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, universality, and parameter efficiency compared to related architectures such as Kolmogorov-Arnold Networks (KANs).

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 $x_1, \ldots, x_d$ [3, 1].

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 inner functions $\phi_{q,p}$ 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.

Recent work has revitalized interest in leveraging Kolmogorov-Arnold representations for modern deep learning, notably with the introduction of Kolmogorov-Arnold Networks (KANs) [5], which place learnable splines on the edges of the network graph rather than on nodes.

2 Motivation and overview of Sprecher Networks

Modern deep learning practice often focuses on conventional feedforward architectures (MLPs), where fixed nonlinear activation functions are applied at nodes, and learnable linear weights operate on edges. However, Sprecher's specific construction offers a foundation for a highly interpretable alternative that may also possess advantages in parameter efficiency. Our *Sprecher Networks* are built upon the following principles:

- Each layer in our network is organized around a shared *monotonic* spline $\phi(\cdot)$ and a shared *general* spline $\Phi(\cdot)$, both learnable.
- The network incorporates trainable shifts via a scalar η, and trainable weights λ that mix each input coordinate before applying ϕ.

- Finally, the network sums over shifted partial evaluations inside the outer spline Φ, closely mimicking Sprecher's formulation.

Our architecture generalizes the classical shift-and-sum construction to a multi-layer network 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 where each edge may have its own nonlinearity, or KANs where splines are assigned per edge, the nodes within each hidden layer of a Sprecher Network are produced uniformly by the block's shared parameters (ϕ, Φ, η). Diversity arises solely from the mixing weights (λ) and the index shift (q).

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 one layer with $2n+1$ summands. Our approach uses L Sprecher blocks to create the progression

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

where $d_0 = d_{\text{in}}$ and d_L is the dimension before potential final aggregation. This provides 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 (widths), and d_{out} is the final 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 subsections describe a single block and the layer-by-layer composition.

3.1 Sprecher block structure

A Sprecher block transforms an input vector from $\mathbb{R}^{d_{\text{in}}}$ to an output vector in $\mathbb{R}^{d_{\text{out}}}$. This transformation is implemented using the following shared, learnable components:

- **Monotonic spline** $\phi(\cdot)$: an increasing piecewise-linear function defined on a prescribed interval (typically $[0, 1]$ or a slightly wider interval). This function is shared across all connections within the block and is learnable.
- **General spline** $\Phi(\cdot)$: a piecewise-linear function defined on a sufficiently large interval (e.g., $[-10, 12]$) that does not require monotonicity. This function is also shared across the block and is learnable.
- 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. These are learnable.
- A scalar **shift parameter** $\eta > 0$, which controls how the index q shifts each coordinate. This is learnable.

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 (indexed by ℓ) uses its own set of shared parameters $\phi^{(\ell)}$, $\Phi^{(\ell)}$, $\eta^{(\ell)}$, and mixing weights $\lambda^{(\ell)}$. This operation implements a shift-and-sum transformation: each input coordinate is shifted by a multiple of η, passed through the shared monotonic spline ϕ, linearly combined with the mixing weights, and then transformed by the shared general spline Φ (with an added constant term q related to the output index). 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 be 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*. 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. Denote the output of the ℓ-th block by $\mathbf{h}^{(\ell)}$ (and set $\mathbf{h}^{(0)} = \mathbf{x}$). Then the ℓ-th block computes

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

There are two cases for composing the blocks, depending on the desired final output dimension m:

(a) Scalar output ($m = 1$): The final block (block L) is designed so its d_L outputs are summed to yield a scalar:

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

If we define the operator for the ℓ-th block as

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

then the overall function is

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

In this case, the network uses L blocks and $2L$ shared spline functions.

(b) Vector-valued output ($m > 1$): When f maps into $\mathbb{R}^m$ with $m > 1$, the network first constructs L hidden layers yielding a representation $\mathbf{h}^{(L)} \in \mathbb{R}^{d_L}$. An additional output block (block $L + 1$) is appended to map $\mathbf{h}^{(L)}$ to $\mathbb{R}^m$ without summation. This output block computes:

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

with $z = \mathbf{h}^{(L)}$. The network output function is then

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

In this configuration, the network uses $L+1$ blocks and $2(L+1)$ shared spline functions. The extra block transforms the final hidden representation into an m-dimensional output without collapsing the index q.

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 $\mathbb{R}^m$:

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

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 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.

Illustrative examples (scalar output)

To clarify the compositional structure, we write out the full expansions for networks with $L = 1, 2, 3$ hidden layers producing a scalar output.

3.2.1 Single hidden layer ($L = 1$)

For a network with architecture $d_{\text{in}} \to [d_1] \to 1$ (i.e., $d_0 = d_{\text{in}}$), 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 when $d_1 = 2d_0 + 1$ and identifying $\phi^{(1)} = \phi,\ \Phi^{(1)} = \Phi$.

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. The intermediate output is

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

The second block computes

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

The final network output is $f(\mathbf{x}) = \sum_{q=0}^{d_2-1} \mathbf{h}_q^{(2)}$. Equivalently, the fully expanded form is:

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

3.2.3 Three hidden layers ($L = 3$)

Let the layer dimensions be $d_0 \to d_1 \to d_2 \to d_3$, with the final block summed. The recursive definition involves:

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

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

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

The network output is $f(\mathbf{x}) = \sum_{q=0}^{d_3-1} \mathbf{h}_q^{(3)}$. The equivalent nested formulation is:

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

Remark 1. *These expansions show how each layer's output is obtained by a shift-and-sum transformation using a shared pair of splines $\phi^{(\ell)}$ and $\Phi^{(\ell)}$. In the scalar-output case, the final layer sums over its output index q. For vector outputs ($m > 1$), an extra block is used without this final summation.*

Remark 2. *It is tempting to try to merge the spline pairs in* (1) *and* (2) *into single composite functions. For example, one might consider replacing the pair*

$$\phi^{(2)}\Big(\Phi^{(1)}(\cdot) + \eta^{(2)} q\Big)$$

in (1) *(or, in the three-layer case, the pairs*

$$\phi^{(2)}\Big(\Phi^{(1)}(\cdot) + \eta^{(2)} s\Big) \quad \textit{and} \quad \phi^{(3)}\Big(\Phi^{(2)}(\cdot) + \eta^{(3)} q\Big)$$

from (2)*) by removing the explicit shift terms (the $\eta^{(\ell)} q$ or $\eta^{(\ell)} s$ terms inside $\phi^{(\ell)}$). However, experiments indicate that such modifications break the network's functionality. The precise reason why these internal shift terms are essential (in addition to the outer $+q$ or $+s$ shifts) remains unclear and is an interesting subject for further investigation.*

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 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 this formula exactly.

Going deeper. While Sprecher's original construction uses a single layer, our approach composes multiple shift-and-sum blocks (each constituting a Sprecher block) to form a deep network. This stacking increases expressivity while preserving the theoretical foundations of the original construction.

5 Implementation highlights

5.1 Trainable splines

Each block employs two learnable spline modules:

- $\phi^{(\ell)}$: a monotonic piecewise-linear spline defined on $[0, 1]$ (or a slightly wider interval), initialized to be increasing.
- $\Phi^{(\ell)}$: a general piecewise-linear spline defined on an interval such as $[-10, 12]$, which may be initialized near the identity. It does not require monotonicity.

In our reference implementation, each spline is represented using 100–300 knots (though fewer may suffice). A flatness penalty (based on the second derivative of the spline coefficients) can be applied to enforce smoothness and act as a regularizer, inspired by [4].

5.2 Shifts and weights

Each block includes:

- A trainable scalar shift parameter $\eta^{(\ell)}$, which controls the amount by which the input is shifted for each index q.
- A matrix of trainable mixing weights $\lambda_{i,q}^{(\ell)}$ (with dimensions $d_{\ell-1} \times d_\ell$) that linearly combine the transformed inputs $\phi^{(\ell)}(\cdot)$.

These parameters, along with the spline coefficients, are learned via gradient-based optimization, typically minimizing a loss function that includes a Mean Squared Error (MSE) term and optionally the spline flatness penalty.

5.3 Parameter counting and comparison with KANs

Throughout this section, we assume cubic splines (order $k = 3$) and an effective grid size of G intervals, so that each spline is represented by approximately $G + k$ parameters (often approximated as G for large G).

The parameterization of Sprecher Networks (SNs) is fundamentally different from Kolmogorov-Arnold Networks (KANs) [5]. In KANs, each edge (with roughly $N_{in} \times N_{out}$ edges per layer, where N_{in}, N_{out} are layer widths) is assigned its own spline function. If each spline has approximately $G + k$ parameters, the parameter count per layer in a KAN is

$$O\big(N_{in} \times N_{out} \times (G + k)\big).$$

Over L layers with average width N, this yields roughly $O(L \cdot N^2 \cdot (G + k))$, which we approximate as $O(L \cdot N^2 \cdot G)$ for large G.

In contrast, an SN block uses only *two shared splines* (one for $\phi^{(\ell)}$ and one for $\Phi^{(\ell)}$) independent of layer widths. The parameters per block include:

- Mixing weights scaling as $d_{\ell-1} \times d_\ell$ (approximately $O(N^2)$ for average width N).
- Shared spline parameters totaling $K_\phi + K_\Phi$, where K_ϕ, K_Φ are the number of parameters for ϕ, Φ respectively. Assuming $K_\phi \approx K_\Phi \approx G + k$, these contribute an overhead of $O(G + k)$ per block.
- One scalar shift parameter $\eta^{(\ell)}$ per block.

Thus, for a network of depth L (meaning L blocks for scalar output, $L + 1$ for vector) and average width N, the total parameter count scales as

$$O(L \cdot N^2 + L \cdot (K_\phi + K_\Phi)).$$

Assuming $K_\phi + K_\Phi$ is of the order of $G + k$, the dominant term in KANs scales with $N^2 \cdot G$, while in SNs the spline parameter count G contributes only an *additive* overhead $L \cdot G$ to the $L \cdot N^2$ weights. In other words, while KAN parameters scale roughly as $O(N^2 \cdot G)$, SN parameters scale roughly as $O(N^2 + G)$ per block. This reduction in dependence on G becomes especially significant when the required spline resolution G is large relative to the layer widths N.

5.4 Sprecher-like theorems

Restricting to a single layer ($L = 1$) with $d_{\text{out}} = 2d + 1$, a monotonic ϕ, and appropriate constants (λ_i, η) reproduces the original Sprecher construction:

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

Thus, Sprecher's argument implies that this network can approximate any continuous function $f : [0,1]^d \to \mathbb{R}$ arbitrarily well if ϕ and Φ are sufficiently flexible.

For vector-valued functions $f : [0,1]^d \to \mathbb{R}^m$ with $m > 1$, our construction uses L hidden blocks and appends an output block (block $L+1$, without final summation) to generate an m-dimensional output. The universality of this specific construction for vector-valued functions is conjectured below.

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 $\phi^{(2)} : J_2 \to \mathbb{R}$, general splines $\Phi^{(1)} : I_1 \to \mathbb{R}$ and $\Phi^{(2)} : J_2 \to \mathbb{R}$ (for suitable intervals I_1 and J_2), 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*

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

This formulation uses $L = 1$ hidden block producing $d_1 = 2n + 1$ nodes, and one output block ($L + 1 = 2$).

Remark 3. *For $m = 1$, this representation reduces (after summing over q) to Sprecher's original formulation.*

5.5 Deeper extensions

A natural conjecture is that stacking multiple Sprecher blocks ($L > 1$) preserves universal approximation capabilities while possibly requiring fewer nodes per layer or offering other benefits like capturing complex compositional structures more efficiently. A complete theoretical treatment of depth versus width trade-offs for Sprecher Networks remains an avenue for future work.

6 Empirical demonstrations

We train Sprecher Networks on small datasets sampled from a target function f. The network learns the parameters $(\eta^{(\ell)}, \lambda^{(\ell)})$ and spline coefficients $(\phi^{(\ell)}, \Phi^{(\ell)})$ via gradient descent. Two classic examples are:

- **1D case:** A polynomial or sinusoidal function on $[0,1]$ where the network learns ϕ and Φ that accurately interpolate the function, effectively acting as a learnable spline interpolant.
- **2D case:** A surface defined 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 using Adam), the network achieves high accuracy. Layerwise spline plots provide an interpretable view of the internal transformation dynamics.

Figures 1 and 2 illustrate typical network outputs for 2D scalar and vector-valued functions, respectively.

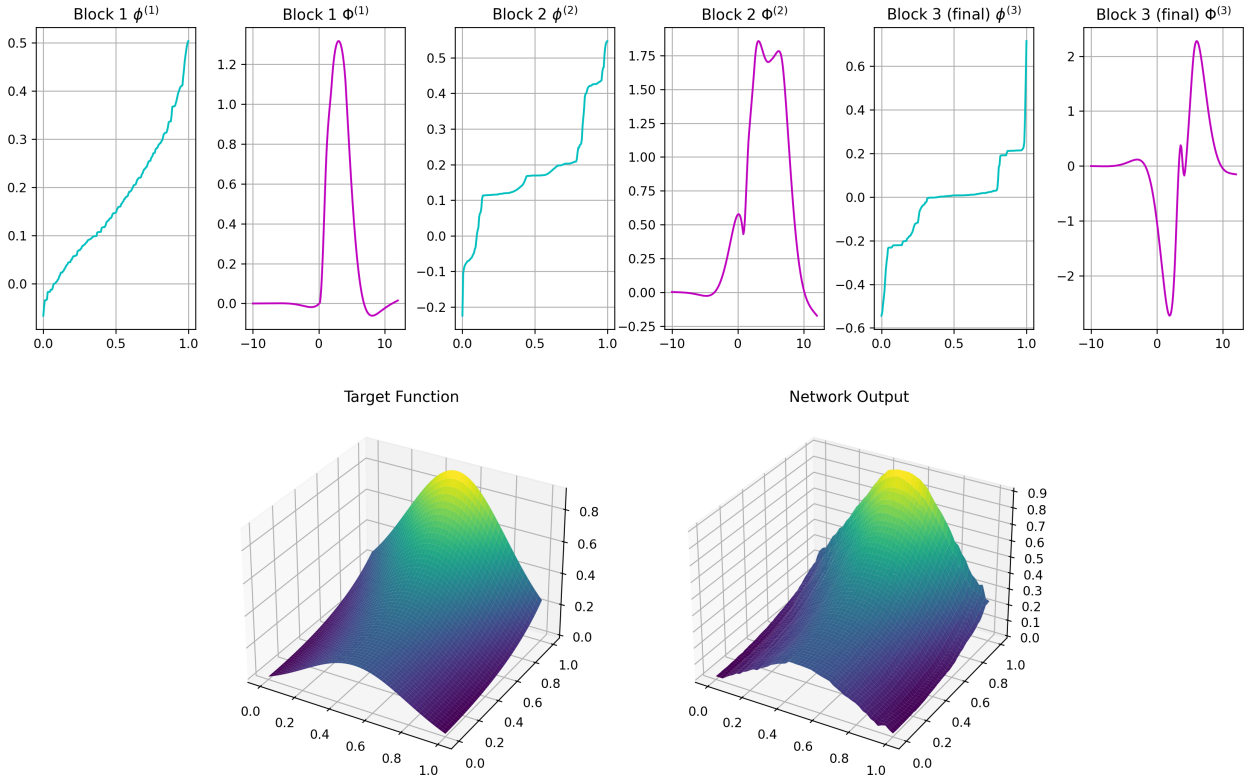

Figure 1: Visualization of a trained $2 \to [5, 8, 5] \to 1$ Sprecher Network approximating a 2D target function $z = f(x, y) = \frac{\exp\left(\sin(\pi x)+y^2\right)-1}{7}$. Top row: Learned spline functions for each block — monotonic splines $\phi^{(\ell)}$ (cyan) and general splines $\Phi^{(\ell)}$ (magenta). Bottom row: Comparison between the target function (left) and the network approximation (right).

6.1 Potential advantages in parameter efficiency: placeholder examples

Preliminary estimates suggest that Sprecher Networks (SNs) could offer significant parameter reductions compared to KANs, based on examples from [5]. These require empirical validation via direct SN training on the same tasks.

PDE solving example (Ref KAN §3.4): A KAN architecture of $[2, 10, 1]$ was reported effective for a Poisson equation task.

- **KAN parameter estimate:** The KAN has $2 \times 10 + 10 \times 1 = 30$ edges/splines. Assuming $G = 20$ grid intervals and cubic splines ($k = 3$, giving approximately $G + k = 23$ parameters per spline), the total KAN parameter count is about $30 \times 23 = 690$ parameters. (Note: The original KAN paper might use different spline implementations or parameter counting conventions).
- **SN parameter estimate (equivalent structure):** An SN architecture $[2, 10, 1]$ corresponds to $d_0 = 2, d_1 = 10, d_2 = 1$ (output dimension). This uses $L = 2$ blocks (scalar output), hence 4 shared splines. With the same spline assumptions ($G = 20, k = 3$), shared spline parameters total roughly $4 \times 23 = 92$. In addition, there are $d_0 \times d_1 + d_1 \times d_2 = 2 \times 10 + 10 \times 1 = 30$ mixing weights and $L = 2$ shift parameters $\eta^{(1)}, \eta^{(2)}$. The approximate total SN parameter count is $92+30+2 = 124$ parameters.
- **Potential advantage:** This represents a potential reduction by roughly a factor of 5.5 (124 versus 690). (*Placeholder: Empirical validation required to confirm if an SN of this structure performs comparably.*).

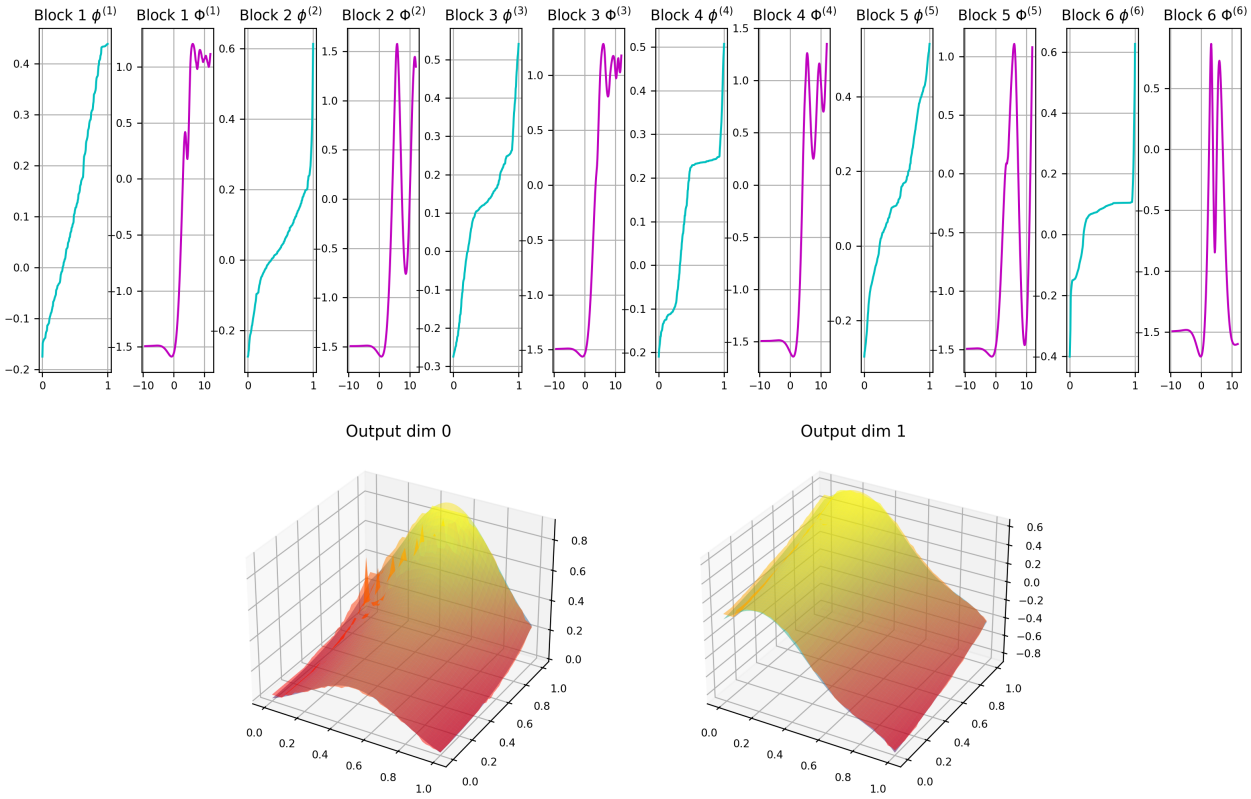

Figure 2: Visualization of a trained Sprecher Network with architecture $2 \to [20, 20, 20, 20, 20] \to 2$, implemented with six Sprecher blocks, approximating a vector-valued function $f(x, y) = \left(\frac{\exp\left(\sin(\pi x)+y^2\right)-1}{7},\ \frac{1}{4}\,y + \frac{1}{5}\,y^2 - x^3 + \frac{1}{5}\sin(7x)\right)$.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

Knot theory example (Ref KAN §4.3): A pruned KAN architecture $[17, 1, 14]$ with $G = 3, k = 3$ was used to predict knot signature.

- **KAN parameters:** The KAN has $17\times1+1\times14 = 31$ edges/splines. Parameters per spline $\approx G+k = 3+3 = 6$. Total KAN params $\approx 31 \times 6 = 186$.
- **SN parameter estimate (equivalent structure):** An SN architecture $[17, 1, 14]$ (vector output $m = 14$) implies $L = 1$ hidden layer ($d_0 = 17, d_1 = 1$) and an output layer mapping $d_1 \to d_2 = 14$. This uses $L + 1 = 2$ blocks total. Number of shared splines is $2(L + 1) = 4$. Spline parameters $\approx 4 \times (G+k) = 4 \times 6 = 24$. Mixing weights: $d_0 \times d_1 + d_1 \times d_2 = 17 \times 1 + 1 \times 14 = 31$. Shift parameters: $L + 1 = 2$. Total SN params $\approx 24 + 31 + 2 = 57$ parameters.
- **Potential Advantage:** This yields a potential reduction of approximately 3.3x (57 versus 186). (*Placeholder: Empirical validation required. Achieving this structure may need SN pruning development, as the intermediate dimension $d_1 = 1$ is very narrow.*).

These calculations illustrate the *potential* for parameter savings in SNs due to the use of shared splines, especially when the effective grid size G is large relative to the mixing weights N^2.

7 Challenges and future directions

While Sprecher Networks offer a theoretically grounded and potentially efficient architecture, several challenges and opportunities for future work exist.

7.1 Implementation challenges

1. **Difficulties in training spline functions:**
   Finding good spline shapes through gradient-based optimization can be challenging. Due to the inherent flexibility of the trainable splines, the network can sometimes settle into local minima where the learned functions flatten out, resulting in "horizontal" outputs (outputs that remain nearly constant regardless of the input). This stagnation significantly impedes the network's ability to accurately approximate the target function. Escaping such flat regions in the loss landscape may require better spline initialization strategies, adaptive learning rates, or second-order optimization methods. Training can be slower compared to simpler models like MLPs.

2. **Initialization and domain/codomain Selection:**
   The original Sprecher construction suggests specific domains/codomains (e.g., $\phi : [0, 1] \to [0, 1]$). In implementations, restricting the domains and codomains of the shared splines to bounded intervals (e.g., $[-10, 10]$ or $[-10, 12]$ for Φ) is a practical necessity. However, determining these intervals optimally a priori is difficult. While trainable range parameters (as used in the provided Python code) offer a solution, they can sometimes exacerbate convergence issues if not carefully regularized or initialized.

3. **Architectural sensitivity:**
   The selection of a suitable network architecture (depth L and widths $d_1, \ldots, d_L$) appears to be highly dependent on the nature of the target function. Simple functions might only require shallow networks, whereas more complex or higher-dimensional functions may need deeper architectures. The optimal balance between network depth and width is not well understood and warrants further investigation. Systematic guidelines for these choices are needed. Although our parameter counts are compared for equivalent structures, the optimal architecture may differ between SNs and KANs.

4. **Knot count selection:**
   The optimal number of knots (K_ϕ, K_Φ) for the shared splines is critical. Too few knots limit expressivity, while too many increase parameters and may lead to overfitting or optimization difficulties. Unlike KANs where spline complexity can vary per edge, the shared nature of SN splines affects the entire layer's nonlinearity uniformly. Adaptive knot placement strategies or methods to automatically determine knot density based on data would be highly beneficial.

7.2 Directions for further research

- **Improved training strategies:** Develop better initialization schemes (e.g., initializing Φ closer to identity, ϕ closer to linear), adaptive optimization methods (e.g., second-order techniques), or alternative regularization approaches tailored for training shared splines in the Sprecher framework.

- **Sparsity and pruning techniques:** Implement L1/entropy-based regularization on mixing weights (λ) and pruning strategies (e.g., removing nodes with low contribution) specifically for SNs to automatically discover compact architectures and further improve parameter efficiency and interpretability.

- **High-dimensional spline adaptations:** Explore if alternative representations for the shared functions ϕ, Φ, such as tensor-product splines or small neural networks, could be beneficial, particularly for higher-dimensional inputs.

- **Residual or skip connections:** Investigate incorporating residual connections around Sprecher blocks to improve gradient flow, potentially enabling deeper and more stable networks, analogous to ResNets.

- **Theoretical analysis:** Further study the approximation capabilities of deep SNs, rigorously analyze the trade-offs between depth and width, and compare their sample complexity and generalization properties with KANs and traditional MLPs. Understanding the role of the internal shifts ($\eta^{(\ell)}q$) remains crucial.

- **Function dimensionality discovery:** Leverage the interpretability of SNs for scientific discovery. By training networks with varying input dimensions $d_{\text{in}} \to [...] \to m$ on data with unknown structure, the architecture achieving the best trade-off between fit quality and complexity could reveal the intrinsic dimensionality of the underlying function. This uses the network structure itself as an investigative tool. 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 *Sprecher Networks*, a modern, trainable extension of Sprecher's single-layer shift-and-sum formulation for function approximation. By utilizing two shared spline functions (ϕ and Φ) per block, along with learnable mixing weights and shifts, SNs retain the theoretical underpinnings of the Kolmogorov–Arnold–Sprecher construction while offering enhanced interpretability and significant potential for parameter efficiency compared to Kolmogorov-Arnold Networks.

Our parameter analysis indicates that, under reasonable assumptions, SNs can significantly reduce the dependence of the total parameter count on the spline grid size G compared to KANs for equivalent network structures, potentially offering substantial savings (e.g., by factors of 3 to 5 in the examples analyzed). Although challenges remain in training stability, speed, hyperparameter selection (e.g., optimal knot counts and domain choices), and rigorous empirical validation across diverse tasks, further work—particularly in improved optimization, regularization, and pruning strategies—could unlock the full potential of Sprecher Networks.

SNs represent a novel direction that bridges classical approximation theory with modern deep learning practices. They offer a unique blend of interpretability derived from their explicit structure and potential parameter efficiency stemming from shared nonlinearities, making them a promising avenue for future research in function approximation and scientific machine learning.

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.