Received 31 July 2020; revised 14 October 2020; accepted 10 November 2020. Date of publication 19 November 2020; date of current version 21 December 2020. The review of this article was arranged by Associate Editor Christopher Rozell.

Digital Object Identifier 10.1109/OJSP.2020.3039379

Learning Activation Functions in Deep (Spline) Neural Networks

PAKSHAL BOHRA (Graduate Student Member, IEEE), JOAQUIM CAMPOS, HARSHIT GUPTA, SHAYAN AZIZNEJAD, AND MICHAEL UNSER (Fellow, IEEE)

Biomedical Imaging Group, École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland

CORRESPONDING AUTHOR: PAKSHAL BOHRA (e-mail: pakshal.bohra@epfl.ch)

(Pakshal Bohra and Joaquim Campos are co-first authors.)

This work was supported in part by the Swiss National Science Foundation under Grant 200020_184646 / 1 and in part by the European Research Council (ERC) under Grant 692726-GlobalBioIm.

ABSTRACT We develop an efficient computational solution to train deep neural networks (DNN) with free-form activation functions. To make the problem well-posed, we augment the cost functional of the DNN by adding an appropriate shape regularization: the sum of the second-order total-variations of the trainable nonlinearities. The representer theorem for DNNs tells us that the optimal activation functions are adaptive piecewise-linear splines, which allows us to recast the problem as a parametric optimization. The challenging point is that the corresponding basis functions (ReLUs) are poorly conditioned and that the determination of their number and positioning is also part of the problem. We circumvent the difficulty by using an equivalent B-spline basis to encode the activation functions and by expressing the regularization as an ℓ_1-penalty. This results in the specification of parametric activation function modules that can be implemented and optimized efficiently on standard development platforms. We present experimental results that demonstrate the benefit of our approach.

INDEX TERMS Activation functions, B-splines, deep learning, regularization, sparsity.

I. INTRODUCTION

During the past decade, deep neural networks (DNNs) have evolved into a major player for machine learning. They have been found to outperform the traditional techniques of statistical learning [1] (*e.g.,* kernel methods, support-vector machines, random forests) in many real-world applications that include image classification [2], speech recognition [3], image segmentation [4], and medical imaging [5].

The basic principle behind DNNs is to construct powerful learning architectures via the composition of simple basic modules; that is, linear (or affine) transformations and pointwise nonlinearities [6]. The qualifier "deep" refers to the depth (or number of layers) of such a composition which is typically much larger than one. Formally, a DNN is a map $\mathbf{f}_{\boldsymbol{\Theta}} : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ that admits a factorized representation of the form

$$\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{x}) : \mathbf{W}_L \circ \cdots \circ \boldsymbol{\sigma}_\ell \circ \mathbf{W}_\ell \circ \cdots \circ \boldsymbol{\sigma}_1 \circ \mathbf{W}_1(\mathbf{x}), \qquad (1)$$

where L is the depth of the neural net and $\boldsymbol{\Theta}$ is a list of parameters that collects all adjustable quantities. Specifically, a given layer ℓ of the network is characterized by

1) a linear transformation $\mathbb{R}^{N_{\ell-1}} \to \mathbb{R}^{N_\ell} : \mathbf{x} \mapsto \mathbf{W}_\ell \mathbf{x}$, where $\mathbf{W}_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ is a matrix of weights, and
2) the pointwise responses of its neurons

$$\boldsymbol{\sigma}_\ell(\mathbf{x}) = \left(\sigma_{\ell,1}(x_1), \ldots, \sigma_{\ell,N_\ell}(x_{N_\ell})\right),$$

where the scalar map $\sigma_{\ell,n} : \mathbb{R} \to \mathbb{R}$ is the activation function of the neuron indexed by (ℓ, n).

In essence, $\mathbf{W}_\ell$ encodes the strength of the neural connections from the previous layer, while $\boldsymbol{\sigma}_\ell$ represents the (parallel) responses of the N_ℓ neurons at layer ℓ. In the conventional setup, the response of the individual neurons is fixed and takes the form

$$\sigma_{\ell,n}(x) = \sigma(x - b_{\ell,n}), \qquad (2)$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is a common activation function—typically, a sigmoid or a rectified linear unit (ReLU)—and $b_{\ell,n} \in \mathbb{R}$ is an adjustable bias [7]. In summary, the parameters $\boldsymbol{\Theta}$ associated with the DNN in (1) are composed of the linear weights of $\mathbf{W}_\ell$ and the biases $\mathbf{b}_\ell \in \mathbb{R}^{N_\ell}$, $\ell = 1, \ldots, L$.

Let $\mathbf{x} \in \mathbb{R}^{N_0}$ denote an input of the network and $\mathbf{y} \in \mathbb{R}^{N_L}$ be the corresponding desired output. Given a series of reference (or training) data $(\mathbf{x}_m, \mathbf{y}_m)$, $m = 1, \ldots, M$, the training task is to adjust the free parameters $\boldsymbol{\Theta}$ of the DNN such that $\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{x}_m) \approx \mathbf{y}_m$ without overfitting. In practice, this training problem is formulated as the (non-convex) optimization of a suitable cost functional with respect to $\boldsymbol{\Theta}$ [6]. Due to the layered structure of the network, such an optimization is achieved effectively by steepest descent with the help of the celebrated backpropagation algorithm [8]. Another important ingredient is the use of stochastic gradient techniques which operate via the subdivision of the data in batches. Since the training of DNNs with respect to the linear weights is key to the success of the approach, there is a rich literature devoted to this subject.

The topic of this article deviates from the standard paradigm in the sense that it explores the option of adapting the responses of the individual neurons in an attempt to further improve the performance of such systems. In other words, instead of assigning a single bias parameter to each neuron as in (2), we are investigating the possibility of redesigning or adjusting the activation functions $\sigma_{\ell,n} : \mathbb{R} \to \mathbb{R}$ on a neuron-by-neuron basis. While the typical way in which this can be achieved is via the introduction of a suitable parametrization, which may be linear or nonlinear, we shall see that one can also formulate the problem in a functional framework with the help of a suitable regularization [9]. At any rate, the main point is that this augmented form of training results in a more difficult optimization problem and that it calls for more powerful algorithms.

The purpose of this work is to unify the parametric and functional approaches by representing the neural activation functions in terms of B-spline basis functions. This is possible as long as we restrict ourselves to the class of deep spline neural networks,[1] which cover the complete family of continuous piecewise-linear (CPWL) mappings [10]–[12]. Our approach builds on the intimate connection between ReLU networks and splines, which has been observed by a number of authors [9], [13]–[16]. The spline interpretation is actually present at two levels: (i) the fact that such DNNs are describable as hierarchical splines and (ii) the property that the global response is CPWL, which allows one to interpret them as piecewise perceptrons [14]. While the local linear (perceptron-like) behavior of deep spline networks is both reassuring and enlightening, the part that is less obvious is the global continuity of the response, which ensures that the linear pieces (facets of polytopes) are seamlessly joined together.

[1]The denomination "deep spline neural network" refers to a DNN whose activation functions are linear splines, which includes ReLUs.

The article is organized as follows: We start with a review of prior work on neural design in Section II. In Section III, we explain the main theoretical results on deep spline networks; namely, the CPWL property and the fact that they are optimal with respect to $\mathrm{TV}^{(2)}$ regularization. We then introduce our parametrization and optimization framework in Section IV and present experimental results in Section V.

II. PRIOR WORK ON NEURAL ACTIVATION FUNCTIONS

We now briefly review the prior works on the design of neural activation functions, which can be broadly classified into three categories.

A. INSPIRATION FROM NEUROPHYSIOLOGY

The traditional activation function for neural networks inspired by neurophysiology is a saturating sigmoid whose sharpness can be tuned for best performance [17]. Since splines have the ability to encode arbitrary functions, they can be used to generate a much richer variety of activation functions, which can then be optimized for best performance. Relevant examples of parametric activation function models for traditional neural networks include B-spline receptive fields [18], Catmull-Rom cubic splines [19], [20], and smooth piecewise polynomials [21].

B. LINK WITH ITERATIVE SOFT-THRESHOLDING ALGORITHMS

One can make an interesting connection between neural networks and sparse-encoding techniques [22], [23] by considering the unrolled version of an iterative soft-thresholding algorithm (ISTA) [24], [25]. This connection suggests that the activation function fulfills the role of the nonlinearity in classical ISTA [26], [27]. Incidentally, the canonical nonlinearity associated with ℓ_1 minimization is an antisymmetric linear spline, which can be expressed as a linear combination of two ReLUs. In recent years, researchers have considered more general parametric nonlinearities whose weights are learned during training. Such models involve linear combinations of Gaussian radial-basis functions [28] and cubic B-splines [29], [30].

C. ReLU VARIATIONS

While many (fixed) activation functions σ in (2) have been considered in the literature, the preferred choice that has emerged over the years is the rectified linear unit $\mathrm{ReLU}(x) = (x)_+ \triangleq \max(0, x)$ [31]. In particular, it has been observed that ReLUs facilitate training [7]. Two ReLU variants, by order of improving performance, are "leaky ReLU" [32], in which the vanishing part of the response is replaced by one with a fixed nonzero linear slope, and "parametric ReLU" (PReLU) [33], where the linear slope is learnable. Also related to ReLU is Agostinelli *et al.*'s model of adaptive piecewise-linear (APL) units [34]. It results in an activation function that is a linear spline with a small fixed number of knots and has been found to outperform plain ReLU activation functions. Another instance is [35], where piecewise-linear units with learnable parameters are used as activation functions.

III. THEORETICAL JUSTIFICATION OF SPLINE ACTIVATION FUNCTIONS

Many of the state-of-the-art DNNs rely on ReLU activation functions or some variant thereof. Beside the issue of practical efficiency, a key feature of ReLU networks is that they result in a global continuous and piecewise-linear (CPWL) input-output relation. This is a fundamental property that generalizes to a wider class of spline activation functions and that also ensures that deep ReLU networks have universal approximation properties [36]–[38].

A. DEEP NEURAL NETS AS HIGH-DIMENSIONAL SPLINES

A polynomial spline of degree 1 is a one-dimensional function that is continuous and piecewise-linear. In fact, the simplest nontrivial example of polynomial spline of degree 1 is $x \mapsto (x - b_k)_+ = \mathrm{ReLU}(x - b_k)$, which is made up of two linear pieces separated by a single knot at b_k. The concept is generalizable to higher dimensions [39], [40].

Definition 1 (CPWL function): A function $f : \mathbb{R}^{N_0} \to \mathbb{R}$ is continuous piecewise-linear if

1) it is continuous $\mathbb{R}^{N_0} \to \mathbb{R}$;
2) its domain $\mathbb{R}^{N_0} = \bigcup_{k=1}^{K} P_k$ can be partitioned into a finite set of non-overlapping polytopes P_k over which it is affine.

Likewise, a vector-valued function $\mathbf{f} = (f_1, \dots, f_N) : \mathbb{R}^{N_0} \to \mathbb{R}^N$ is CPWL if each of the component functions f_n is CPWL.

What is truly remarkable with CPWL functions is that they remain CPWL through the operations that typically occur in a deep neural network [11], [12]. Specifically,

1) any linear combination of CPWL functions is CPWL;
2) the composition of any two CPWL functions is CPWL;
3) the max or min of two CPWL functions is CPWL.

Since the functions $\mathbf{W}_\ell$ in (1) are trivially CPWL, the resulting DNN is CPWL whenever the pointwise nonlinearities $\boldsymbol{\sigma}_\ell$ are CPWL, for instance when they are piecewice-linear splines, which is indeed the case for deep ReLU networks. It is therefore perfectly legitimate to interpret deep ReLU networks—and, by extension, deep spline networks—as multidimensional splines of polynomial degree 1.

B. VARIATIONAL OPTIMALITY OF DEEP SPLINE NETWORKS

Lesser known is the property that the CPWL behavior can also be enforced indirectly through the use of an appropriate regularization [9]. To that end, one simply augments the cost functional that is used to train the network by an additive second-order total-variation regularization term for each adjustable activation function.

In our framework, we consider deep neural networks $\mathbf{f}_{\mathrm{deep}} : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ composed of L layers with the generic feedforward architecture described by (1). The linear transformation in layer ℓ, represented by the matrix $\mathbf{W}_\ell : \mathbb{R}^{N_{\ell-1}} \to \mathbb{R}^{N_\ell}$, is associated with some free (adjustable) parameters $\boldsymbol{\theta}_\ell \in \mathbb{R}^{N_{\mathrm{lin},\ell}}$. In order to specify the latter, one has to distinguish between two configurations. When the layer is fully connected, $\boldsymbol{\theta}_\ell$ is the vectorized version of $\mathbf{W}_\ell$, which amounts to a total of $N_{\mathrm{lin},\ell} = N_{\ell-1} \times N_\ell$ tunable weights. The other important configuration is that of a convolutional layer where $\boldsymbol{\theta}_\ell$ contains much fewer convolution filter weights than $N_{\ell-1} \times N_\ell$. Similarly, to share nonlinearities across neurons, we specify each nonlinear mapping $\boldsymbol{\sigma}_\ell : \mathbb{R}^{N_\ell} \to \mathbb{R}^{N_\ell}$ by the vector $\mathbf{s}_\ell = (s_{\ell,1}, \dots, s_{\ell,N_{\mathrm{nonlin},\ell}})$ of adjustable activation functions $s_{\ell,n} : \mathbb{R} \to \mathbb{R}$, where $N_{\mathrm{nonlin},\ell} \in \mathbb{N}$ denotes the number of unique activation functions used in layer ℓ. For example, in a fully connected layer, it can be advantageous to use an independent activation function for each neuron. In this case, $N_{\mathrm{nonlin},\ell} = N_\ell$ and $s_{\ell,n} = \sigma_{\ell,n}$. By contrast, for convolutional layers, it is natural to use a single activation function per feature map, so that $N_{\mathrm{nonlin},\ell}$ will typically match the number of channels. When the same nonlinearity is shared across all channels, one has that $N_{\mathrm{nonlin},\ell} = 1$.

With this extended notation, the training of the network is formulated as the functional optimization problem

$$\min_{\substack{\boldsymbol{\theta}_\ell \in \mathbb{R}^{N_{\mathrm{lin},\ell}} \\ \mathbf{s}_\ell \in \mathrm{BV}^{(2)}(\mathbb{R})^{N_{\mathrm{nonlin},\ell}}}} \sum_{m=1}^{M} \mathrm{E}(\mathbf{f}_{\mathrm{deep}}(\mathbf{x}_m), \mathbf{y}_m) + \sum_{\ell=1}^{L} \mu_\ell \|\boldsymbol{\theta}_\ell\|_2^2 + \sum_{\ell=1}^{L-1} \lambda_\ell \mathrm{TV}^{(2)}(\mathbf{s}_\ell), \tag{3}$$

where $E : \mathbb{R}^{N_L} \times \mathbb{R}^{N_L} \to \mathbb{R}+$ is an arbitrary proper convex function and $\mathrm{TV}^{(2)}(\mathbf{s}) = \mathrm{TV}^{(2)}(s_1, \dots, s_N) = \sum_{n=1}^{N} \mathrm{TV}^{(2)}(s_n)$, where

$$\mathrm{TV}^{(2)}(s_n) = \left\| \mathrm{D}^2 s_n \right\|_{\mathcal{M}} = \sup_{\varphi \in \mathcal{S}(\mathbb{R}):\ \|\varphi\|_\infty \le 1} \langle s_n, \frac{\mathrm{d}^2\varphi}{\mathrm{d}x^2} \rangle \tag{4}$$

is the second-order total variation of the component function $s_n : \mathbb{R} \to \mathbb{R}$. Let us remark that the two first terms in (3) are the standard criteria used to train deep neural networks. The first (data loss) quantifies the goodness of fit, while the second (the so-called weight decay) favors solutions with a smaller amplitude of the linear weights $\boldsymbol{\theta}_\ell$. The novel element here is the additional optimization over the individual neuronal activation functions $\mathbf{s}_\ell$, which is made possible because of the inclusion of the third term: the sum of the second-order total variations of the trainable nonlinearities. Since this regularization only penalizes the second derivative of the activation function, it favors simple solutions—preferably linear or with "sparse" second derivatives—while ensuring that the activations be differentiable almost everywhere, which is essential for the backpropagation algorithm. For further explanation on the regularization functional $\mathrm{TV}^{(2)}$ and the definition of the search space $\mathrm{BV}^{(2)}(\mathbb{R})$, we refer to Appendix A.

Unser's representer theorem for DNNs states that (3) admits a global minimizer (deep spline network) with neuronal activation functions of form

$$x \mapsto s_{\ell,n}(x) = b_{0,\ell,n} + b_{1,\ell,n}x + \sum_{k=1}^{K_{\ell,n}} a_{k,\ell,n}(x - \tau_{k,\ell,n})_+ \quad (5)$$

with $K_{\ell,n} \leq (M-2)$ and $\mathrm{TV}^{(2)}(s_{\ell,n}) = \sum_{k=1}^{K_{\ell,n}} |a_{k,\ell,n}| = \|\mathbf{a}_{\ell,n}\|_1$. Thus, the optimal activation functions are *adaptive* piecewise-linear splines. Specifically, every nonlinearity has a parametric description that is given by (5). It is characterized by its number $K = K_{\ell,n}$ of knots, the knot locations $\tau_1, \ldots, \tau_K$, and the linear weights $\mathbf{b} \in \mathbb{R}^2$, $\mathbf{a} \in \mathbb{R}^K$, where we have dropped the network indices (ℓ, n) for simplicity. While Characterization (5) is elegant, it does not tell one how to determine the underlying parameters. We thus now face a more challenging optimization problem. The main complication is the allocation of knots—the determination of $K_{\ell,n}$ and the locations $\tau_{k,\ell,n}$ on a neuron-by-neuron basis—which is now also part of the problem.

Let us mention that we can also handle the case where the nonlinear mapping is shared across the layers, so that $\boldsymbol{\sigma}_1 = \cdots = \boldsymbol{\sigma}_L = \boldsymbol{\sigma}$ and $\boldsymbol{\sigma}$ is specified by a vector $\mathbf{s} = (s_1, \ldots, s_{N_{\text{nonlin}}})$ of adjustable scalar maps. Here, the training problem is formulated as

$$\min_{\substack{\boldsymbol{\theta}_\ell \in \mathbb{R}^{N_{\text{lin},\ell}} \\ \mathbf{s} \in \mathrm{BV}^{(2)}(\mathbb{R})^{N_{\text{nonlin}}}}} \sum_{m=1}^{M} \mathrm{E}(\mathbf{f}_{\text{deep}}(\mathbf{x}_m), \mathbf{y}_m) + \lambda \mathrm{TV}^{(2)}(\mathbf{s}) + \sum_{\ell=1}^{L} \mu_\ell \|\boldsymbol{\theta}_\ell\|_2^2. \quad (6)$$

By adapting Unser's representer theorem, we can show that the optimal shared activation functions have the same form as in (5).

Remarkably, the parametric form that results from the functional minimization of (3) is compatible with the model proposed by Agostinelli *et al.* [34]. They represent the activation functions as $s_{\ell,n}(x) = h_{\ell,n}(x - b_{\ell,n})$, where $h_{\ell,n}$ is an APL unit of the form

$$x \mapsto h_{\ell,n}(x) = (x)_+ + \sum_{k=1}^{K} a_{k,\ell,n}(-x + \tau_{k,\ell,n})_+. \quad (7)$$

Here, the number K of knots is fixed beforehand; the bias $b_{\ell,n}$, weights $a_{k,\ell,n}$, and knot locations $\tau_{k,\ell,n}$ are learnable parameters. While (7) bears a close resemblance to (5), there are a few key differences that we highlight here.

1) The justification of APL units in [34] is empirical while the spline parametrization of (5) is based on a global functional optimization.
2) The APL units involve a fixed ReLU positioned at 0, and so, unlike (5), they cannot reproduce all affine functions of the form $b_0 + b_1 x$.
3) The number of spline knots in APL units is fixed (and is the same for all neurons), whereas it is adaptive in our approach. In fact, the determination of $K_{\ell,n}$ is part of the optimization problem that we consider.
4) The ReLU weights of the APL units are either not constrained, or slightly regularized through some empirical ℓ_2-norm weight decay. By contrast, in our approach, the theory dictates the use of a sparsity-promoting ℓ_1-regularization. In fact, as we shall see in Section IV, the ℓ_1-norm regularization is of great practical significance as it allows us to control $K_{\ell,n}$ by removing unnecessary knots.

IV. OPTIMIZATION OF ACTIVATION FUNCTIONS

A. CONVEX PROXY FOR SHALLOW NETWORKS

The major difficulty in optimizing the DNN with respect to the spline parameters in (5) is that the number $K = K_{\ell,n}$ of knots is unknown and that the activation model is nonlinear with respect to the knot locations $\tau_k = \tau_{k,\ell,n}$. Our workaround is to place a fixed but highly redundant set of knots on a uniform grid with a step size T. We then rely on the sparsifying effect of ℓ_1-minimization to nullify the coefficients of $\mathbf{a} = (a_k)$ that are not needed. This amounts to representing the spline activation functions by

$$\sigma(x) = b_0 + b_1 x + \sum_{k=k_{\min}}^{k_{\max}} a_k (x - kT)_+, \quad (8)$$

with $\mathrm{TV}^{(2)}(\sigma) = \|\mathbf{a}\|_1$. The consideration of the linear model (8), thereafter referred to as "gridded ReLU," gives rise to a classical ℓ_1-optimization problem that can be handled by most neural-network software frameworks. In the case of a shallow network with $L = 1$, it even results in a convex problem that is reminiscent of the LASSO [41]. We also note that (8) can be made arbitrarily close to (5) by taking T sufficiently small. While the solution $\mathbf{a}$ is expected to be sparse, with few active knots, the downside of the approach is that the underlying representation is cumbersome and badly conditioned due to the exploding behavior of the basis functions $(\cdot - kT)_+$ at infinity.

B. FROM ReLUs TO B-SPLINES

While the direct connection with ℓ_1-minimization in (8) is very attractive, the less favorable aspect of the model is that its computational cost is proportional to the underlying number of ReLUs (or spline knots); that is, $K = (k_{\max} - k_{\min} + 1)$, which can be arbitrarily large depending on the value of T. Here, we propose a way to bypass this limitation by switching to another equivalent but maximally localized basis: the B-splines. Our model takes the form

$$\sigma(x) = \sum_{k=k_{\min}-1}^{k_{\max}+1} c_k \varphi_k\left(\frac{x}{T}\right), \quad (9)$$

which involves triangular-shaped basis functions that are rescaled versions of B-splines defined on an integer grid. As illustrated in Figure 1, the central bases for $k = (k_{\min} + 1)$ to $(k_{\max} - 1)$ are shifted replicates of the compactly supported linear B-spline

$$\varphi_k(x) = \beta^1(x - k), \text{ for } k_{\min} < k < k_{\max}, \quad (10)$$

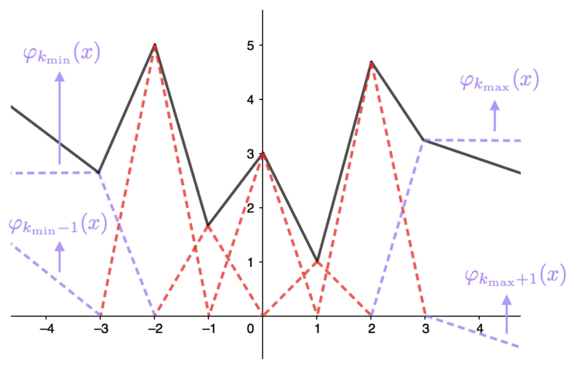


FIGURE 1. Decomposition of a deep spline activation function (solid line) in terms of B-spline basis functions (dashed lines), as expressed by (9) with $T = 1$. The basis is composed of $(K - 2)$ triangular functions, which are compactly supported and shifted replicates of each other, plus 4 one-sided outside functions. The key property is that the evaluation of $\sigma(x)$ for any fixed $x \in \mathbb{R}$ involves no more than two basis functions.

where

$$\beta^1(x) = (x+1)_+ - 2(x)_+ + (x-1)_+ = \begin{cases} 1 - |x|, & x \in [-1, 1] \\ 0, & \text{otherwise.} \end{cases} \quad (11)$$

The four remaining boundary basis functions are one-sided splines that allow the activation function defined in (9) to exhibit a linear behavior at both ends, for $x < k_{\min}T$ as well as for $x > k_{\max}T$. Specifically, we have that

$$\varphi_{k_{\min}-1}(x) = (-x + k_{\min})_+ = \begin{cases} k_{\min} - x, & x < k_{\min} \\ 0, & \text{otherwise} \end{cases} \quad (12)$$

$$\varphi_{k_{\min}}(x) = (-x + k_{\min} + 1)_+ - (-x + k_{\min})_+ = \begin{cases} 1, & x \leq k_{\min} \\ 1 - (x - k_{\min}), & x \in (k_{\min}, k_{\min} + 1) \\ 0, & x \geq k_{\min} + 1 \end{cases} \quad (13)$$

$$\varphi_{k_{\max}}(x) = (x - k_{\max} + 1)_+ - (x - k_{\max})_+ = \begin{cases} 0, & x \leq k_{\max} - 1 \\ x - k_{\max} + 1, & x \in (k_{\max} - 1, k_{\max}) \\ 1, & x \geq k_{\max} \end{cases} \quad (14)$$

$$\varphi_{k_{\max}+1}(x) = (x - k_{\max})_+ = \begin{cases} 0, & x \leq k_{\max} \\ x - k_{\max}, & x > k_{\max}. \end{cases} \quad (15)$$

The B-spline model defined in (9) has the same knots as those of the gridded ReLU representation given by (8). It also has the same number of degrees of freedom; namely, $K + 2 = (k_{\max} + 1) - (k_{\min} - 1) + 1$. By using the property that the φ_k can all be expanded in terms of integer shifts of ReLUs (see the central term of (10)–(15)), we can show that the two sets of basis functions span the same subspace. In doing so, we obtain a formula for the retrieval of the a_k and, hence, the $\mathrm{TV}^{(2)}(\sigma)$—in terms of the second-order difference of the c_k (see Appendix B). While the gridded ReLU and B-spline models (8) and (9) are mathematically equivalent, the advantage of (9) is that there are at most two active basis functions at any given point $x = x_0$, independently of the step size T. This has important implications for the efficiency and scalability of both the evaluation of the DNN at a given point $\mathbf{x}_m$ and the computation of its gradient with respect to c_k (as opposed to a_k in the equivalent ReLU representation). Details of our implementation of the B-spline model are given in Appendix B.

V. EXPERIMENTAL RESULTS

In this section, we illustrate the capabilities of the proposed learning framework. Our main intent is to assess the benefit of optimizing the activation functions and to demonstrate the following claims:

1) The use of learned activation functions tends to improve the testing performance.
2) More complex activation functions can allow for simpler/smaller networks.
3) Learning with gridded ReLUs yields good performance for small values of K. However, the time and memory required for learning explodes as K grows.
4) The B-spline configuration is easy to train and is scalable in time and memory as K grows. Hence, it has the ability to learn more complex activation functions, which then typically also translates into better performance.

Further, we investigate the effect of the regularization parameter λ on the number of active knots in the learned spline activation functions and the performance of the neural network.

We consider both classification and signal-recovery (deconvolution) problems to highlight the versatility of our approach. The code (in PyTorch) is available on GitHub.[2]

A. CLASSIFICATION

1) AREA CLASSIFICATION

First, we discuss a simple two-class classification example with input dimension $N_0 = 2$. It allows us to obtain a better understanding of our learning scheme and to illustrate our claims visually.

Setup: The task is to classify points in the two-dimensional space $[-1, 1] \times [-1, 1]$ as lying inside or outside an S shape (see Figure 2a). Mathematically, this region is represented by the binary function $f : [-1, 1] \times [-1, 1] \mapsto \{0, 1\}$ given by

$$f(x_1, x_2) = \begin{cases} 1, & |x_1 - g(x_2)| \leq 0.3 \text{ and } |x_2| < 0.8, \\ 0, & \text{otherwise,} \end{cases} \quad (16)$$

where $g(x) = 0.4 \sin(-5x)$. We generate training and validation datasets with $M = 1{,}500$ data points each. The

[2][Online]. Available: https://github.com/joaquimcampos/DeepSplines

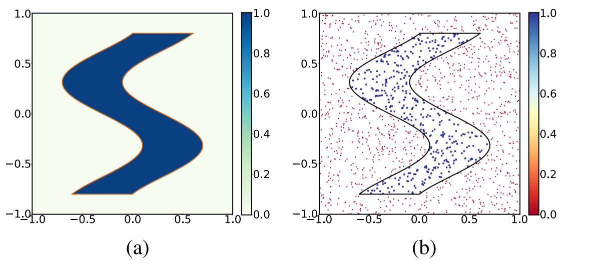


FIGURE 2. Ground truth and training dataset.

coordinates $\mathbf{x}_m = (x_{1,m}, x_{2,m})$ of the data points are sampled from a uniform distribution on $[-1, 1] \times [-1, 1]$ and the labels y_m are assigned according to (16). The training dataset is shown in Figure 2b.

We tackle this problem using a fully connected network with N_h hidden layers, which takes a 2D input $\mathbf{x} = (x_1, x_2)$ and outputs a real value $\hat{f}(\mathbf{x}) \in [0, 1]$. The number of neurons in each hidden layer is W; thus, the layer descriptor $(N_0, \dots, N_L)$ of the network is of the form $(2, W, \dots, W, 1)$. In the B-spline network, spline activation functions with $K = 19$ knots on a grid of size $T = 0.1$ are used as nonlinearities after each linear step, except the final one which involves a fixed sigmoid activation function. In the adaptive piecewise-linear unit (APLU) network, the nonlinearities take the form (7) with the number of adjustable knots set to $K = 19$. We compare the performance of our, ReLU, PReLU, and APLU networks on a test dataset that consists of 40 000 points that lie on a 2D grid of width 0.01×0.01 in $[-1, 1] \times [-1, 1]$. To evaluate the performance of these networks on a dataset, the output values $\hat{f}$ are quantized into predictions

$$\hat{f}_{\text{pred}}(\mathbf{x}) = \begin{cases} 1, & \hat{f}(\mathbf{x}) > 0.5 \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

The classification accuracy is computed as

$$\text{accuracy } (\%) = \frac{\#\text{ correct predictions}}{\#\text{ total predictions}} \times 100. \tag{18}$$

The binary cross-entropy loss is given by

$$\mathcal{L}(\boldsymbol{\Theta}) = \frac{1}{M}\sum_{m=1}^{M}((-y_m)\log(\hat{f}(\mathbf{x}_m)) - (1 - y_m)\log(1 - \hat{f}(\mathbf{x}_m))), \tag{19}$$

where $\boldsymbol{\Theta}$ represents the parameters of the network. This loss is chosen for the training process. In all the networks, the weights are initialized using Xavier's initialization [42]. For the B-spline network, half of the spline activation functions are initialized with σ_{abs} and the other half with σ_{soft}, where

$$\sigma_{\text{abs}}(x) = \begin{cases} -x, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{20}$$

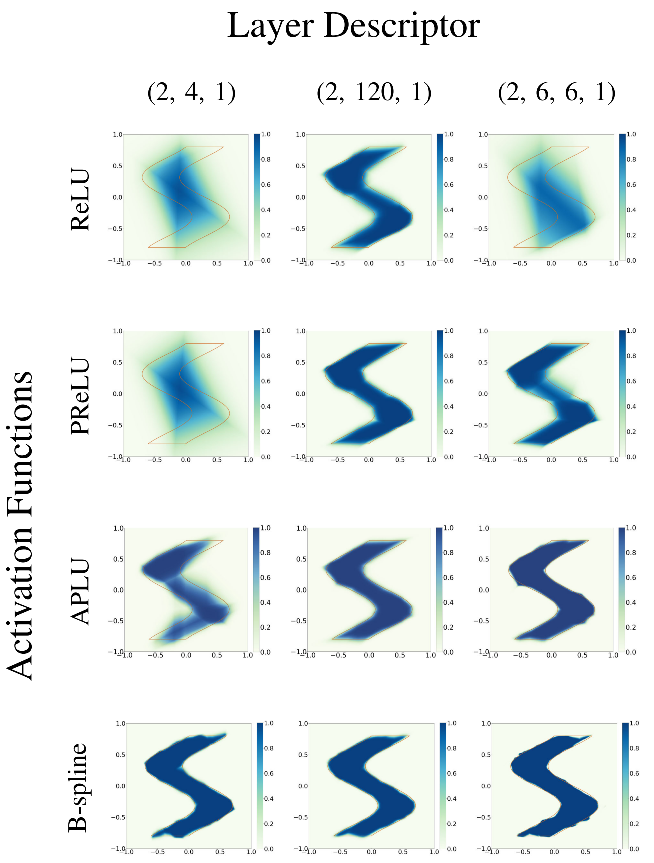


FIGURE 3. Learned probability maps for the area-classification problem.

$$\sigma_{\text{soft}}(x) = \begin{cases} x + \frac{1}{2}, & x \leq -\frac{1}{2} \\ 0, & x \in (-\frac{1}{2}, \frac{1}{2}) \\ x - \frac{1}{2}, & x \geq \frac{1}{2}. \end{cases} \tag{21}$$

This initialization is based on the fact that any function can be represented as the sum of an even and an odd function. In the APLU network, the ReLU weights $a_{k,\ell,n}$ and knot locations $\tau_{k,\ell,n}$ are initialized by randomly sampling them from zero-mean Gaussian distributions with standard deviations 0.1 and 1, respectively. The loss function is minimized over a total of 500 epochs using the ADAM optimizer [43]. The initial learning rate, set to 10^{-3}, is decreased by a factor of 10 at the epochs 440 and 480. A small batch size of 10 is helpful to avoid local minima.

Comparison With ReLU, PReLU, and APLU Networks: We compare in Figure 3 and Table 1 the performance of the ReLU, PReLU, B-spline, and APLU networks for three different architectures. For the B-spline networks, the optimal values of μ_ℓ and λ_ℓ, in terms of the performance for the validation dataset, are found using the method described in Appendix C. The weight decays for the ReLU, PReLU, and APLU networks are tuned with the help of a grid search. In the APLU network, an ℓ_2-norm penalty with scaling factor 10^{-3} is also applied to the activation function parameters $(a_{k,\ell,n}, \tau_{k,\ell,n})$. With these optimal hyperparameters, the networks are then

TABLE 1 Number of Parameters and Classification Error Rate

	Architecture	N_{param}	Error rate (%)
ReLU	(2,4,1)	17	17.02
	(2,120,1)	481	2.59
	(2,6,6,1)	67	15.39
PReLU	(2,4,1)	21	17.00
	(2,120,1)	601	1.87
	(2,6,6,1)	79	2.89
APLU	(2,4,1)	169	5.15
	(2,120,1)	5041	1.64
	(2,6,6,1)	523	1.42
B-spline	(2,4,1)	68	1.66
	(2,120,1)	822	1.40
	(2,6,6,1)	171	1.60

retrained 9 times independently. The median performance (over these 9 runs) for the test dataset is reported in Figure 3 and Table 1.

In the interest of fairness, in Table 1 we also mention the number of parameters associated with the networks. A fully connected network with N_h hidden layers has $3\,W + (N_h - 1)W^2$ linear weights and 1 bias parameter for the fixed sigmoid activation function. The network also has some additional parameters that depend on the choice of the activation function. The ReLU networks have N_hW biases while, in addition to these biases, the PReLU networks have N_hW learnable parameters that represent the linear slopes of the PReLU activation functions in $\mathbb{R}^-$. In the B-spline networks, the number of additional parameters (per activation function) is equal to the number of active knots in the learned linear-spline nonlinearity plus the 2 coefficients that determine its linear (null-space) component. Lastly, the APLU networks have $(2\,K + 1)$ additional parameters per activation function, where the number of adjustable knots K was set to 19 beforehand.

For the simplest architecture (2,4,1), we observe that the B-spline and APLU networks outperform the ReLU and PReLU models which lack capacity and perform rather poorly. This demonstrates that the learning of activation functions improves the accuracy; more so, if the activation function has reasonably many learnable parameters.

Remarkably, the simplest B-spline network outperforms the ReLU and PReLU networks with richer architectures despite having fewer parameters. This is because it is capable of learning more complex activation functions. This, in turn, translates into an overall map that is more faithful to the gold standard—the ideal S shape. This suggests that, instead of making the architecture of the network more complex, for example by including more neurons in the initial layers and/or adding more layers, one can increase the accuracy by relying on more sophisticated, learnable nonlinearities.

The results of Table 1 also illustrate the advantages of our learning scheme over the APL units. For the architecture (2,4,1), the B-spline network yields a better accuracy than the APLU network even though it has fewer parameters. One possible explanation is that the APL units, which have a fixed number of knots, face difficulties in optimizing their knot locations, whereas the adaptive B-splines bypass this problem with the help of a grid and sparsity-promoting ℓ_1-regularization. Another possible reason could be the ill-conditioned nature of expansion (7), in the sense that a small perturbation of one ReLU coefficient has a nonlocal effect on the activation function, which makes the optimization task more challenging. For the other two richer architectures, we get similar performances for the APLU and B-spline networks. However, the B-spline networks require fewer knots.

Effect of the Regularization Parameter λ: We consider now a B-spline network with layer descriptor (2, 4, 1) for the area-classification task. The weight decay is fixed as $\mu_1 = \mu_2 = 10^{-4}$ and λ is varied in the interval $[10^{-10}, 10^2]$. For each value of λ, 10 independent models are trained on the training dataset. The median number of total active knots[3] and the classification error (on the test dataset) of the corresponding model are shown as functions of λ in Figure 4.

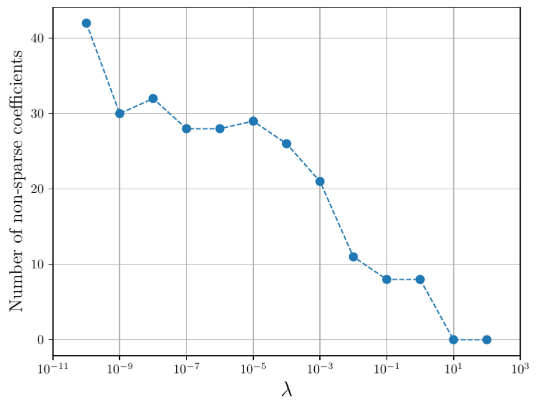
The number of active knots decreases (or, equivalently, the sparsity of the learned activation functions increases) as λ increases, which means that the hyperparameter λ controls the complexity of the network. The performance of the network remains (nearly) constant, up to a critical value of λ, after which it begins to deteriorate. This is crucial since it suggests that, by carefully tuning λ, we can obtain simpler networks that still perform well.

2) CIFAR-10 AND CIFAR-100

Now, we look at the application of the proposed learning scheme to the classification of standard datasets such as CIFAR [44]. We consider two network architectures—the network-in-network [45] (NIN) and a deep residual network [46] (ResNet32) for the CIFAR-10 and CIFAR-100 classification tasks. Each dataset consists of 50,000 training images and 10,000 test images of size (32×32).

First, we compare the performance of the B-spline, ReLU, and APLU networks. We then also demonstrate the advantages of our B-spline solution over its gridded ReLU counterpart. In the B-spline networks (NIN and ResNet), we use spline nonlinearities with $K = 49$ knots on a grid of size $T = 0.16$. We rely on one activation function per output channel for the convolutional layers and one spline activation function per output unit for the fully connected layers. For the APLU

[3]The number of total active knots is the sum of the number of active knots or ReLUs in each learned activation function in the network.



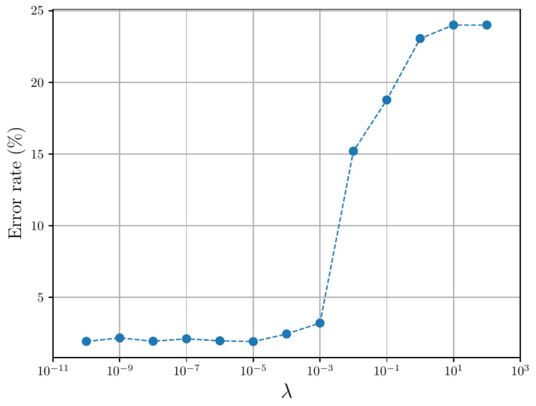


FIGURE 4. Effect of λ on the number of active knots and on the classification error.

networks[4] (NIN and ResNet), we set the number of adjustable knots to $K = 1$, with one APL activation function per output unit for the convolutional layers as well as the fully connected layers. All networks include a softmax unit in the final layer and are trained by minimizing the categorical cross-entropy loss.

For each dataset, 5,000 samples are reserved for validation during training, while the remaining 45,000 samples are augmented as in [46]. The weights in the NINs are initialized by random sampling from a Gaussian distribution with zero mean and a standard deviation of 0.05. The weights in the ResNets are initialized using He's recipe [33]. The B-spline activation functions are initialized as leaky ReLUs while the APL units are initialized in the same manner as in the area-classification experiment. For the B-spline NIN, B-spline ResNet, and APLU ResNet, the parameters of the activation functions are updated using the ADAM optimizer [43] with an initial learning rate of 10^{-3}. The remaining network parameters are updated using an SGD optimizer with an initial learning rate of 10^{-1}. For the APLU NIN, an SGD optimizer with an initial learning rate of 10^{-1} is used to update all the learnable parameters. The NINs are trained for 320 epochs

TABLE 2 NIN Error Rates on CIFAR-10 and CIFAR-100

Activation function	CIFAR-10	CIFAR-100
ReLU	8.78%	32.44%
APLU	8.71%	31.74%
B-spline	8.29%	30.43%

TABLE 3 ResNet Error Rates on CIFAR-10 and CIFAR-100

Activation function	CIFAR-10	CIFAR-100
ReLU	6.31%	29.02%
APLU	6.45%	28.85%
B-spline	6.02%	28.24%

with a batch size of 128. The learning rate is decreased by a factor of 10 in epochs 80, 160, and 240. The ResNets are trained for 300 epochs with a batch size of 128 while the learning rate is divided by 10 in epochs 150, 225, and 262, following the training scheme in [47].

Comparison with ReLU and APLU Networks: For the ReLU networks (NIN and ResNet), we deploy a grid search to optimize the weight decays in terms of the performance on the validation dataset. For the B-spline and APLU networks, we use the same weight decays as those found for the corresponding ReLU networks, and we perform grid searches to find the optimal values of λ and the ℓ_2-norm penalty scaling factor. We then use the optimal hyperparameters and retrain the networks N_T times independently on the complete training datasets, with 50,000 samples. We set $N_T = 5$ for the NINs and $N_T = 9$ for the ResNets. Finally, we compute the error rates over the test datasets. The median test errors are reported in Table 2 and Table 3. We see that the B-spline networks outperform the ReLU and APLU networks here as well. Surprisingly, the APLU ResNet is slightly inferior to the ReLU ResNet for the CIFAR-10 dataset. It turns out that, for residual networks with APL units, a similar observation has been made in [48].

B-Splines vs. Gridded ReLUs vs. APLUs: In this experiment,[5] we record the memory consumption and computation time (per epoch) for the B-spline, gridded ReLU, and APLU ResNets.

As we see in Table 4, the time/memory consumption during forward and backward propagation for gridded ReLUs and APLUs explodes with the number of knots. This is because the point evaluation of an activation function requires a summation over all contributing ReLUs, which results in a time complexity of $O(K)$. Moreover, the corresponding intermediate values need to be stored for backpropagation.

[4]The reported configurations are the ones that were found to give the best performance.

[5]This experiment was run on a TITAN X (Pascal) GPU with 12196 MB of memory.

TABLE 4 **B-Splines *vs.* Gridded ReLUs *vs.* APLUs**

Architecture, Nb. coefficients	Memory (megabytes)	Time per epoch (seconds)
B-splines, $K = 9$	1132	44.92
B-splines, $K = 29$	1133	41.89
B-splines, $K = 499$	1299	41.19
Gridded ReLUs, $K = 9$	3313	49.86
Gridded ReLUs, $K = 29$	9616	81.21
APLUs, $K = 9$	3316	49.72
APLUs, $K = 29$	9618	87.34

For the gridded ReLU and APLU networks, the maximum number of knots allowed by the GPU memory is 31.

For B-splines, by contrast, each evaluation only requires the coefficients of two adjacent basis functions, since the $\varphi_{k,T}$ have minimal overlap, leading to a time complexity of $O(1)$. Accordingly, one only needs to store the coefficients and the index of the two active basis functions.

B. SIGNAL RECOVERY

We further illustrate the benefits of learning the activation functions through the application of convolutional neural networks (CNNs) to inverse problems [49]. Here, the goal is to recover a signal $\mathbf{x} \in \mathbb{R}^N$ from its (noisy) measurements $\mathbf{y} \in \mathbb{R}^M$ given by

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n}, \tag{22}$$

where $\mathbf{H} : \mathbb{R}^N \mapsto \mathbb{R}^M$ is a linear operator that describes the measurement acquisition process and $\mathbf{n} \in \mathbb{R}^M$ is an additive noise.

Classical model-based methods formulate the inverse problem as the optimization task

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathbb{R}^N}{\arg\min} \left(\|\mathbf{y} - \mathbf{H}\mathbf{x}\|_2^2 + \tau R(\mathbf{x})\right), \tag{23}$$

where R is the regularization that incorporates prior information about $\mathbf{x}$ and $\tau \in \mathbb{R}_+$ a parameter that controls the regularization strength. For instance, $R(\mathbf{x}) = \|\mathbf{L}\mathbf{x}\|_1$ [50]–[52] promotes solutions that are sparse in the transform domain specified by $\mathbf{L} : \mathbb{R}^N \to \mathbb{R}^N$. Over the past decade, a variety of learning-based methods were found to outperform the classical model-based ones. These include the use of CNN-based regression schemes that relate an initial estimate of the signal to the desired estimate of the signal [5], [53]. In our experiments, we compare the performance of standard ReLU CNNs with B-spline CNNs in a deconvolution task.

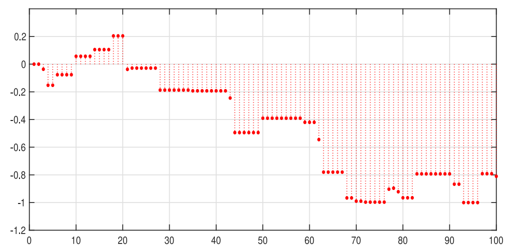


FIGURE 5. **Piecewise-constant signal generated according to (26).**

1) SETUP

We consider the recovery of piecewise-constant statistical signals $\mathbf{x} \in \mathbb{R}^{100}$ that satisfy the discrete innovation model

$$\mathbf{u} = \mathbf{D}\mathbf{x}, \tag{24}$$

where $\mathbf{D} \in \mathbb{R}^{100\times 100}$ is a finite-difference matrix and $\mathbf{u} \in \mathbb{R}^{100}$ is a sparse random vector with independent and identically distributed entries that are drawn from the Bernoulli-Laplace distribution

$$p_U(u) = (0.6)\delta(u) + (0.4)\frac{1}{2}\mathrm{e}^{-|u|}. \tag{25}$$

Under appropriate boundary conditions, we can invert (24) and derive the synthesis formula

$$x_k = \sum_{q=1}^{k} u_q, \quad k = 1, 2, \ldots, 100, \tag{26}$$

which has been used for our experiments. The dynamic range of each generated signal $\mathbf{x}$ is adjusted so that its values lie in $[-1, 1]$. An example of such a signal is shown in Figure 5.

The noiseless measurement vector $\mathbf{y}_0 \in \mathbb{R}^{88}$ is obtained by convolving the signal $\mathbf{x}$ with a Gaussian kernel of standard deviation $\sigma = 2$ and support $(6\sigma + 1) \times 1$. The resulting discrete-system matrix $\mathbf{H} \in \mathbb{R}^{88\times 100}$, such that $\mathbf{y}_0 = \mathbf{H}\mathbf{x}$, is

$$\mathbf{H} = \begin{bmatrix} h_{13} & \cdots & h_1 & 0 & \cdots & 0 \\ 0 & \ddots & & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & & \ddots & 0 \\ 0 & \cdots & 0 & h_{13} & \cdots & h_1 \end{bmatrix}, \tag{27}$$

where $\mathbf{h} \in \mathbb{R}^{13}$ denotes the truncated Gaussian kernel. Finally, we add a white Gaussian noise $\mathbf{n} \in \mathbb{R}^{88}$ to the noiseless measurements $\mathbf{y}_0$ such that the input SNR, defined as

$$\mathrm{SNR}(\mathbf{y}_0 + \mathbf{n}) = 20\log_{10}\left(\|\mathbf{y}_0\|_2/\|\mathbf{n}\|_2\right), \tag{28}$$

is equal to 20 dB.

Our training dataset for the CNN-based approaches consists of $M_t = 10\,000$ samples. Meanwhile, the validation and test datasets contain 1 000 samples each.

Similar to the work in [5], [53], we train CNNs to learn a mapping from an initial estimate of the signal (in our case $\hat{\mathbf{x}}_{\text{init}} = \mathbf{H}^T\mathbf{y}$) to the desired estimate $\hat{\mathbf{x}}$ of the signal. The architecture of the network is shown in Figure 6 and the details

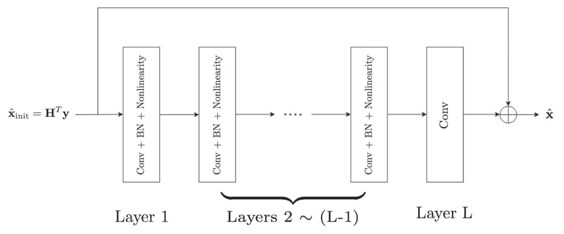


FIGURE 6. **Architecture of the convolutional neural network. In a ReLU CNN, the nonlinearity is the ReLU, while in a B-spline CNN, the nonlinearity is a learnable linear spline.**

TABLE 5 **Convolution Layers**

Layer number	(Filter size, number of input channels, number of output channels)
1	$(3 \times 1,\ 1,\ C)$
$2 \sim (L-1)$	$(3 \times 1,\ C,\ C)$
L	$(3 \times 1,\ C,\ 1)$

of the convolutional layers are provided in Table 5. For all our experiments, the number of channels is set as $C = 5$. In the B-spline CNN, we have learnable linear-spline activation functions with $K = 49$ knots on a grid of size $T = 0.1$.

The loss function used for training is

$$\mathcal{L}(\mathbf{\Theta}) = \sum_{m=1}^{M} \|\mathbf{x}_m - \hat{\mathbf{x}}_m(\mathbf{\Theta})\|_2^2, \tag{29}$$

where $\mathbf{\Theta}$ represents the parameters of the network. All the activation functions in the B-spline CNN are initialized as leaky ReLUs with negative slopes set to 0.1. The loss function is minimized using the ADAM optimizer. The networks are trained for 150 epochs with a batch size of 20. For ReLU CNNs, the initial learning rate is set as 10^{-2} and is decreased by a factor of 0.5 in the epochs [25, 50, 75, 100, 125]. The same learning rate schedule is also used for B-spline CNNs with $L \leq 7$. For B-spline CNNs with more layers ($L > 7$), the initial learning rate is 10^{-3} and is decreased by a factor of 0.5 in the epochs [50, 75, 100, 125].

2) RESULTS AND DISCUSSION

In our experiments, we compare the CNN-based approaches with the total-variation (TV) method [54] which corresponds to $R(\mathbf{x}) = \|\mathbf{D}\mathbf{x}\|_1$ in (23). It is known to promote piecewise-constant solutions and is well matched to the signals that we consider here. In order to make a fair comparison with the CNNs, we use the same regularization parameter τ in the TV method for every signal in the test dataset. This value of τ is the one that gives the best performance in terms of the mean-square error or, equivalently,

$$\text{SNR}(\hat{\mathbf{x}}, \mathbf{x}) = 20\log_{10}\left(\|\mathbf{x}\|_2 / \|\hat{\mathbf{x}} - \mathbf{x}\|_2\right) \tag{30}$$

TABLE 6 **Sharing *vs.* Unsharing of the Linear Spline Activation Functions in B-Spline CNNs ($L = 4$)**

Method	N_{param}	SNR (dB)	Time per epoch (seconds)
ReLU CNN	211	14.95	4.59
Fully shared B-CNN	253	15.09	16.05
Channel shared B-CNN	346	15.16	17.04
Layer shared B-CNN	456	15.23	15.90
Unshared B-CNN	830	15.36	17.78

for the validation dataset.

Sharing vs. Unsharing: We consider four configurations for the B-spline CNN. The first is the fully shared network, where a single learnable spline activation function is shared across all layers and channels. The second and third are the channel (layer, respectively) shared network, where the nonlinearity is shared only across channels (layers, respectively). The fourth is the unshared network, which has an independent nonlinearity in each layer and channel.

In a first experiment, we compare the performance of these four configurations. We fix the number of layers to $L = 4$. In the B-spline CNN, we rely on our hyperparameter-tuning method (see Appendix C) to find the optimal μ_ℓ and λ_ℓ in terms of performance for the validation dataset. The weight decay for the ReLU CNN is chosen via grid search. Using the optimal values, we retrain the networks 9 times independently; the median SNR over these 9 runs is shown for the test dataset in Table 6, where B-CNN means B-spline CNN.

We provide the number of parameters for the networks in Table 6. A ReLU CNN with L layers, C channels, and filter size $(w \times 1)$, has $2wC + (L-2)wC^2$ convolutional-filter weights, 1 bias term for the last convolutional layer, and $2(L-1)C$ batch-normalization parameters. The additional parameters in the B-spline CNN are the total number of active knots in the learned spline activation functions.

We observe that all four versions of the B-spline CNN achieve a higher SNR than the ReLU CNN. This further supports our claim that learning the activation functions tends to improve the performance of the network. Moreover, as expected, configurations with a greater number of parameters perform better. The option of sharing the learnable spline nonlinearities makes our framework flexible and allows us to benefit from the increased capacity of the network while introducing fewer additional parameters. Also, note that Table 6 confirms that the running times for the different versions of the B-spline CNN are nearly the same.

Increase in the Depth of the Networks: Next, we compare the ReLU CNN and the fully shared B-spline CNN when the number of layers increases. The procedure of the previous

TABLE 7 Performance of Deep Networks

Method	Layers	Parameters	Performance
ReLU CNN	4	211	14.95
	5	296	15.27
	6	381	15.47
	7	466	15.68
	8	551	15.74
	10	721	15.80
	15	1146	15.84
Fully shared B-CNN	4	253	15.09
	5	339	15.34
	6	430	15.59
	7	503	15.73
	8	587	15.79
	10	760	15.81
	15	1196	15.85
TV	-	-	14.92

experiment is followed again to set the hyperparameters and to report the performance of the test dataset (see Table 7).

We observe that the CNN-based approaches outperform the TV method, despite it being particularly well matched to the piecewise-constant signals that we consider. This shows the advantage of using learning-based methods over model-based ones when sufficient training data is available. For all values of L, the B-spline CNN achieves a higher SNR than the ReLU CNN. However, this improvement in performance diminishes as L increases and is negligible for $L \geq 10$. We believe that, when the network is sufficiently deep, the ReLU CNN has a sufficient representation power and so the additional capacity offered by the B-spline CNN does not translate into better performance. The main takeaway here is that learning the activation functions results in a noticeable improvement in performance for simpler/smaller networks, which are desirable for a number of reasons such as better interpretability of the networks, computational efficiency, and controlled Lipschitz constants [55].

VI. CONCLUSION

We have presented an efficient computational solution to train deep neural networks with learnable activation functions. Specifically, we have focused on deep spline networks. They form a superset of the traditional ReLU networks and are known to be optimal with respect to the second-order total variation of the adjustable nonlinearities. We have tackled the resulting difficult joint-optimization problem by representing the linear-spline nonlinearities in terms of B-spline basis functions and by expressing the second-order total-variation regularization as an ℓ_1-penalty, thus unifying the parametric and functional approaches for the learning of activation functions. The proposed B-spline representation was instrumental in making the training of the DNN computationally feasible. Indeed, any computation concerning the activation functions involves only two basis elements per data point. Finally, we have demonstrated the benefits of our framework through experiments in the context of classification and deconvolution problems. In particular, we have observed that our method compares favorably to the traditional ReLU networks, the improvement being more pronounced for simpler/smaller networks.

APPENDIX A

SECOND-ORDER TOTAL VARIATION

In this section, we briefly explain the notion of second-order total variation and provide the definition of the corresponding native space $\mathrm{BV}^{(2)}(\mathbb{R})$. We refer to [9] for more details.

The second-order total-variation seminorm of a function $s : \mathbb{R} \to \mathbb{R}$ is defined as

$$\mathrm{TV}^{(2)}(s) = \|\mathrm{D}^2 s\|_{\mathcal{M}}, \tag{31}$$

where D is the (weak) derivative operator and the total-variation norm $\|\cdot\|_{\mathcal{M}}$ is defined over the Banach space $\mathcal{M}(\mathbb{R})$ of bounded Radon measures as

$$\|w\|_{\mathcal{M}} = \sup_{\varphi \in \mathcal{S}(\mathbb{R}):\ \|\varphi\|_\infty = 1} \langle w, \varphi \rangle,$$

where $\mathcal{S}(\mathbb{R})$ is Schwartz' space of smooth and rapidly decaying test functions. The space $\mathcal{M}(\mathbb{R})$ is a generalization of the space $L_1(\mathbb{R})$ of absolutely integrable functions, in the sense that $L_1(\mathbb{R}) \subseteq \mathcal{M}(\mathbb{R})$ and, for any $f \in L_1(\mathbb{R})$, the two norms satisfy $\|f\|_{L_1} = \|f\|_{\mathcal{M}}$. The generalized space $\mathcal{M}(\mathbb{R})$ is, however, larger than $L_1(\mathbb{R})$ as it contains the set of all shifted Dirac impulses $\delta(\cdot - \tau)$ with $\|\delta(\cdot - \tau)\|_{\mathcal{M}} = 1$ for any $\tau \in \mathbb{R}$. In particular, this implies that

$$w_\delta = \sum_{k \in \mathbb{Z}} a[k]\delta(\cdot - \tau_k) \in \mathcal{M}(\mathbb{R}) \quad \text{and} \quad \|w_\delta\|_{\mathcal{M}} = \sum_{k \in \mathbb{Z}} \big|a[k]\big|$$

for any absolutely summable sequence $a[\cdot] \in \ell_1(\mathbb{Z})$. Likewise, since $\mathrm{D}^2\{(\cdot - \tau_k)_+\} = \delta(\cdot - \tau_k)$ (Green's function property), one readily deduces that $\mathrm{TV}^{(2)}(\sigma) = \|\mathbf{a}\|_{\ell_1}$ for the generic spline activation function defined by (8).

Finally, the native space $\mathrm{BV}^{(2)}(\mathbb{R})$ is the space of functions with second-order bounded variation

$$\mathrm{BV}^{(2)}(\mathbb{R}) = \{f : \mathbb{R} \to \mathbb{R} : \mathrm{TV}^{(2)}(f) < +\infty\}.$$

B. APPENDIX B

LEARNABLE SPLINE ACTIVATION FUNCTION MODULE

In this section, we describe our implementation of the B-spline formulation of the learnable linear-spline activation functions. We also detail our sparsification procedure which is a postprocessing step during training; the intent is to control the number of active knots in the network.

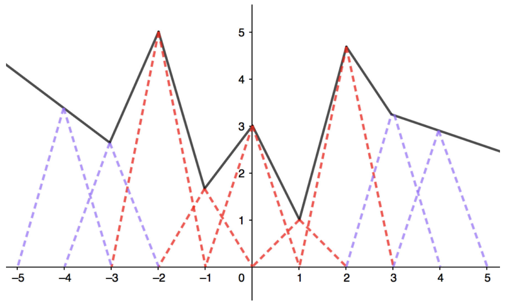


FIGURE 7. The left and right linear extrapolations beyond [−3, 3] of the activation function are computed with the help of two extra B-splines on each side.

A. B-SPLINE FORMULATION

We place a highly redundant set of knots (for the linear spline) on a finite uniform grid of size T. The cardinality of this set of knots is K, with K odd. We define the indices $k_{\min} = -(K-1)/2$ and $k_{\max} = (K-1)/2$. The spline we want to build will extend linearly outside the interval $[k_{\min}T, k_{\max}T]$ and can be represented in the gridded ReLU basis as

$$\sigma(x) = b_0 + b_1 x + \sum_{k=k_{\min}}^{k_{\max}} a_k (x-kT)_+, \tag{32}$$

with $\mathrm{TV}^{(2)}(\sigma) = \|\mathbf{a}\|_1$.

Here, we represent σ in a B-spline basis as (33) shown at the bottom of this page, where φ_T is the triangle-shaped B-spline

$$\varphi_T(x) = \begin{cases} 1 - \left|\frac{x}{T}\right|, & -T \le x \le T, \\ 0, & \text{otherwise.} \end{cases} \tag{34}$$

The B-spline representation in (33) is equivalent to the one in (9). Here, we place $K+2$ triangular basis functions on the grid and, instead of using one-sided boundary basis functions, the linear extrapolations beyond $[k_{\min}T, k_{\max}T]$ are handled with the help of the last two B-spline coefficients on each side: $(k_{\min-1}, k_{\min})$ and $(k_{\max}, k_{\max+1})$. An example of this construction is shown in Figure 7.

The relationship between the ReLU coefficients $\mathbf{a} \in \mathbb{R}^K$ and the B-spline coefficients $\mathbf{c} \in \mathbb{R}^{K+2}$ is given by

$$\begin{bmatrix} a_{k_{\min}} \\ \vdots \\ a_{k_{\max}} \end{bmatrix} = \frac{1}{T} \underbrace{\begin{bmatrix} 1 & -2 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 1 & -2 & 1 \end{bmatrix}}_{\mathbf{L} \in \mathbb{R}^{(K+2)\times K}} \begin{bmatrix} c_{k_{\min-1}} \\ c_{k_{\min}} \\ \vdots \\ c_{k_{\max}} \\ c_{k_{\max+1}} \end{bmatrix}, \tag{35}$$

while the linear-term parameters b_0, b_1 can be determined from $c_{k_{\min-1}}$ and $c_{k_{\min}}$. From (35), we see that the $\mathrm{TV}^{(2)}$ regularization of σ can also be computed from the B-spline coefficients as $\mathrm{TV}^{(2)}(\sigma) = \|\mathbf{Lc}\|_1$.

B. SPARSIFICATION

To train networks with learnable spline nonlinearities, we augment the cost function with the $\mathrm{TV}^{(2)}$ regularization of the activation functions. This translates into an ℓ_1-penalty on the ReLU coefficients $\mathbf{a} = (a_k)$ or, equivalently, on the filtered version of the B-spline coefficients $\mathbf{Lc}$. We rely on the sparsifying effect of the ℓ_1-norm to remove some of the redundant knots. In practice, we observe that, while some of the coefficients $a_k = [\mathbf{Lc}]_k$ attain small values, they never vanish entirely. In order to fix this and have a tight control on the number of knots, we have applied a further "sparsification" as a postprocessing step after training.

The first step is to retrieve the ReLU coefficients $\mathbf{a}$ from the trained B-spline coefficients $\mathbf{c}$ using (35). Then, every coefficient a_k with absolute value below a certain threshold is set to zero, yielding $\hat{\mathbf{a}} = (\hat{a}_k)$. Finally, we transform these modified ReLU coefficients to the new B-spline coefficients $\hat{\mathbf{c}}$. In this step, the coefficients $\hat{c}_{k_{\min-1}}$ and $\hat{c}_{k_{\min}}$ that determine the linear term are assigned the same values as $c_{k_{\min-1}}$ and $c_{k_{\min}}$, respectively. The other coefficients $\hat{c}_k$ are computed from $\hat{a}_k$ using the relations in (35). The sparsification is achieved by selecting the maximum threshold such that the training accuracy does not drop by more than 0.2%.

APPENDIX C
HYPERPARAMETER TUNING USING OPTIMALITY CONDITIONS

In this section, we propose a method to tune the hyperparameters of Problems (3) and (6). Our hyperparameter-tuning method is based on some optimality conditions that we prove for the global minimizers of these problems. It is flexible with respect to the choice of linear layers and architecture and can be applied to any deep spline network.

A. OPTIMALITY CONDITIONS

The main principle of our optimality conditions is based on the scale- and dilation-invariance properties of the second-order total-variation regularization, as we state in Proposition 2.

Proposition 2: The second-order total-variation regularization $\mathrm{TV}^{(2)} : \mathrm{BV}^{(2)}(\mathbb{R}) \to \mathbb{R}$ is scale- and dilation-invariant. Specifically, for any $\sigma \in \mathrm{BV}^{(2)}(\mathbb{R})$ and any $c \neq 0$, we have that

$$\mathrm{TV}^{(2)}(c\sigma) = |c|\,\mathrm{TV}^{(2)}(\sigma), \tag{36}$$

$$\sigma(x) = \begin{cases} c_{k_{\min}} + \frac{1}{T}(c_{k_{\min}} - c_{k_{\min-1}})(x - k_{\min}T), & x \in (-\infty, k_{\min}T) \\ \sum_{k=k_{\min-1}}^{k_{\max+1}} c_k \varphi_T(x-kT), & x \in [k_{\min}T, k_{\max}T] \\ c_{k_{\max}} + \frac{1}{T}(c_{k_{\max+1}} - c_{k_{\max}})(x - k_{\max}T), & x \in (k_{\max}T, \infty) \end{cases} \tag{33}$$

and

$$\mathrm{TV}^{(2)}\left(\sigma(c\cdot)\right) = |c|\mathrm{TV}^{(2)}\left(\sigma\right). \tag{37}$$

Proof: We first recall that

$$\mathrm{TV}^{(2)}(\sigma) = \|\mathrm{D}^2\sigma\|_{\mathcal{M}} = \sup_{\varphi\in\mathcal{S}(\mathbb{R})\setminus\{0\}} \frac{\langle \mathrm{D}^2\sigma, \varphi\rangle}{\|\varphi\|_\infty}. \tag{38}$$

One deduces (36) from the linearity of D^2 and the homogeneity of the $\mathcal{M}$-norm. To derive (37), we use the relation $\mathrm{D}^2\{\sigma(c\cdot)\} = c^2\mathrm{D}^2\{\sigma\}(c\cdot)$ and the equality $\langle f(c\cdot), g\rangle = c^{-1}\langle f, g(\cdot/c)\rangle$ which, together with (38), yields

$$\begin{aligned}\mathrm{TV}^{(2)}(\sigma(c\cdot)) &= \sup_{\varphi\in\mathcal{S}(\mathbb{R})\setminus\{0\}} c\frac{\langle \mathrm{D}^2\sigma, \varphi(\cdot/c)\rangle}{\|\varphi\|_\infty}\\ &= |c| \sup_{\psi\in\mathcal{S}(\mathbb{R})\setminus\{0\}} \frac{\langle \mathrm{D}^2\sigma, \psi\rangle}{\|\psi\|_\infty},\end{aligned} \tag{39}$$

where the latter is obtained via the change of variable $\psi = \mathrm{sgn}(c)\varphi(\cdot/c)$. The last step is to notice that

$$\sup_{\psi\in\mathcal{S}(\mathbb{R})\setminus\{0\}} \frac{\langle \mathrm{D}^2\sigma, \psi\rangle}{\|\psi\|_\infty} = \mathrm{TV}^{(2)}(\sigma)$$

which, when combined with (39), yields (37).

In Theorem (3), we prove that the energies of all linear and nonlinear layers of any global minimizer of (3) are inversely proportional to their corresponding regularization parameters.

Theorem 3: Let $\mathbf{f}_{\Theta^*}$ be a global minimizer of (3) with linear parameters $\boldsymbol{\theta}_\ell^*$ and learned activation functions $\boldsymbol{s}_\ell^*$. Then, we have that

$$\begin{aligned}2\mu_1\|\boldsymbol{\theta}_1^*\|_2^2 &= \lambda_1\mathrm{TV}_1^{(2)}(\mathbf{s}_1^*) = \cdots = \lambda_{L-1}\mathrm{TV}_1^{(2)}(\mathbf{s}_{L-1}^*)\\ &= 2\mu_L\|\boldsymbol{\theta}_L^*\|_2^2.\end{aligned} \tag{40}$$

Proof: Let us denote by G^* the geometric mean of the $L + 2(L-1) = (3L-2)$ quantities

$$\{\mu_\ell\|\boldsymbol{\theta}_\ell^*\|_2^2\}_{\ell=1}^L \bigcup \left\{\frac{\lambda_\ell}{2}\mathrm{TV}_1^{(2)}(\boldsymbol{s}_\ell^*)\right\}_{\ell=1}^{L-1} \bigcup \left\{\frac{\lambda_\ell}{2}\mathrm{TV}_1^{(2)}(\boldsymbol{s}_\ell^*)\right\}_{\ell=1}^{L-1}.$$

It turns out that G^* can be computed via the relation

$$G^{*(3L-2)} = \left(\prod_{\ell=1}^{L}\mu_\ell\|\boldsymbol{\theta}_\ell^*\|_2^2\right)\left(\prod_{\ell=1}^{L-1}\frac{\lambda_\ell}{2}\mathrm{TV}_1^{(2)}(\boldsymbol{s}_\ell^*)\right)^2.$$

Due to the inequality of arithmetic and geometric means (AM and GM, respectively), we have that

$$(3L-2)G^* \le \sum_{\ell=1}^{L-1}\lambda_\ell\mathrm{TV}_1^{(2)}(\mathbf{s}_\ell^*) + \sum_{\ell=1}^{L}\mu_\ell\|\theta_\ell^*\|_2^2, \tag{41}$$

where the inequality is saturated if and only if (40) holds.

Inspired from the mentioned AM-GM inequality, we now define a new set of linear parameters $\tilde{\boldsymbol{\theta}}_\ell$, $\ell = 1, \dots, L$ and adjustable activation functions $\tilde{\boldsymbol{s}}_\ell$, $\ell = 1, \dots, L-1$, as

$$\tilde{\boldsymbol{\theta}}_\ell = c_\ell\boldsymbol{\theta}_\ell, c_\ell = \left(\frac{G^*}{\mu_\ell\|\boldsymbol{\theta}_\ell\|_2^2}\right)^{\frac{1}{2}},$$

$$\tilde{\boldsymbol{s}}_\ell = d_\ell\boldsymbol{s}_\ell\left(\frac{\cdot}{c_\ell d_{\ell-1}}\right), d_\ell = c_\ell d_{\ell-1}\frac{G^*}{\frac{\lambda_\ell}{2}\mathrm{TV}_1^{(2)}(\boldsymbol{s}_\ell)},$$

with the convention that $d_0 = 1$. Let us specify the corresponding linear and nonlinear layers by $\tilde{\boldsymbol{W}}_\ell$ and $\tilde{\boldsymbol{\sigma}}_\ell$, respectively. One readily observes that

$$\tilde{\boldsymbol{W}}_\ell = c_\ell\boldsymbol{W}_\ell, \qquad \tilde{\boldsymbol{\sigma}}_\ell = d_\ell\boldsymbol{\sigma}_\ell\left(\frac{\cdot}{c_\ell d_{\ell-1}}\right)$$

in all layers. Interestingly, the input-output relation of this new neural network is the same as that of $\boldsymbol{f}_{\Theta^*}$. This is due to two simple observations.

1) For $\ell = 1, \dots, L-1$, we have that

$$\tilde{\boldsymbol{W}}_\ell \circ \tilde{\boldsymbol{\sigma}}_\ell(\cdot) = d_\ell\boldsymbol{W}_\ell \circ \boldsymbol{\sigma}(\cdot/d_{\ell-1}).$$

2) For the output-layer, we have that $c_L d_{L-1} = 1$.

Since the input-output relation remains unchanged, the data-fidelity term in the cost functional of the minimization (3) does not change either. Now, due to the optimality of $\boldsymbol{f}_{\Theta^*}$, we deduce that

$$\begin{aligned}&\sum_{\ell=1}^{L-1}\lambda_\ell\mathrm{TV}_1^{(2)}(\boldsymbol{s}_\ell^*) + \sum_{\ell=1}^{L}\mu_\ell\|\boldsymbol{\theta}_\ell^*\|_2^2\\ &\le \sum_{\ell=1}^{L-1}\lambda_\ell\mathrm{TV}_1^{(2)}(\tilde{\boldsymbol{s}}_\ell) + \sum_{\ell=1}^{L}\mu_\ell\|\tilde{\boldsymbol{\theta}}_\ell\|_2^2.\end{aligned} \tag{42}$$

Using Proposition 2, we have that

$$\lambda_\ell\mathrm{TV}_1^{(2)}(\tilde{\boldsymbol{s}}_\ell) = \lambda_\ell\frac{d_\ell}{c_\ell d_{\ell-1}}\mathrm{TV}_1^{(2)}(\boldsymbol{s}_\ell) = 2G^*,$$

for $\ell = 1, \dots, L-1$. Similarly, from the scale invariance of the ℓ_2-norm, we deduce that

$$\mu_\ell\|\tilde{\boldsymbol{\theta}}_\ell\|_2^2 = \mu_\ell c_\ell^2\|\boldsymbol{\theta}_\ell^*\|_2^2 = G^*.$$

Replacing these in (42), we obtain that

$$\sum_{\ell=1}^{L-1}\lambda_\ell\mathrm{TV}_1^{(2)}(\boldsymbol{s}_\ell^*) + \sum_{\ell=1}^{L}\mu_\ell\|\boldsymbol{\theta}_\ell^*\|_2^2 \le (3L-2)G^*,$$

which is the converse of the AM-GM inequality (41). This shows that (41) is saturated and, hence, that (40) holds. ■

For the case where the activation functions are shared across layers, we show in Theorem 4 that the optimal configuration is such that there would be a balance between the total energy of linear layers and the second-order total variation of the learned activation functions.

Theorem 4: Let f_{Θ^*} be a global minimizer of (6). Then, we have that

$$\lambda\mathrm{TV}_1^{(2)}(\mathbf{s}^*) = 2\sum_{\ell=1}^{L-1}\mu_\ell\|\theta_1^*\|_2^2. \tag{43}$$

Proof: The proof is very similar to the one for Theorem 3. We define G^* as

$$G^* = \left(\frac{\lambda}{2}\mathrm{TV}_1^{(2)}(\boldsymbol{s}^*)\right)^{\frac{2}{3}}\left(\sum_{\ell=1}^{L}\mu_\ell\|\boldsymbol{\theta}_\ell\|_2^2\right)^{\frac{1}{3}}.$$

The AM-GM inequality implies in this case that

$$3G^* \leq \lambda_\ell \mathrm{TV}_1^{(2)}(\mathbf{s}^*) + \sum_{\ell=1}^{L} \mu_\ell \|\theta_\ell^*\|_2^2, \tag{44}$$

with equality if and only if (43) holds. Now, we define a new set of linear parameters and adjustable activation functions as

$$\tilde{\boldsymbol{\theta}}_\ell = c^{-1}\boldsymbol{\theta}_\ell, \qquad \ell = 1, \dots, L,$$
$$\tilde{\mathbf{s}} = c\boldsymbol{s}(c\cdot),$$

where the constant $c > 0$ is

$$c = \frac{G^*}{\frac{\lambda}{2}\mathrm{TV}_1^{(2)}(\boldsymbol{s})}.$$

Again, the data-fidelity term remains unchanged. From the optimality of $\boldsymbol{f}_{\boldsymbol{\Theta}^*}$, we deduce that

$$\lambda \mathrm{TV}_1^{(2)}(\boldsymbol{s}^*) + \sum_{\ell=1}^{L} \mu_\ell \|\boldsymbol{\theta}_\ell^*\|_2^2 \leq \lambda \mathrm{TV}_1^{(2)}(\tilde{\mathbf{s}}) + \sum_{\ell=1}^{L} \mu_\ell \|\tilde{\boldsymbol{\theta}}_\ell\|_2^2.$$

By direct calculations, similar to what we did in Theorem 3, we simplify the above inequality into

$$\lambda \mathrm{TV}_1^{(2)}(\boldsymbol{s}^*) + \sum_{\ell=1}^{L} \mu_\ell \|\boldsymbol{\theta}_\ell^*\|_2^2 \leq 3G^*$$

which, together with (44) implies that the AM-GM equality holds, ultimately leading to (43).

B. HYPERPARAMETER TUNING

Using Theorems 3 and 4, we now introduce a way to tune the hyperparameters of our optimization problems. The main idea is to enforce the optimality condition in the initial settings (before training) and, consequently, to reduce the dimension of the hyperparameter space so that it is sufficient to perform a grid search over a single parameter.

Our scheme is described as follows:

1) Initialize the linear parameters $\boldsymbol{\theta}_\ell^0$ (*e.g.,* using Xavier's rule) and the activation functions $\boldsymbol{s}_\ell^0$ (*e.g.,* soft-threshold/absolute value) and compute the quantities $\|\boldsymbol{\theta}_\ell^0\|_2^2$ and $\mathrm{TV}^{(2)}(\boldsymbol{s}_\ell^0)$ for all layers.
2) Set

$$\mu_\ell = \frac{C}{2\|\boldsymbol{\theta}_\ell^0\|_2^2}, \tag{45}$$

where $C > 0$ is the unique hyperparameter that is required to be tuned.
3) If the activation functions are shared across layers, set

$$\lambda = \frac{(L-1)C}{\mathrm{TV}_1^{(2)}(\mathbf{s}^0)}. \tag{46}$$

Otherwise, set

$$\lambda_\ell = \frac{C}{\mathrm{TV}_1^{(2)}(\mathbf{s}_\ell^0)}. \tag{47}$$

4) Perform a grid search to find the optimal value of $C > 0$.

REFERENCES

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Germany: Springer, 2006.
[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1106–1114.
[3] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Oct. 2012.
[4] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assisted Interv.*, vol. 9351, Berlin, Germany: Springer, 2015, pp. 234–241.
[5] K. H. Jin, M. T. McCann, E. Froustey, and M. Unser, "Deep convolutional neural network for inverse problems in imaging," *IEEE Trans. Image Process.*, vol. 26, no. 9, pp. 4509–4522, Sep. 2017.
[6] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, vol. 1, Cambridge, MA, USA: MIT Press, 2016.
[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.
[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
[9] M. Unser, "A representer theorem for deep neural networks," *J. Mach. Learn. Res.*, vol. 20, no. 110, pp. 1–30, 2019.
[10] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2924–2932.
[11] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, "Understanding deep neural networks with rectified linear units," 2016, *arXiv:1611.01491*.
[12] G. Strang, "The functions of deep learning," *SIAM News*, vol. 51, no. 10, pp. 1–4, 2018.
[13] T. Poggio, L. Rosasco, A. Shashua, N. Cohen, and F. Anselmi, "Notes on hierarchical splines, DCLNs and i-theory," Center for Brains, Minds and Machines (CBMM), Tech. Rep., 2015.
[14] R. Balestriero and R. G. Baraniuk, "A spline theory of deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 374–383.
[15] R. Parhi and R. D. Nowak, "The role of neural network activation functions," *IEEE Signal Process. Lett.*, vol. 27, pp. 1779–1783, 2020.
[16] R. Parhi and R. D. Nowak, "Banach space representer theorems for neural networks and ridge splines," 2020, *arXiv:2006.05626*.
[17] C. T. Chen and W. D. Chang, "A feedforward neural network with function shape autotuning," *Neural Netw.*, vol. 9, no. 4, pp. 627–641, 1996.
[18] S. H. Lane, M. Flax, D. Handelman, and J. Gelfand, "Multi-layer perceptrons with B-spline receptive field functions," in *Proc. Adv. Neural Inf. Process. Syst.*, 1991, pp. 684–692.
[19] L. Vecci, F. Piazza, and A. Uncini, "Learning and approximation capabilities of adaptive spline activation function neural networks," *Neural Netw.*, vol. 11, no. 2, pp. 259–270, 1998.
[20] S. Guarnieri, F. Piazza, and A. Uncini, "Multilayer feedforward networks with adaptive spline activation function," *IEEE Trans. Neural Netw.*, vol. 10, no. 3, pp. 672–683, May 1999.
[21] L. Hou, D. Samaras, T. M. Kurc, Y. Gao, and J. H. Saltz, "Convnets with smooth adaptive activation functions for regression," *Proc. Mach. Learn. Res.*, vol. 54, pp. 430, 2017.
[22] R. G. Baraniuk, E. Candès, M. Elad, and M. Yi, "Applications of sparse representation and compressive sensing," *Proc. IEEE*, vol. 98, no. 6, pp. 906–909, 2010.
[23] M. Elad, *Sparse and Redundant Representations. From Theory to Applications in Signal and Image Processing*. Berlin, Germany: Springer, 2010.
[24] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 399–406.
[25] V. Papyan, Y. Romano, J. Sulam, and M. Elad, "Theoretical foundations of deep learning via sparse representations: A multilayer sparse model and its connection to convolutional neural networks," *IEEE Signal Process. Mag.*, vol. 35, no. 4, pp. 72–89, Jul. 2018.
[26] M. A. T. Figueiredo and R. D. Nowak, "An EM algorithm for wavelet-based image restoration," *IEEE Trans. Image Process.*, vol. 12, no. 8, pp. 906–916, Aug. 2003.
[27] I. Daubechies, M. Defrise, and C. D. Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Commun. Pure Appl. Math.*, vol. 57, no. 11, pp. 1413–1457, 2004.

[28] Y. Chen and T. Pock, “Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1256–1272, Jun. 2017.
[29] U. Kamilov and H. Mansour, “Learning optimal nonlinearities for iterative thresholding algorithms,” *IEEE Signal Process. Lett.*, vol. 23, no. 5, pp. 747–751, May 2016.
[30] H. Q. Nguyen, E. Bostan, and M. Unser, “Learning convex regularizers for optimal Bayesian denoising,” *IEEE Trans. Signal Process.*, vol. 66, no. 4, pp. 1093–1105, Feb. 2018.
[31] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proc. Int. Conf. Artif. Intell. Statist.*, 2011, pp. 315–323.
[32] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. Int. Conf. Mach. Learn. Workshop Deep Learn. Audio, Speech Lang. Process.*, 2013.
[33] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1026–1034.
[34] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, “Learning activation functions to improve deep neural networks,” 2014, *arXiv:1412.6830*.
[35] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan, “Deep learning with s-shaped rectified linear activation units,” in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 1737–1743.
[36] D. Perekrestenko, P. Grohs, D. Elbrächter, and H. Bölcskei, “The universal approximation power of finite-width deep Relu networks,” 2018, *arXiv:1806.01528*.
[37] H. Bölcskei, P. Grohs, G. Kutyniok, and P. Petersen, “Optimal approximation with sparsely connected deep neural networks,” *SIAM J. Math. Data Sci.*, vol. 1, no. 1, pp. 8–45, 2019.
[38] R. Gribonval, G. Kutyniok, M. Nielsen, and F. Voigtlaender, “Approximation spaces of deep neural networks,” 2019, *arXiv:1905.01208*.
[39] J. M. Tarela and M. V. Martinez, “Region configurations for realizability of lattice piecewise-linear models,” *Math. Comput. Modelling*, vol. 30, no. 11, pp. 17–27, 1999.
[40] S. Wang and X. Sun, “Generalization of hinging hyperplanes,” *IEEE Trans. Inf. Theory*, vol. 51, no. 12, pp. 4425–4431, Dec. 2005.
[41] R. Tibshirani, “Regression shrinkage and selection via the Lasso,” *J. Roy. Stat. Soc. Ser. B*, vol. 58, no. 1, pp. 265–288, 1996.
[42] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proc. Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.
[43] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *Proc. Int. Conf. Learn. Representations*, 2015.
[44] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.
[45] M. Lin, Q. Chen, and S. Yan, “Network in network,” 2013, *arXiv:1312.4400*.
[46] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[47] N. S. Keskar and R. Socher, “Improving generalization performance by switching from ADAM to SGD,” 2017, *arXiv:1712.07628*.
[48] A. Molina, P. Schramowski, and K. Kersting, “Padé activation units: End-to-end learning of flexible activation functions in deep networks,” in *Proc. Int. Conf. Learn. Representations*, 2020.
[49] M. T. McCann, K. H. Jin, and M. Unser, “Convolutional neural networks for inverse problems in imaging—A review,” *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 85–95, Nov. 2017.
[50] D. L. Donoho, “Compressed sensing,” *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.
[51] S. Foucart and H. Rauhut, *A Mathematical Introduction to Compressive Sensing*. Basel, Switzerland: Birkhäuser, 2013.
[52] M. Lustig, D. L. Donoho, and J. M. Pauly, “Sparse MRI: The application of compressed sensing for rapid MR imaging,” *Magn. Reson. Med.*, vol. 58, no. 6, pp. 1182–1195, 2007.
[53] C. M. Hyun, H. P. Kim, S. M. Lee, S. Lee, and J. K. Seo, “Deep learning for undersampled MRI reconstruction,” *Phys. Med. Biol.*, vol. 63, no. 13, 2018, Art. no. 135007.
[54] L. I. Rudin, S. Osher, and E. Fatemi, “Nonlinear total variation based noise removal algorithms,” *Physica D: Nonlinear Phenom.*, vol. 60, nos. 1–4, pp. 259–268, 1992.
[55] S. Aziznejad, H. Gupta, J. Campos, and M. Unser, “Deep neural networks with trainable activations and controlled Lipschitz constant,” *IEEE Trans. Signal Process.*, vol. 68, pp. 4688–4699, 2020.

PAKSHAL BOHRA (Graduate Student Member, IEEE) received the B.Tech and M.Tech degrees in electrical engineering from the Indian Institute of Technology, Bombay, India, in 2018. He is currently working toward the Ph.D. degree with the Biomedical Imaging Group, École Polytechnique Fédérale de Lausanne, Switzerland, under the direction of Michael Unser. His research interest includes splines, stochastic processes and learning for inverse problems.

JOAQUIM CAMPOS received the B.Sc. degree in electrical engineering from Instituto Superior Tècnico, Lisbon, Portugal, in 2016, and the M.Sc. degree in communication systems from the École Polytechnique Fédérale de Lausanne (EPFL), Switzerland, in 2020. He is currently working toward the Ph.D. degree with the Biomedical Imaging Group, École Polytechnique Fédérale de Lausanne, under the direction of Michael Unser. His research interests include machine learning and inverse problems.

HARSHIT GUPTA received the B.Tech degree in electronics and communications engineering from the Indian Institute of Technology (IIT), Guwahati, in 2015. He received the Ph.D. degree in 2020 from the Biomedical Imaging Group, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, under the direction of Prof. Michael Unser. His research interests include aimed at designing deep-learning-based algorithms, backed with sound mathemat-ics, in order to solve inverse problems in imaging. He is interested in the modalities of cryo electron microscopy (Cryo-EM), computational tomography (CT), and magnetic resonance imaging (MRI). He has also worked on spline theory, representer theorems, and variational methods.

SHAYAN AZIZNEJAD received the B.Sc. degree in electrical engineering and mathematics from the Sharif University of Technology, Tehran, Iran, in 2017. He is currently working toward the Ph.D. degree under the direction of Michael Unser with the Biomedical Imaging Group, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland. His research interests include variational formulation of inverse problems and machine learning, spline theory, and stochastic processes. He was the recipient of the Best Student Paper Award at ICASSP 2019.

MICHAEL UNSER (Fellow, IEEE) is a Professor and Director of EPFL’s Biomedical Imaging Group, Lausanne, Switzerland. His primary area of investigation is biomedical image processing. He has authored or coauthored more than 350 journal papers on those topics. He is the author with P. Tafti of the book *An introduction to sparse stochastic processes* (Cambridge University Press 2014). From 1985 to 1997, he was with the Biomedical Engineering and Instrumentation Program, National Institutes of Health, Bethesda USA, conducting research on bioimaging. He is internationally recognized for his research contributions to sampling theory, wavelets, the use of splines for image processing, stochastic processes, and computational bioimaging. Prof. Unser has served on the editorial board of most of the primary journals in his field including the IEEE Transactions on Medical Imaging (Associate Editor-in-Chief 2003–2005), IEEE Transactions on Image Processing, Proceedings of the IEEE, and *SIAM Journal of Imaging Sciences*. He is the Founding Chair of the technical committee on Bio Imaging and Signal Processing (BISP) of the IEEE Signal Processing Society. Prof. Unser is a EURASIP fellow (2009), and a member of the Swiss Academy of Engineering Sciences. He was the recipient of several international prizes including five IEEE-SPS Best Paper Awards, two Technical Achievement Awards from the IEEE (2008 SPS and EMBS 2010), the Technical Achievement Award from EURASIP (2018), and a recent Career Achievement Award (IEEE EMBS 2020).