Sprecher Networks: A generalized neural architecture for multivariate function approximation

Christian Hägg Boris Shapiro

Abstract

We present *Sprecher Networks*, a family of trainable neural architectures inspired by the classical Kolmogorov–Arnold–Sprecher construction for approximating multivariate continuous functions. Unlike typical deep networks, these models rely on *monotonic and general splines* to implement learnable transformations with explicit shifts and summations. Our approach reproduces Sprecher's original one-layer "sum of shifted splines" formula and extends it to deeper, multi-layer compositions. We discuss theoretical motivations, implementation details, and potential advantages in interpretability and universality.

1 Introduction and historical background

Approximation of continuous functions by sums of univariate functions has been a recurring theme in mathematical analysis and neural networks. Kolmogorov's Superposition Theorem (1957), refined by Arnold, established that any multivariate continuous function $f : [0,1]^d \to \mathbb{R}$ can be represented as a finite sum of strictly increasing univariate functions applied to affine transformations of $x_1, \ldots, x_d$.

David Sprecher's 1965 construction.

$$f(\mathbf{x}) = \sum_{q=0}^{2n} \Phi\Big(\sum_{p=1}^{n} \lambda_p \, \phi\big(x_p + \eta\, q\big) + q \Big)$$

In his 1965 landmark paper, David Sprecher [2] proposed a simplified version of the Kolmogorov–Arnold representation that specifically replaces families of $\phi_{q,p}$ by a single monotonic function ϕ with suitable argument shifts. Concretely, Sprecher showed that this single-layer construction suffices to approximate any d-variate continuous function when $n \geq d$ and the parameters $(\eta, \lambda_1, \ldots, \lambda_n)$ and splines (ϕ, Φ) are chosen carefully. His result preserves the key insight of *shifting the input coordinates* by multiples of η and summing the partial evaluations under a final univariate map Φ.

2 Motivation and overview of Sprecher Networks

Modern deep learning practice often focuses on conventional feedforward architectures. However, Sprecher's approach offers a highly interpretable alternative:

- Each layer in our network is organized around a fixed monotonic spline $\phi(\cdot)$ and a more general spline $\Phi(\cdot)$.
- It incorporates trainable shifts via a scalar η, and trainable weights λ that mix each input coordinate before passing them into ϕ.
- Finally, the network sums over shifted partial evaluations inside an outer spline, closely mimicking Sprecher's formula.

Our *Sprecher Networks* generalize the classical shift-and-sum construction to a multi-layer architecture by stacking hidden layers. In our design, the mapping from one hidden layer (i.e. the collection of nodes in that layer) to the next is realized by a *Sprecher block*: a functional operator that applies a shift-and-sum

transformation using a shared pair of spline functions. Unlike conventional networks, the nodes within each hidden layer are not individually parameterized; rather, they are produced uniformly by the common transformation defined by the Sprecher block. That is, each Sprecher block applies

$$(x_i)_{i=1}^{d_{\text{in}}} \;\mapsto\; \Big[\Phi\Big(\sum_i \lambda_{i,q}\,\phi(x_i+\eta\,q)+q\Big)\Big]_{q=0}^{d_{\text{out}}-1},$$

and in the scalar-output case the outputs of the final Sprecher block are aggregated (via summation over q) to produce a single-dimensional output. In the original 1965 paper, Sprecher used only one such layer (with $2n+1$ summands). Our approach allows the network to follow the progression

$$d_0 \to d_1 \to \cdots \to d_{L-1} \to 1,$$

by stacking L Sprecher blocks (each embedding the shift-and-sum transformation). In this way, we recover a deeper analog of the Kolmogorov–Arnold–Sprecher construction with enhanced expressive power while maintaining a clear theoretical connection to the classical result.

Definition 1 (Network notation)**.** *Throughout this paper, we denote Sprecher Network architectures using arrow notation of the form $d_{\text{in}} \to [d_1, d_2, \ldots, d_L] \to d_{\text{out}}$, where d_{in} is the input dimension, $[d_1, d_2, \ldots, d_L]$ represents the hidden layer dimensions, and d_{out} is the output dimension. For example, $2 \to [5, 3, 8] \to 1$ describes a network with 2-dimensional input, three hidden layers of widths 5, 3, and 8 respectively, and a scalar output. When input or output dimensions are clear from context, we may use the abbreviated notation $[d_1, d_2, \ldots, d_L]$ to focus on the hidden layer structure.*

3 Core architectural details

In our architecture, the fundamental building unit is the *Sprecher block*. The network is composed of a sequence of Sprecher blocks, each of which performs a shift-and-sum transformation inspired by Sprecher's original construction. The following subsection details the structure of a single Sprecher block.

3.1 Sprecher block structure

A Sprecher block transforms an input vector from $\mathbb{R}^{d_{\text{in}}}$ to an output vector in $\mathbb{R}^{d_{\text{out}}}$ and consists of the following components:

- **Monotonic spline** $\phi(\cdot)$: an increasing piecewise-linear function defined on a prescribed interval (typically $[0, 1]$ or a slightly wider interval).
- **General spline** $\Phi(\cdot)$: a piecewise-linear function defined on a sufficiently large interval (e.g., $[-10, 12]$) that does not require monotonicity.
- A matrix of **mixing weights** $\{\lambda_{i,q}\}$ of size $d_{\text{in}} \times d_{\text{out}}$, which linearly combines the outputs of the monotonic spline.
- A scalar **shift parameter** $\eta > 0$, which controls how the index q shifts each coordinate.

Concretely, given an input vector $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$, a single Sprecher block (with its own parameters ϕ, Φ, η, and λ) computes its q-th output (with $q = 0, \ldots, d_{\text{out}} - 1$) via

$$\text{Block}_{\phi,\Phi,\eta,\lambda}(\mathbf{x})_q = \Phi\left(\sum_{i=1}^{d_{\text{in}}} \lambda_{i,q}\,\phi\Big(x_i + \eta\,q\Big) + q\right).$$

Note that in a network with two or more hidden layers, each Sprecher block is endowed with its own set of parameters (which may be denoted by $\phi^{(\ell)}$, $\Phi^{(\ell)}$, $\eta^{(\ell)}$, and $\lambda^{(\ell)}$ for the ℓ-th block). This operation implements a shift-and-sum transformation: each input coordinate is shifted by a multiple of η, passed through the monotonic spline ϕ, linearly combined with the mixing weights, and then transformed by the general spline Φ (with an added constant term q to preserve the shift). By stacking Sprecher blocks as hidden layers, our network constructs a deep, compositional representation that generalizes Sprecher's original one-layer shift-and-sum formulation.

3.2 Layer composition and final mapping

Let L denote the number of hidden layers in the network. In our architecture, a "hidden layer" refers to the collection of nodes produced by a transformation, while the mapping from one hidden layer to the next is implemented by a *Sprecher block*: a function that applies a shift-and-sum operation via a shared pair of spline functions. This means that, unlike traditional neural networks where each neuron is separately parameterized, the nodes in each hidden layer here are generated uniformly by the Sprecher block's common spline parameters, with diversity arising only through their associated mixing weights and index shifts.

Suppose that the ℓ-th Sprecher block maps

$$\mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell},$$

with d_0 equal to the input dimension. In the ℓ-th Sprecher block, a monotonic spline $\phi^{(\ell)}$ and a general spline $\Phi^{(\ell)}$ are applied together with the mixing weights $\{\lambda_{i,q}^{(\ell)}\}$ and a shift parameter $\eta^{(\ell)}$. If we denote the output of the ℓ-th block by $\mathbf{h}^{(\ell)}$ (with $\mathbf{h}^{(0)} = \mathbf{x}$), then the ℓ-th block computes its q-th output as

$$\mathbf{h}_q^{(\ell)} = \Phi^{(\ell)}\left(\sum_{i=1}^{d_{\ell-1}} \lambda_{i,q}^{(\ell)}\, \phi^{(\ell)}\Big(\mathbf{h}_i^{(\ell-1)} + \eta^{(\ell)}\, q\Big) + q\right), \quad q = 0, \ldots, d_\ell - 1.$$

There are two cases for composing the blocks, depending on the desired final output dimension m (i.e. the number of nodes in the output layer):

(a) Scalar output ($m = 1$): In this configuration, the final Sprecher block is designed so that its d_L outputs are aggregated by summation to yield a scalar output. Formally, we define

$$f(\mathbf{x}) = \sum_{q=0}^{d_L-1} \mathbf{h}_q^{(L)}.$$

Thus, the overall function can be written in terms of the operators

$$T^{(\ell)} : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}, \quad \text{with} \quad \Big(T^{(\ell)}(z)\Big)_q = \Phi^{(\ell)}\left(\sum_{i=1}^{d_{\ell-1}} \lambda_{i,q}^{(\ell)}\, \phi^{(\ell)}\Big(z_i + \eta^{(\ell)}\, q\Big) + q\right),$$

as

$$f(\mathbf{x}) = \sum_{q=0}^{d_L-1} \Big(T^{(L)} \circ T^{(L-1)} \circ \cdots \circ T^{(1)}\Big)(\mathbf{x})_q. \tag{1}$$

(b) Vector-valued output ($m > 1$): When the target function f maps into $\mathbb{R}^m$ with $m > 1$, the network first constructs L hidden layers (each implemented as a Sprecher block), yielding a hidden representation $\mathbf{h}^{(L)} \in \mathbb{R}^{d_L}$. An additional output block is then appended to map $\mathbf{h}^{(L)}$ to $\mathbb{R}^m$ without performing any summation (i.e., leaving the output as a q-dependent vector rather than collapsing it to a scalar, to preserve the distinct contributions of each channel). In particular, define

$$\Big(T^{(L+1)}(z)\Big)_q = \Phi^{(L+1)}\left(\sum_{r=0}^{d_L-1} \lambda_{r,q}^{(L+1)}\, \phi^{(L+1)}\Big(z_r + \eta^{(L+1)}\, q\Big) + q\right), \quad q = 0, \ldots, m-1,$$

with $z = \mathbf{h}^{(L)}$. Then the overall network function is given by

$$f(\mathbf{x}) = \Big(T^{(L+1)} \circ T^{(L)} \circ \cdots \circ T^{(1)}\Big)(\mathbf{x}) \in \mathbb{R}^m. \tag{2}$$

Note that this output block is constructed in the same manner as the hidden Sprecher blocks (with its own $\phi^{(L+1)}$ and $\Phi^{(L+1)}$), except that it does not include a summation over its outputs. Its role is solely to transform the final hidden representation into an m-dimensional output.

In summary, when $L \geq 1$ and the final output is scalar, the network consists of exactly L blocks and thus $2L$ spline functions (one pair per block). When the output is vector-valued (with $m > 1$), an additional block is appended, resulting in $L+1$ blocks and $2(L+1)$ spline functions. This extra block naturally extends Sprecher's original construction to the vector-valued setting.

We illustrate the case of a network with architecture $d_0 \to [d_1, d_2, d_3] \to m$ (i.e., $L = 3$ hidden layers with $m > 1$ outputs) below. Let $X^{(0)}$ be the input, and $X^{(3)}$ the output of the third block. The "extra" final block then maps $X^{(3)}$ to $\mathbb{R}^m$:

$$X^{(0)} \xrightarrow{\text{Sprecher block 1}} X^{(1)} \xrightarrow{\text{Sprecher block 2}} X^{(2)} \xrightarrow{\text{Sprecher block 3}} X^{(3)} \downarrow \text{output block} \downarrow \mathbb{R}^m \text{ (output)}$$

Note: Here, $X^{(\ell)}$ denotes the output of the ℓ-th Sprecher block (i.e., the hidden-layer representation after ℓ transformations). Each Sprecher Block internally applies a pair of spline functions $(\phi^{(\ell)}, \Phi^{(\ell)})$, a weight matrix, and a shift parameter to transform its input. The diagram now emphasizes that the hidden representations $X^{(1)}, X^{(2)}, X^{(3)}$ are the complete outputs of their respective blocks, and the final (output) block transforms $X^{(3)}$ into the m-dimensional output without an additional summation.

This clarifies that while the classical Sprecher formula involves two splines in a single block (with summation to yield a scalar), a vector-valued extension naturally introduces an extra block and an additional pair of splines.

We will now illustrate three examples with $L = 1, 2, 3$ hidden layers in the common scalar-output case (i.e. $m = 1$). These examples demonstrate how the architecture—when used in its most common setting—retains the simplicity and interpretability of Sprecher's original one-layer, shift-and-sum formulation while allowing for deeper, compositional representations.

3.2.1 Single hidden layer ($L = 1$)

In our architecture, the transformation between hidden layers is performed by a Sprecher block — a function that applies a pair of spline functions (ϕ and Φ), a weight matrix, and a shift parameter to its input. Here, "hidden layer" specifically denotes the collection of nodes produced by this transformation, which are not individually parameterized as in traditional neural networks. For a network with architecture $d_{\text{in}} \to [d_1] \to 1$ (i.e., $d_0 = d_{\text{in}}$ inputs and one hidden layer of dimension d_1), the network computes

$$f(\mathbf{x}) = \sum_{q=0}^{d_1-1} \Phi^{(1)} \left(\sum_{i=1}^{d_0} \lambda_{i,q}^{(1)} \, \phi^{(1)} \Big(x_i + \eta^{(1)} \, q \Big) + q \right).$$

This precisely reproduces Sprecher's 1965 construction

$$f(\mathbf{x}) = \sum_{q=0}^{2n} \Phi \left(\sum_{p=1}^{n} \lambda_p \, \phi \Big(x_p + \eta \, q \Big) + q \right)$$

when we set $d_1 = 2d_0 + 1$ and identify $\phi^{(1)} = \phi,\ \Phi^{(1)} = \Phi$.

3.2.2 Two hidden layers ($L = 2$)

Let $d_0 = d_{\text{in}}, d_1$ be the size of the first hidden layer, and d_2 the size of the second (final) layer, which is summed to yield a scalar output. Denoting the intermediate output as

$$\mathbf{h}_r^{(1)} = \Phi^{(1)} \Big(\sum_{i=1}^{d_0} \lambda_{i,r}^{(1)} \, \phi^{(1)} \Big(x_i + \eta^{(1)} \, r \Big) + r \Big), \quad r = 0, \ldots, d_1 - 1,$$

the second block becomes

$$\mathbf{h}_q^{(2)} = \Phi^{(2)}\Big(\sum_{r=0}^{d_1-1} \lambda_{r,q}^{(2)}\, \phi^{(2)}\Big(\mathbf{h}_r^{(1)} + \eta^{(2)}\, q\Big) + q\Big), \quad q = 0, \ldots, d_2 - 1.$$

Finally, the network output is

$$f(\mathbf{x}) = \sum_{q=0}^{d_2-1} \mathbf{h}_q^{(2)}.$$

Equivalently, we can write this as

$$f(\mathbf{x}) = \sum_{q=0}^{d_2-1} \Phi^{(2)}\Big(\sum_{r=0}^{d_1-1} \lambda_{r,q}^{(2)}\, \phi^{(2)}\Big(\Phi^{(1)}\Big(\sum_{i=1}^{d_0} \lambda_{i,r}^{(1)}\, \phi^{(1)}\Big(x_i + \eta^{(1)}\, r\Big) + r\Big) + \eta^{(2)}\, q\Big) + q\Big).$$

3.2.3 Three hidden layers ($L = 3$)

Now let the layer dimensions be $d_0 \to d_1 \to d_2 \to d_3$, with the final block summed to yield a scalar output. In recursive form, we have

$$\mathbf{h}_r^{(1)} = \Phi^{(1)}\Big(\sum_{i=1}^{d_0} \lambda_{i,r}^{(1)}\, \phi^{(1)}\Big(x_i + \eta^{(1)}\, r\Big) + r\Big),$$

$$\mathbf{h}_s^{(2)} = \Phi^{(2)}\Big(\sum_{r=0}^{d_1-1} \lambda_{r,s}^{(2)}\, \phi^{(2)}\Big(\mathbf{h}_r^{(1)} + \eta^{(2)}\, s\Big) + s\Big),$$

$$\mathbf{h}_q^{(3)} = \Phi^{(3)}\Big(\sum_{s=0}^{d_2-1} \lambda_{s,q}^{(3)}\, \phi^{(3)}\Big(\mathbf{h}_s^{(2)} + \eta^{(3)}\, q\Big) + q\Big).$$

Then the network output is

$$f(\mathbf{x}) = \sum_{q=0}^{d_3-1} \mathbf{h}_q^{(3)}.$$

The equivalent formulation with nested sums is

$$f(\mathbf{x}) = \sum_{q=0}^{d_3-1} \Phi^{(3)}\Big(\sum_{s=0}^{d_2-1} \lambda_{s,q}^{(3)}\, \phi^{(3)}\Big(\Phi^{(2)}\Big(\sum_{r=0}^{d_1-1} \lambda_{r,s}^{(2)}\, \phi^{(2)}\Big(\Phi^{(1)}\Big(\sum_{i=1}^{d_0} \lambda_{i,r}^{(1)}\, \phi^{(1)}\Big(x_i + \eta^{(1)}\, r\Big) + r\Big) + \eta^{(2)}\, s\Big) + s\Big) + \eta^{(3)}\, q\Big) + q\Big).$$

Remark 1. *These expansions show how each layer's output is obtained by a shift-and-sum transformation using a pair of splines $\phi^{(\ell)}$ and $\Phi^{(\ell)}$. In the scalar-output case, the final layer sums over its output index q to yield a single scalar. In the vector-valued output case (i.e. when $f : [0,1]^d \to \mathbb{R}^m$ with $m > 1$), we append an extra output block that is constructed exactly as a Sprecher block but, unlike the final block in the scalar-output case, it does not perform a summation over its outputs. In other words, the extra block produces an m-dimensional vector directly (the formula remains the same as for a hidden block, except that the summation over the output index q is omitted).*

4 Relation to Sprecher (1965) and its generalization

In his 1965 paper [2], David Sprecher proved a version of the Kolmogorov–Arnold theorem by showing that

$$f(\mathbf{x}) = \sum_{q=0}^{2n} \Phi\Big(\sum_{p=1}^{n} \lambda_p\, \phi\big(x_p + \eta\, q\big) + q\Big),$$

for suitable choices of monotonic ϕ and continuous Φ. Our single-layer Sprecher Network with $d = n$ inputs and $d_{\text{out}} = 2n + 1$ outputs, *followed by a sum over q*, reproduces precisely that formula.

Going deeper. While Sprecher's original construction used one layer, we can compose multiple shift-and-sum blocks to form a deeper architecture. Each block can reduce or transform the intermediate dimension, thereby creating new function classes that remain "Sprecher-like" but with greater representational flexibility.

5 Implementation Highlights

5.1 Trainable splines

Each block has two spline modules:

- $\phi^{(\ell)}$, a monotonic piecewise-linear spline defined on $[0, 1]$ (or a slightly wider interval), initialized so that it is increasing.
- $\Phi^{(\ell)}$, a general piecewise-linear spline defined on, say, $[-10, 12]$ (or any sufficiently wide domain). This spline is not necessarily monotonic and can be initialized near the identity function.

Both spline types typically have many knots (e.g. 100–300), which goes well beyond the requirements of the original theorem that only demanded continuity and monotonicity for ϕ.

In practice, a flatness penalty can be introduced to encourage smooth approximations, a design choice partially inspired by earlier numerical methods (see [4]). This penalty serves to regularize the spline functions.

5.2 Shifts and weights

Each block has a scalar $\eta^{(\ell)}$ controlling how the block's index q shifts its input coordinates, plus a matrix $\lambda_{i,q}^{(\ell)}$ mixing each input. These free parameters are updated via gradient-based optimizers to minimize a loss function of the form

$$(\text{MSE on training data}) + \alpha \times (\text{penalty on spline second derivatives}),$$

where the penalty encourages smoother spline functions.

5.3 Parameter counting

Since each block contains two splines (each with potentially hundreds of knot parameters), the total number of trainable parameters is typically dominated by the spline coefficients. If each $\phi^{(\ell)}$ has K_ϕ knots and each $\Phi^{(\ell)}$ has K_Φ knots, then each block contributes roughly $K_\phi + K_\Phi$ spline parameters, plus $d_{\ell-1}\, d_\ell$ weight parameters and one $\eta^{(\ell)}$. Summing over all blocks yields the total parameter count. Note that Sprecher Networks have a significant advantage over KANs (see [5]) in that their parameter counts don't explode for wide networks.

5.4 Sprecher-like theorems

If we restrict to one layer with $d_{\text{out}} = 2d + 1$, a monotonic ϕ, and suitable constants (λ_i, η), then we reproduce the original Sprecher formula. Sprecher's 1965 argument implies that this single-layer network can approximate any continuous function $f : [0, 1]^d \to \mathbb{R}$ arbitrarily well, provided ϕ and Φ are sufficiently flexible.

An interesting possibility arises when we consider the case of a single hidden layer with vector output. In particular, for a network approximating a function $f : [0, 1]^d \to \mathbb{R}^m$ with $m > 1$, one could set $d_{\text{out}} = 2d+1$ in the hidden layer and then append an output block that omits the final summation over q. This construction would then possibly generalize Sprecher's original scalar formulation to the vector-valued setting. Although the precise relationship between the number of hidden nodes and the output dimension may depend on further architectural details—and the universality of such an extension remains highly conjectural—it suggests a natural direction for future theoretical investigation.

Explicitly, we can formulate this conjecture as follows:

Conjecture 1 (Vector-valued Sprecher representation)**.** *Let $n \in \mathbb{N}$ and let $f : [0,1]^n \to \mathbb{R}^m$ be any continuous function with $m > 1$. Then there exist monotonic splines $\phi^{(1)} : [0,1] \to \mathbb{R}$ and $\phi^{(2)} : J \to \mathbb{R}$, general splines $\Phi^{(1)} : I_1 \to \mathbb{R}$ and $\Phi^{(2)} : J \to \mathbb{R}$ (defined on suitable intervals I_1 and J), mixing weight matrices $\lambda_{i,r}^{(1)}$ for $1 \le i \le n$ and $0 \le r \le 2n$ and $\lambda_{r,q}^{(2)}$ for $0 \le r \le 2n$ and $0 \le q \le m-1$, and shift parameters $\eta^{(1)}, \eta^{(2)} > 0$ such that for every $\mathbf{x} = (x_1, \ldots, x_n) \in [0,1]^n$ one has*

$$f(\mathbf{x}) = \left[\Phi^{(2)} \left(\sum_{r=0}^{2n} \lambda_{r,q}^{(2)} \, \phi^{(2)} \left(\Phi^{(1)} \Big(\sum_{i=1}^{n} \lambda_{i,r}^{(1)} \, \phi^{(1)} \Big(x_i + \eta^{(1)} \, r \Big) + r \Big) + \eta^{(2)} \, q \right) + q \right) \right]_{q=0}^{m-1}. \tag{3}$$

Remark 2. *In the case $m = 1$, the representation in* (3) *reduces (upon summing over q) to Sprecher's original formulation.*

5.5 Deeper extensions

A natural conjecture is that stacking these blocks preserves universal approximation capability while potentially requiring fewer summation nodes. Intuitively, multi-layer compositions can capture complex structure more efficiently. However, a complete theoretical treatment of depth versus width for Sprecher Networks warrants further research.

6 Empirical demonstrations

In practice, we train Sprecher Networks on small data sets sampled from a target function f. Two classic examples are:

- **1D case:** A polynomial or sinusoidal function on $[0,1]$. The network learns ϕ and Φ splines that approximate f very accurately, effectively acting as a learnable spline interpolant.
- **2D case:** A surface such as

  $$f(x,y) = \frac{1}{7} \Big(\exp\big(\sin(\pi x) + y^2\big) - 1 \Big).$$

  With sufficient training (typically after $O(10^5)$ gradient updates), the network achieves good approximation. Layerwise spline plots provide an interpretable view of how the network transforms inputs.

Figure 1 demonstrates a trained Sprecher Network with architecture $2 \to [5,8,5] \to 1$, approximating a 2D surface. Similarly, Figure 2 shows a Sprecher Network approximating a vector-valued function

$$f(x,y) = \Big(\frac{\exp\big(\sin(\pi x) + y^2\big) - 1}{7}, \; \frac{1}{4}\, y + \frac{1}{5}\, y^2 - x^3 + \frac{1}{5} \sin(7x) \Big)$$

with domain and codomain in $\mathbb{R}^2$. In this case the network has architecture $2 \to [20,20,20,20,20] \to 2$, which is implemented with six Sprecher blocks (where the sixth Sprecher block is used for the vector-valued output). In both figures the top row displays the learned spline functions for each block (monotonic splines in cyan and general splines in magenta), while the bottom row compares the target surfaces with the network outputs. This illustrates that the network, whether approximating an $\mathbb{R}^2 \to \mathbb{R}$ or an $\mathbb{R}^2 \to \mathbb{R}^2$ function, is capable of capturing the underlying function's intricacies.

7 Challenges and future directions

7.1 Implementation challenges

1. **Difficulties in training spline functions:**
   Finding good spline shapes through gradient-based optimization can be quite challenging. Due to the inherent flexibility of the trainable splines, the network can sometimes settle into local minima where

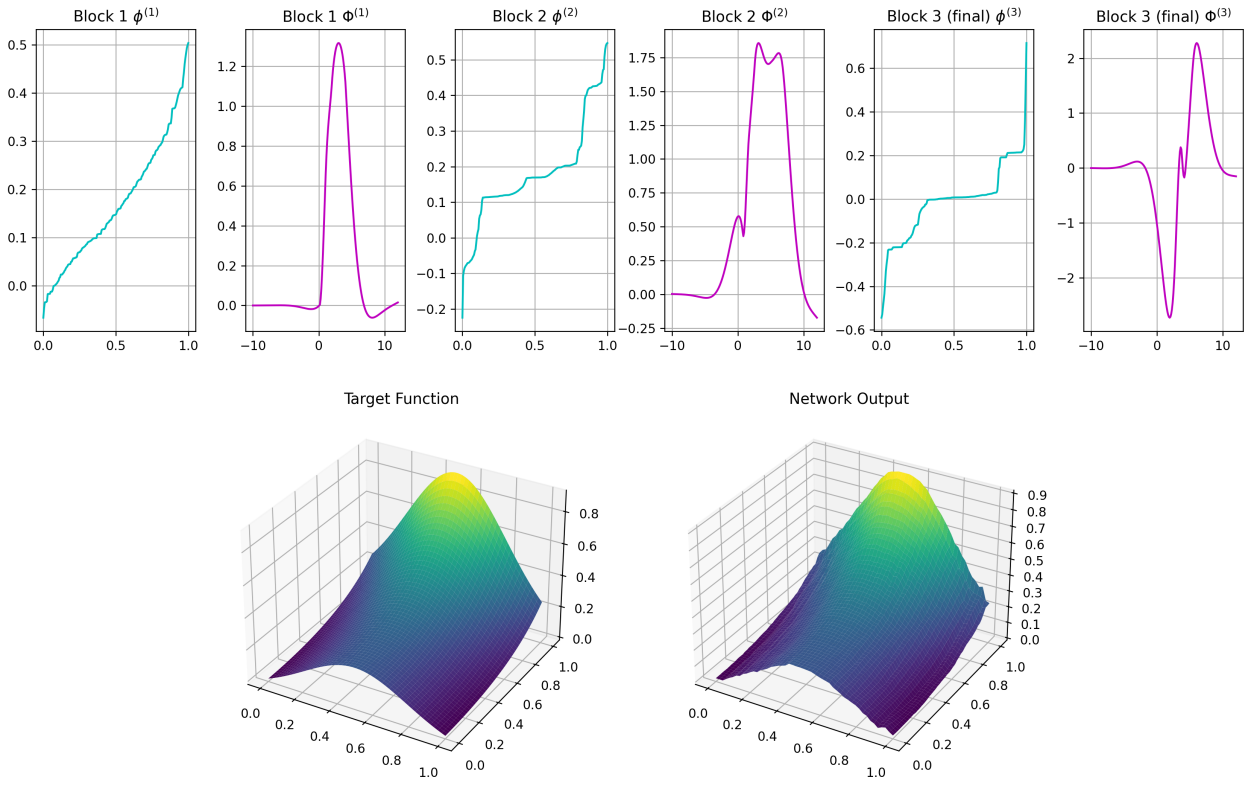

Figure 1: Visualization of a trained $2 \to [5, 8, 5] \to 1$ Sprecher Network approximating a 2D target function $z = f(x, y)$. **Top row:** The learned spline functions for each block - monotonic splines $\phi^{(\ell)}$ (cyan) and general splines $\Phi^{(\ell)}$ (magenta). **Bottom row:** Comparison between the target function (left) and the network approximation (right) showing the high-quality fit achieved after training.

the learned functions flatten out, resulting in "horizontal" outputs; that is, outputs that remain nearly constant regardless of the input. For example, when training on data from a scalar function $f : \mathbb{R} \to \mathbb{R}$, the network may produce an output that is nearly a horizontal line; for data from a function $f : \mathbb{R}^2 \to \mathbb{R}$, the output may form a nearly horizontal plane; and more generally, for a function $f : \mathbb{R}^m \to \mathbb{R}$ with $m > 2$, the network output can yield horizontal hyperplanes. Similarly, for vector-valued outputs the network may settle on multiple flat outputs, often resulting in parallel horizontal hyperplanes. This stagnation significantly impedes the network's ability to accurately approximate the target function, and escaping such flat regions in the loss landscape may require better spline initialization strategies.

2. **Initialization and domain/codomain selection:**
The original Sprecher construction suggests that the monotonic spline ϕ should be defined as a mapping from $[0, 1]$ to $[0, 1]$ while the general spline Φ is a function from $\mathbb{R}$ to $\mathbb{R}$. In implementations, however, restricting the domains and codomains of these splines to bounded intervals is necessary, e.g. by choosing intervals of the form $[-a, a]$ with a sufficiently large a (for example, $a \approx 10$). Although this empirical choice works well across different function approximation tasks, it remains unclear how to determine these intervals optimally. A natural idea is to initialize the Φ functions as $[0, 1] \to [0, 1]$ mappings and then let the network learn appropriate scaling parameters. However, this approach often exacerbates the problem of converging to flat, uninformative solutions.

3. **Architectural sensitivity:**
The selection of a suitable network architecture appears to be highly dependent on the nature of the target function. For example, when training on data from a scalar function $f : \mathbb{R} \to \mathbb{R}$, a network with a single hidden layer typically suffices. In contrast, data from functions $f : \mathbb{R}^2 \to \mathbb{R}$ may require deeper networks, often with around four hidden layers, to achieve smooth and accurate outputs with

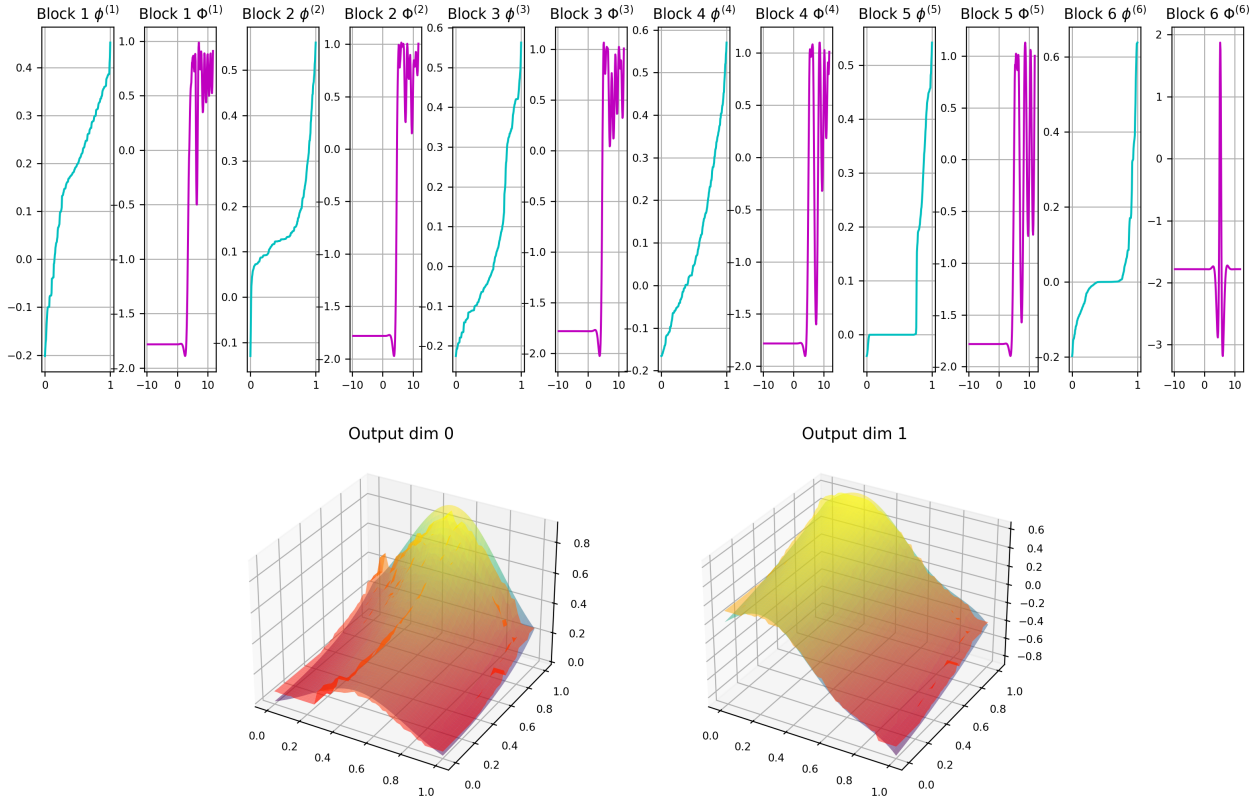

Figure 2: Visualization of a trained Sprecher Network with architecture $2 \to [20, 20, 20, 20, 20] \to 2$, implemented with six Sprecher blocks, that approximates the vector-valued function $f(x, y) = \left(\frac{\exp\left(\sin(\pi x)+y^2\right)-1}{7},\ \frac{1}{4}\,y + \frac{1}{5}\,y^2 - x^3 + \frac{1}{5}\sin(7x)\right)$. The top row shows the learned spline functions for each block (monotonic splines in cyan and general splines in magenta), while the bottom row compares the target surfaces with the network outputs for both output dimensions.

piecewise smooth splines. The situation becomes even more complex for vector-valued functions, where determining the optimal balance between network depth and width is even less straightforward and warrants further investigation. More systematic experimentation is needed to develop general hypotheses and design guidelines for choosing the appropriate network architecture.

7.2 Directions for further research

- *High-dimensional* spline adaptations (e.g., tensor-product or multi-dimensional ϕ, Φ).
- Incorporation of *residual* or *skip connections* for more powerful deep architectures.
- Theoretical comparisons of depth versus width in Sprecher-style networks.
- *Function dimensionality discovery:* Sprecher Networks offer a promising approach for discovering the intrinsic dimensionality of unknown functions. Given a dataset of output values in $\mathbb{R}^m$ (e.g., m-dimensional measurements), one could train multiple Sprecher Networks with varying input dimensions $d_{\text{in}} \to [...] \to m$ and compare their performance. The network achieving optimal approximation with minimal complexity would likely reveal the true input dimensionality of the underlying function, assuming that it exists and is sufficiently well-behaved. This approach leverages the network's interpretable spline components to visualize functional relationships, effectively "reverse-engineering" complex data manifolds. For instance, when approximating data generated by an unknown function $f : \mathbb{R}^2 \to \mathbb{R}^5$, a Sprecher Network with architecture $2 \to [...] \to 5$ should yield more regular splines and

lower error than architectures with incorrect input dimensionality. Applications include scientific data analysis, where understanding the minimal parametrization of complex phenomena is often as valuable as the predictive model itself.

8 Conclusion

We have introduced and motivated the design of *Sprecher Networks*, a modern, trainable extension of Sprecher's single-layer shift-and-sum formula for function approximation. By using two splines (ϕ and Φ) per block, along with careful index shifting and summation, we retain the theoretical grounding of the Kolmogorov–Arnold approach, thereby enabling deeper networks with potentially greater expressive power.

Further related work: Recent work in neural network design has re-examined the Kolmogorov–Arnold theorem from fresh perspectives. In particular, *Kolmogorov–Arnold Networks (KANs)* [5] propose learnable spline functions on edges instead of nodes, showing that such architectures can sometimes surpass MLPs in accuracy and interpretability for certain tasks. While KANs share the general KA motivation, the *Sprecher Network* approach here uses one monotonic and one non-monotonic spline in each layer, with a summation over shifted inputs. It would be interesting in future work to compare these designs more thoroughly and unify them under the broader KA umbrella.

References

[1] Arnold, V. I. (1963). "On functions of three variables," *Doklady Akademii Nauk SSSR*, **48**.

[2] Sprecher, D. A. (1965). "On the Structure of Continuous Functions of Several Variables," *Transactions of the American Mathematical Society*, **115**, 340–355.

[3] Kolmogorov, A. N. (1957). "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition," *Doklady Akademii Nauk SSSR*, **114**.

[4] Köppen, M. (2002). "On the training of a Kolmogorov Network," in *Artificial Neural Networks—ICANN 2002: International Conference, Madrid, Spain, August 28–30, 2002 Proceedings 12*, pp. 474–479. Springer.

[5] Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., Hou, T.Y., Tegmark, M. (2025). "KAN: Kolmogorov-Arnold Networks," *ICLR 2025 (to appear)*. arXiv preprint arXiv:2404.19756.