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

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

August 1, 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:

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

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 a shallow theorem to a deep architecture. Sprecher's 1965 result is remarkable because it guarantees universal approximation with a single hidden layer (i.e., a shallow network). This mirrors the history of Multi-Layer Perceptrons, where the Universal Approximation Theorem also guaranteed the sufficiency of a single hidden layer. However, the entire deep learning revolution was built on the empirical

---

*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`

discovery that composing multiple layers, while not theoretically necessary for universality, provides vast practical benefits in terms of efficiency and learnability.

This history motivates our central research question: can the components of Sprecher's shallow, highly-structured formula be used as a new kind of building block in a *deep*, compositional architecture? We propose to investigate this by composing what we term *Sprecher blocks*, where the vector output of one block becomes the input to the next. It is crucial to emphasize that this deep, compositional structure is our own architectural proposal, inspired by the paradigm of deep learning, and is not part of Sprecher's original construction or proof of universality. The goal of this paper is not to generalize Sprecher's theorem, but to empirically evaluate whether this theorem-inspired design is a viable and efficient alternative to existing deep learning models when extended into a deep framework.

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:

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

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 6).

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** $\phi(\cdot)$**:** A non-decreasing piecewise-linear function mapping $\mathbb{R} \to [0, 1]$. While we denote the domain as $\mathbb{R}$ for generality, in practice each $\phi^{(\ell)}$ operates on a bounded domain determined dynamically by the network's structure and current parameters, as detailed in Section 7.3. This function is shared across all input-output connections within the block and its coefficients are learnable. Monotonicity is enforced during training through appropriate parameterization (see Section 7).

- **General spline** $\Phi(\cdot)$**:** A piecewise-linear function (without monotonicity constraints) whose domain and codomain are determined by the network structure and can be either fixed or made trainable. If trainable codomains are used, they can be parameterized in various ways, such as by center and radius parameters, allowing the spline to adapt its output range 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 η. This parameter controls the magnitude of the input shift $x_i + \eta q$, which depends on the output index q. While Sprecher's original construction requires $\eta > 0$, practical implementations may allow η to take any real value during training.

Concretely, we first define a single Sprecher block as an operator. Let $B^{(\ell)} : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}$ denote the ℓ-th Sprecher block with parameters $\phi^{(\ell)}, \Phi^{(\ell)}, \eta^{(\ell)}, \lambda^{(\ell)}$. Given an input vector $\mathbf{x} = (x_1, \ldots, x_{d_{\text{in}}}) \in \mathbb{R}^{d_{\text{in}}}$, the block computes its output vector $\mathbf{h} \in \mathbb{R}^{d_{\text{out}}}$ component-wise as:

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

for $q = 0, \ldots, d_{\text{out}} - 1$, where α is a scaling factor (typically set to 1 to maintain consistency with Sprecher's original construction). While $\alpha = 1$ maintains theoretical fidelity, 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 (Learnable Soft Routing):** When enabled, residual connections adapt to dimensional mismatches through a learnable routing mechanism that maintains architectural coherence while providing gradient highways. The implementation varies based on dimensional relationships:

  - **Identity** ($d_{\text{in}} = d_{\text{out}}$): Uses a single learnable scalar weight w_{res}:

  $$[B_{\text{res}}^{(\ell)}(\mathbf{x})]_q = [B^{(\ell)}(\mathbf{x})]_q + w_{\text{res}} \cdot x_q$$

  - **Pooling** ($d_{\text{in}} > d_{\text{out}}$): Each input dimension has a learnable selectivity parameter $\rho_i \in \mathbb{R}$ that determines how strongly it connects to different outputs. The routing is based on circular distances:

  $$p_i = i/d_{\text{in}}, \quad q_j = j/d_{\text{out}} \quad \text{(normalized positions)} \tag{2}$$
  $$d_{ij} = \min(|p_i - q_j|, 1 - |p_i - q_j|) \quad \text{(circular distance)} \tag{3}$$
  $$a_{ij} = \exp(-\sigma(\rho_i) \cdot s \cdot d_{ij}^2) \quad \text{(affinity)} \tag{4}$$
  $$r_{ij} = \frac{\exp(a_{ij}/\tau)}{\sum_k \exp(a_{ik}/\tau)} \quad \text{(routing weights)} \tag{5}$$

  where $\sigma(\cdot)$ is the sigmoid function, s is a selectivity scale (typically 10), and τ is a temperature parameter. The residual contribution becomes:

  $$[B_{\text{res}}^{(\ell)}(\mathbf{x})]_j = [B^{(\ell)}(\mathbf{x})]_j + \sum_{i=1}^{d_{\text{in}}} r_{ij} \cdot \sigma(\rho_i) \cdot s_{\text{input}} \cdot x_i$$

  where s_{input} is an input scaling factor (typically 2).

– **Broadcasting** ($d_{\text{in}} < d_{\text{out}}$): Each output dimension has a learnable selectivity parameter that determines which inputs it draws from:

$$r_{ji} = \frac{\exp(a_{ji}/\tau)}{\sum_k \exp(a_{jk}/\tau)} \quad \text{(routing weights for output } j) \tag{6}$$

$$[B_{\text{res}}^{(\ell)}(\mathbf{x})]_j = [B^{(\ell)}(\mathbf{x})]_j + \sum_{i=1}^{d_{\text{in}}} r_{ji} \cdot x_i \tag{7}$$

This learnable routing mechanism uses $O(\max(d_{\text{in}}, d_{\text{out}}))$ parameters while providing flexible, data-driven connections that can learn to focus on relevant input-output pairs.

- **Normalization:** Batch or layer normalization can be applied to block outputs to improve training stability, particularly in deeper networks. See Section 3.3 for details.

- **Output scaling:** The network may include learnable output scaling parameters γ (scale) and β (bias) applied to the final network output: $f_{\text{final}}(\mathbf{x}) = \gamma \cdot f(\mathbf{x}) + \beta$. These parameters can improve optimization dynamics and are typically initialized with $\gamma = 0.1$ and β set to match the mean of the target distribution when available.

Remark 1 (Learnable routing as attention-like mechanism)**.** *The learnable soft routing can be viewed as a lightweight attention mechanism specifically designed for dimensional adaptation. Unlike full attention which would require $O(d_{\text{in}} \times d_{\text{out}})$ parameters, our approach uses only $O(\max(d_{\text{in}}, d_{\text{out}}))$ selectivity parameters that modulate pre-defined circular distance relationships. The circular distance metric ensures that: (1) nearby dimensions in the normalized position space have higher default affinity, (2) the first and last dimensions are considered neighbors (wraparound), and (3) the learned selectivity can either sharpen these relationships (high selectivity) or broaden them (low selectivity). This design is motivated by the observation that in many structured problems, there exists a natural ordering or topology in the feature space that can be exploited.*

Remark 2 (Structured sparsity and architectural coherence)**.** *The learnable soft routing design can be viewed as implementing a structured sparse projection where each output connects to a weighted combination of inputs based on learned affinities. In the pooling case, inputs with higher selectivity values will concentrate their contributions to fewer outputs, while inputs with lower selectivity distribute more broadly. The unexpected effectiveness of this constrained routing suggests that for gradient flow and information propagation, the specific connection pattern may matter less than ensuring smooth, learnable paths that can adapt to the data distribution while maintaining computational efficiency.*

3.3 Normalization considerations

The unique structure of Sprecher Networks requires careful consideration when incorporating normalization techniques. While standard neural networks typically normalize individual neuron activations, the shared-spline architecture of SNs suggests a different approach: normalizing the entire vector output $\mathbf{h}^{(\ell)} \in \mathbb{R}^{d_\ell}$ of each Sprecher block as a cohesive unit.

When normalization is applied after block ℓ, the transformation takes the familiar form:

$$\tilde{\mathbf{h}}^{(\ell)} = \gamma^{(\ell)} \odot \frac{\mathbf{h}^{(\ell)} - \mu^{(\ell)}}{\sigma^{(\ell)} + \epsilon} + \beta^{(\ell)}$$

where $\gamma^{(\ell)}, \beta^{(\ell)} \in \mathbb{R}^{d_\ell}$ are learnable affine parameters. The statistics $\mu^{(\ell)}$ and $\sigma^{(\ell)}$ are computed per-dimension across the batch for batch normalization or across dimensions for layer normalization.

This approach maintains the parameter efficiency central to SNs: each normalization layer adds only $2d_\ell$ parameters, preserving the $O(LN)$ scaling. Moreover, by treating the block output as a unified representation rather than normalizing individual components, the method respects the architectural philosophy that all output dimensions arise from shared transformations through $\phi^{(\ell)}$ and $\Phi^{(\ell)}$.

Normalization can be positioned either before or after each Sprecher block, with post-block normalization often providing better gradient flow through the constrained parameter space. For networks with scalar

output, where the final block's outputs are summed, normalization on the scalar sum effectively becomes a simple affine transformation. Practitioners often find it beneficial to skip normalization for the first block, allowing the network to directly process the input features while still stabilizing deeper layers. This selective application is particularly important when theoretical domain computation assumes inputs in $[0, 1]^n$, as normalization could shift the data outside expected ranges.

Our empirical findings suggest that the combination of normalization with residual connections proves particularly effective for deeper SNs, enabling successful training of networks with many blocks while maintaining the characteristic parameter efficiency of the architecture.

3.4 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:

$$\mathbf{h}_q^{(\ell)} = \Phi^{(\ell)}\left(\sum_{i=1}^{d_{\ell-1}} \lambda_i^{(\ell)}\, \phi^{(\ell)}\left(\mathbf{h}_i^{(\ell-1)} + \eta^{(\ell)}\, q\right) + q\right), \quad q = 0, \ldots, d_\ell - 1. \tag{8}$$

Remark 3 (On the nature of composition). *Note that in this $L > 1$ composition (Eq. 8), 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. The motivation for this compositional structure comes not from Sprecher's work, but from the empirical success of the deep learning paradigm. 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

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

then the overall function is

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

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:

$$y_q = \left(T^{(L+1)}(\mathbf{h}^{(L)})\right)_q = \Phi^{(L+1)}\left(\sum_{r=0}^{d_L-1} \lambda_r^{(L+1)}\, \phi^{(L+1)}\left(\mathbf{h}_r^{(L)} + \eta^{(L+1)}\, q\right) + q\right),$$
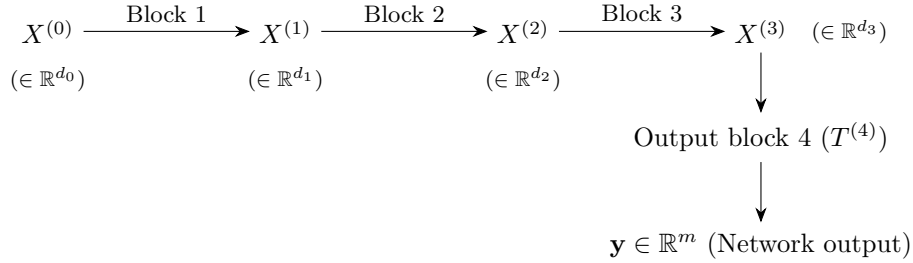
for $q = 0, \ldots, m - 1$. The network output function is then:

$$f(\mathbf{x}) = \mathbf{y} = \left(T^{(L+1)} \circ T^{(L)} \circ \cdots \circ T^{(1)} \right)(\mathbf{x}) \in \mathbb{R}^m. \tag{9}$$

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 $\mathbf{x}$.

$$X^{(0)} \xrightarrow{\text{Block 1}} X^{(1)} \xrightarrow{\text{Block 2}} X^{(2)} \xrightarrow{\text{Block 3}} X^{(3)} \quad (\in \mathbb{R}^{d_3})$$

$(\in \mathbb{R}^{d_0}) \qquad\qquad (\in \mathbb{R}^{d_1}) \qquad\qquad (\in \mathbb{R}^{d_2})$

Output block 4 $(T^{(4)})$

$\mathbf{y} \in \mathbb{R}^m$ (Network output)

Here, $X^{(\ell)} = \mathbf{h}^{(\ell)}$ denotes the output vector of the ℓ-th Sprecher block (we use both notations interchangeably for clarity in different contexts). 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.

3.5 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.

3.5.1 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:

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

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

3.5.2 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:

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

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

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

7

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:

$$f(\mathbf{x}) = \sum_{q=0}^{d_2-1} \Phi^{(2)} \left(\sum_{r=0}^{d_1-1} \lambda_r^{(2)} \, \phi^{(2)} \left(\Phi^{(1)} \left(\sum_{i=1}^{d_0} \lambda_i^{(1)} \, \phi^{(1)} \left(x_i + \eta^{(1)} \, r \right) + r \right) + \eta^{(2)} \, q \right) + q \right). \qquad (10)$$

3.5.3 Three hidden layers ($L = 3$)

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

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

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

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

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

$$f(\mathbf{x}) = \sum_{q=0}^{d_3-1} \Phi^{(3)} \left(\sum_{s=0}^{d_2-1} \lambda_s^{(3)} \, \phi^{(3)} \left(\Phi^{(2)} \left(\sum_{r=0}^{d_1-1} \lambda_r^{(2)} \, \phi^{(2)} \left(\Phi^{(1)} \left(\sum_{i=1}^{d_0} \lambda_i^{(1)} \, \phi^{(1)} \left(x_i + \eta^{(1)} \, r \right) + r \right) \right. \right. \right. \right.$$
$$\left. \left. \left. \left. + \eta^{(2)} \, s \right) + s \right) + \eta^{(3)} \, q \right) + q \right). \qquad (11)$$

Remark 4. *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 5 (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 (10), or $\eta^{(2)}s$ inside $\phi^{(2)}$ and $\eta^{(3)}q$ inside $\phi^{(3)}$ in (11). 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 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

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 $\sigma_{\text{learn}}(\cdot)$	$\sum(\text{inputs})$	Implicit in Block Formula
Parameter sharing	None (typically)	Activations? (Maybe)	None (typically)	Splines (ϕ, Φ) per block
Residual design	Matrix projection $O(N^2)$ per skip	Matrix projection $O(N^2)$ per skip	None (typically)	Learnable Soft Routing $O(N)$ per skip
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]. All architectures may optionally include normalization layers; SNs apply normalization to entire block outputs rather than individual activations. Learnable Soft Routing in SNs uses selectivity parameters to modulate circular distance-based connections, providing flexible routing while maintaining $O(N)$ parameter scaling. 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.

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' $O(LN^2G)$.[1]

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 6 (Note on weight structure)**.** *The following proposition uses matrix weights $\lambda_{i,q}^{(\ell)}$ to establish the connection between SNs and LANs. However, the architecture we propose and analyze throughout this paper uses vector weights $\lambda_i^{(\ell)}$ (following Sprecher's original formulation), where the same weights are shared across all output dimensions. This vector version represents an even more constrained special case of the*

---

[1]The $O(LN)$ term includes contributions from mixing weights (λ), normalization parameters when used, and learnable soft routing residual connections. Each residual connection contributes at most $\max(d_{\text{in}}, d_{\text{out}})$ parameters, maintaining the linear scaling in width.

LAN formulation shown below. The practical SN architecture with vector weights would require the weight matrices in the proposition to have repeated rows, i.e., $\lambda_{i,q}^{(\ell)} = \lambda_i^{(\ell)}$ for all q. This constraint is fundamental to maintaining the "true Sprecher" structure and achieving the characteristic $O(LN)$ parameter scaling.

Proposition 1. *An SN with matrix weights (cf. (9)) 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 (8). □

Remark 7 (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.
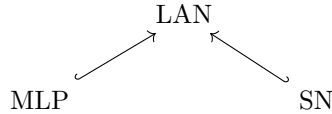

Figure 1: Diagram illustrating the dependencies between the models, in terms of learnable parameters. MLPs are LANs with fixed activation function, while SNs are LANs with a particular parameter structure (Proposition 1).

Remark 8 (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 $\eta^{(\ell)}q$. While Sprecher's theoretical construction extends ϕ appropriately, practical implementations typically compute exact theoretical bounds for all spline domains based on the network structure, ensuring that all evaluations fall within the spline's operating range through dynamic domain updates.*

5 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 ℓ contains mixing weights $\lambda^{(\ell)}$ with $O(d_{\ell-1})$ parameters, where $d_{\ell-1}$ is the input dimension to that block (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. When normalization is used, each block may have an associated normalization layer adding $O(N)$ parameters (specifically $2N$ for the affine transformation). When residual connections are used, they add at most $O(\max(d_{\ell-1}, d_\ell))$ parameters per block using the learnable soft routing approach. Specifically:

- Identity case ($d_{\ell-1} = d_\ell$): 1 scalar parameter

- Pooling case ($d_{\ell-1} > d_\ell$): $d_{\ell-1}$ selectivity parameters

- Broadcasting case ($d_{\ell-1} < d_\ell$): d_ℓ selectivity parameters

This maintains the linear scaling in width while providing significantly more flexibility than fixed routing patterns. Summing over L (or $L+1$) blocks, the total parameter count scales approximately as $O(LN + LG + L)$, where the linear dependence on N now includes mixing weights, optional normalization parameters, and residual connections.

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.

5.1 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.

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. KANs use B-splines with $G = 20$ intervals, $k = 3$ $(G + k \approx 23$ 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. SNs use piecewise linear splines with $G = 20$ intervals $(G + 1 = 21$ params/spline). Shared splines: $2 \times 21 = 42$. Mixing weights: $d_0 = 2$ (vector weights). Shift $\eta^{(1)}$: 1. Total SN params $\approx 42 + 2 + 1 = 45$.

- **Potential advantage:** For equivalent structure, theoretical reduction factor $\approx 15x$ (45 vs 690). Actual performance requires empirical validation.

Pruned KAN $[17, 1, 14]$, with B-splines using $G = 3$ intervals, $k = 3$ $(G + k \approx 6$ 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. Using piecewise linear splines with $G = 3$ intervals: $2(L+1) \times (G+1) = 4 \times 4 = 16$. Mixing weights: $d_0 + d_1 = 17 + 1 = 18$ (vector weights). Shifts $\eta^{(1)}, \eta^{(2)}$: 2. Total SN params $\approx 16 + 18 + 2 = 36$.

- **Potential advantage:** Theoretical reduction factor $\approx 5.2x$ (36 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.

6 Theoretical aspects and open questions

6.1 Relation to Sprecher (1965) and universality

As shown in Section 3.5.1, 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.

Remark 9 (Piecewise linear splines in practice). *While the following theoretical analysis considers general degree-k splines for completeness, the Sprecher Network architecture as implemented specifically requires piecewise linear splines ($k = 1$). This constraint is not merely an implementation choice but a fundamental requirement for the efficient domain computation algorithm described in Section 7.3.*

For piecewise linear splines, the extrema occur only at knot points, enabling exact output range computation in $O(G)$ time by simply evaluating the spline at its knots. Higher-degree splines would require finding critical points through differentiation and root-finding in each interval, making the domain propagation algorithm computationally prohibitive and potentially numerically unstable.

The following analysis with general k is provided to illustrate how approximation rates would scale with spline degree in principle, but readers should understand that $k = 1$ is architecturally required for the domain management system that is central to SNs' practical implementation.

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. Note that while this theorem assumes degree-k splines for generality, practical implementations often use piecewise linear splines ($k = 1$) for computational efficiency. 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

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

13

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

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

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. $\qquad\qquad\square$

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:

$$E_\ell \le \sum_{j=1}^{\ell} \left(\prod_{m=j+1}^{\ell} L_{T^{(m)}} \right) K_{T^{(j)}} \varepsilon_j \tag{12}$$

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 $\mathbf{x} \in [0,1]^n$:

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

Substituting the induction hypothesis completes the proof. $\qquad\square$

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:

- $\varepsilon_j = \max\left\{ \left(\dfrac{2B_{j-1}+2\eta^{(j)}(d_j-1)}{G_\phi^{(j)}} \right)^{k+1}, \left(\dfrac{2R_j}{G_\Phi^{(j)}} \right)^{k+1} \right\}$

- $R_j = \Lambda d_{j-1}\|\phi^{(j)}\|_{L^\infty} + d_j$

- $K_{T^{(j)}} = \dfrac{L_{\Phi^{(j)}}\Lambda d_{j-1}\|\phi^{(j)(k+1)}\|_{L^\infty} + \|\Phi^{(j)(k+1)}\|_{L^\infty}}{(k+1)!}$

- $L_{T^{(m)}} = L_{\Phi^{(m)}}\Lambda d_{m-1}L_{\phi^{(m)}}$ *(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 (12).

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. $\qquad\square$

Remark 10 (Monotonic spline approximation). *The approximation rates in Lemma 1 assume we can achieve standard spline approximation rates while maintaining monotonicity for $\phi^{(\ell)}$. While the theoretical analysis considers general degree-k splines, practical implementations typically use piecewise linear splines (k = 1) with a log-space parameterization that maintains monotonicity by construction. For piecewise linear splines approximating smooth functions, the approximation rate becomes $O(G^{-2})$.*

Remark 11 (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.*

6.2 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 $\eta^{(\ell)}$.

7 Implementation considerations

7.1 Trainable splines

For practical implementations, piecewise-linear splines are used for both $\phi^{(\ell)}$ and $\Phi^{(\ell)}$. While these are only C^0 continuous (not smooth in the classical sense), they offer excellent computational efficiency and sufficient expressiveness for function approximation. Each piecewise linear spline with G intervals requires $G + 1$ parameters (the values at the $G + 1$ knot locations). The piecewise linear structure also simplifies domain analysis, as the extrema of such splines occur at knot locations, enabling exact computation of output ranges during forward propagation. 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 codomain is typically fixed to $[0, 1]$. Strict monotonicity (strictly increasing) is enforced during training. This can be achieved through various parameterization strategies, such as using a transformation that guarantees positive increments between successive spline values. One effective approach parameterizes the differences between knot values in log-space and applies a positive activation function (e.g., softplus), ensuring that the spline remains strictly increasing throughout training without requiring explicit constraints or projection steps. Specifically, the monotonic spline coefficients can be computed as the cumulative sum of positive increments, normalized to the [0,1] range. 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 $\Phi^{(\ell)}$ operates on a domain determined by the network structure. Both the domain and codomain can be fixed or made trainable via parameters (e.g., center and radius). Trainable ranges add flexibility but also complexity to the optimization. Monotonicity is not required for $\Phi^{(\ell)}$.

7.2 Shifts, weights, and optimization

Each block includes the learnable scalar shift $\eta^{(\ell)}$ and the learnable mixing weight vector $\lambda^{(\ell)} \in \mathbb{R}^{d_{\ell-1}}$. While Sprecher's original construction requires the shift parameter to be positive ($\eta > 0$), practical implementations can relax this constraint. As shown in Lemma 3, the theoretical domain computation handles both positive and negative values of $\eta^{(\ell)}$, allowing it to be trained as an unconstrained parameter initialized to a small positive value.

Learnable routing parameters. When residual connections are enabled, each block may include additional routing selectivity parameters $\rho^{(\ell)}$. These are initialized near zero (using a small random initialization with configurable standard deviation) to begin with approximately uniform routing. During training, these parameters learn to create focused or distributed routing patterns based on the data. The sigmoid activation ensures selectivity values remain bounded, while the exponential affinity function creates smooth gradients for optimization. Key hyperparameters include the selectivity scale s (default: 10.0), temperature τ (default: 1.0), and input scale s_{input} (default: 2.0), which can be tuned based on the specific application.

All learnable parameters (spline coefficients, $\eta^{(\ell)}$, $\lambda^{(\ell)}$, routing selectivity $\rho^{(\ell)}$, 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.

7.3 Theoretical domain computation

One advantage of the structured Sprecher architecture is the ability to compute exact theoretical bounds for all intermediate values when using piecewise-linear splines, as the extrema of such functions occur at knot locations. Throughout this analysis, we assume inputs lie in $[0, 1]^n$, though the methodology extends naturally to any bounded domain. This assumption enables principled initialization and can aid in debugging and analysis.

Lemma 3 (Domain propagation through Sprecher blocks). *Consider a Sprecher block where each input component lies in the same range $[a, b]$, i.e., inputs are in $[a, b]^{d_{in}}$, shift parameter η, weights λ_i, and output dimension d_{out}. Let us distinguish between a spline's domain (the interval of valid inputs) and its codomain (the interval of possible outputs). Then:*

1. *The domain required for ϕ to handle all possible inputs is:*

$$\mathcal{D}_\phi = \begin{cases} [a, b + \eta(d_{out} - 1)] & \text{if } \eta \geq 0 \\ [a + \eta(d_{out} - 1), b] & \text{if } \eta < 0 \end{cases}$$

2. *Given that ϕ maps to $[0, 1]$, the domain required for Φ is:*

$$\left[\sum_{i:\lambda_i < 0} \lambda_i, \sum_{i:\lambda_i > 0} \lambda_i + \alpha(d_{out} - 1) \right]$$

   where α is the scaling factor for the q term (typically $\alpha = 1$).

3. *If Φ has trainable codomain parameters (c_c, c_r) defining codomain $[c_c - c_r, c_c + c_r]$, then the actual output range is determined by examining the spline values at its knots. For piecewise linear splines, since the function is linear between knots, the extrema occur at knot points. Thus, the actual output range is $[\min_j \Phi(t_j), \max_j \Phi(t_j)]$ where t_j are the knot locations within the domain $\mathcal{D}_\Phi$. This ensures accurate range propagation even for highly oscillatory splines.*

4. *When residual connections are present, the output range must be adjusted by the residual contribution. For a scalar residual weight w, if $w \geq 0$, the residual adds $[wa, wb]$ to the range where $[a, b]$ is the input range. If $w < 0$, it adds $[wb, wa]$ (note the reversal). For learnable soft routing:*

   - ***Pooling***: *The contribution is bounded by $[-s_{input}|a|, s_{input}|b|]$ where s_{input} is the input scaling factor, as the routing weights are normalized and selectivity-scaled inputs are bounded.*

   - ***Broadcasting***: *Each output's range is expanded by at most $[-\max(|a|, |b|), \max(|a|, |b|)]$ as the routing weights sum to 1.*

Proof. The proof proceeds by interval arithmetic, tracking the range of possible values through each operation.

For (1), the argument to ϕ is $x_i + \eta q$ where $x_i \in [a, b]$ and $q \in \{0, 1, \ldots, d_{out} - 1\}$. When $\eta \geq 0$, the minimum occurs at $q = 0$ (giving a) and maximum at $q = d_{out} - 1$ (giving $b + \eta(d_{out} - 1)$). When $\eta < 0$, these are reversed.

For (2), the argument to Φ is $\sum_i \lambda_i \phi(x_i + \eta q) + \alpha q$. Since ϕ outputs in $[0, 1]$, the weighted sum achieves its minimum when positive λ_i multiply 0 and negative λ_i multiply 1, yielding $\sum_{i:\lambda_i < 0} \lambda_i$. The maximum adds the contribution from positive weights and the maximum αq term.

For (4), the learnable soft routing contributions follow from analyzing the normalized routing weights and bounded selectivity values. $\quad\square$

This lemma enables a forward propagation algorithm for computing all spline domains throughout the network. Crucially, the algorithm's efficiency relies on the property of piecewise-linear splines (formalized in Lemma 3, part 3), which allows for the exact computation of output ranges by simply finding the extrema of the spline's values at its knots. In practical implementations, this computation is performed periodically during training as the parameters $\lambda^{(\ell)}$ and $\eta^{(\ell)}$ evolve, ensuring that splines operate on appropriate domains: where interval addition accounts for the appropriate sign handling based on weight values.

Remark 12 (Dynamic spline updates and function preservation). *A critical challenge in training Sprecher Networks is that the domains of the splines $\phi^{(\ell)}$ and $\Phi^{(\ell)}$ evolve as the parameters $\eta^{(\ell)}$ and $\lambda^{(\ell)}$ are updated. This evolution could be mitigated by clamping the spline inputs to fixed (or slowly trainable) intervals. However, this would break the direct correspondence with Sprecher's formula. His construction, for a function on $[0, 1]^n$, explicitly relies on evaluating the inner function ϕ on shifted inputs $x_i + \eta q$. Since each $x_i \in [0, 1]$, these shifted values necessarily extend beyond this initial interval, requiring ϕ to operate on a wider domain.*

Algorithm 1 Forward domain propagation

1: **Input:** Network parameters $\{\lambda^{(\ell)}, \eta^{(\ell)}\}_{\ell=1}^{L}$, input domain $[0,1]^n$
2: **Output:** Domains and ranges for all splines
3: Initialize current range $\mathcal{R}_0 \leftarrow [0,1]^n$
4: **for** each block $\ell = 1, \ldots, L$ **do**
5: $\quad$ $\mathcal{D}_\phi^{(\ell)} \leftarrow$ apply Lemma 3 part (1) to $\mathcal{R}_{\ell-1}$
6: $\quad$ $\mathcal{D}_\Phi^{(\ell)} \leftarrow$ apply Lemma 3 part (2)
7: $\quad$ $\mathcal{R}_\ell \leftarrow [\min_j \Phi^{(\ell)}(t_j), \max_j \Phi^{(\ell)}(t_j)]$ where $t_j \in \mathcal{D}_\Phi^{(\ell)}$ are knots
8: $\quad$ **if** block ℓ has residual connections **then**
9: $\quad\quad$ Adjust $\mathcal{R}_\ell$ according to Lemma 3 part (4)
10: $\quad$ **end if**
11: **end for**

Therefore, to maintain theoretical fidelity, the splines must adapt their domains dynamically. A naive approach would be to simply perform a "horizontal rescaling" by compressing or stretching the spline's knot positions to fit the new domain while keeping the knot coefficients fixed. However, this method has a severe drawback: it alters the function's derivatives at every point, which can disrupt gradient flow and amplify oscillations, leading to numerical instability.

*The shortcomings of naive rescaling motivate a more sophisticated approach: a **function-preserving resampling** strategy that is applied selectively to preserve learned representations while adapting to evolving domains:*

General Splines ($\Phi^{(\ell)}$): *For the general splines, which learn arbitrary and often complex functional forms, preserving the learned shape is paramount. When the domain of a $\Phi^{(\ell)}$ spline changes, the procedure is to first establish a new set of knot locations, typically spaced uniformly across the new computed domain. The original spline is then treated as a continuous function, and its evaluation at these new knot locations yields the updated knot coefficients. This process effectively "resamples" the learned shape onto the new domain.*

Monotonic Splines ($\phi^{(\ell)}$): *For the monotonic splines, whose purpose is to provide an increasing map to $[0,1]$, this complex resampling is not required. Their learnable parameters define relative increments between knots, not an arbitrary shape. Therefore, a straightforward update of the knot positions to the new theoretical domain is sufficient and computationally efficient.*

This targeted, dual-strategy approach is theoretically sounder than naive rescaling, as it avoids the most direct sources of information loss and instability from domain changes. However, the fundamental difficulty of optimizing highly oscillatory splines within dynamically shifting domains remains a key challenge, and this approach should be seen as a mitigation strategy rather than a complete solution to training stability.

One particularly elegant application of this domain computation is the initialization of $\Phi^{(\ell)}$ splines. By computing each $\Phi^{(\ell)}$'s theoretical domain before training begins, we can initialize both its domain and codomain parameters to match this computed range, making each $\Phi^{(\ell)}$ start as an approximate identity function on its natural operating range. This provides a principled initialization strategy: by starting each $\Phi^{(\ell)}$ as an approximate identity function on its computed domain, we ensure that the initial network performs a series of near-identity transformations. However, the effectiveness of this initialization strategy remains an empirical question, as identity initialization may not always lead to better optimization dynamics than alternatives such as variance-preserving initialization or carefully chosen random initialization.

Remark 13 (Practical benefits)**.** *The ability to compute exact domains through interval arithmetic provides several practical benefits: (i) it enables theoretically grounded initialization without arbitrary hyperparameters, (ii) it can help diagnose training issues by detecting when values fall outside expected ranges, and (iii) it ensures numerical stability by keeping spline evaluations within their intended operating regions. These benefits distinguish Sprecher Networks from architectures with less structured internal representations.*

Remark 14 (Conservative domain estimates and oscillatory splines)**.** *The interval arithmetic used for domain computation provides worst-case bounds that may be overly conservative in practice. When domains*

become unnecessarily wide, splines must distribute their knots sparsely, often leading to the oscillatory behavior observed in learned $\Phi^{(\ell)}$ functions. This suggests that tighter domain estimation or regularization of weight magnitudes during early training could improve both optimization dynamics and the interpretability of learned splines. Future work might explore adaptive knot placement strategies that concentrate knots in regions of high data density while maintaining the fixed parameter count that is central to SNs' theoretical appeal.

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 $1 \rightarrow [W] \rightarrow 1$ (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 $2 \rightarrow [5, 8, 5] \rightarrow 1$ (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 $2 \rightarrow [20, \dots, 20] \rightarrow 2$ (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 15 (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.*

The domain management strategy also plays a crucial role in these oscillations. The current implementation's use of horizontal rescaling to adapt spline domains, while theoretically sound, may exacerbate oscillatory behavior by compressing existing oscillations when domains shrink. This suggests that the interplay between domain computation, parameter constraints, and spline smoothness is more subtle than initially apparent and deserves further theoretical and empirical investigation.

8.2 Systematic benchmarks

While comprehensive benchmarking remains future work, the architectural differences suggest SNs might excel on functions with compositional structure aligned with their shift-based design (e.g., additive functions, functions with known Sprecher representations) while potentially struggling with functions requiring independent processing of different output dimensions. Systematic comparison with MLPs and KANs across diverse function classes would clarify these trade-offs.
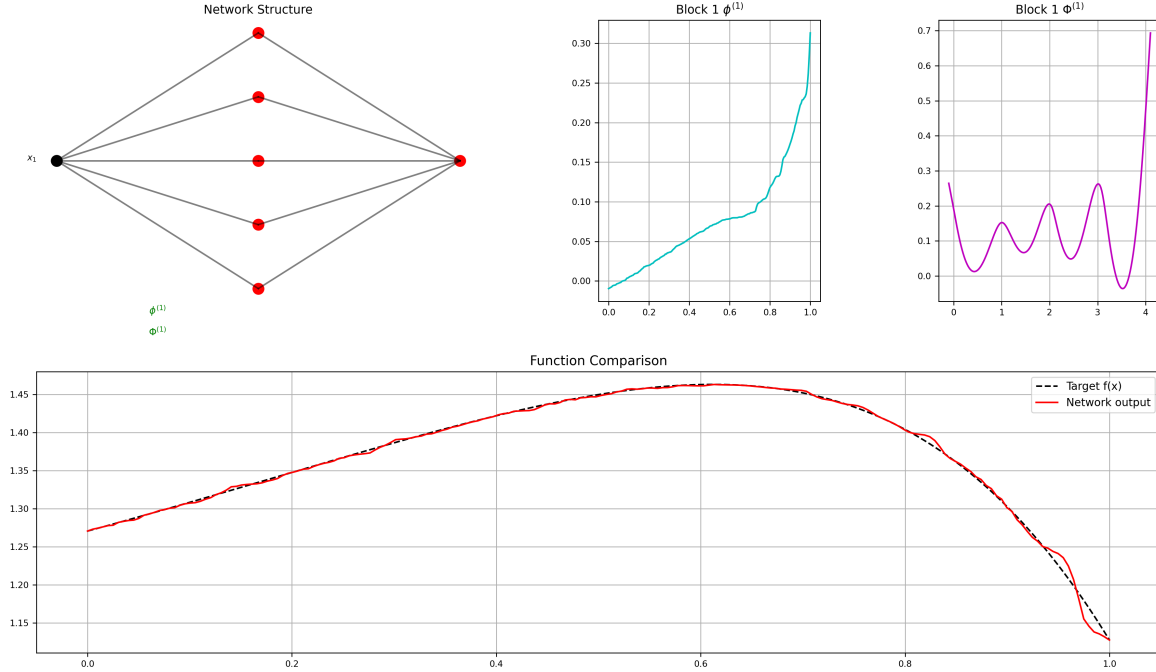
Figure 2: Visualization of a trained Sprecher Network with architecture $1 \to [5] \to 1$ (one block) trained on data sampled from $f(x) = \sum_{q=0}^{4} \Phi(\lambda_{q+1}\phi(x + \eta q) + q)$, where the ground truth functions were $\phi(x) = (e^x - 1)/(e - 1)$ and $\Phi(x) = \sin x$, with shift $\eta = 1/10$ 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.

8.3 Ablation study: role of the shift η

To empirically validate the importance of the internal shift parameter $\eta^{(\ell)}$ noted in Remark 5, 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.

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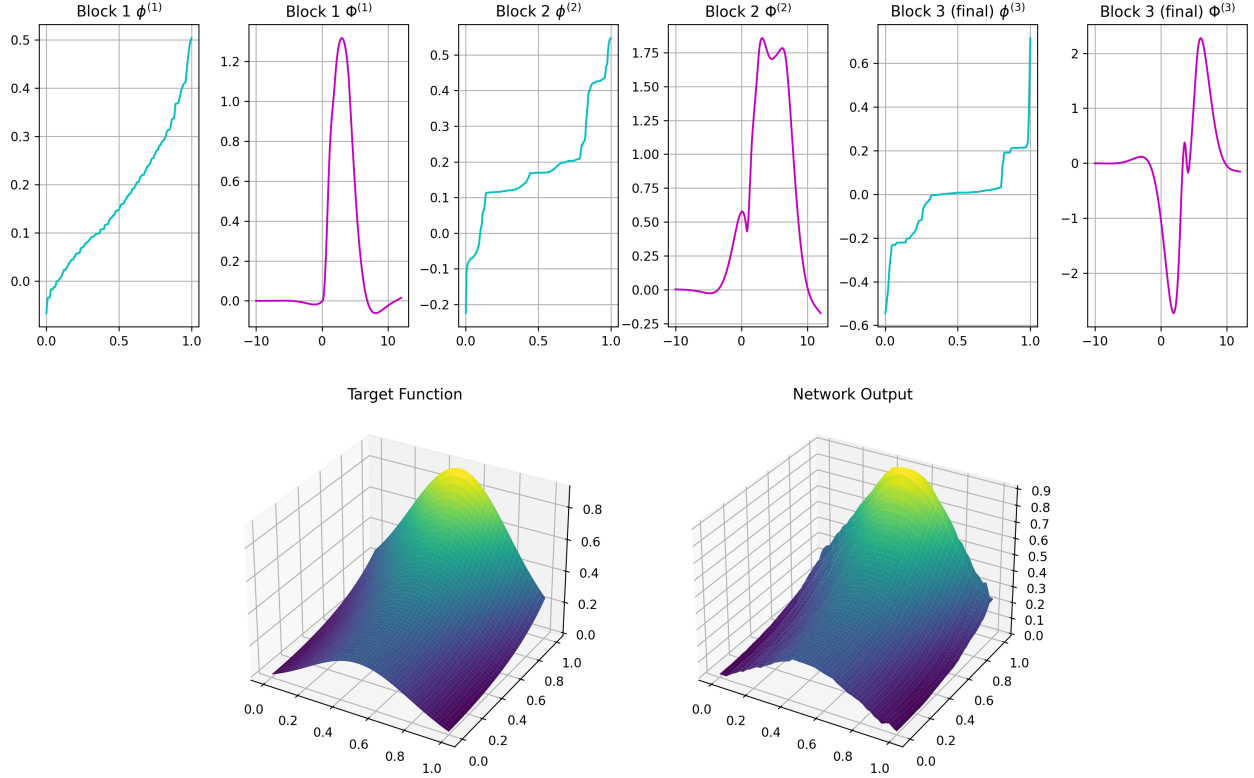
Figure 3: Visualization of a trained $2 \rightarrow [5, 8, 5] \rightarrow 1$ Sprecher Network (3 blocks) approximating the scalar 2D target function $z = f(x, y) = (\exp(\sin(\pi x) + y^2) - 1)/7$. Top row: Learned spline functions for each block — monotonic splines $\phi^{(\ell)}$ (cyan) and general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

9 Limitations and future work

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 (Conjectures 1 and 2).

The Sprecher block design imposes strong constraints: forcing all feature interactions through two shared splines and using weight vectors rather than matrices heavily restricts expressive power compared to standard architectures. This represents a fundamental trade-off between parameter efficiency and flexibility that may limit performance on functions not aligned with this compositional structure.

Current implementations face practical challenges including significantly slower training than MLPs (empirically 3-10× slower) and optimization difficulties manifesting as oscillatory splines. These oscillations appear to compensate for the reduced parameter flexibility but complicate optimization. The development of adaptive knot placement strategies that concentrate resolution where data lives while maintaining fixed parameter counts could address this issue, as could specialized training algorithms tailored to the shared-spline structure.

Beyond addressing these immediate challenges, SNs open several intriguing research directions. The weight-sharing structure raises fundamental theoretical questions about potential equivariance properties. Just as CNNs exhibit translation equivariance through spatial weight sharing, SNs' sharing across output dimensions may satisfy a related form of equivariance, possibly connected to permutations of output indices or transformations of the function domain. Understanding such properties could provide deeper insight into when and why the architecture succeeds.
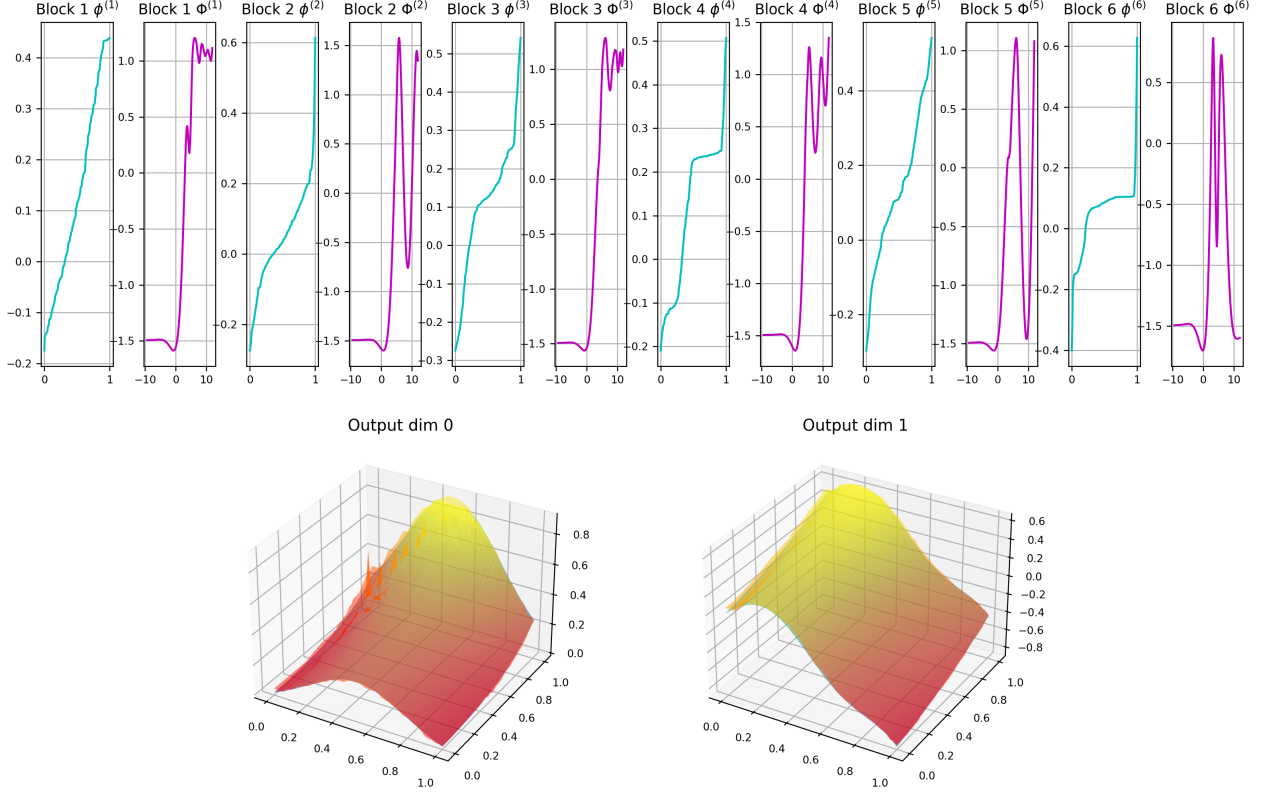
Figure 4: Visualization of a trained Sprecher Network with architecture $2 \to [20, 20, 20, 20, 20] \to 2$ (5 hidden layers, 6 blocks total), approximating a vector-valued function $f(x,y) = ((\exp(\sin(\pi x) + y^2) - 1)/7,\ \frac{1}{4}y + \frac{1}{5}y^2 - x^3 + \frac{1}{5}\sin(7x))$. Top row: Learned 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).

The architecture's properties suggest unique opportunities in scientific machine learning where both interpretability and parameter efficiency are valued. One particularly compelling possibility is dimensionality discovery: the architecture's sensitive dependence on input dimension could enable inference of the intrinsic dimensionality of data-generating processes. By training SNs with varying d_{in} and using model selection criteria that balance fit quality against complexity, researchers might determine the true number of relevant variables in systems where this is unknown, a valuable capability in many scientific domains. Furthermore, the explicit structure of learned splines could enable integration with symbolic regression tools to extract closed-form expressions for the learned $\phi^{(\ell)}$ and $\Phi^{(\ell)}$ functions, potentially revealing underlying mathematical relationships in data.

Architectural enhancements also merit exploration. Unlike KANs where edge pruning can significantly reduce parameters, SNs' vector structure suggests different optimization strategies. Automatic pruning of entire Sprecher blocks based on their contribution to network output could yield more compact architectures. For very high-dimensional problems where even $O(N)$ scaling becomes prohibitive, hybrid approaches using low-rank approximations for $\lambda^{(\ell)}$ or replacing splines with small neural sub-networks in certain blocks could maintain efficiency while improving expressivity. These directions highlight how theorem-inspired architectures can open new research avenues beyond simply providing alternative implementations of existing methods.

10 Conclusion

We have introduced Sprecher Networks (SNs), a trainable architecture built by re-imagining the components of David Sprecher's 1965 constructive proof as the building blocks for a modern deep learning model. 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 contribution is demonstrating that Sprecher's shallow, highly-structured formula can be extended into an effective deep architecture with remarkable parameter efficiency: $O(LN + LG)$ compared to MLPs' $O(LN^2)$ or KANs' $O(LN^2G)$. This efficiency comes from adhering to Sprecher's use of weight vectors rather than matrices, representing a strong architectural constraint that may serve as either beneficial inductive bias or limitation depending on the problem domain.

Our initial demonstrations show SNs can successfully learn diverse functions and achieve competitive performance on tasks like MNIST classification, with the added benefit of interpretable spline visualizations. However, practical challenges in training speed and theoretical gaps regarding deep network universality remain open questions. Whether SNs prove broadly useful or remain a fascinating special case, they demonstrate the value of mining classical mathematical results for architectural inspiration in modern 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.