Kolmogorov-Arnold networks

Rafał Osadnik

06/06/2024

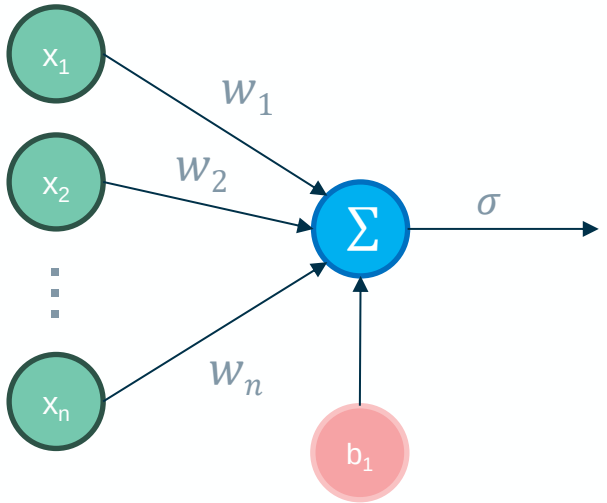
Agenda

- MLP Briefing
  - Theoretical basis
  - Structure
  - Importance
- KANs
  - Theoretical basis
  - Problem with past KANs in ML
  - Paper contribution
  - KANs structure
  - KANs parametrization
- KAN’s sparsity enforcement
- KAN’s Properties
  - Parameter efficiency
  - Interpretability

MLP Briefing

Theoretical basis: Universal approximation theorem

Given any continuous function there exists a neural network that approximates said function according to set criteria (as accurately as one desires). Perceptrons **apply linear transformations** onto inputs **and then** apply some **non-linearity**. **Weights and biases are the learned parameters**.



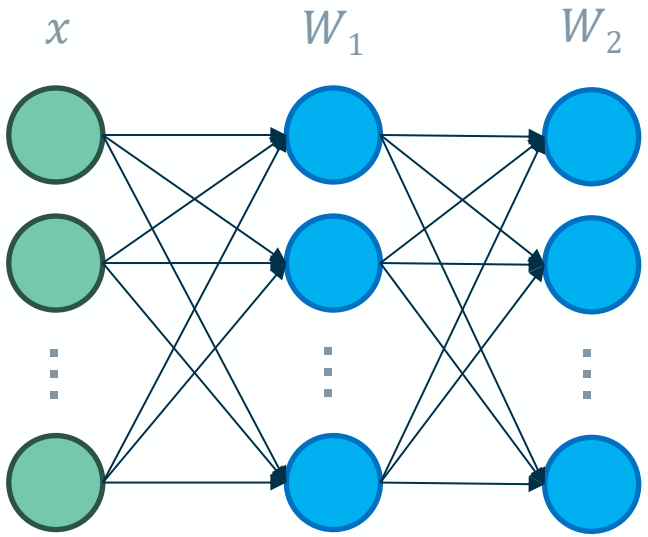
$\sigma\left(b_1 + \Sigma_i^n w_i x_i\right)$

Perceptron

$$\mathrm{PER}(x) = (\sigma \circ W_1)x$$

MLP Briefing

Now let's just multiply the perceptron arbitrary number of times and we get an MLP!



...

$$\mathrm{MLP}(x) = (\sigma_n \circ W_n \circ \cdots \circ \sigma_2 \circ W_2 \circ \sigma_1 \circ W_1)x$$

Multi Layer Perceptron

MLP Briefing

Importance of MLPs:

- Tranformers (LLMs, ViTs)
- Reward models (alignment of LLMs)
- Reinforcement learning (decision networks)
- Many, many more...

Llama2 source code

```
super().__init__()
self.n_heads = args.n_heads
self.dim = args.dim
self.head_dim = args.dim // args.n_heads
self.attention = Attention(args)
self.feed_forward = FeedForward(
    dim=args.dim,
    hidden_dim=4 * args.dim,
    multiple_of=args.multiple_of,
    ffn_dim_multiplier=args.ffn_dim_multiplier,
)
```

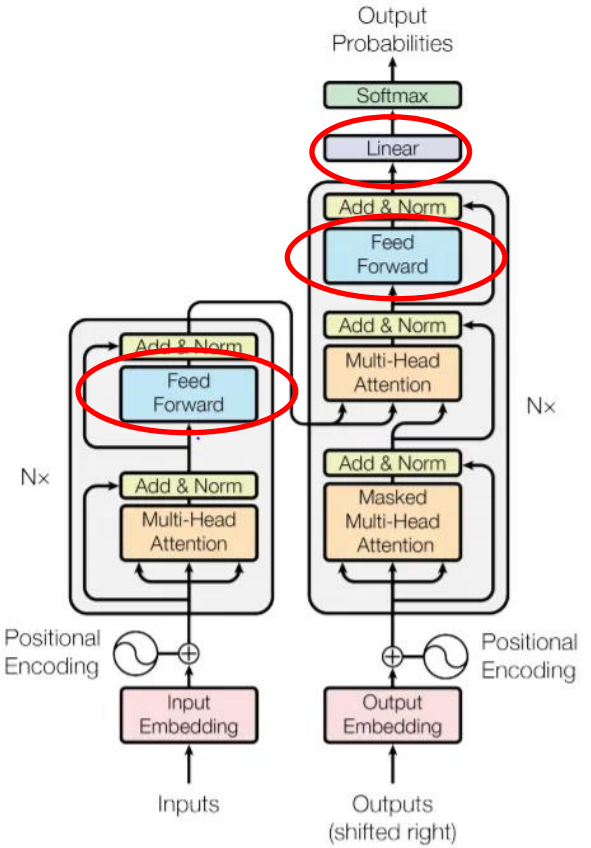


Figure 1: The Transformer - model architecture.

KANs

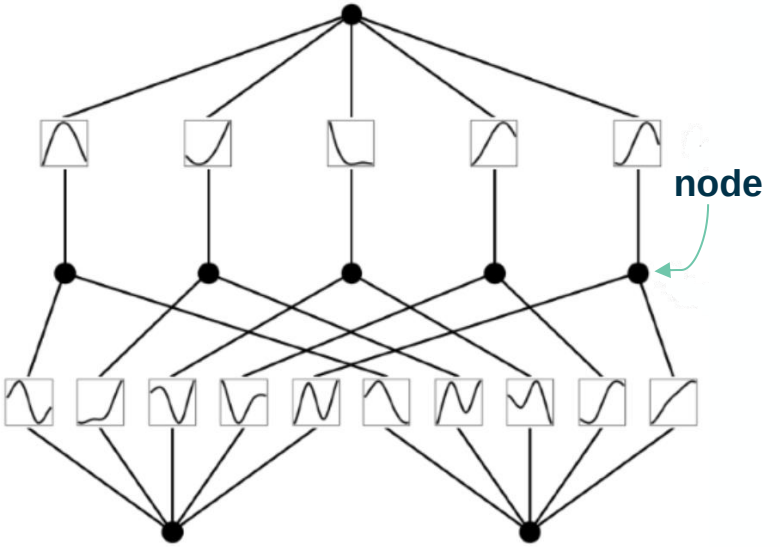
Theoretical basis: Kolmogorov-Arnold representation theorem

Any multivariate continuous function can be expressed as a finite composition of univariate functions and the operation of addition (the only true multivariate function is addition). This means that **KANs** instead of learning just weights they actually **learn univariate functions**.

only 2 layers

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

$n = 2$



KANs

KANs have been studied before in context of machine learning. There are a number of works that researched their capabilities and limitations.

Problem with past KANs:

Main limitation was their architecture, as it conformed to the original formulation of the KA theorem. Only networks of depth ***2*** and width of ***2n+1*** were studied.

Such small and shallow architectures **couldn't approximate multivariate functions using only smooth univariate functions**. Sometimes the **functions had to be non-smooth or even fractal**. This lead to abandonment of the **KAN** idea initially.

KANs

The main contribution of the recent paper is the **extension of this theorem to arbitrary widths and depths of networks**, allowing for deeper and more complex architectures. Making the network bigger allows for the representation through smooth and continuous functions only.

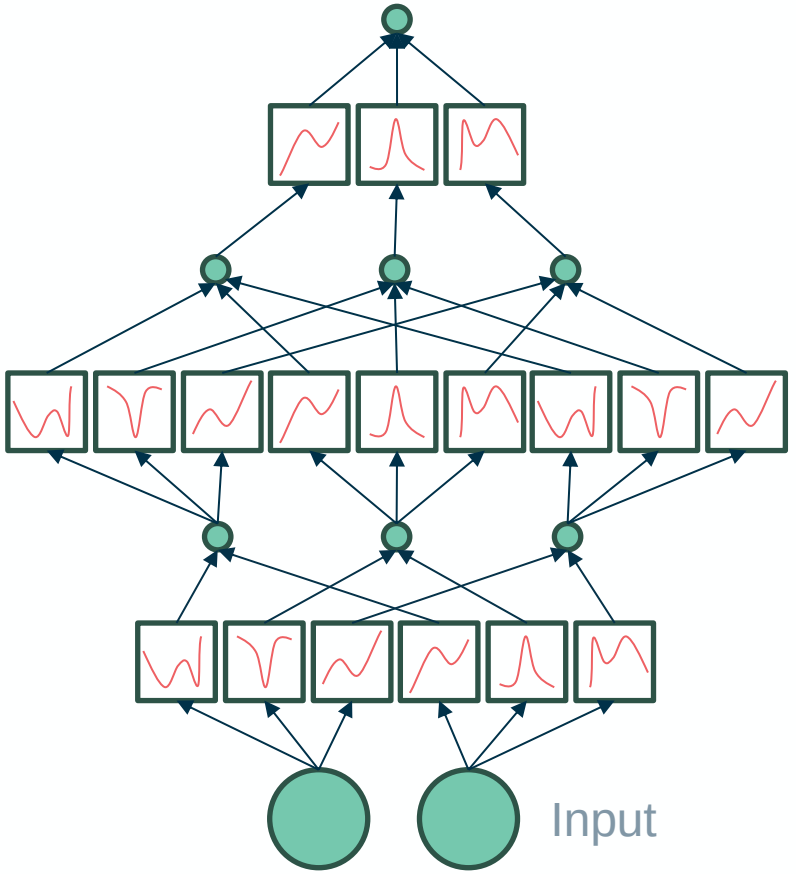
They simply define what a KAN layer is and stack them together. A shape of **KAN is represented by an integer array**:

$$[n_0, n_1, \ldots, n_L]$$

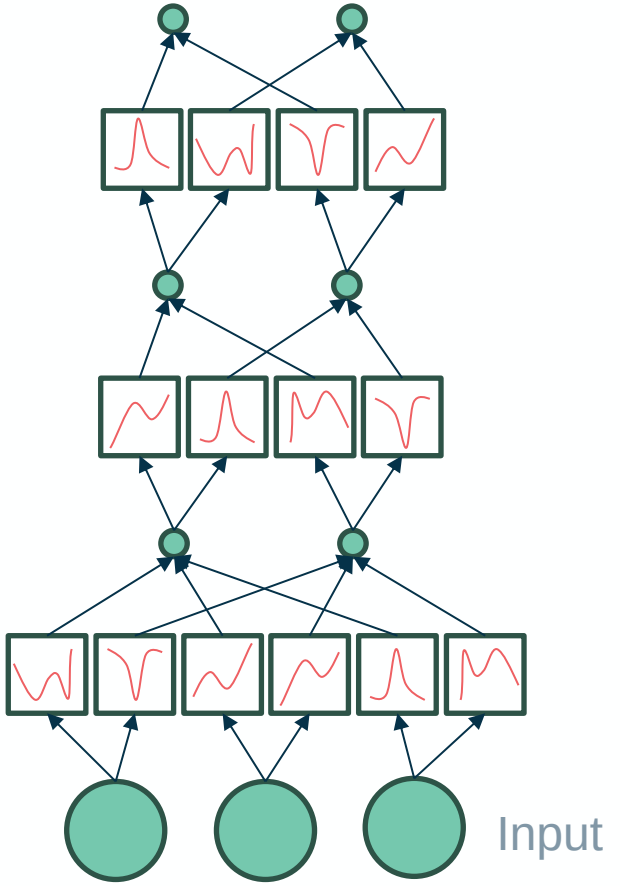
Where $\boldsymbol{n_L}$ is the number of nodes in $\boldsymbol{L}$**-th** layer. The number of activation functions between layers with $\boldsymbol{n_L}$ and $\boldsymbol{n_{L\text{-}1}}$ number of nodes is always $\boldsymbol{n_L n_{L\text{-}1}}$.

KANs

Structure notation of KANs with examples



[2, 3, 3, 1] **KAN**



[3, 2, 2, 2] **KAN**

KANs

How do KANs learn?

The authors employed B-splines as their method of function parametrization. Splines are just linear combinations of polynomial functions fitted to corresponding knot points. Activation functions in KANs are defined as follows:

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

basis function

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

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

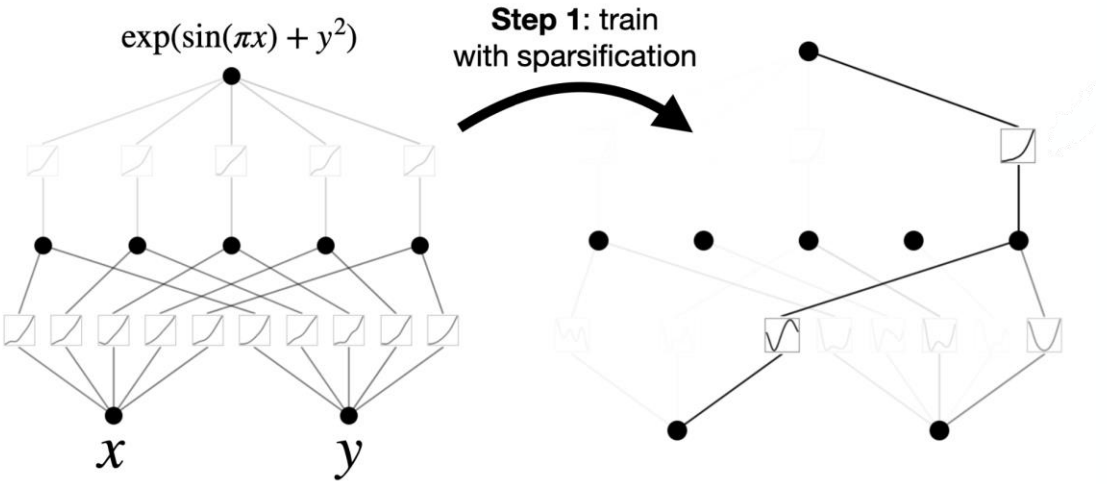
w_b, w_s, c_i and spline parameters are learnable!

KAN's sparsity enforcement

Similarly to MLPs – KANs can be regularized during training to enforce sparse representations. In MLPs L1 regularization applied to weights is used for that purpose. **For KANs L1 is too weak**, and **additional entropy regularization is used**.

KAN loss function with sparsity enforcement

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



KAN’s properties

Using simple and different functions to approximate complex relationships allows for easier understanding of the network decision making process. The functions allow for local explainability (output dependence on variable) as well as compositional explainability (how variables influence each other).

- High parameter efficiency (lower memory requirements)
- High accuracy
- Extreme interpretability
- Accuracy / simplicity trade-off
- The same architecture can be reparametrized differently (spline grids) and retrained

KAN's properties

Parameter efficiency

For the same structure KANs have more parameters than MLPs, meaning they are less parameter efficient in terms of their structure. Typically though, much smaller KANs are needed to achieve the same or higher accuracy for the same tasks as MLPs.

Signature classification

Method	Architecture	Parameter Count	Accuracy
Deepmind's MLP	4 layer, width-300	3×10^5	78.0%
KANs	2 layer, $[17, 1, 14]$ $(G = 3, k = 3)$	2×10^2	81.6%

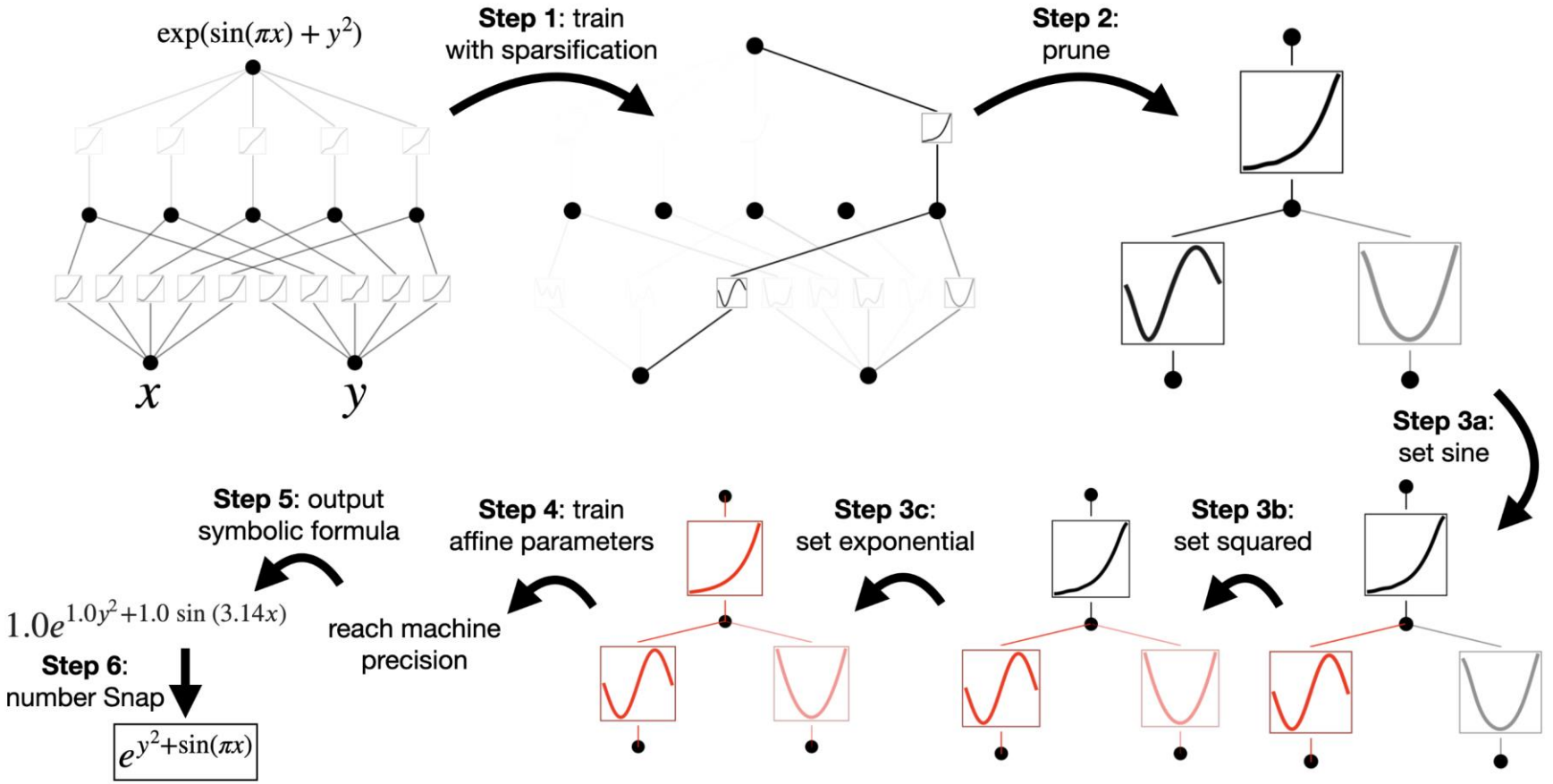
KAN’s properties

Interpretability

Novel approach to problem solving which automatically offers extreme interpretability out of the box. KANs are expected to be the most useful for natural sciences, where most representations are expected to be sparse compositionally and smooth. Interpretability / accuracy trade-off can be made, favouring simple architectures with very few activation functions.

Symbolic regression can be extremely helpful in obtaining accurate relationships.

Symbolic regression with KANs



PyKAN

The authors prepared a Python package with KAN building, visualization and training tools:

https://pypi.org/project/pykan/

Their repository:

https://github.com/KindXiaoming/pykan

Sources

[1]. **KAN: Kolmogorov-Arnold Networks,** https://arxiv.org/abs/2404.19756v3