Sprecher Networks: A Trainable Architecture Based on the Kolmogorov–Arnold–Sprecher Theorem

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

June 7, 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 discuss theoretical motivations, implementation details, compare SNs with related architectures (MLPs, KANs, and networks with learnable node activations), and explore potential advantages in interpretability, universality, and parameter efficiency.

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.

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.

*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

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 $\lambda_{i,q}$ that combine contributions from different input 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. Diversity in the transformation arises from the mixing weights (λ) and the index-dependent shifts (q).

Concretely, each Sprecher block applies the transformation:

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

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. However, it's important to note that rigorous universality is established only for a single block with scalar output (directly from Sprecher's theorem); the universality of networks with $L > 1$ blocks or vector-valued outputs remains conjectural (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 matrix** λ**:** A matrix $\{\lambda_{i,q}\}$ of size $d_{\text{in}} \times d_{\text{out}}$, whose entries are learnable. These weights linearly combine the contributions from different input dimensions after transformation by ϕ.
- **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:

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

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 $\lambda_{i,q}^{(\ell)}$, 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:

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

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

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

then the overall function is

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

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:

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

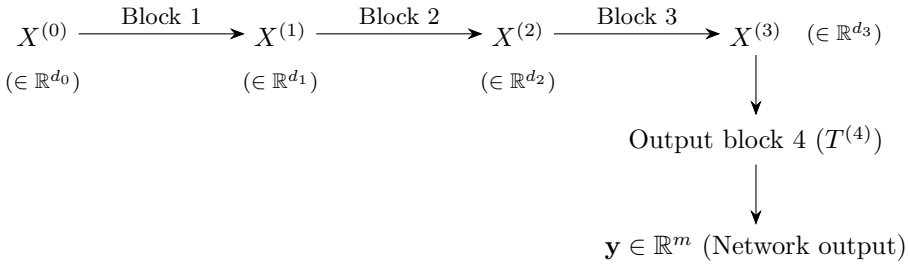
for $q = 0, \ldots, m-1$. The network output function is then:

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

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:

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

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

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:

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

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

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

The final network output is 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:

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

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:

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

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

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

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

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

Remark 1. *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 2 (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 Non-linearity	Nodes (Fixed)	Nodes (Learnable)	Edges (Learnable)	Blocks (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.)	$O(LN^2G)$	$O(LN^2 + LG)$ (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 (involving L or $L+1$ weight matrices depending on scalar/vector output) should be understood as $\sum_\ell d_{\ell-1}d_\ell$ (MLP, LAN, SN) and the LN^2G term for KAN as $(\sum_\ell d_{\ell-1}d_\ell)G$, where the sum is over the relevant blocks/matrices, for precise counts.*

Table 1 summarizes the key distinctions:

- **Location of learnability:** MLPs learn edge weights. LANs add learnable node activations. KANs move learnability entirely to edge splines, eliminating linear weights. SNs have learnability in shared block-level splines (ϕ, Φ), block-level shifts (η), and mixing weights (λ).
- **Linear weights:** Present in MLPs, LANs, and SNs (λ). Absent in KANs.
- **Node operation:** MLPs/LANs apply activations at nodes. KANs perform simple summation at nodes. SNs encapsulate the transformation within the block formula, which implicitly involves summations but also specific shifts.
- **Sharing**: KANs typically have independent splines per edge. SNs enforce sharing of ϕ and Φ within each block, a key factor in their parameterization.
- **Parameter scaling**: The sharing mechanism in SNs leads to a different parameter scaling ($O(LN^2 + LG)$) compared to KANs ($O(LN^2G)$), potentially offering significant savings when the spline complexity G is large relative to width N.

This comparison clarifies that SNs represent a distinct architecture, related to but not a direct subset of KANs, primarily due to the presence of mixing weights (λ), the specific shift structure ($\eta q, q$), and the block-level sharing of splines.

Here, we provide a precise comparison between LANs and SNs. More specifically, we argue that SNs correspond to LANs with (almost) fixed biases that have dependencies between pairs of neighboring layers and with the weights in every other layer constrained to the identity. 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.

Proposition 1. *An SN (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 3 (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.

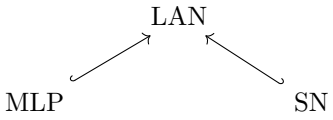


Figure 1: Diagram illustrating the dependencies between the models, in terms of learnable parameters. MLPs are LANs with fixed activation function, while SNs are LANs with a particular parameter structure (Proposition 1).

Remark 4 (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.*

5 Theoretical aspects and universality

5.1 Relation to Sprecher (1965) and universality

As shown in Section 3.3.1, 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 $\lambda_{i,q}^{(1)} = \lambda_p$ 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.

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.

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

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

where $\phi \in C^{k+1}(\mathbb{R})$ is monotonic, $\Phi \in C^{k+1}(\mathbb{R})$, $|\lambda_{i,q}| \leq \Lambda$, 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 $q \in \{0, \dots, d_{out} - 1\}$. Define:

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

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

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

Moreover, $|s_q| \le \Lambda d_{in} \|\phi\|_{L^\infty(I_\phi)} + |q| \le \Lambda d_{in} \|\phi\|_{L^\infty(I_\phi)} + d_{out} = R_{out}$, 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:

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

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:

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

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 $E_1 \le K_{T^{(1)}} \varepsilon_1$ directly from Lemma 1.

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

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

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* $|\lambda_{i,q}^{(\ell)}| \leq \Lambda$ *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 5 (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.*
- *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 6 (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. We conjecture its validity:

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 $\Phi^{(\ell)}$.

- **Representation:** 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.
- **Inner spline $\phi^{(\ell)}$:**
  - **Domain/range:** Typically fixed domain $[0,1]$ and range $[0,1]$.
  - **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.
  - **Initialization:** Should be initialized to provide a reasonable starting point, such as a roughly linear and increasing function within its range. The specific initialization strategy may depend on the chosen parameterization method for ensuring monotonicity.
- **Outer Spline $\Phi^{(\ell)}$:**
  - **Domain/range:** 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:** Not required.
  - **Initialization:** 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

- **Parameters:** Each block includes the learnable scalar shift $\eta^{(\ell)} > 0$ and the learnable mixing weight matrix $\lambda^{(\ell)} \in \mathbb{R}^{d_{\ell-1} \times d_\ell}$. 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.
- **Optimization:** 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.
- **Loss function:** Typically Mean Squared Error (MSE) for regression tasks. Additional regularization terms may be included to encourage specific properties.

Remark 7 (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:*

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

where:

- $Penalty(\Phi^{(\ell)})$ *is a flatness penalty encouraging smoothness, calculated as the mean squared second difference of spline coefficients:* $\frac{1}{K-2} \sum_{k=1}^{K-1} (c_{k+1} - 2c_k + c_{k-1})^2$
- $Sparsity(\lambda^{(\ell)})$ *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.

- **Schedule:** Start training with a coarse grid (e.g., $G = 5$). Periodically (e.g., every few thousand steps or based on loss stagnation), increase the number of knots (e.g., $G \to 10, \to 20, \dots$).
- **Initialization:** When increasing the grid resolution from G_1 knots to $G_2 > G_1$ knots, initialize the coefficients for the new, finer grid 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.
- **Effect:** 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

A key motivation for exploring architectures like SNs and KANs is the potential for improved parameter efficiency compared to standard MLPs, especially for functions exhibiting certain structures. The specific design of SNs, particularly the sharing of splines, leads to a distinct parameter scaling.

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. Let G be the number of intervals used for the piecewise-linear splines (implying $G + 1$ knots), and let k be the order of the spline (e.g., $k = 1$ for linear, $k = 3$ for cubic B-splines). For simplicity, we approximate the number of parameters per spline as $O(G)$.

- **MLP:** Primarily consists of linear weight matrices. The total parameter count is dominated by these weights, scaling as $O(LN^2)$. The number of parameters associated with fixed activations is negligible.
- **LAN / Adaptive-MLP:** Has 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. Total: $O(LN^2 + LNG)$.
- **KAN [9]:** Replaces linear weights with learnable edge splines. There are $O(N^2)$ 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 Network (SN):** Each block has:
  - Mixing weights $\lambda^{(\ell)}$: $O(N^2)$ parameters per block.
  - Shared splines $\phi^{(\ell)}, \Phi^{(\ell)}$: $2 \times O(G) = O(G)$ parameters per block, *independent* of N.
  - Shift parameter $\eta^{(\ell)}$: $O(1)$ parameter per block.

  Summing over L (or $L+1$) blocks, the total parameter count scales approximately as:

  $$O(LN^2 + LG + L).$$

  Additional optional components include: output scaling parameters (2 for scalar output, $2m$ for vector output when applied per dimension), and residual connection parameters when enabled (at most L scalar weights when dimensions match, or absorbed into the LN^2 term through projection matrices when dimensions differ). In practice, these additional terms are negligible compared to the dominant LN^2 and LG terms.

Comparison insight: The crucial difference lies in how the spline complexity G contributes. In KANs, it multiplies the N^2 term associated with layer connectivity ($O(LN^2G)$). In SNs, due to spline sharing within blocks, it appears as an additive term ($O(LG)$) alongside the mixing weight term ($O(LN^2)$).

This suggests that SNs have the *potential* for significant parameter savings compared to KANs, particularly when:

1. High spline resolution (G) is required for accuracy.
2. The layer width (N) is relatively small or moderate compared to G.

In such regimes ($G \gg N$), the LN^2 term might dominate in SNs, while the LN^2G term dominates in KANs.

7.1 Illustrative parameter comparisons

We present parameter count comparisons based on architectures reported in [9] to highlight the potential parameter efficiency of SNs. Note that optimal architectures for SNs may differ from those of KANs due to the shared spline structure, and these calculations serve primarily to illustrate the parameter scaling differences.

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 (equiv. structure):** 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: $d_0 \times d_1 = 2 \times 10 = 20$. Shift $\eta^{(1)}$: 1. Total SN params $\approx 46 + 20 + 1 = 67$.
- **Potential advantage:** For equivalent structure $2 \to [10] \to 1$ (one hidden layer, $L = 1$ block for scalar output): Total SN params ≈ 67, giving a factor ≈ 10x reduction (67 vs 690). (*Note: Direct architectural equivalence between KAN layers and SN blocks requires careful consideration. The comparison assumes similar expressivity, which needs 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 (equiv. structure):** 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: $(d_0 \times d_1) + (d_1 \times d_2) = (17 \times 1) + (1 \times 14) = 31$. Shifts $\eta^{(1)}, \eta^{(2)}$: 2. Total SN params $\approx 24 + 31 + 2 = 57$.
- **Potential advantage:** Factor ≈ 3.3x reduction (57 vs 186). (*Note: Requires validation. The narrow intermediate dimension $d_1 = 1$ might pose challenges for SN training.*).

These calculations serve only to illustrate the *potential* parameter scaling differences arising from the architectural choice of shared block splines versus per-edge splines. Actual performance and optimal architectures need empirical study.

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.

- **1D case:** For $f(x)$ 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 ($f(x) = \sum_q \Phi(\lambda_q \phi(x + \eta q) + q)$ with known $\phi, \Phi, \lambda, \eta$). 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, especially given the flexibility of splines and the complex interplay between parameters during optimization. This highlights a potential challenge in uniquely identifying underlying generating functions solely from learned SN parameters, although the learned structure often provides valuable insights.
- **2D case (scalar):** Moving to multivariate functions, consider $f(x, y) = (\exp(\sin(\pi x) + y^2) - 1)/7$. 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.
- **2D case (vector):** E.g., $f(x, y) = (f_1(x, y), f_2(x, y))$. A deeper network like $2 \to [20, \ldots, 20] \to 2$ (e.g., 5 hidden layers, requiring 6 blocks) is 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 8 (Spline artifacts)**.** *Learned outer splines $\Phi^{(\ell)}$ occasionally develop pronounced peaks with adjacent undershoots, resembling biphasic waveforms, see $\Phi^{(1)}$ in Block 1 of Figure 3. While such shapes can arise from approximating sharp features with finite-resolution splines, the precise reason for this specific pattern appearing in $\Phi^{(1)}$, especially when contrasted with $\Phi^{(2)}$ that features the same number of knots and handles high-curvature features very well, is not determined and likely relates to the complex interplay during optimization for this particular network.*

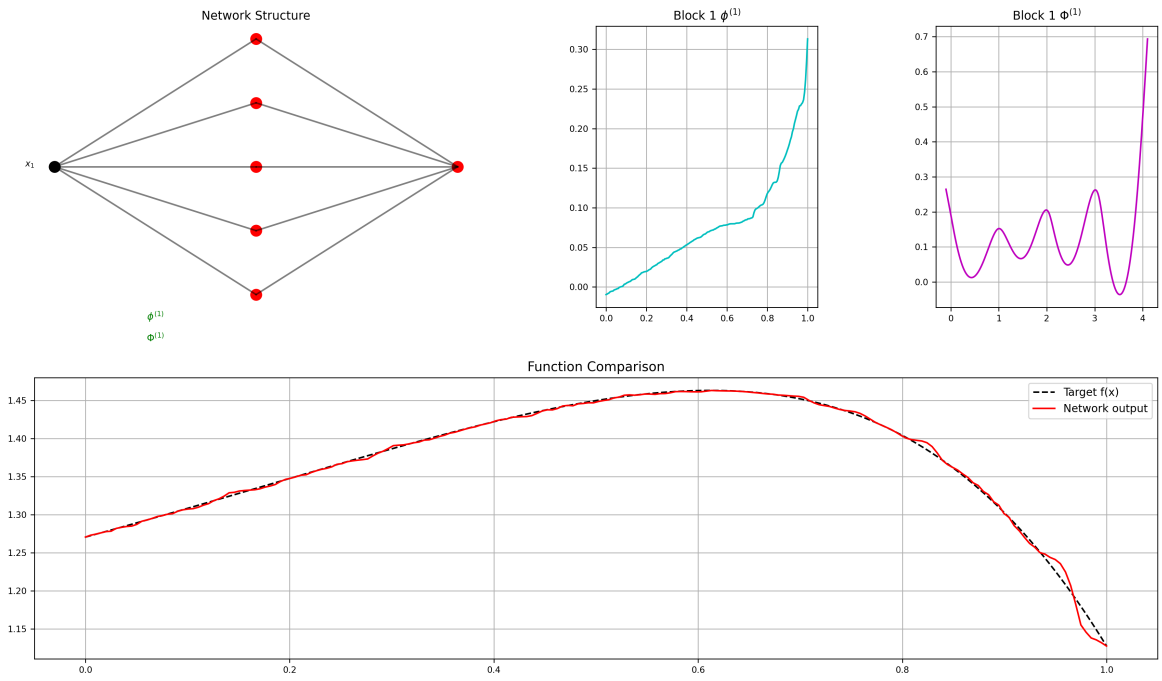


Figure 2: Visualization of a trained Sprecher Network with architecture $1 \to [5] \to 1$ (one block) trained on data sampled from $f(x) = \sum_{q=0}^{4} \Phi(\lambda_{q+1}\phi(x + \eta q) + q)$, where the ground truth functions were $\phi(x) = (e^x - 1)/(e - 1)$ and $\Phi(x) = \sin x$, with shift $\eta = 1/10$ 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 illustrate SNs in a few examples from the KAN article, focusing on parameter efficiency and scaling. (These examples illustrate potential parameter efficiency; actual optimal architectures may vary.)

Synthetic families with known structure:

- **Smooth additive $f(x, y) = \exp(\sin \pi x + y^2)$:** KAS theory suggests a simple structure exists. Empirical scaling plots comparing SN test RMSE vs. parameter count against MLP scaling would be informative. SNs might achieve better approximation error scaling with respect to the number of parameters (e.g., error scaling closer to N_{params}^{-4}) compared to MLPs (which often exhibit slower rates or saturate earlier).
- **Multiplicative xy:** A known simple KAS representation exists ($xy = [(x + y)/2]^2 - [(x - y)/2]^2$). An SN like $[2, 2, 1]$ is expected to be highly efficient compared to MLPs requiring significantly more parameters for similar accuracy.
- **High-dimensional additive $f(\mathbf{x}) = \exp(\frac{1}{d}\sum \sin^2(\pi x_i))$:** For large d, the structure is predominantly additive composed with univariate functions. SNs (potentially shallow like $d \to [1] \to 1$) might significantly outperform deep, wide MLPs due to SNs ability to directly represent the univariate components and summation structure efficiently.

Special function representation: The idea of using SNs for "compressing" representations of special functions (e.g., from SciPy library) is compelling. Comparing the parameter count of a pruned, trained SN achieving a target RMSE (e.g., 10^{-3}) against an MLP trained for the same task could demonstrate practical

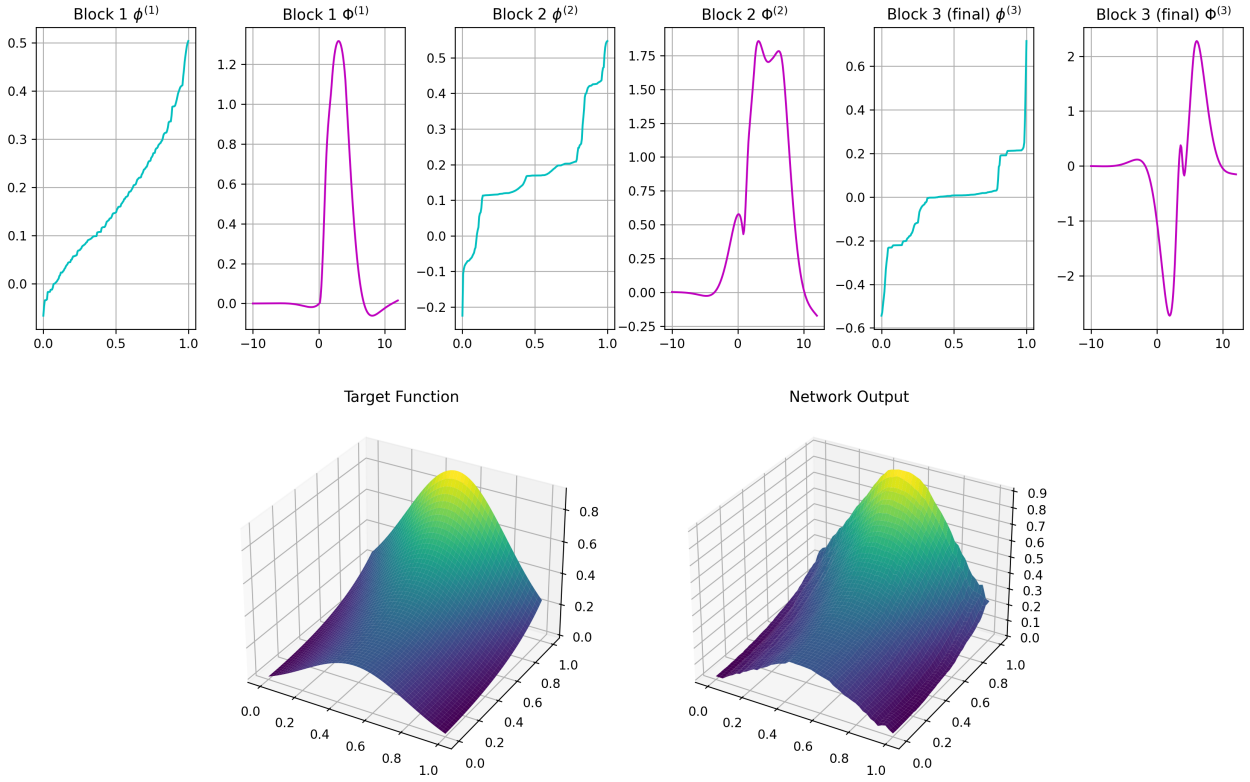


Figure 3: Visualization of a trained $2 \to [5, 8, 5] \to 1$ Sprecher Network (3 blocks) approximating the scalar 2D target function $z = f(x, y) = (\exp(\sin(\pi x) + y^2) - 1)/7$. Top row: Learned spline functions for each block — monotonic splines $\phi^{(\ell)}$ (cyan) and general splines $\Phi^{(\ell)}$ (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.

parameter savings. A table summarizing median compression factors across several special functions would quantify this advantage.

Scientific ML benchmarks (qualitative discussion):

- **Poisson PDE:** While direct MSE comparison might be premature due to SN training speed, the *parameter count* difference calculated earlier (e.g., ≈ 124 for SN vs. ≈ 690 for KAN for a similar structure) remains a significant point illustrating potential efficiency gains if comparable accuracy can eventually be reached.
- **Knot signature:** Similarly, the potential parameter reduction (e.g., ≈ 57 for SN vs. ≈ 186 for KAN) is noteworthy, even if achieving the exact same accuracy requires further tuning or potentially different optimal architectures for SNs. The ability of SNs to highlight important input features via weight magnitudes (λ) also contributes to interpretability.

8.3 Ablation study: role of the shift η

To empirically validate the importance of the internal shift parameter $\eta^{(\ell)}$ noted in Remark 2, 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 and plays a crucial role in the expressivity or optimization dynamics of multi-layer SNs.

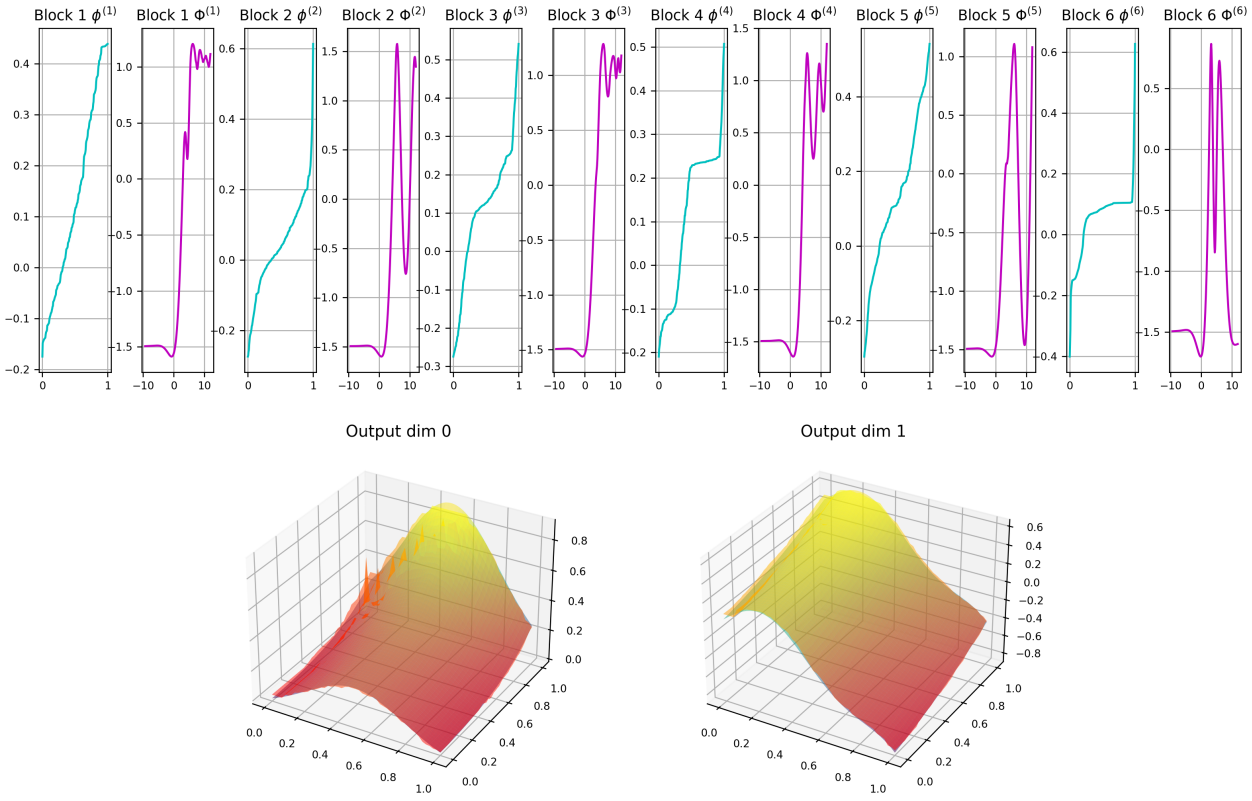


Figure 4: Visualization of a trained Sprecher Network with architecture $2 \to [20, 20, 20, 20, 20] \to 2$ (5 hidden layers, 6 blocks total), approximating a vector-valued function $f(x, y) = ((\exp(\sin(\pi x) + y^2) - 1)/7,\ \frac{1}{4}y + \frac{1}{5}y^2 - x^3 + \frac{1}{5}\sin(7x))$. Top row: Learned 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).

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]. Notable results include:

- 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) 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 Challenges and future directions

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

9.1 Implementation and training challenges

1. **Training speed and stability:** Current implementations based on standard automatic differentiation libraries and optimizers (like Adam or LBFGS) can be slow, potentially 10x or more slower than comparable MLPs per epoch, although convergence rates might differ. Training shared splines, especially highly flexible ones (many knots), can be challenging. Networks may converge to suboptimal local minima where splines become flat or overly simplistic, hindering accurate approximation. Escaping such minima might require advanced optimization techniques (e.g., second-order methods, adaptive learning rates tailored for spline parameters) or better initialization strategies.

2. **Initialization and range selection:** Initializing the splines (ϕ, Φ), shifts (η), and weights (λ) appropriately is crucial. While initializing ϕ as linear and Φ near identity often works, optimal strategies are not fully established. Determining the ideal domain/codomain for Φ a priori is difficult. Using trainable range parameters offers flexibility but adds complexity and potential optimization instability if not carefully managed (e.g., through regularization or constraints on the rate of change).

3. **Architectural sensitivity:** Choosing the network depth (L) and hidden layer widths ($d_1, \ldots, d_L$) significantly impacts performance. The optimal architecture likely depends heavily on the target function's structure and complexity. Finding the right balance between depth and width for SNs, and developing guidelines or automated architecture search methods, is an important area for future work. While parameter counts might be lower for *equivalent* structures compared to KANs, the *optimal* structures might differ.

4. **Knot count and placement:** The number of knots (G) for the shared splines is a critical hyperparameter. Too few knots limit expressivity; too many increase parameter counts and risk overfitting or optimization issues. Unlike KANs where spline complexity can vary per edge, the shared nature in SNs means G affects the entire block's nonlinearity. Adaptive knot strategies (e.g., adding knots where function curvature is high) or automated methods for selecting G based on data complexity would be highly valuable.

5. **Monotonicity enforcement:** While methods like parameterizing via cumulative sums or sorting coefficients exist, ensuring strict monotonicity robustly and efficiently during optimization without hindering learning requires careful implementation.

9.2 Directions for further research

- **Improved training strategies:** Develop robust initialization schemes (e.g., based on data statistics), adaptive optimization methods specifically considering the different parameter types (spline coeffs, η, λ), and effective regularization techniques (beyond basic flatness/sparsity) tailored to the shared-spline structure. Explore second-order optimization possibilities.

- **Theoretical analysis:**
  - Prove the universality conjectures (Conjecture 1, 2) for vector-valued and deep SNs.
  - Rigorously analyze approximation rates ($G(\varepsilon)$ scaling) and compare them theoretically with KANs and MLPs under various function smoothness assumptions.
  - Investigate the precise role of the internal shifts ($\eta^{(\ell)}q$) and the outer shifts (q) in deep SNs: why are they essential?
  - Study sample complexity and generalization properties.

- **Architectural enhancements:**
  - **Sparsity and pruning:** Develop targeted L1/entropy regularization for mixing weights (λ) and potentially node pruning strategies (based on contribution analysis) to discover compact SN architectures automatically. (Unlike for KANs, however, pruning edges in SNs leads to relatively small decreases in parameter counts. Automatic pruning of Sprecher blocks may be a more promising direction.)

  - **Alternative spline bases:** Explore higher-order B-splines or other basis functions beyond piecewise linear, potentially offering better smoothness or approximation properties.
  - **High-dimensional adaptations:** For very high-dimensional inputs where even $O(N^2)$ weights become prohibitive, investigate alternative representations for the interaction, possibly using low-rank approximations for λ or different structures for ϕ, Φ (e.g., small neural networks, tensor methods).
  - **Residual connections:** Residual connections in SNs can be more sophisticated than typical skip connections. When the input and output dimensions of a block match ($d_{\ell-1} = d_\ell$), a learnable scalar weight multiplies the input before adding it to the block output. When dimensions differ, a learnable linear projection (without bias) maps the input to the output dimension. These residual connections can significantly improve training stability when enabled. Further exploration of more sophisticated skip connection patterns, such as dense connections or learnable gating mechanisms, could enhance training stability for very deep networks.

- **Scientific discovery and interpretability:**
  - **Function dimensionality discovery:** Leverage the architecture's sensitivity to input dimension. Train SNs with varying d_{in} on unstructured data and use model selection criteria (balancing fit quality and complexity) to infer the intrinsic dimensionality of the underlying process generating the data.
  - **Symbolic integration:** Enhance integration with symbolic regression tools to automatically suggest or fit closed-form expressions for the learned $\phi^{(\ell)}$ and $\Phi^{(\ell)}$ splines after training, further boosting interpretability.
  - **Application case studies:** Apply SNs to challenging problems in scientific machine learning (e.g., developing SN-based PINNs for complex PDEs, modeling physical systems) where interpretability and potential parameter efficiency are highly valued.

- **Hardware acceleration:** Explore possibilities for custom hardware kernels that could exploit the specific computational pattern of SNs (two shared spline evaluations per block coupled with matrix-vector operations) to mitigate the current training speed limitations.

10 Conclusion

We have introduced Sprecher Networks (SNs), a trainable neural architecture directly inspired by David Sprecher's 1965 constructive proof of the Kolmogorov-Arnold theorem. By composing functional blocks that utilize shared monotonic (ϕ) and general (Φ) splines, learnable mixing weights (λ), and explicit shifts (η, q), SNs offer a distinct alternative to traditional MLPs and the more recent KANs.

SNs retain a strong theoretical connection to the classical KAS representation, inheriting universality properties in the single-layer case. Their architecture, particularly the sharing of splines within blocks, leads to a parameter scaling of $O(LN^2 + LG)$, contrasting with KANs' $O(LN^2G)$. This suggests a significant potential for parameter efficiency, especially when high spline resolution (G) is needed relative to network width (N). The explicit structure with shared, learnable splines also offers avenues for interpretability not readily available in standard MLPs or potentially even KANs (where splines are per-edge).

While promising, SNs face implementation challenges that require further research, including optimization techniques tailored to their unique structure and comprehensive empirical validation. The architecture represents a novel point in the design space of neural networks, bridging classical approximation theory with modern deep learning. Their theoretical grounding, potential parameter efficiency, and structural interpretability make them a compelling direction 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] 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.