Sprecher Networks: A Parameter-Efficient Architecture Inspired by the Kolmogorov–Arnold–Sprecher Theorem

Christian Hägg* Kathlén Kohn† Giovanni Luca Marchetti‡ Boris Shapiro§

June 12, 2025

Abstract

We present *Sprecher Networks* (SNs), a family of trainable neural architectures inspired by the classical Kolmogorov–Arnold–Sprecher (KAS) construction for approximating multivariate continuous functions. Distinct from Multi-Layer Perceptrons (MLPs) with fixed node activations and Kolmogorov-Arnold Networks (KANs) featuring learnable edge activations, SNs utilize shared, learnable splines (*monotonic* and *general*) within structured blocks incorporating explicit learnable shifts and mixing weights. Our approach directly realizes Sprecher's specific 1965 "sum of shifted splines" formula in its single-layer variant and extends it to deeper, multi-layer compositions. We demonstrate empirically that composing these blocks into deep networks leads to highly parameter-efficient models, discuss theoretical motivations, and compare SNs with related architectures (MLPs, KANs, and networks with learnable node activations).

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. The Kolmogorov–Arnold Representation Theorem [7, 1] established that any multivariate continuous function $f : [0,1]^d \to \mathbb{R}$ can be represented as a finite composition of continuous functions of a single variable and the addition operation. Specifically, Kolmogorov (1957) showed that such functions can be represented as a finite sum involving univariate functions applied to sums of other univariate functions of the inputs.

David Sprecher's 1965 construction. In his 1965 landmark paper [10], David Sprecher provided a constructive proof and a specific formula realizing the Kolmogorov–Arnold representation. He showed that any continuous function $f : [0,1]^n \to \mathbb{R}$ could be represented as:

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

for a single *monotonic* inner function ϕ, a continuous outer function Φ, a constant shift parameter $\eta > 0$, and constants λ_p. This construction simplified the representation by using only one inner function ϕ, relying on shifts of the input coordinates ($x_p + \eta q$) and an outer summation index shift ($+q$) to achieve universality. The key insight of *shifting input coordinates* and summing evaluations under inner and outer univariate maps is central to Sprecher's specific result.

From theorem to architectural building block. While Sprecher's theorem provides a blueprint for a powerful shallow network, modern deep learning has repeatedly demonstrated the benefits of compositional depth. This motivates our central research question: can the core components of Sprecher's formula be used as building blocks in a deep, compositional architecture? We propose to do so by composing what we term *Sprecher blocks*, where the vector output of one block becomes the input to the next. It is crucial to note

*Department of Mathematics, Stockholm University, Stockholm, Sweden. Email: hagg@math.su.se
†Department of Mathematics, KTH Royal Institute of Technology, Stockholm, Sweden. Email: kathlen@kth.se
‡Department of Mathematics, KTH Royal Institute of Technology, Stockholm, Sweden. Email: glma@kth.se
§Department of Mathematics, Stockholm University, Stockholm, Sweden. Email: shapiro@math.su.se

that this compositional structure is our own architectural proposal and is not part of Sprecher's original construction or proof of universality. This paper's central goal is to empirically evaluate whether this highly structured, theorem-inspired design is a viable and efficient alternative to existing deep learning models.

Modern context. Recent work has revitalized interest in leveraging Kolmogorov-Arnold representations for modern deep learning. Notably, Kolmogorov-Arnold Networks (KANs) [9] were introduced, proposing an architecture with learnable activation functions (splines) placed on the *edges* of the network graph, replacing traditional linear weights and fixed node activations.

Architectural landscape. Understanding how novel architectures relate to established ones is crucial. Standard Multi-Layer Perceptrons (MLPs) [4] employ fixed nonlinear activation functions at nodes and learnable linear weights on edges, justified by the Universal Approximation Theorem [2, 5]. Extensions include networks with *learnable activations on nodes*, sometimes called Adaptive-MLPs or Learnable Activation Networks (LANs) [3, 11, 9] (Appendix B), which retain linear edge weights but make the node non-linearity trainable. KANs [9] represent a more significant departure, moving learnable splines to edges and eliminating linear weights entirely, using simple summation at nodes. Sprecher Networks (SNs), as we detail below, propose a distinct approach derived directly from Sprecher's 1965 formula. SNs employ function blocks containing shared learnable splines (ϕ, Φ), learnable mixing weights (λ), and explicit structural shifts (η, q). This structure offers a different alternative within the landscape of function approximation networks.

2 Motivation and overview of Sprecher Networks

While MLPs are the workhorse of deep learning, architectures inspired by KAS representations offer potential benefits, particularly in interpretability and potentially parameter efficiency for certain function classes. KANs explore one direction by placing learnable functions on edges. Our *Sprecher Networks* (SNs) explore a different direction, aiming to directly implement Sprecher's constructive formula within a trainable framework and extend it to deeper architectures.

SNs are built upon the following principles, directly reflecting Sprecher's formula:

- Each functional block (mapping between layers) is organized around a shared *monotonic* spline $\phi(\cdot)$ and a shared *general* spline $\Phi(\cdot)$, both learnable.
- Each block incorporates a learnable scalar shift η applied to inputs based on the output index q.
- Each block includes learnable mixing weights λ_i (a vector, not a matrix) that combine contributions from different input dimensions, with the same weights shared across all output dimensions.
- The structure explicitly includes the additive shift q inside the outer spline Φ, mimicking Sprecher's formulation.

Our architecture generalizes this classical single-layer shift-and-sum construction to a multi-layer network by composing these functional units, which we term *Sprecher blocks*. The mapping from one hidden layer representation to the next is realized by such a block. Unlike MLPs with fixed node activations, LANs with learnable node activations, or KANs with learnable edge activations, SNs concentrate their learnable non-linearity into the two shared splines per block, applied in a specific structure involving shifts and learnable linear weights. This imposes a strong inductive bias, trading the flexibility of independent weights/splines for extreme parameter sharing. Diversity in the transformation arises from the mixing weights (λ) and the index-dependent shifts (q).

Concretely, each Sprecher block applies the transformation:

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

Note that λ_i depends only on the input index i, not on the output index q, maintaining fidelity to Sprecher's original 1965 construction. For scalar outputs, the outputs of the final Sprecher block are aggregated (via summation over q). For vector outputs, an additional block is used without final summation.

In Sprecher's original work, one layer (block) with $d_{\text{out}} = 2n + 1$ outputs (where $n = d_{\text{in}}$) was sufficient for universality. Our approach stacks L Sprecher blocks to create a deep network progression:

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

where $d_0 = d_{\text{in}}$ is the input dimension, and d_L is the dimension of the final hidden representation before potential aggregation or final mapping. This multi-block composition provides a deeper analog of the KAS construction, aiming for potentially enhanced expressive power or efficiency for complex compositional functions. The universality of networks with $L > 1$ blocks or vector-valued outputs is an open question we explore empirically (see Section 5).

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 of the network. For scalar output ($d_{\text{out}} = 1$), the final block's outputs are summed. For vector output ($d_{\text{out}} > 1$), an additional non-summed block maps from d_L to d_{out}. 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 (implying the final block's outputs of dimension 8 are summed). $2 \to [5, 3] \to 4$ describes a network with 2-dimensional input, two hidden layers of widths 5 and 3, and a 4-dimensional vector output (implying an additional output block maps from dimension 3 to 4 without summation). 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 performing a shift-and-sum transformation inspired by Sprecher's original construction.

3.1 Sprecher block structure

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

- **Monotonic spline $\phi(\cdot)$:** An increasing piecewise-linear function, typically defined on a fixed interval like $[0, 1]$. This function is shared across all input-output connections within the block and its coefficients are learnable. Strict monotonicity (strictly increasing) is enforced during training (see Section 6).
- **General spline $\Phi(\cdot)$:** A piecewise-linear function (without monotonicity constraints) defined on a potentially wider interval. The domain and codomain can be either fixed (e.g., $[-10, 10]$) or made trainable. If trainable ranges are used, they can be parameterized in various ways, such as by center and radius parameters for both domain and codomain, allowing the spline to adapt its input/output ranges during training. This function is also shared across the block and its coefficients are learnable.
- **Mixing weights vector λ:** A vector $\{\lambda_i\}$ of size d_{in}, whose entries are learnable. These weights linearly combine the contributions from different input dimensions after transformation by ϕ. Crucially, these weights are shared across all output dimensions within the block, maintaining fidelity to Sprecher's original formulation.
- **Shift parameter η:** A learnable scalar $\eta > 0$. This parameter controls the magnitude of the input shift $x_i + \eta q$, which depends on the output index q.

Concretely, given an input vector $\mathbf{x} = (x_1, \ldots, x_{d_{\text{in}}}) \in \mathbb{R}^{d_{\text{in}}}$, a single Sprecher block (indexed implicitly by ℓ, with parameters $\phi^{(\ell)}, \Phi^{(\ell)}, \eta^{(\ell)}, \lambda^{(\ell)}$) computes the q-th component of its output vector $\mathbf{h} \in \mathbb{R}^{d_{\text{out}}}$ (where $q = 0, \ldots, d_{\text{out}} - 1$) via:

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

where α is a scaling factor (set to 1 for the original Sprecher construction). While $\alpha = 1$ maintains theoretical consistency with Sprecher's formula, alternative values may be explored to improve optimization dynamics in deeper networks. Note that q serves dual roles here: as an output index ($q = 0, \ldots, d_{\text{out}} - 1$) and as an additive shift parameter within the formula.

In a network with multiple layers, each Sprecher block (indexed by $\ell = 1, \ldots, L$ or $L + 1$) uses its own independent set of shared parameters $(\phi^{(\ell)}, \Phi^{(\ell)}, \eta^{(\ell)}, \lambda^{(\ell)})$. The block operation implements a specific form of transformation: each input coordinate x_i is first shifted by an amount depending on the output index q and the shared shift parameter $\eta^{(\ell)}$, then passed through the shared monotonic spline $\phi^{(\ell)}$. The results are linearly combined using the learnable mixing weights $\lambda_i^{(\ell)}$, shifted again by the output index q, and finally passed through the shared general spline $\Phi^{(\ell)}$. Stacking these blocks creates a deep, compositional representation.

3.2 Optional enhancements

Several optional components can enhance the basic Sprecher block:

- **Residual connections:** When enabled, learnable parameters facilitate skip connections around each block. If input and output dimensions match ($d_{\text{in}} = d_{\text{out}}$), a scalar weight w_{res} is used. When dimensions differ, a learnable linear projection matrix $W_{\text{res}} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ (without bias) maps the input to the output dimension. The block output with residual connection becomes:

$$h_q^{\text{final}} = h_q + \begin{cases} w_{\text{res}} \cdot x_q & \text{if } d_{\text{in}} = d_{\text{out}} \\ (W_{\text{res}}\mathbf{x})_q & \text{if } d_{\text{in}} \neq d_{\text{out}} \end{cases}$$

- **Output scaling:** The network may include learnable output scaling parameters γ (scale) and β (bias) applied to the final network output. These can be particularly useful for optimization, with appropriate initialization strategies (e.g., initializing β to match target statistics and γ to control initial output magnitudes).

3.3 Layer composition and final mapping

Let L be the number of hidden layers specified by the architecture $[d_1, \ldots, d_L]$. In our framework, a "hidden layer" corresponds to the vector output of a Sprecher block. The mapping from the representation at layer $\ell - 1$ to layer ℓ is implemented by the ℓ-th Sprecher block.

Let the input to the network be $\mathbf{h}^{(0)} = \mathbf{x} \in \mathbb{R}^{d_0}$ (where $d_0 = d_{\text{in}}$). The output of the ℓ-th Sprecher block ($\ell = 1, \ldots, L$) is the vector $\mathbf{h}^{(\ell)} \in \mathbb{R}^{d_\ell}$, computed component-wise as:

$$\mathbf{h}_q^{(\ell)} = \Phi^{(\ell)}\left(\sum_{i=1}^{d_{\ell-1}} \lambda_i^{(\ell)}\, \phi^{(\ell)}\left(\mathbf{h}_i^{(\ell-1)} + \eta^{(\ell)}\, q\right) + q\right), \quad q = 0, \ldots, d_\ell - 1. \tag{2}$$

Remark 1 (On the nature of composition)**.** *Note that in this $L > 1$ composition (Eq. 2), the argument to the inner spline $\phi^{(\ell)}$ is $\mathbf{h}_i^{(\ell-1)}$, the output of the previous layer, not the original input coordinate x_i. This is the fundamental departure from Sprecher's construction and is the defining feature of our proposed deep architecture. Each layer processes the complex, transformed output of the layer before it, enabling the network to learn hierarchical representations.*

The composition of these blocks and the final output generation depend on the desired final output dimension $m = d_{\text{out}}$:

(a) Scalar output ($m = 1$): The network consists of exactly L Sprecher blocks. The output of the final block, $\mathbf{h}^{(L)} \in \mathbb{R}^{d_L}$, is aggregated by summation to yield the scalar output:

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

If we define the operator for the ℓ-th block as $T^{(\ell)} : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}$, where

$$\Big(T^{(\ell)}(z)\Big)_q = \Phi^{(\ell)}\left(\sum_{i=1}^{d_{\ell-1}} \lambda_i^{(\ell)}\, \phi^{(\ell)}\Big(z_i + \eta^{(\ell)}\, q\Big) + q\right),$$

then the overall function is

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

This network uses L blocks and $2L$ shared spline functions in total (one pair $(\phi^{(\ell)}, \Phi^{(\ell)})$ per block).

(b) Vector-valued output ($m > 1$): When the target function f maps to $\mathbb{R}^m$ with $m > 1$, the network first constructs the L hidden layers as above, yielding a final hidden representation $\mathbf{h}^{(L)} \in \mathbb{R}^{d_L}$. An *additional* output block (block $L+1$) is then appended to map this representation $\mathbf{h}^{(L)}$ to the final output space $\mathbb{R}^m$. This $(L+1)$-th block operates *without* a final summation over its output index. It computes the final output vector $\mathbf{y} \in \mathbb{R}^m$ as:

$$y_q = \Big(T^{(L+1)}(\mathbf{h}^{(L)})\Big)_q = \Phi^{(L+1)}\left(\sum_{r=0}^{d_L-1} \lambda_r^{(L+1)}\, \phi^{(L+1)}\Big(\mathbf{h}_r^{(L)} + \eta^{(L+1)}\, q\Big) + q\right),$$

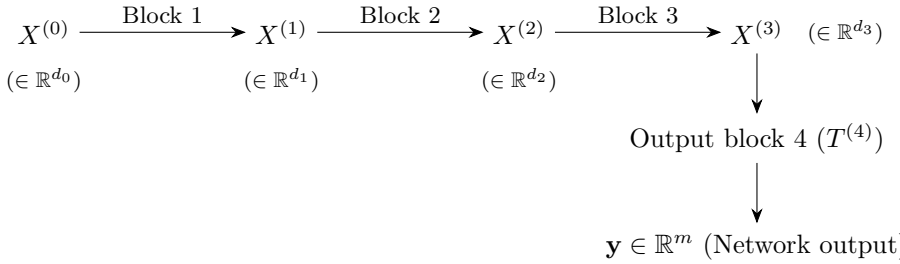
for $q = 0, \ldots, m-1$. The network output function is then:

$$f(\mathbf{x}) = \mathbf{y} = \Big(T^{(L+1)} \circ T^{(L)} \circ \cdots \circ T^{(1)}\Big)(\mathbf{x}) \in \mathbb{R}^m. \tag{3}$$

In this configuration, the network uses $L+1$ blocks and involves $2(L+1)$ shared spline functions. The extra block serves as a trainable output mapping layer, transforming the final hidden representation $\mathbf{h}^{(L)}$ into the desired m-dimensional output vector.

In summary: for an architecture with L hidden layers, a scalar-output SN uses L blocks and $2L$ shared splines. A vector-output SN (with $m > 1$) uses $L+1$ blocks and $2(L+1)$ shared splines. When output scaling is used, this adds 2 parameters for scalar output or $2m$ parameters for vector output (when applied per dimension). This structure provides a natural extension of Sprecher's original scalar formula to the vector-valued setting.

We illustrate the vector-output case ($m > 1$) for a network architecture $d_0 \to [d_1, d_2, d_3] \to m$ (i.e., $L = 3$ hidden layers). Let $X^{(0)}$ be the input $\mathbf{x}$.



Here, $X^{(\ell)} = \mathbf{h}^{(\ell)}$ denotes the output vector of the ℓ-th Sprecher block. Each block $T^{(\ell)}$ internally uses its own pair of shared splines $(\phi^{(\ell)}, \Phi^{(\ell)})$, mixing weights $\lambda^{(\ell)}$, and shift $\eta^{(\ell)}$. The final output block $T^{(4)}$ maps the representation $X^{(3)}$ to the final m-dimensional output $\mathbf{y}$ without subsequent summation.

3.4 Illustrative expansions (scalar output)

To further clarify the compositional structure for the scalar output case ($m = 1$), we write out the full expansions for networks with $L = 1, 2, 3$ hidden layers.

3.4.1 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:

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

This precisely reproduces Sprecher's 1965 construction if we choose $d_1 = 2d_0 + 1$ and identify $\phi^{(1)} = \phi$, $\Phi^{(1)} = \Phi$, and $\lambda_i^{(1)} = \lambda_i$.

3.4.2 Two hidden layers ($L = 2$)

Let the architecture be $d_0 \to [d_1, d_2] \to 1$. The intermediate output $\mathbf{h}^{(1)} \in \mathbb{R}^{d_1}$ is computed as:

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

The second block computes $\mathbf{h}^{(2)} \in \mathbb{R}^{d_2}$ using $\mathbf{h}^{(1)}$ as input:

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

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

$$f(\mathbf{x}) = \sum_{q=0}^{d_2-1} \Phi^{(2)}\left(\sum_{r=0}^{d_1-1} \lambda_r^{(2)}\, \phi^{(2)}\left(\Phi^{(1)}\left(\sum_{i=1}^{d_0} \lambda_i^{(1)}\, \phi^{(1)}\Big(x_i + \eta^{(1)}\, r\Big) + r\right) + \eta^{(2)}\, q\right) + q\right). \tag{4}$$

3.4.3 Three hidden layers ($L = 3$)

Let the architecture be $d_0 \to [d_1, d_2, d_3] \to 1$. The recursive definition involves:

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

$$\mathbf{h}_s^{(2)} = \Phi^{(2)}\Big(\sum_{r=0}^{d_1-1} \lambda_r^{(2)}\, \phi^{(2)}\Big(\mathbf{h}_r^{(1)} + \eta^{(2)}\, s\Big) + s\Big), \quad s = 0, \ldots, d_2 - 1,$$

$$\mathbf{h}_q^{(3)} = \Phi^{(3)}\Big(\sum_{s=0}^{d_2-1} \lambda_s^{(3)}\, \phi^{(3)}\Big(\mathbf{h}_s^{(2)} + \eta^{(3)}\, q\Big) + q\Big), \quad q = 0, \ldots, d_3 - 1.$$

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

$$f(\mathbf{x}) = \sum_{q=0}^{d_3-1} \Phi^{(3)}\left(\sum_{s=0}^{d_2-1} \lambda_s^{(3)}\, \phi^{(3)}\left(\Phi^{(2)}\left(\sum_{r=0}^{d_1-1} \lambda_r^{(2)}\, \phi^{(2)}\left(\Phi^{(1)}\left(\sum_{i=1}^{d_0} \lambda_i^{(1)}\, \phi^{(1)}\Big(x_i + \eta^{(1)}\, r\Big) + r\right) + \eta^{(2)}\, s\right) + s\right) + \eta^{(3)}\, q\right) + q\right). \tag{5}$$

Remark 2. *These expansions highlight the compositional nature where the output of one Sprecher block, which is a vector of transformed values, serves as the input to the next. Each transformation layer involves its own pair of shared splines and learnable parameters.*

Remark 3 (Necessity of internal shifts)**.** *It is tempting to simplify the nested structures, for instance by removing the inner shift terms like $\eta^{(2)}q$ inside $\phi^{(2)}$ in (4), or $\eta^{(2)}s$ inside $\phi^{(2)}$ and $\eta^{(3)}q$ inside $\phi^{(3)}$ in (5). One might hypothesize that the outer splines $\Phi^{(\ell)}$ could absorb this shifting effect (yielding a single composite spline per Sprecher block). However, experiments (see Section 8.3) suggest that these internal shifts $\eta^{(\ell)}q$ (or $\eta^{(\ell)}s$) applied to the inputs of the $\phi^{(\ell)}$ splines are crucial for the effective functioning of deeper Sprecher Networks. Removing them significantly degrades performance. The precise theoretical reason for their necessity in the multi-layer case, beyond their presence in Sprecher's original single-layer formula, warrants further investigation.*

4 Comparison with related architectures

To position Sprecher Networks accurately, we compare their core architectural features with Multi-Layer Perceptrons (MLPs), networks with learnable node activations (LANs/Adaptive-MLPs), and Kolmogorov-Arnold Networks (KANs).

Table 1: Architectural comparison of neural network families.

Feature	MLP	LAN / Adaptive-MLP	KAN	Sprecher Network (SN)
Learnable Components	Linear Weights (on edges)	Linear Weights + Node Activations	Edge Splines	Block Splines (ϕ, Φ) + Mixing Weights (λ) + Shift Parameter (η)
Fixed components	Node Activations	—	Node Summation	Node Summation (implicit) + Fixed Shifts ($+q$)
Location of	Nodes	Nodes	Edges	Blocks
Non-linearity	(Fixed)	(Learnable)	(Learnable)	(Shared, Learnable)
Node operation	Apply $\sigma(\cdot)$	Apply $\sigma_{\text{learn}}(\cdot)$	$\sum$(inputs)	Implicit in Block Formula
Parameter sharing	None (typically)	Activations? (Maybe)	None (typically)	Splines (ϕ, Φ) per block
Theoretical basis	UAT	UAT	KAT (inspired)	KAS (Sprecher, direct)
Param scaling	$O(LN^2)$	$O(LN^2 + LNG)$ (Approx.)	$O(LN^2G)$	$O(LN + LG)$ (Approx.)

Notes: L=depth, N=average width, G=spline grid size/complexity. UAT=Universal Approx. Theorem, KAT=Kolmogorov-Arnold Theorem, KAS=Kolmogorov-Arnold-Sprecher. LAN details often follow KAN Appendix B [9]. *The parameter scaling notation uses N to denote a typical or average layer width for simplicity, following [9]. For architectures with varying widths d_ℓ, the LN^2 terms should be understood as $\sum_\ell d_{\ell-1}d_\ell$ (MLP, LAN), the LN^2G term for KAN as $(\sum_\ell d_{\ell-1}d_\ell)G$, and the LN term for SN as $\sum_\ell d_{\ell-1}$ (since SN uses weight vectors, not matrices), where the sum is over the relevant blocks/layers, for precise counts.*

Table 1 summarizes the key distinctions between these architectures. MLPs learn edge weights with fixed node activations, LANs add learnable node activations to this structure, KANs move learnability entirely to edge splines while eliminating linear weights, and SNs concentrate learnability in shared block-level splines, block-level shifts, and mixing weights. The critical difference for SNs is their use of weight vectors rather than matrices, which fundamentally reduces the dependence on width from quadratic to linear. This architectural choice can be understood through an analogy with convolutional neural networks: just as CNNs achieve parameter efficiency and improved generalization by sharing weights across spatial locations, SNs share weights across output dimensions within each block. In CNNs, spatial shifts provide the necessary diversity despite weight sharing; in SNs, the shifts ηq and the additive term $+q$ play this diversifying role. This perspective reframes our architectural constraint not as a limitation, but as a principled form of weight sharing motivated by Sprecher's theorem—suggesting that SNs might be viewed as a "convolutional" approach to function approximation networks. While this weight sharing is theoretically justified for single-layer networks, its effectiveness in deep compositions remains an empirical finding that warrants further theoretical investigation. This combination of choices leads to SNs' distinctive parameter scaling of $O(LN + LG)$ compared to KANs' $O(LN^2G)$.

Here, we provide a precise comparison between LANs and SNs. While the following proposition shows that SNs can be expressed as special cases of LANs with specific structural constraints, it is important to note that Sprecher's construction guarantees that this constrained form retains full expressivity in the

single-layer case. This suggests that the extreme parameter sharing and structural constraints in SNs may serve as a beneficial inductive bias rather than a limitation.

Definition 2. *A LAN is an MLP with learnable activation. More precisely, the model is defined as:*

$$f(\mathbf{x}) = A^{(L)} \circ \sigma^{(L-1)} \circ A^{(L-1)} \circ \sigma^{(L-2)} \circ \cdots \circ \sigma^{(1)} \circ A^{(1)}(\mathbf{x}),$$

where $A^{(k)} \colon \mathbb{R}^{d_{k-1}} \to \mathbb{R}^{d_k}$ is an affine map, and $\sigma^{(k)} \colon \mathbb{R} \to \mathbb{R}$ is the activation function (applied coordinate-wise). In an MLP, the trainable parameters are the weights $W^{(k)}$ and biases $b^{(k)}$ of $A^{(k)}(\mathbf{x}) = W^{(k)}\mathbf{x} + b^{(k)}$ for $k = 1, \ldots, L$. In a LAN, σ contains additional trainable parameters, e.g., the coefficients of a spline.

Remark 4 (Note on weight structure)**.** *The following proposition assumes matrix weights $\lambda_{i,q}^{(\ell)}$ as originally described. However, the actual implementation uses vector weights $\lambda_i^{(\ell)}$, which would require modifying the weight matrix structure in the proposition accordingly. We present the matrix version here for completeness, noting that the vector version represents an even more constrained (and parameter-efficient) special case of LANs.*

Proposition 1. *An SN with matrix weights (cf. (3)) is a LAN, where:*

- *in odd layers $k = 2\ell - 1$, the weight matrix $W^{(k)} \in \mathbb{R}^{d_\ell d_{\ell-1} \times d_{\ell-1}}$ is fixed to $[I|\cdots|I]^\top$, where I is the $d_{\ell-1} \times d_{\ell-1}$ identity matrix, the bias vector has only one learnable parameter $\eta^{(\ell)}$ and is structured as $b^{(k)} = \eta^{(\ell)}(0, \ldots, 0, 1, \ldots, 1, \ldots, d_\ell - 1, \ldots, d_\ell - 1)^\top \in \mathbb{R}^{d_\ell d_{\ell-1}}$, and the activation is $\sigma^{(k)} = \phi^{(\ell)}$,*
- *in even layers $k = 2\ell$, the learnable weight matrix $W^{(k)} \in \mathbb{R}^{d_\ell \times d_\ell d_{\ell-1}}$ is structured as*

$$\begin{bmatrix} \lambda_{1,0}^{(\ell)} & \cdots & \lambda_{d_{\ell-1},0}^{(\ell)} & 0 & & & \cdots & & & 0 \\ 0 & \cdots & 0 & \lambda_{1,1}^{(\ell)} & \cdots & \lambda_{d_{\ell-1},1}^{(\ell)} & 0 & & \cdots & 0 \\ & & & & & & \ddots & & & \\ 0 & & & \cdots & & & 0 & \lambda_{1,d_\ell-1}^{(\ell)} & \cdots & \lambda_{d_{\ell-1},d_\ell-1}^{(\ell)} \end{bmatrix},$$

  the bias is fixed to $b^{(k)} = (0, \ldots, d_\ell - 1)^\top \in \mathbb{R}^{d_\ell}$, and the activation is $\sigma^{(k)} = \Phi^{(\ell)}$.

Proof. Follows immediately by inspecting (2). □

Remark 5 (Understanding the LAN representation)**.** *The representation of SNs as LANs in Proposition 1 uses an expanded intermediate state space. Each Sprecher block is decomposed into two LAN layers:*

- *The first layer expands the input $\mathbf{h}^{(\ell-1)} \in \mathbb{R}^{d_{\ell-1}}$ to $\mathbb{R}^{d_\ell d_{\ell-1}}$ by creating d_ℓ shifted copies, where the $(q \cdot d_{\ell-1} + i)$-th component contains $\phi^{(\ell)}(h_i^{(\ell-1)} + \eta^{(\ell)} q)$.*
- *The second layer applies the mixing weights $\lambda_{i,q}^{(\ell)}$ to select and sum the appropriate components for each output q, adds the shift q, and applies $\Phi^{(\ell)}$.*

This construction shows that while SNs can be expressed within the LAN framework, they represent a highly structured special case with specific weight patterns and an expanded intermediate dimension of $O(d_{\ell-1}d_\ell)$ between each pair of SN layers.

While this proposition shows that SNs are special cases of LANs with specific structural constraints, Sprecher's construction guarantees that this constrained form retains full expressivity. In Sprecher's original construction, η can be chosen as a universal constant rather than a learnable parameter. This, combined with Proposition 1, implies that LANs do not lose expressivity when constraining biases to specific structured forms.

Remark 6 (Domain considerations)**.** *The above proposition assumes that the spline functions $\phi^{(\ell)}$ can handle inputs outside their original domain $[0, 1]$, which may arise due to the shifts $\eta^{(\ell)} q$. In Sprecher's original construction with inputs in $[0, 1]^n$, the shifted values $x_i + \eta q$ can indeed fall outside $[0, 1]$. While the theoretical construction can extend ϕ to a larger domain, practical implementations typically clamp inputs to ensure they remain in $[0, 1]$ after shifting, which has proven effective in practice. Alternative approaches include extending the spline domain appropriately or relying on the optimization process to learn suitable η values that keep most shifted inputs within a reasonable range.*

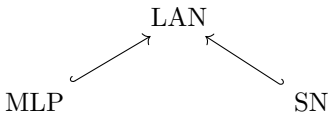


Figure 1: Diagram illustrating the dependencies between the models, in terms of learnable parameters. MLPs are LANs with fixed activation function, while SNs are LANs with a particular parameter structure (Proposition 1).

5 Theoretical aspects and open questions

5.1 Relation to Sprecher (1965) and universality

As shown in Section 3.4.1, a single-layer ($L = 1$) Sprecher Network with input dimension $d_0 = n$ and output dimension $d_1 = 2n + 1$, when configured with appropriate (potentially fixed) weights $\lambda_i^{(1)} = \lambda_i$ and shift $\eta^{(1)} = \eta$, directly reproduces Sprecher's 1965 formula (1). Sprecher proved that for any continuous function $f : [0, 1]^n \to \mathbb{R}$, there exist a suitable monotonic ϕ, a continuous Φ, and constants λ_p, η such that this representation holds [10]. This immediately implies:

Theorem 1 (Universality of single-layer SNs)**.** *For any dimension $n \geq 1$ and any continuous function $f : [0, 1]^n \to \mathbb{R}$, and any $\epsilon > 0$, there exists a single-layer Sprecher Network with architecture $n \to [2n + 1] \to 1$, using sufficiently flexible (e.g., high knot count) continuous splines $\phi^{(1)}$ (monotonic) and $\Phi^{(1)}$, and appropriate parameters $\lambda^{(1)}, \eta^{(1)}$, such that the network output $\hat{f}(\mathbf{x})$ satisfies $\sup_{\mathbf{x}\in[0,1]^n} |f(\mathbf{x}) - \hat{f}(\mathbf{x})| < \epsilon$.*

Thus, single-layer SNs inherit the universal approximation property directly from Sprecher's constructive proof for functions defined on the unit hypercube. Notably, our use of weight vectors λ_i rather than matrices $\lambda_{i,q}$ maintains complete fidelity to Sprecher's original formulation, which used constants λ_p depending only on the input index. This is more than a historical curiosity: Sprecher's proof demonstrates that this constrained form is sufficient for universal approximation, suggesting that the additional parameters in a full matrix formulation may be redundant. The vector formulation enforces a specific structure where all output dimensions share the same linear combination of transformed inputs, with diversity arising solely from the shifts—a constraint that Sprecher showed does not limit expressivity in the single-layer case.

While universality guarantees that an approximation exists, it doesn't quantify how the error behaves as the approximating functions (splines) become more refined. The following lemmas and theorem address the approximation rate achievable by replacing the ideal continuous functions in the Sprecher structure with finite-resolution splines, assuming such an ideal network exists for the target function f.

Lemma 1 (Single block approximation)**.** *Consider a single Sprecher block $T : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$ defined by*

$$T(\mathbf{z})_q = \Phi\left(\sum_{i=1}^{d_{in}} \lambda_i \phi(z_i + \eta q) + q\right), \quad q = 0, \ldots, d_{out} - 1$$

where $\phi \in C^{k+1}(\mathbb{R})$ is monotonic, $\Phi \in C^{k+1}(\mathbb{R})$, $|\lambda_i| \leq \Lambda$, and $\eta > 0$.

Suppose the input satisfies $\mathbf{z} \in \mathcal{B}_{in} := [-B_{in}, B_{in}]^{d_{in}}$. Define:

- *$I_\phi := [-B_{in} - \eta(d_{out} - 1), B_{in} + \eta(d_{out} - 1)]$ (domain containing all possible arguments to ϕ)*
- *$I_\Phi := [-R_{out}, R_{out}]$ where $R_{out} := \Lambda d_{in} \|\phi\|_{L^\infty(I_\phi)} + d_{out}$ (domain containing all possible arguments to Φ)*

Let $\hat{\phi}$ be a monotonic spline of degree k with G_ϕ uniformly spaced knots on I_ϕ, and $\hat{\Phi}$ be a spline of degree k with G_Φ uniformly spaced knots on I_Φ. If

$$\|\phi - \hat{\phi}\|_{L^\infty(I_\phi)} \leq \frac{\|\phi^{(k+1)}\|_{L^\infty(I_\phi)}}{(k+1)!} \left(\frac{|I_\phi|}{G_\phi}\right)^{k+1}$$

$$\|\Phi - \hat{\Phi}\|_{L^\infty(I_\Phi)} \leq \frac{\|\Phi^{(k+1)}\|_{L^\infty(I_\Phi)}}{(k+1)!} \left(\frac{|I_\Phi|}{G_\Phi}\right)^{k+1}$$

then the approximate block $\hat{T}$ using $\hat{\phi}, \hat{\Phi}$ satisfies

$$\sup_{\mathbf{z}\in\mathcal{B}_{in}} \|T(\mathbf{z}) - \hat{T}(\mathbf{z})\|_\infty \leq K_T \max\left\{\left(\frac{|I_\phi|}{G_\phi}\right)^{k+1}, \left(\frac{|I_\Phi|}{G_\Phi}\right)^{k+1}\right\}$$

where

$$K_T := \frac{L_\Phi \Lambda d_{in} \|\phi^{(k+1)}\|_{L^\infty(I_\phi)} + \|\Phi^{(k+1)}\|_{L^\infty(I_\Phi)}}{(k+1)!}$$

and L_Φ is the Lipschitz constant of Φ on I_Φ.

Proof. Fix $\mathbf{z} \in \mathcal{B}_{in}$ and $q \in \{0, \ldots, d_{out} - 1\}$. Define:

$$s_q := \sum_{i=1}^{d_{in}} \lambda_i \phi(z_i + \eta q) + q, \quad \hat{s}_q := \sum_{i=1}^{d_{in}} \lambda_i \hat{\phi}(z_i + \eta q) + q$$

Since $z_i \in [-B_{in}, B_{in}]$ and $0 \leq \eta q \leq \eta(d_{out} - 1)$, we have $z_i + \eta q \in I_\phi$. Thus:

$$\begin{aligned}
|s_q - \hat{s}_q| &= \left|\sum_{i=1}^{d_{in}} \lambda_i [\phi(z_i + \eta q) - \hat{\phi}(z_i + \eta q)]\right| \\
&\leq \sum_{i=1}^{d_{in}} |\lambda_i| \cdot \|\phi - \hat{\phi}\|_{L^\infty(I_\phi)} \\
&\leq \Lambda d_{in} \cdot \frac{\|\phi^{(k+1)}\|_{L^\infty(I_\phi)}}{(k+1)!} \left(\frac{|I_\phi|}{G_\phi}\right)^{k+1}
\end{aligned}$$

Moreover, $|s_q| \leq \Lambda d_{in} \|\phi\|_{L^\infty(I_\phi)} + |q| \leq \Lambda d_{in} \|\phi\|_{L^\infty(I_\phi)} + d_{out} = R_{out}$, so $s_q \in I_\Phi$. Similarly, since $\hat{\phi}$ is bounded on I_ϕ (as a continuous function on a compact set), $\hat{s}_q \in I_\Phi$ for sufficiently large G_ϕ.

Now:

$$\begin{aligned}
|T(\mathbf{z})_q - \hat{T}(\mathbf{z})_q| &= |\Phi(s_q) - \hat{\Phi}(\hat{s}_q)| \\
&\leq |\Phi(s_q) - \Phi(\hat{s}_q)| + |\Phi(\hat{s}_q) - \hat{\Phi}(\hat{s}_q)| \\
&\leq L_\Phi |s_q - \hat{s}_q| + \|\Phi - \hat{\Phi}\|_{L^\infty(I_\Phi)} \\
&\leq L_\Phi \Lambda d_{in} \frac{\|\phi^{(k+1)}\|_{L^\infty(I_\phi)}}{(k+1)!} \left(\frac{|I_\phi|}{G_\phi}\right)^{k+1} + \frac{\|\Phi^{(k+1)}\|_{L^\infty(I_\Phi)}}{(k+1)!} \left(\frac{|I_\Phi|}{G_\Phi}\right)^{k+1}
\end{aligned}$$

Taking the maximum over q and $\mathbf{z}$ gives the result. □

Lemma 2 (Error composition)**.** *Consider an L-block Sprecher network where block ℓ has approximation constant $K_{T^{(\ell)}}$ and Lipschitz constant $L_{T^{(\ell)}}$ (viewing the block as a function $T^{(\ell)} : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}$). Let E_ℓ denote the worst-case error after block ℓ:*

$$E_\ell := \sup_{\mathbf{x}\in[0,1]^n} \|\mathbf{h}^{(\ell)}(\mathbf{x}) - \hat{\mathbf{h}}^{(\ell)}(\mathbf{x})\|_\infty$$

Then:

$$E_\ell \leq \sum_{j=1}^{\ell} \left(\prod_{m=j+1}^{\ell} L_{T^{(m)}}\right) K_{T^{(j)}} \varepsilon_j \tag{6}$$

where ε_j is the approximation error bound for block j from Lemma 1, and the empty product equals 1.

Proof. We proceed by induction. For $\ell = 1$, we have $E_1 \leq K_{T^{(1)}}\varepsilon_1$ directly from Lemma 1.

Assume the result holds for $\ell - 1$. For any $\mathbf{x} \in [0,1]^n$:

$$\begin{aligned}
\|\mathbf{h}^{(\ell)}(\mathbf{x}) - \hat{\mathbf{h}}^{(\ell)}(\mathbf{x})\|_\infty &= \|T^{(\ell)}(\mathbf{h}^{(\ell-1)}(\mathbf{x})) - \hat{T}^{(\ell)}(\hat{\mathbf{h}}^{(\ell-1)}(\mathbf{x}))\|_\infty \\
&\leq \|T^{(\ell)}(\mathbf{h}^{(\ell-1)}(\mathbf{x})) - T^{(\ell)}(\hat{\mathbf{h}}^{(\ell-1)}(\mathbf{x}))\|_\infty \\
&\quad + \|T^{(\ell)}(\hat{\mathbf{h}}^{(\ell-1)}(\mathbf{x})) - \hat{T}^{(\ell)}(\hat{\mathbf{h}}^{(\ell-1)}(\mathbf{x}))\|_\infty \\
&\leq L_{T^{(\ell)}}\|\mathbf{h}^{(\ell-1)}(\mathbf{x}) - \hat{\mathbf{h}}^{(\ell-1)}(\mathbf{x})\|_\infty + K_{T^{(\ell)}}\varepsilon_\ell \\
&\leq L_{T^{(\ell)}}E_{\ell-1} + K_{T^{(\ell)}}\varepsilon_\ell
\end{aligned}$$

Substituting the induction hypothesis completes the proof. □

Theorem 2 (Spline approximation rate for Sprecher Networks)**.** *Let $f : [0,1]^n \to \mathbb{R}$ be realized by an ideal L-block Sprecher network with scalar output. Assume:*

(i) *Each $\phi^{(\ell)}, \Phi^{(\ell)} \in C^{k+1}$ on their respective domains.*

(ii) *Each block satisfies $|\lambda_i^{(\ell)}| \leq \Lambda$ and has shift parameter $\eta^{(\ell)} > 0$.*

(iii) *The network propagates through bounded sets: $\mathbf{h}^{(\ell)}(\mathbf{x}) \in [-B_\ell, B_\ell]^{d_\ell}$ for all $\mathbf{x} \in [0,1]^n$.*

For each block ℓ, approximate $\phi^{(\ell)}$ and $\Phi^{(\ell)}$ with degree-k splines using $G_\phi^{(\ell)}$ and $G_\Phi^{(\ell)}$ knots respectively. Then:

$$\sup_{\mathbf{x}\in[0,1]^n} |f(\mathbf{x}) - \hat{f}(\mathbf{x})| \leq d_L \sum_{j=1}^{L} \left(\prod_{m=j+1}^{L} L_{T^{(m)}} \right) K_{T^{(j)}}\varepsilon_j$$

where:

- $\varepsilon_j = \max\left\{ \left(\frac{2B_{j-1} + 2\eta^{(j)}(d_j - 1)}{G_\phi^{(j)}} \right)^{k+1}, \left(\frac{2R_j}{G_\Phi^{(j)}} \right)^{k+1} \right\}$
- $R_j = \Lambda d_{j-1}\|\phi^{(j)}\|_{L^\infty} + d_j$
- $K_{T^{(j)}} = \frac{L_{\Phi^{(j)}}\Lambda d_{j-1}\|\phi^{(j)\,(k+1)}\|_{L^\infty} + \|\Phi^{(j)\,(k+1)}\|_{L^\infty}}{(k+1)!}$
- $L_{T^{(m)}} = L_{\Phi^{(m)}}\Lambda d_{m-1}L_{\phi^{(m)}}$ *(Lipschitz constant of block m)*

If all blocks use the same number of knots G (i.e., $G_\phi^{(\ell)} = G_\Phi^{(\ell)} = G$ for all ℓ), the error scales as $O(G^{-(k+1)})$ with constants that may grow exponentially with depth L.

Proof. By Lemma 1, each block ℓ can be approximated with error bounded by $K_{T^{(\ell)}}\varepsilon_\ell$. By Lemma 2, the error after L blocks is bounded by the sum given in (6).

For the scalar output case, we have:

$$\begin{aligned}
|f(\mathbf{x}) - \hat{f}(\mathbf{x})| &= \left| \sum_{q=0}^{d_L-1} h_q^{(L)}(\mathbf{x}) - \sum_{q=0}^{d_L-1} \hat{h}_q^{(L)}(\mathbf{x}) \right| \\
&\leq \sum_{q=0}^{d_L-1} |h_q^{(L)}(\mathbf{x}) - \hat{h}_q^{(L)}(\mathbf{x})| \\
&\leq d_L\|\mathbf{h}^{(L)}(\mathbf{x}) - \hat{\mathbf{h}}^{(L)}(\mathbf{x})\|_\infty \\
&\leq d_L E_L
\end{aligned}$$

Substituting the bound for E_L gives the result. □

Remark 7 (Monotonic spline approximation)**.** *The approximation rates in Lemma 1 assume we can achieve standard spline approximation rates while maintaining monotonicity for $\phi^{(\ell)}$. This can be achieved using positive B-splines of degree k with appropriate knot sequences, Bernstein polynomial approximation followed by degree elevation, or the specific log-space parameterization used in implementation, which maintains monotonicity by construction. Each approach may have slightly different constants but achieves the same $O(G^{-(k+1)})$ rate for smooth monotonic functions.*

Remark 8 (Dependence on depth)**.** *The constant in the error bound depends on the layer dimensions d_ℓ and the Lipschitz constants $L_\phi^{(\ell)}, L_\Phi^{(\ell)}$ through the factors $L_{T^{(m)}}$. If these factors are consistently greater than 1, the constant can grow exponentially with depth L. This highlights a potential challenge for approximation with very deep networks, similar to error bounds seen in other deep learning contexts.*

5.2 Vector-valued functions and deeper extensions

For vector-valued functions $f : [0,1]^n \to \mathbb{R}^m$ with $m > 1$, our construction appends an $(L+1)$-th block without final summation. While intuitively extending the representation, the universality of this specific construction is not directly covered by Sprecher's original theorem. The composition of multiple Sprecher blocks to create deep networks represents a natural but theoretically uncharted extension of Sprecher's construction. While single-layer universality is guaranteed, the expressive power of deep SNs remains an open question with several competing hypotheses. Depth might provide benefits analogous to those in standard neural networks: enabling more efficient representation of compositional functions, creating a more favorable optimization landscape despite the constrained parameter space, or allowing the network to gradually transform inputs into representations that are progressively easier to process. Alternatively, the specific constraints of the SN architecture might interact with depth in unexpected ways, either amplifying the benefits of the structured representation or creating new challenges not present in single-layer networks.

Conjecture 1 (Vector-valued Sprecher Representation)**.** *Let $n, m \in \mathbb{N}$ with $m > 1$, and let $f : [0,1]^n \to \mathbb{R}^m$ be any continuous function. Then for any $\epsilon > 0$, there exists a Sprecher Network with architecture $n \to [d_1] \to m$ (using $L = 1$ hidden block of width $d_1 \geq 2n+1$ and one output block), with sufficiently flexible continuous splines $\phi^{(1)}, \Phi^{(1)}, \phi^{(2)}, \Phi^{(2)}$ ($\phi^{(1)}, \phi^{(2)}$ monotonic) and appropriate parameters $\lambda^{(1)}, \eta^{(1)}, \lambda^{(2)}, \eta^{(2)}$, such that the network output $\hat{f}(\mathbf{x})$ satisfies $\sup_{\mathbf{x}\in[0,1]^n} \|f(\mathbf{x}) - \hat{f}(\mathbf{x})\|_{\mathbb{R}^m} < \epsilon$.*

Furthermore, stacking multiple Sprecher blocks ($L > 1$) creates deeper networks. It is natural to hypothesize that these deeper networks also possess universal approximation capabilities, potentially offering advantages in efficiency or learning dynamics for certain function classes, similar to depth advantages observed in MLPs.

Conjecture 2 (Deep universality)**.** *For any input dimension $n \geq 1$, any number of hidden blocks $L \geq 1$, and any continuous function $f : [0,1]^n \to \mathbb{R}$ (or $f : [0,1]^n \to \mathbb{R}^m$), and any $\epsilon > 0$, there exists a Sprecher Network with architecture $n \to [d_1, \ldots, d_L] \to 1$ (or $\to m$), provided the hidden widths $d_1, \ldots, d_L$ are sufficiently large (e.g., perhaps $d_\ell \geq 2d_{\ell-1} + 1$ is sufficient, although likely not necessary), with sufficiently flexible continuous splines $\phi^{(\ell)}, \Phi^{(\ell)}$ and appropriate parameters $\lambda^{(\ell)}, \eta^{(\ell)}$, such that the network output $\hat{f}(\mathbf{x})$ satisfies $\sup_{\mathbf{x}\in[0,1]^n} |f(\mathbf{x}) - \hat{f}(\mathbf{x})| < \epsilon$ (or the vector norm equivalent).*

Proving Conjectures 1 and 2 rigorously would require analyzing the compositional properties and ensuring that the range of intermediate representations covers the domain needed by subsequent blocks, potentially involving careful control over the spline ranges and the effect of the shifts $\eta^{(\ell)}$.

6 Implementation considerations

6.1 Trainable splines

For practical implementations, piecewise-linear splines are a natural choice for both $\phi^{(\ell)}$ and $\Phi^{(\ell)}$. Each spline can be defined by a set of knots (x-coordinates) and corresponding coefficients (y-coordinates). A common approach is to use fixed, uniformly spaced knots over the spline's domain, with learnable coefficients. The number of knots is a hyperparameter that affects both expressivity and computational cost.

For the inner spline $\phi^{(\ell)}$, the domain and range are typically fixed to $[0, 1]$. Monotonicity can be enforced through various parameterization strategies. One effective approach uses a transformation that guarantees positive increments between successive spline values, such as parameterizing the differences in log-space and applying a positive activation function. This ensures strict monotonicity throughout training without requiring explicit constraints or post-processing operations, while maintaining good gradient flow properties. The spline should be initialized to provide a reasonable starting point, such as a roughly linear and increasing function within its range.

The outer spline $\Phi^{(\ell)}$ is defined on a wider interval, e.g., $[-10, 10]$. The exact range can be fixed or made trainable via range parameters (center and radius). Trainable ranges add flexibility but also complexity to the optimization. Monotonicity is not required for $\Phi^{(\ell)}$, and it is often initialized close to the identity function ($y = x$) or a scaled identity within its domain/codomain range to facilitate initial learning.

6.2 Shifts, weights, and optimization

Each block includes the learnable scalar shift $\eta^{(\ell)} > 0$ and the learnable mixing weight vector $\lambda^{(\ell)} \in \mathbb{R}^{d_{\ell-1}}$. The shift parameter $\eta^{(\ell)}$ is required to be positive ($\eta > 0$) in Sprecher's original construction. This positivity constraint can be handled through appropriate initialization and optimization strategies, or enforced explicitly through parameter transformations.

All learnable parameters (spline coefficients, $\eta^{(\ell)}$, $\lambda^{(\ell)}$, and potentially range parameters for $\Phi^{(\ell)}$) are trained jointly using gradient-based optimization methods like Adam [6] or LBFGS. The loss function is typically Mean Squared Error (MSE) for regression tasks. Additional regularization terms may be included to encourage specific properties.

Remark 9 (Regularization for smoother splines)**.** *While training with MSE alone is sufficient, the splines learned under pure MSE optimization can exhibit rapid oscillations or jagged behavior. To encourage smoother, more interpretable splines (as shown in Figures 2 and 3), additional regularization terms can be incorporated:*

$$\mathcal{L}_{total} = \mathcal{L}_{MSE} + w_{flat} \sum_{\ell} \mathit{Penalty}(\Phi^{(\ell)}) + w_{sparse} \sum_{\ell} \mathit{Sparsity}(\lambda^{(\ell)}) + w_{var} \mathcal{L}_{variance}$$

where:

- $\mathit{Penalty}(\Phi^{(\ell)})$ *is a flatness penalty encouraging smoothness, calculated as the mean squared second difference of spline coefficients:* $\frac{1}{K-2} \sum_{k=1}^{K-1} (c_{k+1} - 2c_k + c_{k-1})^2$
- $\mathit{Sparsity}(\lambda^{(\ell)})$ *encourages sparse mixing weights (e.g., using L1 norm combined with entropy)*
- $\mathcal{L}_{variance} = (\sigma_{output} - \sigma_{target})^2$ *encourages the network output to match the target variance*

These regularization terms are not necessary for successful training but can improve interpretability and generalization. The weights w_{flat}, w_{sparse}, and w_{var} control the strength of each regularization component.

6.3 Grid extension for splines

Inspired by techniques used for KANs [9], the spline resolution (number of knots G) can be adaptively increased during training. This approach starts training with a coarse grid (e.g., $G = 5$) and periodically increases the number of knots (e.g., $G \to 10, \to 20, \dots$) based on a schedule or loss stagnation. When increasing the grid resolution from G_1 knots to $G_2 > G_1$ knots, the coefficients for the new, finer grid are initialized by fitting the fine spline to the current coarse spline using least squares projection on a dense set of evaluation points. This preserves the learned function while providing more degrees of freedom. This often leads to "staircase" learning curves, where the loss drops significantly after each grid extension, allowing the model to achieve higher accuracy than possible with a fixed coarse grid, while potentially starting optimization in a smoother landscape.

7 Parameter counting and efficiency as a trade-off

A key consequence of adhering to the vector-based weighting scheme inspired by Sprecher's formula is a dramatic reduction in parameters compared to standard architectures. This represents a strong architectural constraint that may serve as either a beneficial inductive bias or a limitation, depending on the target function class. The specific design of SNs, particularly the sharing of splines and use of weight vectors rather than matrices, leads to a distinctive parameter scaling that warrants careful analysis.

Let's assume a network of depth L (meaning L hidden layers, thus L blocks for scalar output or $L+1$ for vector output), with an average layer width N. We denote the input dimension as N_{in} when it differs significantly from the hidden layer widths. Let G be the number of intervals used for the piecewise-linear splines (implying $G+1$ knots). For simplicity, we approximate the number of parameters per spline as $O(G)$.

The parameter counts for different architectures scale as follows. MLPs primarily consist of linear weight matrices, leading to a total parameter count dominated by these weights, scaling as $O(LN^2)$. LANs (Adaptive-MLPs) have both linear weights ($O(LN^2)$) and learnable activations; if each of the N nodes per layer has a learnable spline, this adds $O(LNG)$ parameters, for a total of $O(LN^2 + LNG)$. KANs replace linear weights with learnable edge splines, with $O(N^2)$ edges between layers. If each edge has a spline with $O(G)$ parameters, the total count per layer is $O(N^2G)$, leading to an overall scaling of $O(LN^2G)$.

Sprecher Networks have a fundamentally different structure. Each block contains mixing weights $\lambda^{(\ell)}$ with $O(N_{in})$ parameters for the first block and $O(N)$ for subsequent blocks (crucially, these are vectors, not matrices), shared splines $\phi^{(\ell)}, \Phi^{(\ell)}$ with $2 \times O(G) = O(G)$ parameters per block independent of N, and a shift parameter $\eta^{(\ell)}$ with $O(1)$ parameter per block. Summing over L (or $L+1$) blocks, the total parameter count scales approximately as $O(LN + LG + L)$.

This scaling reveals the crucial trade-off: SNs achieve a reduction from $O(LN^2)$ to $O(LN)$ in the width-dependent term by using weight vectors rather than matrices. Additionally, in KANs, the spline complexity G multiplies the N^2 term ($O(LN^2G)$), while in SNs, due to spline sharing within blocks, it appears as an additive term ($O(LG)$). This suggests potential for significant parameter savings, particularly when high spline resolution (G) is required for accuracy or when the layer width (N) is moderate to large.

This extreme parameter sharing represents a fundamental architectural bet: that the structure imposed by Sprecher's theorem—using only weight vectors with output diversity through shifts—provides a beneficial inductive bias that outweighs the reduction in parameter flexibility. The empirical observation that training remains feasible (albeit slower) with vector weights suggests this bet may be justified for certain function classes. Moreover, viewing this constraint through the lens of weight sharing in CNNs provides a new perspective: both architectures sacrifice parameter flexibility for a structured representation that has proven effective in practice, though for SNs the theoretical justification comes from Sprecher's theorem rather than domain-specific intuitions about spatial invariance. Whether this constraint serves as beneficial regularization or harmful limitation likely depends on the specific problem domain and the alignment between the target function's structure and the inductive bias imposed by the SN architecture.

7.1 Illustrative parameter comparisons (hypothetical examples)

We present parameter count comparisons based on architectures reported in [9] to illustrate the potential parameter efficiency of SNs. Note that these examples are hypothetical – while the KAN architectures are from their paper, the SN results are theoretical projections that require empirical validation. Optimal architectures for SNs may differ from those of KANs due to the shared spline structure.

PDE solving example (KAN Ref §3.4): KAN architecture $[2, 10, 1]$, reported effective. $N_{in} = 2, N_{hid} = 10, N_{out} = 1$.

- **KAN (est.):** $(2 \times 10 + 10 \times 1) = 30$ edges/splines. Assume $G = 20$ intervals, $k = 3$ B-splines ($G + k \approx 23$ params/spline). Total KAN params $\approx 30 \times 23 = 690$.
- **SN (hypothetical):** Architecture $2 \to [10] \to 1$. $L = 1$ hidden layer, scalar output means $L = 1$ block total. Shared splines: $2 \times (G + k) \approx 2 \times 23 = 46$. Mixing weights: $d_0 = 2$ (vector weights). Shift $\eta^{(1)}$: 1. Total SN params $\approx 46 + 2 + 1 = 49$.

- **Potential advantage:** For equivalent structure, theoretical reduction factor $\approx$ 14x (49 vs 690). Actual performance requires empirical validation.

Knot theory example (KAN Ref §4.3): Pruned KAN $[17, 1, 14]$, $G = 3, k = 3$ ($G+k \approx 6$ params/spline).

- **KAN (est.):** $(17 \times 1 + 1 \times 14) = 31$ edges/splines. Total KAN params $\approx 31 \times 6 = 186$.
- **SN (hypothetical):** Architecture $17 \to [1] \to 14$. $L = 1$ hidden layer, vector output means $L + 1 = 2$ blocks total. Shared splines: $2(L + 1) \times (G + k) \approx 4 \times 6 = 24$. Mixing weights: $d_0 + d_1 = 17 + 1 = 18$ (vector weights). Shifts $\eta^{(1)}, \eta^{(2)}$: 2. Total SN params $\approx 24 + 18 + 2 = 44$.
- **Potential advantage:** Theoretical reduction factor $\approx$ 4.2x (44 vs 186). The narrow intermediate dimension $d_1 = 1$ might pose challenges for SN training.

These calculations illustrate the dramatic parameter reduction possible with SNs, but they also highlight a crucial practical consideration: the optimal architecture for SNs likely differs substantially from that of MLPs or KANs. With vector weights providing only linear scaling in width, SNs may require wider or deeper architectures to achieve comparable expressivity. The art of architecture selection for SNs involves balancing the parameter efficiency against the need for sufficient expressivity—a trade-off that remains poorly understood and likely depends strongly on the problem domain. Early empirical evidence suggests that SNs excel when the target function aligns well with their compositional, shift-based structure, but struggle when forced to approximate functions that require truly independent processing of different output dimensions.

8 Empirical demonstrations and case studies

While comprehensive benchmarking remains future work, we provide initial empirical demonstrations to illustrate the feasibility and characteristics of SNs.

8.1 Basic function approximation

We train SNs on datasets sampled from known target functions f. The network learns the parameters ($\eta^{(\ell)}$, $\lambda^{(\ell)}$) and spline coefficients ($\phi^{(\ell)}$, $\Phi^{(\ell)}$) via gradient descent (Adam optimizer, typically $O(10^5)$ updates) using MSE loss plus regularization.

For 1D functions $f(x)$ on $[0, 1]$, an SN like $1 \to [W] \to 1$ (one block) learns $\phi^{(1)}$ and $\Phi^{(1)}$ that accurately interpolate f, effectively acting as a learnable spline interpolant structured according to Sprecher's formula. Figure 2 provides an example where an SN is trained on data generated from a known Sprecher structure. While the network achieves a very accurate approximation of the overall function $f(x)$, the learned components (splines $\hat{\phi}, \hat{\Phi}$, weights $\hat{\lambda}$, shift $\hat{\eta}$) only partially resemble the ground truth functions and parameters used to generate the data. Perfect recovery of internal components is generally not guaranteed, as multiple parameter combinations might yield similar final outputs.

Moving to multivariate functions, consider the 2D scalar case $f(x, y) = (\exp(\sin(\pi x) + y^2) - 1)/7$. A network like $2 \to [5, 8, 5] \to 1$ (3 blocks) can achieve high accuracy. Figure 3 shows the interpretable layerwise spline plots and the final fit quality. For 2D vector-valued functions $f(x, y) = (f_1(x, y), f_2(x, y))$, deeper networks like $2 \to [20, \ldots, 20] \to 2$ (e.g., 5 hidden layers, requiring 6 blocks) are used. Figure 4 illustrates the learned splines and the approximation of both output surfaces.

These examples demonstrate the feasibility of training SNs and the potential interpretability offered by visualizing the learned shared splines $\phi^{(\ell)}$ and $\Phi^{(\ell)}$ for each block.

Remark 10 (Spline artifacts and compensation mechanisms)**.** *Learned outer splines $\Phi^{(\ell)}$ often develop pronounced oscillations, particularly when using vector weights rather than matrix weights. These oscillations can be understood as a compensation mechanism: with fewer degrees of freedom in the mixing weights, the outer splines must work harder to achieve the necessary transformations. This phenomenon parallels observations in other constrained architectures, where limiting one component's flexibility forces other components to develop more complex behaviors. From an optimization perspective, these oscillations may indicate that the network is operating near the limits of its expressivity for the given architecture, suggesting that wider or deeper networks might achieve the same function approximation with smoother splines.*

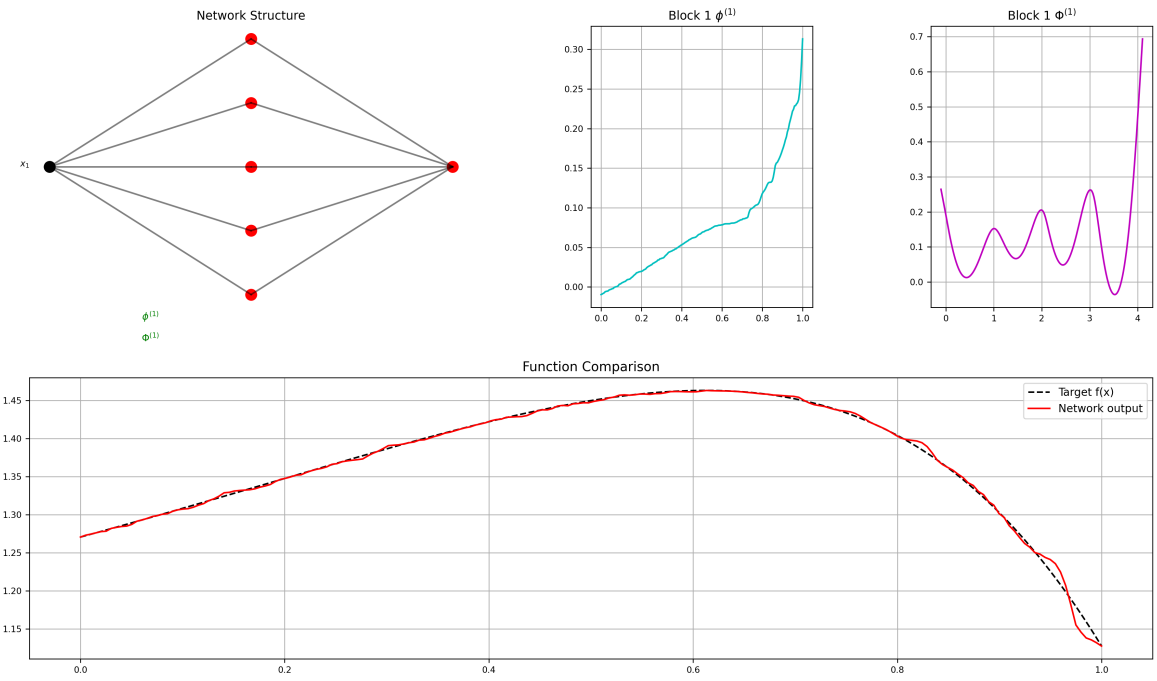


Figure 2: Visualization of a trained Sprecher Network with architecture $1 \to [5] \to 1$ (one block) trained on data sampled from $f(x) = \sum_{q=0}^{4} \Phi(\lambda_{q+1}\phi(x+\eta q)+q)$, where the ground truth functions were $\phi(x) = (e^x-1)/(e-1)$ and $\Phi(x) = \sin x$, with shift $\eta = 1/10$ and mixing weights $\lambda = \{1/2, -4/5, 1, 1/5, -6/5\}$. Top row: Network structure, learned monotonic spline $\phi^{(1)}$ (cyan), learned general spline $\Phi^{(1)}$ (magenta). Bottom row: Comparison between the target function $f(x)$ (dashed black) and the network output (solid red). This network was trained with regularization terms to encourage smoother splines.

8.2 Systematic benchmarks (selected examples)

To provide more quantitative insight, we consider how SNs might perform on examples from the KAN article, focusing on parameter efficiency and scaling. These projections are based on the architectural differences and require empirical validation.

For synthetic families with known structure, such as smooth additive functions like $f(x,y) = \exp(\sin \pi x + y^2)$, KAS theory suggests a simple structure exists. SNs might achieve better approximation error scaling with respect to the number of parameters compared to MLPs, though this needs empirical verification. For multiplicative functions like xy, which has a known simple KAS representation ($xy = [(x+y)/2]^2 - [(x-y)/2]^2$), an SN architecture like $[2, 2, 1]$ might be highly efficient compared to MLPs requiring significantly more parameters for similar accuracy. High-dimensional additive functions of the form $f(\mathbf{x}) = \exp(\frac{1}{d}\sum \sin^2(\pi x_i))$ present an interesting case where the predominantly additive structure composed with univariate functions might align well with SNs' architectural bias.

The idea of using SNs for compressing representations of special functions (e.g., from SciPy library) is compelling. Comparing the parameter count of a trained SN achieving a target RMSE against an MLP trained for the same task could demonstrate practical parameter savings, though such comparisons await implementation.

8.3 Ablation study: role of the shift η

To empirically validate the importance of the internal shift parameter $\eta^{(\ell)}$ noted in Remark 3, ablation studies can be performed. Training SNs with $\eta^{(\ell)}$ fixed to 0 across all blocks typically results in significantly higher final RMSE (often 1-2 orders of magnitude worse) and potentially slower convergence or saturation at poorer loss values, especially for deeper networks. This confirms that the ηq shift inside $\phi^{(\ell)}$ is not redundant

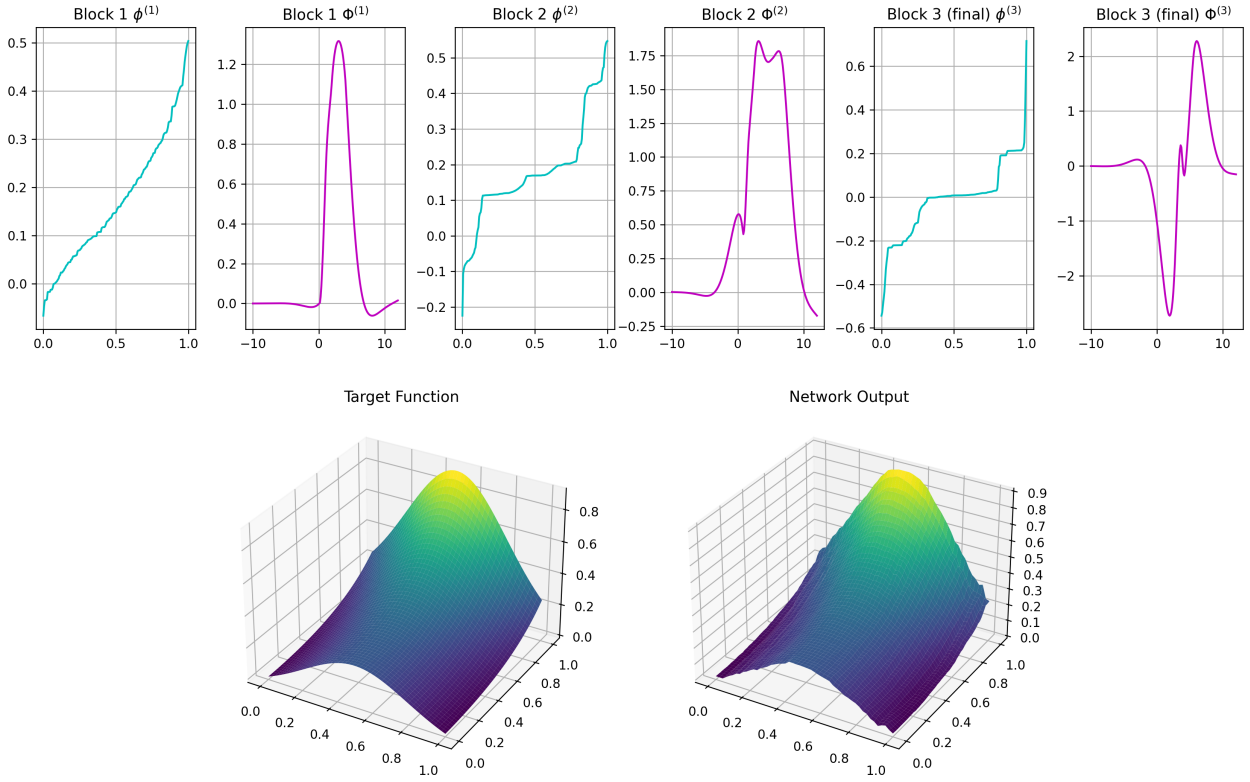


Figure 3: Visualization of a trained $2 \to [5, 8, 5] \to 1$ Sprecher Network (3 blocks) approximating the scalar 2D target function $z = f(x, y) = (\exp(\sin(\pi x) + y^2) - 1)/7$. Top row: Learned spline functions for each block — monotonic splines $\phi^{(\ell)}$ (cyan) and general splines $\Phi^{(\ell)}$ (magenta). Bottom row: Comparison between the target function surface (left) and the network approximation (right). This network was trained with regularization terms to encourage smoother splines.

and plays a crucial role in the expressivity or optimization dynamics of multi-layer SNs.

8.4 MNIST classification

To demonstrate SNs' applicability beyond function approximation, we tested them on the MNIST digit classification task. The 28×28 grayscale images were flattened to 784-dimensional vectors and normalized to [0,1]. A deeper network with architecture $784 \to [100, 100, 100] \to 10$ achieved over 99.5% test accuracy after training for several hours on consumer hardware. A shallower network with architecture $784 \to [100] \to 10$ (using Sprecher's theoretical minimum of one hidden layer, but with significantly fewer than the $2n+1 = 1569$ nodes to guarantee universality) achieved approximately 92% test accuracy. These results demonstrate that while achieving state-of-the-art performance on image tasks would likely require a convolutional design, SNs can attain competitive results on standard benchmarks. More importantly, they confirm that deeper architectures provide significant benefits for complex tasks, consistent with observations in traditional deep learning.

9 Limitations and Discussion

The primary limitation of this work is the gap between our theoretically-grounded single-layer model and our empirically-driven deep architecture. While single-layer SNs inherit universal approximation properties directly from Sprecher's theorem, the universality and approximation bounds of deep, compositional SNs remain open theoretical questions. Our work provides empirical evidence that these architectures are effective,

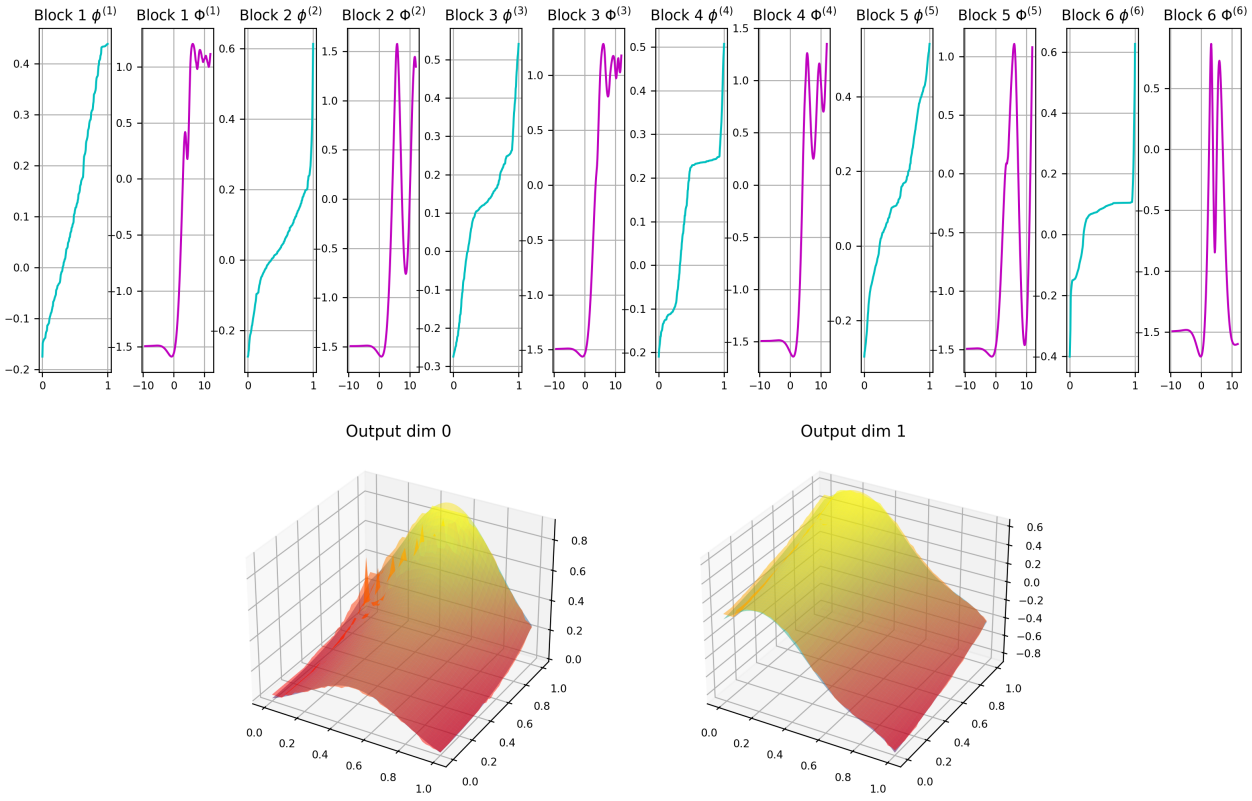


Figure 4: Visualization of a trained Sprecher Network with architecture $2 \to [20, 20, 20, 20, 20] \to 2$ (5 hidden layers, 6 blocks total), approximating a vector-valued function $f(x, y) = ((\exp(\sin(\pi x) + y^2) - 1)/7,\ \frac{1}{4}y + \frac{1}{5}y^2 - x^3 + \frac{1}{5}\sin(7x))$. Top row: Learned spline functions for each block ($\phi^{(\ell)}$ in cyan, $\Phi^{(\ell)}$ in magenta). Bottom row: Comparison between the target surfaces and the network outputs for both output dimensions (dim 0 left, dim 1 right; target=viridis/blue-green, prediction=autumn/red-yellow overlay).

but formal proofs of their expressivity are lacking.

The core of our proposed architecture, the Sprecher block, imposes a very strong inductive bias. By forcing all feature interactions within a block through two shared splines and using weight vectors rather than matrices, the network's expressive power is heavily constrained compared to standard architectures. This design choice represents a fundamental trade-off: it leads to remarkable parameter efficiency but may limit the model's ability to learn functions that do not align well with this specific compositional structure. Consequently, our claims of parameter efficiency should be understood in the context of this trade-off – the architecture may excel for certain function classes while struggling with others.

Furthermore, the current implementation faces practical challenges. Training speed is significantly slower than comparable MLPs, potentially limiting scalability. The optimization landscape created by the shared spline structure presents unique challenges. With weight vectors rather than matrices, the network must learn to extract features that are simultaneously useful for all output dimensions—a more constrained task than learning output-specific features. This constraint manifests in practice as increased training time (empirically 3-10× slower) and sometimes (always?) requires the outer splines $\Phi^{(\ell)}$ to develop pronounced oscillations to compensate for the reduced parameter flexibility. These oscillations, while mathematically valid, can complicate optimization and suggest that specialized training strategies or architectural modifications (such as residual connections or adapted spline domains) may be beneficial. These implementation challenges, combined with the theoretical gaps, suggest that SNs are currently best viewed as an exploration of how classical mathematical theorems can inspire novel architectural designs rather than as a ready replacement for existing methods.

10 Challenges and future directions

While Sprecher Networks offer a theoretically-inspired architecture with potential advantages, several challenges remain alongside exciting opportunities for future research.

10.1 Implementation and training challenges

Current implementations face several significant obstacles. Training speed and stability remain primary concerns, with SNs often requiring 10x or more computation time compared to comparable MLPs per epoch (*** is this really true? ***). The shared spline structure can lead to optimization difficulties where networks converge to suboptimal local minima with flat or overly simplistic splines. Escaping such minima likely requires advanced optimization techniques tailored specifically to the unique parameter structure of SNs.

The initialization of splines, shifts, and weights significantly impacts training success, yet optimal strategies remain elusive. Similarly, architectural choices such as network depth and hidden layer widths have profound effects on performance, but principled methods for architecture selection are lacking. The number of knots for shared splines represents a critical hyperparameter that, unlike in KANs where complexity can vary per edge, affects the entire block's expressivity in SNs. Developing adaptive strategies for knot placement and automated methods for selecting appropriate spline resolution based on data complexity would be highly valuable.

10.2 Directions for further research

The theoretical foundations of deep SNs require significant development. One particularly intriguing direction concerns potential equivariance properties of the architecture. The weight-sharing structure of SNs, analogous to convolutions, raises the question of whether SNs satisfy any form of equivariance. While CNNs exhibit translation equivariance, the relevant group action for SNs is less clear—it may relate to permutations of output indices or transformations of the function domain. Understanding such properties could provide deeper insight into the inductive bias of the architecture and guide its application to problems with known symmetries.

Beyond equivariance, several other theoretical questions merit investigation.

Proving the universality conjectures for vector-valued and deep SNs would establish whether the empirically observed expressivity has theoretical backing. Rigorous analysis of approximation rates and comparison with existing architectures under various smoothness assumptions would clarify when SNs might offer advantages. Understanding the precise role of internal shifts in deep networks, i.e., why the $\eta^{(\ell)}q$ terms appear essential beyond their presence in Sprecher's original formula, remains an intriguing theoretical question. Additionally, investigating whether the parameter reduction from $O(N^2)$ to $O(N)$ per block comes with any expressivity trade-offs in practice, despite Sprecher's theorem guaranteeing universality with vector weights in the single-layer case, would provide valuable insights into the depth-efficiency trade-offs. Sample complexity and generalization properties of SNs compared to standard architectures also warrant theoretical investigation.

Improved training strategies represent another critical research direction. Developing robust initialization schemes based on data statistics, adaptive optimization methods that specifically account for the different parameter types (spline coefficients, shift parameters η, mixing weights λ), and effective regularization techniques tailored to the shared-spline structure could significantly improve training efficiency. The potential for second-order optimization methods that exploit the specific structure of SNs deserves exploration, as the shared spline parameterization might enable more efficient Hessian approximations than in standard neural networks.

Architectural enhancements offer multiple avenues for improvement. Unlike KANs where edge pruning can significantly reduce parameters, pruning individual weights in SNs yields relatively small gains due to the vector structure. However, automatic pruning of entire Sprecher blocks based on their contribution to the network output could lead to more compact architectures. Exploring higher-order B-splines or other basis functions beyond piecewise linear representations might offer better smoothness or approximation properties, particularly for functions with higher-order continuity requirements. For very high-dimensional

inputs where even $O(N)$ scaling becomes prohibitive, investigating alternative representations such as low-rank approximations for λ or replacing splines with small neural networks or tensor decompositions could maintain efficiency while improving expressivity. The sophisticated residual connection structure in SNs, using scalar weights when dimensions match and learnable projections otherwise, suggests that even more advanced skip connection patterns, such as dense connections or learnable gating mechanisms, could enhance training stability for very deep networks.

Applications to scientific machine learning present particularly promising opportunities where SNs' unique properties could provide concrete advantages. The architecture's sensitive dependence on input dimension suggests an intriguing possibility for function dimensionality discovery: by training SNs with varying d_{in} on unstructured data and using model selection criteria that balance fit quality and complexity, one might infer the intrinsic dimensionality of the underlying process generating the data. This could prove valuable in scientific domains where determining the true number of relevant variables is itself a research question. The interpretability offered by visualizing learned splines could provide insights in domains where understanding the learned function is as important as prediction accuracy. Furthermore, the explicit structure of learned splines might enable integration with symbolic regression tools to automatically suggest or fit closed-form expressions for the learned $\phi^{(\ell)}$ and $\Phi^{(\ell)}$ functions after training, potentially revealing underlying mathematical relationships in the data. Developing SN-based approaches for physics-informed neural networks or other scientific computing applications where both parameter efficiency and interpretability are valued could demonstrate practical advantages over black-box approaches.

Hardware acceleration tailored to the specific computational patterns of SNs could address current speed limitations. The architecture's reliance on shared spline evaluations and vector-vector operations, rather than the matrix multiplications that dominate standard neural networks, suggests opportunities for specialized implementations. Custom kernels that efficiently handle the repeated evaluation of the same splines with different shifted inputs could significantly improve training speed. The regular structure and extreme parameter sharing might enable hardware optimizations that are impossible with less structured architectures, potentially narrowing or even reversing the current performance gap with standard implementations.

11 Conclusion

We have introduced Sprecher Networks (SNs), a trainable neural architecture inspired by David Sprecher's 1965 constructive proof of the Kolmogorov-Arnold theorem. By composing functional blocks that utilize shared monotonic and general splines, learnable mixing weights, and explicit shifts, SNs offer a distinct approach to function approximation that differs fundamentally from MLPs, KANs, and other existing architectures.

The key innovation lies in how we have repurposed the components of a classical approximation theorem as building blocks for deep learning. While Sprecher's theorem guarantees universal approximation for single-layer networks with specific width requirements, our empirical findings suggest that deep compositions of these constrained blocks can be surprisingly effective. This effectiveness comes with very high (theoretical) parameter efficiency, scaling as $O(LN + LG)$ rather than the $O(LN^2)$ of MLPs or $O(LN^2G)$ of KANs, achieved through the combination of shared splines within blocks and the use of weight vectors rather than matrices.

However, this parameter efficiency represents a trade-off rather than a pure advantage. The strong architectural constraints imposed by adhering to Sprecher's formula may limit expressivity for certain function classes even as they provide beneficial inductive bias for others. Our work thus raises fundamental questions about the role of theorem-inspired constraints in deep learning architectures: when do such constraints guide learning toward efficient solutions, and when do they overly restrict the hypothesis space?

While comprehensive benchmarking and theoretical analysis remain as future work, our initial demonstrations show that SNs can successfully learn diverse functions and achieve competitive performance on standard tasks like MNIST classification. The ability to visualize and interpret the learned splines offers potential advantages for scientific applications where understanding the learned function is crucial. Yet practical challenges, particularly in training speed and optimization, must be addressed before SNs can be considered a mature alternative to existing methods.

This work demonstrates how classical mathematical results can inspire novel architectural designs that

challenge conventional assumptions in deep learning. The journey from Sprecher's 1965 theorem to trainable neural networks reveals surprising connections: weight sharing patterns reminiscent of convolutional networks emerge naturally from theoretical considerations, extreme parameter constraints prove compatible with effective learning, and decades-old mathematical structures anticipate modern architectural innovations. Whether SNs represent a practical advance or primarily a theoretical curiosity, they underscore the value of revisiting foundational results with fresh perspectives informed by contemporary machine learning insights. Whether the specific constraints of SNs prove broadly useful or remain a fascinating special case, the approach of mining theoretical results for architectural inspiration offers a promising direction for developing more efficient and interpretable neural networks. The empirical success of deep SNs, despite lacking theoretical guarantees beyond the single-layer case, highlights both the power of compositional architectures and the continuing gap between theoretical understanding and practical effectiveness in deep learning.

References

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

[2] Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, **2**(4), 303–314.

[3] Goyal, M., Goyal, R., Lall, B. (2019). "Learning activation functions: A new paradigm for understanding neural networks," arXiv preprint arXiv:1906.09529.

[4] Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

[5] Hornik, K., Stinchcombe, M., White, H. (1989). "Multilayer feedforward networks are universal approximators," *Neural networks*, **2**(5), 359–366.

[6] Kingma, D. P., Ba, J. (2014). "Adam: A Method for Stochastic Optimization," arXiv preprint arXiv:1412.6980.

[7] 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**(5), 953–956.

[8] 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.

[9] 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.

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

[11] Zhang, S., Shen, Z., Yang, H. (2022). "Neural network architecture beyond width and depth," *Advances in Neural Information Processing Systems*, **35**, 5669–5681.