Relating Piecewise Linear Kolmogorov Arnold Networks to ReLU Networks

Nandi Schoots
University of Oxford

Mattia Jacopo Villani
King's College London

Niels uit de Bos
MATS

Abstract

Kolmogorov-Arnold Networks are a new family of neural network architectures which holds promise for overcoming the curse of dimensionality and has interpretability benefits (Liu et al., 2024). In this paper, we explore the connection between Kolmogorov Arnold Networks (KANs) with piecewise linear (univariate real) functions and ReLU networks. We provide completely explicit constructions to convert a piecewise linear KAN into a ReLU network and vice versa.

1 INTRODUCTION

Architectural innovations are key drivers in the evolution of deep learning. Advances in architecture design, such as the introduction of convolutional (LeCun et al., 1989) or attention layers (Vaswani et al., 2017), yield significant performance improvements in AI systems. Very recently, Liu et al. (2024) introduced Kolmogorov Arnold Networks (KANs), an alternative to feedforward-style architectures in deep learning. The authors argue that KANs are more interpretable than traditional feedforward networks. However, they also find that KANs are typically 10x slower to train than MLPs, given the same number of parameters.

Our paper introduces completely explicit constructions for converting a ReLU network into a KAN with piecewise linear activation functions and vice versa (Section 4). This means users can train a ReLU network, translate it to a KAN, and benefit from the enhanced interpretability of the KAN. Moreover, unifying the architectures under a common framework facilitates the application of existing tools and theories developed for the ReLU network, such as analyzing symmetries and polyhedral regions, or initialisation techniques and research on generalisation bounds. In other words, we can have the best of both worlds.

Our conversion process is efficient in terms of the number of non-zero parameters of the converted network: the KAN-to-ReLU conversion does not increase the number of non-zero parameters, while the ReLU-to-KAN conversion increases the number of non-zero parameters by a term that is linear in the number of neurons (Section 5). However, in the KAN-to-ReLU conversion, we end up with a very wide network with sparse weight matrices (Section 4.3).

We show that, for a given parameter budget, KANs produce a finer polyhedral complex than ReLU networks. Specifically, we show that the upper bound on the number of linear regions implemented by a KAN is higher (Section 6). Parameter efficiency is key to enabling the use of lightweight models at inference time.

Throughout this paper, the term KAN refers to a KAN with piecewise linear activation functions.

2 RELATED WORKS

Kolmogorov Arnold's result, also known as the *Kolmogorov Superposition Theorem* (KST) shows that every function can be written using univariate functions and summing (Kolmogorov, 1956). The recent Liu et al. (2024) construction relies on this result.

Previously, several other attempts to unify KST and Deep Learning theory have been made (Schmidt-Hieber, 2021; Ismayilova and Ismailov, 2024).

KANs represent multivariate functions as compositions and superpositions of univariate functions. These representations are often considered more interpretable (Yang et al., 2021) because they are based on univariate functions. However, these functions can be very complex, and, e.g., in the case of piecewise linear univariate functions, they may have a large number of pieces (on which the function does not necessarily monotonically increase). Additionally, constructive

Proceedings of the 28[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

proofs of the Kolmogorov Arnold theorem typically find highly irregular and erratic univariate inner and outer functions (Braun and Griebel, 2009), decreasing their interpretability. Some authors try to impose Lipschitz continuity to enforce higher regularity in the inner and outer functions (Actor and Knepley, 2017); however, this comes at a cost of a large number of total functions. Moreover, the large number of univariates increases network complexity.

3 BACKGROUND

In this section, we recall some of the core ideas and definitions from Liu et al. (2024) for the reader's benefit. We also discuss piecewise linear functions and ReLU networks.

The **Kolmogorov Arnold Theorem** (or the superposition theorem) states the following. Let $f\colon [0,1]^n \to \mathbb{R}$ be a continuous multivariate real function. Then there are a finite number of continuous univariate real functions Φ_q and ϕ_p^q such that f can be written as

$$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_p^q(x_p) \right).$$

Kolmogorov Arnold Networks (KANs) were recently introduced and generalize Kolmogorov Arnold representations.

Definition 1. *A* KAN layer *with input dimension n_{in} and output dimension n_{out} is given by a n_{out}-by-n_{in} matrix $\mathbf{\Phi} = \{\phi_p^q\}_{p=1,\dots,n_{\text{in}}}^{q=1,\dots,n_{\text{out}}}$ of univariate real functions $\phi_p^q\colon \mathbb{R} \to \mathbb{R}$ that we call* activation functions. *It represents the function*

$$\mathbf{\Phi}\colon \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}^{n_{\text{out}}},$$
$$\mathbf{x} = (x_i)_{i=1,\dots,n_{\text{in}}} \mapsto \mathbf{\Phi}(\mathbf{x}) = \left(\sum_{i=1}^{n_{\text{in}}} \phi_i^j(x_i) \right)_{j=1,\dots,n_{\text{out}}}$$

Definition 2. *A* Kolmogorov Arnold network (KAN) *is a composition of L KAN layers $\mathbf{\Phi}_{L-1} \circ \ldots \circ \mathbf{\Phi}_0$. In the case that the last layer has output dimension 1, the function represented by the KAN takes the form*

$$f(\mathbf{x}) = \sum_{i_{L-1}=1}^{n_{L-1}} \phi_{i_{L-1}}^{L-1,i_L} \left(\ldots \sum_{i_1=1}^{n_1} \phi_{i_1}^{1,i_2} \left(\sum_{i_0=1}^{n_0} \phi_{i_0}^{0,i_1}(x_{i_0}) \right) \ldots \right) \tag{1}$$

where n_ℓ is the input dimension of the ℓ-th KAN layer $\mathbf{\Phi}_\ell = \{\phi_p^{\ell,q}\}_{p=1,\dots,n_\ell}^{q=1,\dots,n_{\ell+1}}$. If a KAN has L layers, we also sometimes say that it has $L-1$ hidden layers.

A **piecewise linear KAN** is a KAN in which each activation function $\phi_{i_{l-1}}^{l-1,i_l} : \mathbb{R} \to \mathbb{R}$ is piecewise linear with a finite number of segments. Going forward, in the interest of presentation, whenever we say KAN we typically mean a piecewise linear KAN, but sometimes we will emphasize the piecewise linearity explicitly.

Any piecewise linear function f can be represented as a **polyhedral complex** $\mathcal{C}(f) = (\Omega, (\alpha_\omega, \beta_\omega)_{\omega\in\Omega})$, where Ω is a partition of the input space $\mathbb{R}^n$, and $(\alpha_\omega, \beta_\omega) \in \mathbb{R} \times \mathbb{R}$ are linear coefficients for $f|_\omega$, i.e., $f|_\omega(x) = \alpha_\omega x + \beta_\omega$.

Rectified **L**inear **U**nit (ReLU) networks are a popular family of architectures for deep learning. Both ReLU networks and piecewise linear KANs are examples of piecewise linear functions. This means that both can be represented as polyhedral complexes through a polyhedral decomposition. In the ReLU case, such decompositions have received large theoretical (Montufar et al., 2014; Arora et al., 2016; Serra et al., 2018) and empirical (Raghu et al., 2017; Humayun et al., 2022; Berzins, 2023; Masden, 2022) attention, and their properties are an object of interest. In this paper we develop the first analysis of the polyhedral decomposition of piecewise linear KANs.

4 CONVERTING KANS TO RELU NETWORKS AND VICE VERSA

In this section we provide methods for translating a ReLU to a KAN with piecewise linear activation functions and vice versa. See Appendix A for a discussion of how the ideas in this section can be extended to convert KANs with B-spline activation functions to and from a more unconventional feedforward architecture with both ReLU activation functions and monomial activation functions.

4.1 For every ReLU there's a KAN

Theorem 1. *Let $g\colon \mathbb{R}^n \to \mathbb{R}^m$ be a feedforward network with activation functions from a family $\mathcal{F}$. There exists a KAN $f\colon \mathbb{R}^n \to \mathbb{R}^m$ with activation functions that are either affine linear or from $\mathcal{F}$ such that $f(x) = g(x)$ for all $x \in \mathbb{R}^n$. In particular, if g is a ReLU network, then there exists a piecewise linear KAN f with $f(x) = g(x)$ for all $x \in \mathbb{R}^n$.*

Proof. Suppose that g is a one layer network with weight vector W and bias b, then we can write a KAN as follows

$$f(x) = \sigma \left(\sum_{i_0=1}^{n} \phi_{i_0}(x_{i_0}) \right),$$

where σ is the univariate real (ReLU) activation function, $\phi_{i_0}(x_{i_0}) = w_{i_0}x_{i_0} + b$ for $i_0 = 1$ and $\phi_{i_0}(x_{i_0}) = w_{i_0}x_{i_0}$ for $i_0 > 1$.

We will now give a general formulation of the KAN corresponding to a network with L layers. Let $g\colon \mathbb{R}^n \to \mathbb{R}^m$ be a composition of $L \in \mathbb{N}$ layers, where for each $l \in \{0, \dots, L-2\}$ the layer is given by:

$$\chi^{(l+1)} = \sigma(W^{(l)}\chi^{(l)} + B^{(l)}),$$

where σ is an element-wise activation function and $\chi^{(l)}$ are the activations, with $\chi^{(0)} = x$ and output layer $g(x) = W^{(L-1)}\chi^{(L-1)} + B^{(L-1)}$.

We define

$$f(\mathbf{x}) = \sum_{i_{L-1}=1}^{n_{L-1}} \phi_{i_{L-1}}^{L-1,i_L}\left(\dots \sum_{i_1=1}^{n_1} \phi_{i_1}^{1,i_2}\left(\sum_{i_0=1}^{n_0} \phi_{i_0}^{0,i_1}(x_{i_0})\right)\dots\right)$$

where for $l = 0$ we define

$$\phi_{i_0}^{0,i_1}(x_{i_0}) = W_{i_0,i_1}^{(0)}x_{i_0} + B_{i_1}^{(0)} \text{ for } i_0 = 1 \text{ and}$$
$$\phi_{i_0}^{0,i_1}(x_{i_0}) = W_{i_0,i_1}^{(0)}x_{i_0} \qquad \text{for } i_0 > 1;$$

and for $l > 0$ we define

$$\phi_{i_l}^{l,i_{l+1}}(s) = W_{i_l,i_{l+1}}^{(l)}\sigma(s) + B_{i_{l+1}}^{(l)} \text{ for } i_l = 1 \text{ and}$$
$$\phi_{i_l}^{l,i_{l+1}}(s) = W_{i_l,i_{l+1}}^{(l)}\sigma(s) \qquad \text{for } i_l > 1.$$

This function f is a KAN and by construction $f(x) = g(x)$ for all $x \in \mathbb{R}^n$. □

Arora et al. (2016) proves that every piecewise linear function with finitely many pieces can be represented exactly by a ReLU network of finite depth and width. A corollary of Theorem 1 is that any piecewise linear function with finitely many pieces can also be represented exactly as a piecewise linear KAN. Moreover, it is possible to do so with a KAN network of the same depth as the ReLU network. Upper bounds on this depth are given by Arora et al. (2016).

4.2 For Every KAN there's a ReLU

In this section we show that we can express any KAN as a ReLU network. We begin with an example of a piecewise linear activation function ϕ with two breakpoints b_1, b_2 and three different inclinations a_1, a_2, a_3, as in Figure 1. If we assume the first segment passes through the origin, then we can write this activation function as

$$g(x) := a_1 \cdot x + (a_2 - a_1) \cdot \mathrm{ReLU}(x - b_1) + (a_3 - a_2) \cdot \mathrm{ReLU}(x - b_2).$$

We will now check that $\phi(x) = g(x)$ in all three segments:

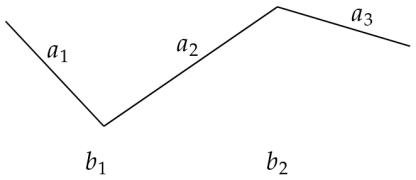

Figure 1: Example of a piecewise linear activation function.

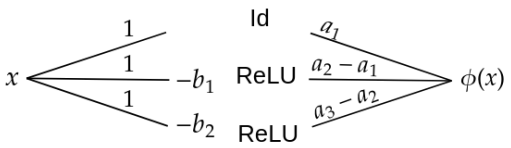

Figure 2: Network implementing the activation function.

- For $x < b_1$ we get $g(x) = a_1 \cdot x$.
- For $b_1 < x < b_2$ we get $g(x) = a_1 \cdot x + (a_2 - a_1) \cdot (x - b_1)$, the derivative of g here is a_2 and the value of g at point b_1 is $g(b_1) = a_1 \cdot b_1 = \phi(b_1)$.
- For $x > b_2$ we get $g(x) = a_1 \cdot x + (a_2 - a_1) \cdot (x - b_1) + (a_3 - a_2) \cdot (x - b_2)$, the derivative here is a_3 and the value of g at point b_2 is $g(b_2) = a_1 \cdot b_2 + (a_2 - a_1) \cdot (b_2 - b_1) = \phi(b_2)$.

We will now prove a lemma about a single piecewise linear activation function, a lemma about a single KAN layer, and finally the general theorem about KANs.

Lemma 1. *Let $\phi\colon \mathbb{R} \to \mathbb{R}$ be a piecewise linear function with a finite number n of segments. Then there exist a n-by-1 matrix $W^{(1)}$, a 1-by-n matrix $W^{(2)}$, a bias vector $B^{(1)} \in \mathbb{R}^n$, and a bias $B^{(2)} \in b$ such that for all $x \in \mathbb{R}$*

$$\phi(x) = W^{(2)}\,\mathrm{ReLU}(W^{(1)}x + B^{(1)}) + B^{(2)};$$

in other words, we can write ϕ as a feedforward network with one hidden layer with n neurons.

Proof. Let $b_1, \dots, b_{n-1} \in \mathbb{R}$ be the breakpoints of the piecewise linear map ϕ, and let a_i for $1 < i < n$ denote the slope of ϕ on the interval $[b_{i-1}, b_i) \subset \mathbb{R}$; by a_1 we denote the slope on $(-\infty, b_1)$, and by a_n we denote the slope on $[b_{n-1}, \infty)$. Let $c \in \mathbb{R}$ be the y-intercept of the first segment, i.e., $\phi(x) = a_0x + c$ for $x \in (-\infty, b_1)$.

Then we can take

$$\begin{aligned} W^{(1)} &= (a_1, a_2 - a_1, a_3 - a_2, \ldots, a_n - a_{n-1})^T \\ W^{(2)} &= (1, 1, \ldots, 1) \\ B^{(1)} &= (0, -b_1, -b_2, \ldots, -b_{n-1}) \\ B^{(2)} &= c \end{aligned}$$

A simple calculation shows that this is correct. Indeed, $W^{(1)}x+B^{(1)}$ is equal to $(x, x-x_1, x-x_2, \ldots, x-x_{n-1})$, so for $1 \leq i < n-1$ and $x \in [x_i, x_{i+1})$, we see that $\mathrm{ReLU}(W^{(1)}x + B^{(1)})$ is equal to $(x, x - x_1, \ldots, x - x_i, 0, \ldots, 0)$. Multiplying this by $W^{(2)}$, we get

$$a_1 x + (a_2 - a_1)(x - b_1) + \ldots + (a_{i+1} - a_i)(x - b_i)$$

which has a linear coefficient for x equal to a_{i+1}, as expected. Similarly, we can show that the slopes on the first and last segment are correct. The coefficient $B^{(2)}$ then ensures the right intercept on the first segment. It follows that the other intercepts are also correct, because the function is continuous and has the correct derivative everywhere. □

This demonstrates that we can convert a single activation function in a piecewise linear KAN to a ReLU network. We use this to convert a single piecewise linear KAN layer.

Lemma 2. *Let $\mathbf{\Phi} = \{\phi_p^q\}_{p=1,\ldots,n_{\text{in}}}^{q=1,\ldots,n_{\text{out}}} : \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}^{n_{\text{out}}}$ be a KAN layer with piecewise linear activation functions $\phi_{q,p}$ with finite numbers of segments. Then there exist matrices $W^{(1)}, W^{(2)}$ and bias vectors $B^{(1)}, B^{(2)}$ such that for all $x \in \mathbb{R}^{n_{\text{in}}}$*

$$\mathbf{\Phi}(x) = W^{(2)}\,\mathrm{ReLU}(W^{(1)}x + B^{(1)}) + B^{(2)};$$

in other words, we can write $\mathbf{\Phi}$ as a feedforward network with one hidden layer.

Proof. First assume $n_{\text{out}} = 1$; the function we need to represent is then $\mathbf{\Phi}(x) = \sum_{p=1}^{n_{\text{in}}} \phi_p^1(x_p)$. By Lemma 1, we can write each ϕ_p^1 as

$$\phi_p^1(x_p) = W_p^{(2)}\,\mathrm{ReLU}(W_p^{(1)}x_p + B_p^{(1)}) + B_p^{(2)}.$$

We can combine these results together. This process is illustrated in Figure 3 and the equation below outlines the matrix calculus:

$$\begin{aligned} \mathbf{\Phi}(x) &= \sum_{p=1}^{n_{\text{in}}} \phi_{1,p}(x_p) \\ &= \sum_{p=1}^{n_{\text{in}}} W_p^{(2)}\,\mathrm{ReLU}(W_p^{(1)}x_p + B_p^{(1)}) + B_p^{(2)} \\ &= \sum_{p=1}^{n_{\text{in}}} B_p^{(2)} + \begin{pmatrix} W_1^{(2)} & W_2^{(2)} & \cdots & W_{n_{\text{in}}}^{(2)} \end{pmatrix} \cdot \mathrm{ReLU} \\ &\quad \left(\begin{pmatrix} W_1^{(1)} & 0 & \ldots & 0 \\ 0 & W_2^{(1)} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & W_{n_{\text{in}}}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_{\text{in}}} \end{pmatrix} + \begin{pmatrix} B_1^{(1)} \\ B_2^{(1)} \\ \vdots \\ B_{n_{\text{in}}}^{(1)} \end{pmatrix} \right) \end{aligned}$$

This shows that we can also write

$$\mathbf{\Phi}(x) = W^{(2)}\,\mathrm{ReLU}(W^{(1)}x + B^{(1)}) + B^{(2)}.$$

Now consider again the general case of $n_{\text{out}} \geq 1$. By what we have just proven, for every $q = 1, \ldots, n_{\text{out}}$, we can write

$$\sum_{p=1}^{n_{\text{in}}} \phi_p^q(x_p) = W_q^{(2)}\,\mathrm{ReLU}(W_q^{(1)}x + B_q^{(1)})$$

for some matrices $W_q^{(1)}, W_q^{(2)}, B_q^{(1)}, B_q^{(2)}$. We can again do all these computations in parallel, as illustrated on the left-hand side of Figure 4; more rigorously, we have the following block matrix computation:

$$\begin{aligned} \mathbf{\Phi}(x) &= \begin{pmatrix} \sum_{p=1}^{n_{\text{in}}} \phi_{1,p}(x_p) \\ \sum_{p=1}^{n_{\text{in}}} \phi_{2,p}(x_p) \\ \vdots \\ \sum_{p=1}^{n_{\text{in}}} \phi_{n_{\text{out}},p}(x_p) \end{pmatrix} \\ &= \begin{pmatrix} W_1^{(2)}\,\mathrm{ReLU}(W_1^{(1)}x + B_1^{(1)}) + B_1^{(2)} \\ W_2^{(2)}\,\mathrm{ReLU}(W_2^{(1)}x + B_2^{(1)}) + B_2^{(2)} \\ \vdots \\ W_{n_{\text{out}}}^{(2)}\,\mathrm{ReLU}(W_{n_{\text{out}}}^{(1)}x + B_{n_{\text{out}}}^{(1)}) + B_{n_{\text{out}}}^{(2)} \end{pmatrix} \\ &= \begin{pmatrix} W_1^{(2)} & 0 & \cdots & 0 \\ 0 & W_2^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & W_{n_{\text{out}}}^{(2)} \end{pmatrix} \cdot \mathrm{ReLU} \\ &\quad \left(\begin{pmatrix} W_1^{(1)} \\ W_2^{(1)} \\ \vdots \\ W_{n_{\text{out}}}^{(1)} \end{pmatrix} x + \begin{pmatrix} B_1^{(1)} \\ B_2^{(1)} \\ \vdots \\ B_{n_{\text{out}}}^{(1)} \end{pmatrix} \right) + \begin{pmatrix} B_1^{(2)} \\ B_2^{(2)} \\ \vdots \\ B_{n_{\text{out}}}^{(2)} \end{pmatrix} \end{aligned}$$

so again we see that we can write

$$\mathbf{\Phi}(x) = W^{(2)}\,\mathrm{ReLU}(W^{(1)}x + B^{(1)}) + B^{(2)}.$$

□

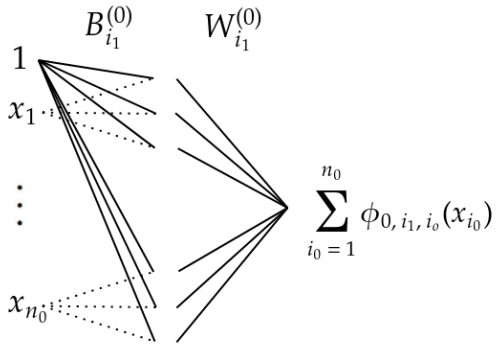

Figure 3: Concatenating vectors $W^{(1)}_{i_1,i_0}$ and $B^{(1)}_{i_1,i_0}$ into vectors $W^{(1)}_{i_1}$ and $B^{(1)}_{i_1}$.

Theorem 2. *For every piecewise linear KAN $f : \mathbb{R}^n \to \mathbb{R}$ there exists a ReLU network $g : \mathbb{R}^n \to \mathbb{R}$ such that $f(x) = g(x)$ for all $x \in \mathbb{R}^n$.*

Proof. A KAN with L layers is a composition $\mathbf{\Phi}_{L-1} \circ \mathbf{\Phi}_{L-2} \circ \ldots \circ \mathbf{\Phi}_0$ of L KAN layers. By Lemma 2, each KAN layer $\mathbf{\Phi}_\ell$ can be written as

$$\mathbf{\Phi}_\ell(x^{(\ell)}) = W^{(\ell,2)}\,\mathrm{ReLU}(W^{(\ell,1)}x^{(\ell)} + B^{(\ell,1)}) + B^{(\ell,1)}.$$

When we compose two layers $\mathbf{\Phi}_{\ell+1} \circ \mathbf{\Phi}_\ell$, the last layer of the feedforward architecture of $\mathbf{\Phi}_\ell$ is not followed by a non-linear activation function, so it can be combined with the first layer of $\mathbf{\Phi}_{\ell+1}$; this combination looks like this:

$$\begin{aligned}
&W^{(\ell+1,1)}\mathbf{\Phi}_\ell(x^{(\ell)}) + B^{(\ell+1,1)} \\
=&W^{(\ell+1,1)}\left(W^{(\ell,2)}\,\mathrm{ReLU}(W^{(\ell,1)}x^{(\ell)} + B^{(\ell,1)}) + B^{(\ell,2)}\right) \\
&+ B^{(\ell+1,1)} \\
=&\underbrace{W^{(\ell+1,1)}W^{(\ell,2)}}_{\text{combined weights}}\mathrm{ReLU}\left(W^{(\ell,1)}x^{(\ell)} + B^{(\ell,1)}\right) \\
&+ \underbrace{W^{(\ell+1,1)}B^{(\ell,2)} + B^{(\ell+1,1)}}_{\text{combined bias}}.
\end{aligned}$$

□

4.3 Class Embeddings

Let $\mathbf{KAN}(L, n, k)$ be the functional class of KANs with L layers, width $n = \max_{i=1,\ldots,L}(n_i)$, and activation functions with at most $k+1$ segments. Similarly, let $\mathbf{ReLU}(L, n)$ denote the class of ReLU networks with width n and L layers.

Theorem 3. *Using the notation from above, the constructions in Theorem 2 and Theorem 1 define the following embeddings:*

$$\begin{aligned}
\boldsymbol{KAN}(L, n, k) &\subseteq \boldsymbol{ReLU}(L+1, n^2(k+1)) \\
&\subseteq \boldsymbol{KAN}(L+1, n^2(k+1), 1).
\end{aligned}$$

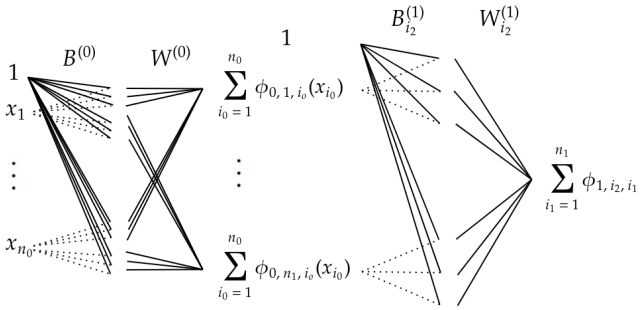

Figure 4: Three hidden layer network implementing a KAN of depth two.

Proof. Our KAN-to-ReLU conversion in Theorem 2 converts a KAN with L layers into a ReLU network with $L+1$ layers (where, as is our convention, the last layer does not include a ReLU, but is simply an affine linear map). In the construction, each activation function with $k+1$ segments needs $k+1$ hidden neurons to be converted into a feed-forward architecture (Lemma 1). Since every KAN layer has n^2 activation functions, this means we need a width of $n^2(k+1)$. This establishes the first embedding

$$\mathbf{KAN}(L, n, k) \subseteq \mathbf{ReLU}(L+1, n^2(k+1)).$$

Conversely, the ReLU-to-KAN conversion in Theorem 1 keeps the number of layers constant. The equations at the end of the proof of Theorem 1 that define the activation functions $\phi^{l,i_{l+1}}_{i_l}$, show that we use activation functions that consist of at most 2 segments, and the width also remains unchanged. This establishes the second embedding

$$\mathbf{ReLU}(L+1, n^2(k+1)) \subseteq \mathbf{KAN}(L+1, n^2(k+1), 1)$$

concluding the proof.

□

5 CONVERSION EFFICIENCY

Let $\# : \mathcal{F} \to \mathbb{N}$ be a parameter counting function on a parameterised functional class $\mathcal{F}$. In the following propositions, f, g represent KANs and ReLU networks. In contrast, $\hat{f}, \hat{g}$ represent the KANs and ReLU networks that have been converted from ReLU (as in Theorem 1) and KAN networks (as in Theorem 2) respectively.

5.1 Conversion from ReLU to KAN

Lemma 3. *Let g be a ReLU network $g\colon \mathbb{R}^n \to \mathbb{R}$ with $L \in \mathbb{N}$ hidden layers and $\mathbf{N} = [n_1, n_2, \ldots, n_L]$ neurons.*

Then the network has at most $n \times n_1 + n_L + \sum_{i=1}^{L-1} n_i \times n_{i+1}$ *parameters in the weight matrices and at most* $1 + \sum_{i=1}^{L} n_i$ *parameters in the bias vectors, for a total of*

$$\#(g) = 1 + n \times n_1 + 2 \cdot n_L + \sum_{i=1}^{L-1} (n_i \times n_{i+1} + n_i).$$

Proposition 1. *Let* g *be a ReLU network* $g \colon \mathbb{R}^n \to \mathbb{R}$ *with* $L \in \mathbb{N}$ *hidden layers and* $\mathbf{N} = [n_1, n_2, \ldots, n_L]$ *neurons. Let* $\hat{f}$ *be the KAN as constructed in the proof of Theorem 1. Then,*

$$\#(\hat{f}) = \#(g) + 4 \cdot (n_1 + n_2 + \ldots + n_L + 1).$$

Proof. The construction in the proof of Theorem 1 uses the same number of parameters as are in g_θ and additionally requires four parameters per application of ReLU, of which there are $n_1 + n_2 + \ldots + n_L + 1$. □

This translation is efficient as it scales linearly with the number of neurons in the original ReLU architecture.

5.2 Conversion from KAN to ReLU

Proposition 2. *Let* $f \colon \mathbb{R}^n \to \mathbb{R}^m$ *be a* L *layer KAN network as described in Equation 1. If each* ϕ *has exactly* k *segments, the total number of parameters is given by:*

$$\#(f) = 2k \sum_{l=1}^{L+1} n_l n_{l-1}.$$

Proof. For each $\phi_{i_{l-1}}^{l,i_l}, i_l = 1, ..., n_l, i_{l-1} = 1, ..., n_{l-1}, l = 1, ..., L+1$ there are exactly k scalar parameters $a_j^{l,i_l,i_{l-1}}$ representing the slopes of the univariate linear segments and $k-1$ scalar parameters representing breakpoints $b_j^{i_l}$ and the initial bias for a total of $2k$ scalar parameters. Since there are $n_l n_{l-1}$ activation functions in layer l, the layer has $n_l n_{l-1} \cdot 2k$ parameters, for a total of $\#(f) = 2k \sum_{l=1}^{L} n_l n_{l-1}$. □

Proposition 3. *Let* $f \colon \mathbb{R}^n \to \mathbb{R}$ *be a* L *layer KAN network as described in Equation 1. Let each* ϕ *have exactly* k *segments. Let* $\hat{g}$ *be the ReLU network as constructed in the proof of Theorem 2, then*

$$\#(\hat{g}) = 2k \sum_{l=1}^{L+1} n_l n_{l-1}.$$

Proof. Each weight matrix $W^{(l)}$ has n_l columns and each column has length $\sum_{i_l=1}^{n_l} \sum_{i_{l-1}=1}^{n_{l-1}} k_{\phi_{l,i_l,i_{l-1}}}$, and has at most $\sum_{i_{l-1}}^{n_{l-1}} k_{\phi_{l,i_l,i_{l-1}}}$ non-zero entries, where $k_{\phi_{l,i_l,i_{l-1}}}$ is the number of segments in the activation function $\phi_{i_{l-1}}^{l,i_l}$.

Given our assumption that every activation function has k segments this means every column has at most $k \cdot n_{l-1}$ non-zero entries. This means that every matrix $W^{(l)}$ has $k \cdot n_{l-1} \cdot n_l$ parameters.

Analogously every bias vector $B^{(l)}$ has $k \cdot n_{l-1} \cdot n_l$ parameters.

For a KAN with L hidden layers and one output layer, we get

$$\sum_{l=1}^{L+1} k \cdot n_{l-1} \cdot n_l + k \cdot n_{l-1} \cdot n_l = 2k \sum_{l=1}^{L+1} n_l n_{l-1}.$$

□

This computation relies on the assumption that only non-zero values of $W^{(l)}$ are considered parameters. Computationally, this can be implemented with sparse matrices, in PyTorch. However, the width of the architecture is increased by a multiplicative factor: every layer has now $k \cdot n_l$ neurons.

6 POLYHEDRAL DECOMPOSITION OF KANS AND RELUS

In this section, we will relate the number of input regions a model differentiates between to the number of parameters needed to implement the model.

Let $\mathcal{R} : \mathcal{F} \to \mathbb{N}$ be the total number of regions in the polyhedral complex (the cardinality of Ω). For both ReLU neworks and KANs we will consider how many parameters are needed per polytope. In other words we will consider their representational power.

6.1 Upper bound number of regions ReLU network

We begin with ReLU networks, and we simply state the results of previous work. The below proposition states that the input space of a ReLU network can be decomposed into a finite number of regions such that the network is linear in each region, and such that the network non-linearity occurs exactly on the region boundaries.

Proposition 4. *[Sudjianto et al. (2020)] For a ReLU network* $\mathcal{N} \colon \mathbb{R}^n \to \mathbb{R}$ *there is a finite partition* Ω *of* $\mathbb{R}^n$ *of cardinality* $p := \#\Omega$ *such that for each part* $\omega \in \Omega$ *there exists a piecewise linear function* $f \colon \mathbb{R}^n \to \mathbb{R}^m$*, and its restriction on* ω*, denoted* $f|_\omega$*, is linear. Each part is a polytope, given by the intersection of a collection of half-spaces. All of the half-spaces are induced by neurons.*

The below proposition gives an upper bound for the number of regions that the input space can be decomposed into.

Proposition 5. *[Montufar (2017)] Let g be a ReLU network $g\colon \mathbb{R}^n \to \mathbb{R}$ with $L \in \mathbb{N}$ hidden layers and $\mathbf{N} = [n_1, n_2, \ldots, n_L]$ neurons. Then the number of linear regions is upper bounded by $\prod_{l=1}^{L} \sum_{j=0}^{d_l} \binom{n_l}{j}$, where $d_l = \min\{n, n_1, n_2, \ldots, n_l\}$, i.e.*

$$\mathcal{R}(g) \leq \prod_{l=1}^{L} \sum_{j=0}^{d_l} \binom{n_l}{j}.$$

Serra and Ramalingam (2020) find a tighter upper bound, by considering which combinations of turned off and turned on ReLUs are possible (activation patterns).

6.2 Upper bound number of regions of a KAN

Now we calculate an upper bound for the number of linear regions of a KAN.

Lemma 4. *Let $f\colon \mathbb{R}^n \to \mathbb{R}^m$ be a piecewise linear map with k segments, and let $g\colon \mathbb{R}^m \to \mathbb{R}^l$ be a piecewise linear map with k' segments. Then the composition $g \circ f$ is a piecewise linear map with at most $k \cdot k'$ segments.*

Proof. Let ω be one of the segments of f. This means that $f|_\omega$ is linear.

For every segment η of g, define $\omega_\eta = \omega \cap f^{-1}(\eta)$, the inverse image of η in ω. Then $(g \circ f)|_{\omega_\eta} = g|_\eta \circ f|_{\omega_\eta}$ is the composition two linear functions, and hence also linear. So if we partition ω into at most k' subsegments ω_η, then $g \circ f$ is linear on all those subsegments. Doing this for all segments ω of f, we have found a partition of the input space of f of at most $k \cdot k'$ segments on which $g \circ f$ is linear. □

Theorem 4. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be KAN network with L hidden layers as described in Equation 1. Suppose that each activation function ϕ in f has at most k segments. The total number of regions $\mathcal{R}(f)$ of f has the following upper bound:*

$$\mathcal{R}(f) \leq k^{n_L + \sum_{i=0}^{L-1} n_i n_{i+1}}$$

where the n_i are the widths of the layers.

Proof. We can write f as a composition $f = \mathbf{\Phi}_L \circ \ldots \circ \mathbf{\Phi}_0$ of KAN layers $\mathbf{\Phi}_i\colon \mathbb{R}^{n_i} \to \mathbb{R}^{n_{i+1}}$. We are going to prove that each KAN layer $\mathbf{\Phi}_i$ is piecewise linear with at most $k^{n_i n_{i+1}}$ segments, so that the conclusion follows from Lemma 4.

First, consider a function

$$\phi\colon \mathbb{R}^n \to \mathbb{R}, \quad x \mapsto \sum_{i=1}^{n} \phi_i(x_i)$$

where each ϕ_i is piecewise linear with at most k segments. An example of such a function is $\mathbf{\Phi}_L$. We will now prove that this map is piecewise linear with at most k^n segments.

For any $1 \leq i \leq n$, let $\omega_{i,1}, \ldots, \omega_{i,k} \subset \mathbb{R}$ denote the k segments of ϕ_i, and write $\omega_{i_1,i_2,\ldots,i_n} = \omega_{1,i_1} \times \cdots \times \omega_{n,i_n}$ for the Cartesian product of one segment for every axis i. These $\omega_{i_1,i_2,\ldots,i_n} = \omega_{1,i_1} \times \cdots \times \omega_{n,i_n}$ partition the space $\mathbb{R}^n$ into k^n segments, and on every such segment, $\sum_i \phi_i$ is linear because each ϕ_i is linear. This proves that ϕ is piecewise linear with at most k^n segments.

Now we consider the more general function

$$\phi\colon \mathbb{R}^n \to \mathbb{R}^m, \quad x \mapsto \left(\sum_{i=1}^{n} \phi_i^j(x_i) \right)_{j=1,\ldots,m}$$

where each ϕ_i^j is a piecewise linear map with at most k segments. All the $\mathbf{\Phi}_\ell$ with $0 \leq \ell < L$ are of this form. As we will now prove, this map is piecewise linear with at most k^{nm} segments.

We write $\phi = (\phi_1, \ldots, \phi_m)$, i.e., we write ϕ_j for the component maps $\phi_j : \mathbb{R}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^{n} \phi_i^j(x_i)$. Our previous result shows that each ϕ_j is piecewise-linear with at most k^n segments, so for each j we have a partition of the input space $\mathbb{R}^n$ into at most k^n segments $\omega_{j,1}, \ldots, \omega_{j,k^n}$ such that ϕ_j is continuous when restricted to that segment. By picking one such segment for each $1 \leq j \leq m$ and intersecting those chosen segments, we get $(k^n)^m$ subsegments of the form

$$\omega_{j_1,\ldots,j_m} := \omega_{1,j_1} \cap \cdots \cap \omega_{m,j_m}$$

that together partition the input space $\mathbb{R}^n$. On each of these subsegments $\omega_{j_1,\ldots,j_m}$, each $\phi_{j'}$ is linear, because $\omega_{j_1,\ldots,j_m}$ is a subset of the segment $\omega_{j',j_{j'}}$ of $\phi_{j'}$. Since this is true for each j', the map ϕ is linear on each such segment. This proves that ϕ is piecewise-linear with at most k^{nm} segments, concluding the proof. □

Using this upper bound, we can approximate the number of regions that can be expressed per network parameter. For ReLU networks this ratio can be approximated by

$$\frac{\prod_{l=1}^{L} \sum_{j=0}^{d_l} \binom{n_l}{j}}{1 + n \times n_1 + 2 \cdot n_L + \sum_{i=1}^{L-1} (n_i \times n_{i+1} + n_i)}.$$

For KANs the ratio is

$$\frac{k^{n_L + \sum_{i=0}^{L-1} n_i n_{i+1}}}{2k \sum_{l=1}^{L+1} n_l}.$$

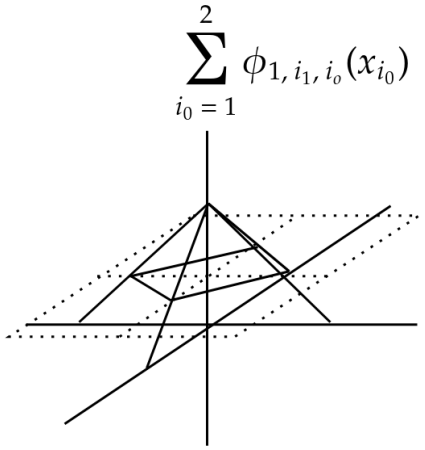

Figure 5: Sum of two activation functions that each have one breakpoint at the origin. A 2-dimensional hyperplane cuts through the pyramid.

This ratio is much bigger for KANs, which means that KANs are able to represent a finer polyhedral partition with fewer parameters. While in general this may suggest that this is a more expressive class of piecewise linear functions, further research is needed to understand what functional class is represented.

7 CONCLUSION

Our work develops the first analytical bridge between feedforward architectures and KANs. Specifically, in the context of piecewise linear functions, we are able to switch between the two representations. This allows users to leverage the trainability of ReLU networks and convert them to (piecewise linear) KANs in order to grab the interpretability benefits. In the other direction, transforming KANs into ReLU networks enables researchers and users to deploy existing techniques to analyse KANs, by importing tools from the rich literature on polyhedral decompositions (Huchette et al., 2023), for example analysing symmetries in parameter space (Grigsby et al., 2023) and extracting the polyhedral complex computationally (Montufar et al., 2014; Villani and Schoots, 2023; Berzins, 2023).

An important corollary of our work is that we show that any piecewise linear function can be expressed as a piecewise linear KAN. This statement was already shown to be true for ReLU functions, i.e. any piecewise linear function can be expressed as a ReLU network (He et al., 2018). Based on this fact and our transformation from ReLU networks to KANs, we can conclude the corollary.

In both directions we show the efficiency of our transformations: the transformed model in one direction (from ReLU to KANs) requires an extra linear term, and in the other direction (from KANs to ReLU) it requires no any extra non-zero parameters.

7.1 Limitations and Future Work

There are still a variety of open questions regarding the expressivity of KANs. For example, suppose we have a piecewise linear function ψ, what is the smallest (in terms of parameter count) KAN f and ReLU network g that can represent this function?

Polyhedral extraction methods, that compute the polyhedral partition and linear coefficients of each part, for KANs would unlock further interpretability benefits. In particular, this would enable new insight into how parameters affect the linear parts of KANs.

Future work could also focus on exploring methods to represent arbitrary finite piecewise linear functions with KANs and ReLU networks with a minimal number of parameters. This would clarify the parameter efficiency of each architecture class, which could be useful for reducing storage costs and in mobile applications.

Acknowledgements

This work was supported by the EPSRC Grant EP/S023356/1 (www.safeandtrustedai.org). We would like to thank Peter McBurney and the reviewers for their helpful comments.

References

Actor, J. and Knepley, M. G. (2017). An algorithm for computing lipschitz inner functions in kolmogorov's superposition theorem. *arXiv preprint arXiv:1712.08286*.

Arora, R., Basu, A., Mianjy, P., and Mukherjee, A. (2016). Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*.

Berzins, A. (2023). Polyhedral complex extraction from relu networks using edge subdivision. In *International Conference on Machine Learning*, pages 2234–2244. PMLR.

Braun, J. and Griebel, M. (2009). On a constructive proof of kolmogorov's superposition theorem. *Constructive approximation*, 30:653–675.

Grigsby, E., Lindsey, K., and Rolnick, D. (2023). Hidden symmetries of relu networks. In *Inter-*

national Conference on Machine Learning, pages 11734–11760. PMLR.

He, J., Li, L., Xu, J., and Zheng, C. (2018). Relu deep neural networks and linear finite elements. *arXiv preprint arXiv:1807.03973*.

Huchette, J., Muñoz, G., Serra, T., and Tsay, C. (2023). When deep learning meets polyhedral theory: A survey. *arXiv preprint arXiv:2305.00241*.

Humayun, A. I., Balestriero, R., and Baraniuk, R. (2022). Exact visualization of deep neural network geometry and decision boundary. In *NeurIPS 2022 Workshop on Symmetry and Geometry in Neural Representations*.

Ismayilova, A. and Ismailov, V. E. (2024). On the kolmogorov neural networks. *Neural Networks*, 176:106333.

Kolmogorov, A. (1956). On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables. *Proceedings of the USSR Academy of Sciences*.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1989). Handwritten digit recognition with a back-propagation network. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pages 396–404. Morgan Kaufmann.

Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljacic, M., Hou, T. Y., and Tegmark, M. (2024). KAN: kolmogorov-arnold networks. *CoRR*, abs/2404.19756.

Masden, M. (2022). Algorithmic determination of the combinatorial structure of the linear regions of relu neural networks. *arXiv preprint arXiv:2207.07696*.

Montufar, G. (2017). Notes on the number of linear regions of deep neural networks. *SampTA*.

Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. *Advances in Neural Information Processing Systems*, 27(NeurIPS).

Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., and Sohl-Dickstein, J. (2017). On the expressive power of deep neural networks. In *International Conference on Machine Learning*, pages 2847–2854. PMLR.

Schmidt-Hieber, J. (2021). The kolmogorov–arnold representation theorem revisited. *Neural networks*, 137:119–126.

Serra, T. and Ramalingam, S. (2020). Empirical bounds on linear regions of deep rectifier networks. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press.

Serra, T., Tjandraatmadja, C., and Ramalingam, S. (2018). Bounding and counting linear regions of deep neural networks. In *ICML*. PMLR.

Sudjianto, A., Knauth, W., Singh, R., Yang, Z., and Zhang, A. (2020). Unwrapping the black box of deep relu networks: interpretability, diagnostics, and simplification. *arXiv*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Villani, M. J. and Schoots, N. (2023). Any deep relu network is shallow. *arXiv preprint arXiv:2306.11827*.

Yang, Z., Zhang, A., and Sudjianto, A. (2021). Gami-net: An explainable neural network based on generalized additive models with structured interactions. *Pattern Recognition*, 120:108192.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Not Applicable]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Not Applicable]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes/No/Not Applicable]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Not Applicable]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Not Applicable]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Not Applicable]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Not Applicable]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

A AN ANALOGOUS CONVERSION FOR KANS WITH B-SPLINE ACTIVATION FUNCTIONS

In this appendix, we describe a conversion that applies to KANs with B-spline activation functions rather than KANs with piecewise linear activation functions, but that is very similar in other respects. The target of this conversion is not a standard ReLU network: feedforward networks with ReLU activations represent piecewise linear functions, and since KANs with B-spline activations are not necessarily piecewise linear, this is impossible. Instead, to carry out an analogous operation, we need to introduce an unconventional architecture that combines both ReLU activations (for the breakpoints in the splines) and monomial activations (for the polynomials).

Specifically, we convert a KAN with B-spline activations of degree at most r, to the following architecture that we call a (ReLU, x^r)-architecture. A block in this architecture consists of the following:

1. an affine linear layer $\mathbb{R}^n \to \mathbb{R}^{(r+1)n'}$ for any integers $n, n' \geq 1$; followed by
2. a ReLU layer $\mathbb{R}^{(r+1)n'} \to \mathbb{R}^{(r+1)n'}$, i.e., a component-wise application of ReLU ; and lastly
3. the monomial activation functions $\sigma_r \colon \mathbb{R}^{(r+1)n'} \to \mathbb{R}^{(r+1)n'}$ that on each of the n' copies of $\mathbb{R}^{r+1}$ inside $\mathbb{R}^{(r+1)n'}$ are defined by $\sigma_r(y_0, y_1, \ldots, y_r) = (1, y_1^1, y_2^2, \ldots, y_r^r)$, i.e., it is the monomial x^j on the j-th component.

Converting a (ReLU, x^r)-architecture to a KAN with B-splines This direction is simple, and is based on the same idea as for piecewise linear KANs (see Theorem 1): every activation function in the (ReLU, x^r)-architecture is in particular a spline, so you can directly replace each of the 3 types of layers in a block directly with a KAN layer.

Converting a KAN with B-splines to a (ReLU, x^r)-architecture This direction is more involved, but also roughly mimics the core ideas from the proof of Theorem 2: for each breakpoint b_i in a spline, we create intermediate neurons to represent the function $x \mapsto \mathrm{ReLU}(x - b_i)$ (similar to Figure 2), and then post-compose these functions with monomials to create a telescoping sum of polynomials that is equal to the original spline. (For piecewise linear KANs, we do not use monomials, but linear functions, as illustrated in Figure 2.) We do this for every layer in the KAN, and the result is a (ReLU, x^r)-architecture. The rest of this appendix explains this construction in more detail.

Let's first take the example of a polynomial of degree r:

$$p = \sum_{i=0}^{r} a_i x^i.$$

We can write:

$$p = P_{\mathbf{a}}(x) := \mathbf{a}\sigma_r(\mathbf{1}_{r+1}^T x),$$

where $x \in \mathbb{R}, \mathbf{1}_{r+1} \in \mathbb{R}^{r+1}$ is a vector of ones, $\mathbf{a} = (a_0, \ldots a_r) \in \mathbb{R}^{r+1}$ is a vector of coefficients, and where $\sigma_r \colon \mathbb{R}^{r+1} \to \mathbb{R}^{r+1}$ is the map of monomial activation functions defined above, i.e., it operates element-wise and raises the i-th coordinate to the i-th power:

$$\sigma_r(\mathbf{y})_i = y_i^i \qquad \text{for } \mathbf{y} = (y_0, y_1, \ldots, y_r) \in \mathbb{R}^{r+1}.$$

This expression shows that we can use the monomial activation functions to create a (ReLU, x^r)-network that represents an arbitrary polynomial. It remains to show that we can combine this with the ReLU activation functions to create splines.

For a polynomial $P_{\mathbf{a}}$ as above and a threshold $b \in \mathbb{R}$, we define $P_{\mathbf{a}}^b$ as the polynomial that satisfies

$$P_{\mathbf{a}}^b(x) := P_{\mathbf{a}}(x + b).$$

With this polynomial and the ReLU activation function, we can create a (ReLU, x^r)-network that exactly represents the function

$$P_{\mathbf{a}}^b(\text{ReLU}(x-b)).$$

This function has the useful property that on $x \geq b$, it is equal to $P_{\mathbf{a}}(x)$, and on $x \leq b$, it is constant and equal to $P_{\mathbf{a}}(b)$.

Having shown that we can represent the functions $P_{\mathbf{a}}^b(x)$ in a (ReLU, x^r)-architecture, we can now use these functions to represent any single-valued B-spline with finitely many pieces as follows. Given a B-spline, let $b_1, \ldots, b_{k-1}$ denote the breakpoints between the segments, and denote the k polynomial functions on the polynomial segments by $P_{\mathbf{a}_1}(x), \ldots, P_{\mathbf{a}_k}(x)$. (This is similar to Figure 2, but with polynomials instead of linear functions.) This B-spline is exactly represented by the function

$$P_{\mathbf{a}_1}(x) + (P_{\mathbf{a}_2}^{b_1} - P_{\mathbf{a}_1}^{b_1})(\text{ReLU}(x-b_1)) + ... + (P_{\mathbf{a}_k}^{b_{k-1}} - P_{\mathbf{a}_{k-1}}^{b_{k-1}})(\text{ReLU}(x-b_{k-1})). \tag{2}$$

This can be easily checked using the property for $P_{\mathbf{a}}^b$ mentioned above, from which it follows that for any $1 \leq i \leq k$ and $x \leq b_i$, all but the first i terms cancel out, and the first i terms form a telescoping sum that is equal to $P_{\mathbf{a}_1}(x) + (P_{\mathbf{a}_2}(x) - P_{\mathbf{a}_1}(x)) + \ldots + (P_{\mathbf{a}_i}(x) - P_{\mathbf{a}_{i-1}}(x)) = P_{\mathbf{a}_i}(x)$. This is analogous to what we do in the proof of Lemma 1.

The function in Equation (2) can be represented by the (ReLU, x^r)-architecture because it is a sum of polynomials that can be represented by the (ReLU, x^r)-architecture. We can now reason in the same way as in the main text: because we can convert the activation functions in the KAN, we can stack and concatenate (ReLU, x^r)-networks to convert the entire KAN.