u_i}{h}))^2$$



Why it works ?

- **Reflection and Quadratic Structure of TANH()**:
  - `u_i` are the knot points and `h` is a modulation parameter, which can be considered as a basis function for B-Spline.
  - Using the tanh function and squaring it creates a boundary (0,1) similar to the B-Spline basis function.
  - This function is symmetrical around the point `u_i`. This means the function values will be the same when moved at equal distances on both sides of `u_i`, thus creating a smooth transition.

$$b_i(u+\delta) = b_i(u-\delta)$$

- **Adjustment of Steepness and Center**:
  - The parameter `h` adjusts the steepness of the function, i.e., how quickly the tanh function changes, similar to adjusting the curves of a B-Spline.
  - The parameter `u_i` determines the central position of the function, corresponding to the knot points in a B-Spline. By adjusting these values, the function can emulate different segments of a B-Spline curve.

$\Longrightarrow$ The forward time of FaskerKAN is 1.5x faster FastKAN

C. Experiment and Conclusion

- You could use model in this colab link: EXPERIMENT

- We will use the best model of KAN to compare with MLP to make a conclusion.

MLP MODEL WITH FEATURE EXTRACTOR

```
Total parameters: 3272843
Trainable parameters: 3272843
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 28, 28]             320
              ReLU-2           [-1, 32, 28, 28]               0
       BatchNorm2d-3           [-1, 32, 28, 28]              64
         MaxPool2d-4           [-1, 32, 14, 14]               0
           Dropout-5           [-1, 32, 14, 14]               0
            Conv2d-6           [-1, 64, 14, 14]          18,496
       BatchNorm2d-7           [-1, 64, 14, 14]             128
            Conv2d-8           [-1, 64, 14, 14]          36,928
       BatchNorm2d-9           [-1, 64, 14, 14]             128
           Conv2d-10           [-1, 64, 14, 14]          36,928
      BatchNorm2d-11           [-1, 64, 14, 14]             128
           Conv2d-12           [-1, 64, 14, 14]          36,928
      BatchNorm2d-13           [-1, 64, 14, 14]             128
             ReLU-14           [-1, 64, 14, 14]               0
           Conv2d-15           [-1, 64, 14, 14]          36,928
      BatchNorm2d-16           [-1, 64, 14, 14]             128
             ReLU-17           [-1, 64, 14, 14]               0
    BasicResBlock-18           [-1, 64, 14, 14]               0
AdaptiveAvgPool2d-19             [-1, 64, 1, 1]               0
           Linear-20                   [-1, 4]             256
             ReLU-21                   [-1, 4]               0
           Linear-22                  [-1, 64]             256
          Sigmoid-23                  [-1, 64]               0
          SEBlock-24           [-1, 64, 14, 14]               0
         MaxPool2d-25             [-1, 64, 7, 7]               0
           Dropout-26             [-1, 64, 7, 7]               0
            Conv2d-27            [-1, 128, 7, 7]          73,856
            Conv2d-28            [-1, 128, 7, 7]           1,280
            Conv2d-29            [-1, 128, 7, 7]          16,512
DepthwiseSeparableConv-30            [-1, 128, 7, 7]               0
             ReLU-31            [-1, 128, 7, 7]               0
           Conv2d-32            [-1, 256, 7, 7]         295,168
      BatchNorm2d-33            [-1, 256, 7, 7]             512
           Conv2d-34            [-1, 256, 7, 7]         590,080
      BatchNorm2d-35            [-1, 256, 7, 7]             512
           Conv2d-36            [-1, 256, 7, 7]         590,080
      BatchNorm2d-37            [-1, 256, 7, 7]             512
           Conv2d-38            [-1, 256, 7, 7]         590,080
      BatchNorm2d-39            [-1, 256, 7, 7]             512
             ReLU-40            [-1, 256, 7, 7]               0
           Conv2d-41            [-1, 256, 7, 7]         590,080
      BatchNorm2d-42            [-1, 256, 7, 7]             512
             ReLU-43            [-1, 256, 7, 7]               0
```

```
    BasicResBlock-44            [-1, 256, 7, 7]               0
AdaptiveAvgPool2d-45            [-1, 256, 1, 1]               0
           Linear-46                  [-1, 16]           4,096
             ReLU-47                  [-1, 16]               0
           Linear-48                 [-1, 256]           4,096
          Sigmoid-49                 [-1, 256]               0
          SEBlock-50            [-1, 256, 7, 7]               0
        MaxPool2d-51            [-1, 256, 3, 3]               0
          Dropout-52            [-1, 256, 3, 3]               0
           Conv2d-53            [-1, 256, 2, 2]         262,400
           Conv2d-54             [-1, 32, 2, 2]           8,224
           Conv2d-55             [-1, 32, 2, 2]           8,224
           Conv2d-56            [-1, 256, 2, 2]          65,792
    SelfAttention-57            [-1, 256, 2, 2]               0
AdaptiveAvgPool2d-58            [-1, 256, 1, 1]               0
           Linear-59                  [-1, 10]           2,570
================================================================
Total params: 3,272,842
Trainable params: 3,272,842
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 3.71
Params size (MB): 12.48
Estimated Total Size (MB): 16.19
```

FasterKAN WITH FEATURE EXTRACTOR

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
           Conv2d-1           [-1, 32, 28, 28]             320
             ReLU-2           [-1, 32, 28, 28]               0
      BatchNorm2d-3           [-1, 32, 28, 28]              64
        MaxPool2d-4           [-1, 32, 14, 14]               0
          Dropout-5           [-1, 32, 14, 14]               0
           Conv2d-6           [-1, 64, 14, 14]           2,048
      BatchNorm2d-7           [-1, 64, 14, 14]             128
           Conv2d-8           [-1, 64, 14, 14]          18,432
      BatchNorm2d-9           [-1, 64, 14, 14]             128
          Conv2d-10           [-1, 64, 14, 14]          36,864
     BatchNorm2d-11           [-1, 64, 14, 14]             128
    BasicResBlock-12           [-1, 64, 14, 14]               0
AdaptiveAvgPool2d-13             [-1, 64, 1, 1]               0
          Linear-14                    [-1, 4]             256
            ReLU-15                    [-1, 4]               0
          Linear-16                   [-1, 64]             256
         Sigmoid-17                   [-1, 64]               0
         SEBlock-18           [-1, 64, 14, 14]               0
       MaxPool2d-19             [-1, 64, 7, 7]               0
         Dropout-20             [-1, 64, 7, 7]               0
```

```
           Conv2d-21             [-1, 64, 5, 5]             640
           Conv2d-22            [-1, 128, 5, 5]           8,320
DepthwiseSeparableConv-23            [-1, 128, 5, 5]               0
             ReLU-24            [-1, 128, 5, 5]               0
           Conv2d-25            [-1, 256, 5, 5]          32,768
      BatchNorm2d-26            [-1, 256, 5, 5]             512
           Conv2d-27            [-1, 256, 5, 5]         294,912
      BatchNorm2d-28            [-1, 256, 5, 5]             512
           Conv2d-29            [-1, 256, 5, 5]         589,824
      BatchNorm2d-30            [-1, 256, 5, 5]             512
    BasicResBlock-31            [-1, 256, 5, 5]               0
AdaptiveAvgPool2d-32            [-1, 256, 1, 1]               0
           Linear-33                  [-1, 16]           4,096
             ReLU-34                  [-1, 16]               0
           Linear-35                 [-1, 256]           4,096
          Sigmoid-36                 [-1, 256]               0
          SEBlock-37            [-1, 256, 5, 5]               0
        MaxPool2d-38            [-1, 256, 2, 2]               0
          Dropout-39            [-1, 256, 2, 2]               0
           Conv2d-40             [-1, 32, 2, 2]           8,224
           Conv2d-41             [-1, 32, 2, 2]           8,224
           Conv2d-42            [-1, 256, 2, 2]          65,792
    SelfAttention-43            [-1, 256, 2, 2]               0
AdaptiveAvgPool2d-44            [-1, 256, 1, 1]               0
EnhancedFeatureExtractor-45                 [-1, 256]               0
        LayerNorm-46                 [-1, 256]             512
ReflectionalSwitchFunction-47               [-1, 256, 8]               0
     SplineLinear-48                 [-1, 128]         262,144
   FasterKANLayer-49                 [-1, 128]               0
        LayerNorm-50                 [-1, 128]             256
ReflectionalSwitchFunction-51               [-1, 128, 8]               0
     SplineLinear-52                  [-1, 64]          65,536
   FasterKANLayer-53                  [-1, 64]               0
        LayerNorm-54                  [-1, 64]             128
ReflectionalSwitchFunction-55                [-1, 64, 8]               0
     SplineLinear-56                  [-1, 32]          16,384
   FasterKANLayer-57                  [-1, 32]               0
        LayerNorm-58                  [-1, 32]              64
ReflectionalSwitchFunction-59                [-1, 32, 8]               0
     SplineLinear-60                  [-1, 16]           4,096
   FasterKANLayer-61                  [-1, 16]               0
        LayerNorm-62                  [-1, 16]              32
ReflectionalSwitchFunction-63                [-1, 16, 8]               0
     SplineLinear-64                  [-1, 10]           1,280
   FasterKANLayer-65                  [-1, 10]               0
================================================================
Total params: 1,427,488
Trainable params: 1,427,488
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 2.04
```

```
Params size (MB): 5.45
Estimated Total Size (MB): 7.49
```

Experiment Report:

Model	Accuracy (%)	Time Per Epoch	Amount of Epochs	Amount of Parameters
MLP	~99.5	58s	17	3,272,842
FasterKAN	~99.5	7m50s	22	1,427,488

Summary

- **Accuracy**:
  - Both the MLP and FasterKAN models exhibit similar high accuracy levels, approximately 99.5%.
- **Time Per Epoch**:
  - **MLP**: The MLP is considerably faster per epoch, requiring only 58 seconds.
  - **FasterKAN**: Takes longer per epoch, with each epoch lasting 7 minutes and 50 seconds.
- **Amount of Epochs**:
  - **MLP**: Needs fewer epochs to achieve its final accuracy
  - **FasterKAN**: Requires more epochs
- **Amount of Parameters**:
  - **MLP**: Contains more parameters (3,272,842)
  - **FasterKAN**: Has fewer parameters (1,427,488)

⟹ KAN could not beat MLP in most aspects. The only aspect where I think KAN could make sense is its ability to interpret.

D. References

- Ziyao Li. (n.d.). Fast-KAN. GitHub repository. Retrieved from https://github.com/ZiyaoLi/fast-kan
- Athanasios Delis. (n.d.). Faster-KAN. GitHub repository. Retrieved from https://github.com/AthanasiosDelis/faster-kan
- Xiaoming Kind. (n.d.). PyKAN. GitHub repository. Retrieved from https://github.com/KindXiaoming/pykan