INTERNATIONAL JOURNAL OF
NUMERICAL ANALYSIS AND MODELING
Volume 21, Number 5, Pages 739–763

doi: 10.4208/ijnam2024-1030

IMPROVING THE EXPRESSIVE POWER OF DEEP NEURAL NETWORKS THROUGH INTEGRAL ACTIVATION TRANSFORM

ZEZHONG ZHANG, FENG BAO, AND GUANNAN ZHANG*

Abstract. The impressive expressive power of deep neural networks (DNNs) underlies their widespread applicability. However, while the theoretical capacity of deep architectures is high, the practical expressive power achieved through successful training often falls short. Building on the insights gained from Neural ODEs, which explore the depth of DNNs as a continuous variable, in this work, we generalize the traditional fully connected DNN through the concept of continuous width. In the Generalized Deep Neural Network (GDNN), the traditional notion of neurons in each layer is replaced by a continuous state function. Using the finite rank parameterization of the weight integral kernel, we establish that GDNN can be obtained by employing the Integral Activation Transform (IAT) as activation layers within the traditional DNN framework. The IAT maps the input vector to a function space using some basis functions, followed by nonlinear activation in the function space, and then extracts information through the integration with another collection of basis functions. A specific variant, IAT-ReLU, featuring the ReLU nonlinearity, serves as a smooth generalization of the scalar ReLU activation. Notably, IAT-ReLU exhibits a continuous activation pattern when continuous basis functions are employed, making it smooth and enhancing the trainability of the DNN. Our numerical experiments demonstrate that IAT-ReLU outperforms regular ReLU in terms of trainability and better smoothness.

Key words. Integral transform, generalized neural network, continuous ReLU activation pattern, expressive power of neural network.

1. Introduction

Deep learning, particularly deep neural networks (DNNs), has not only achieved remarkable success in traditional computer science fields such as computer vision [1] and natural language processing [2] but has also gained rapid popularity in other scientific communities, such as numerical partial differential equations (PDEs) [3] and biology [4]. Their cross-disciplinary popularity stems from their remarkable ability as highly expressive black-box function approximators. With high enough expressive power, as long as we can define a desired input-output map by a loss function, they can approximate such functions through brute force optimization, making them useful in many applications.

Despite the work of Hornik showing the universal approximation ability of DNNs with one hidden layer [5], in practice, selecting an appropriate architecture with good hyperparameters, such as width and depth, is crucial. The architecture choice plays a significant role in achieving good practical expressive power, which refers to the model's trainability through gradient-based optimization [6, 7]. In other words, even with a large number of parameters, a poorly chosen architecture can result in limited practical expressive power, especially for deep architectures. By considering the model depth as a continuous variable, Neural ODEs, as introduced in [8], offer

Received by the editors on December 13, 2023 and, accepted on May 16, 2024.
2000 *Mathematics Subject Classification.* 35R35, 49J40, 60G40.
*Corresponding author.

improved expressive power without increasing the number of parameters. In this continuous depth setting, the forward propagation is formulated as an integral with respect to the depth variable.

Inspired by the idea of continuous depth, in this work, we introduce a General Deep Neural Network (GDNN) that explores the concept of continuous width. This approach is also rooted in the classical conceptual extension in functional analysis, where finite-dimensional vectors are generalized to infinite-dimensional univariate functions and matrix-vector multiplications become integral transforms. With this concept, in GDNN, the state vector is regarded as a (continuous) function defined over the interval [-1,1]. Consequently, the weight matrices and bias vectors are also generalized to weight integral kernels and bias functions. The forward propagation of GDNN becomes recursively integrating the activated state functions with the weight kernel. A normal DNN can be viewed as a discretization of GDNN, where the standard weight matrices and bias vectors are obtained by evaluating the weight integral kernel and bias function on 2D and 1D mesh points, respectively. There are many existing methods to parameterize the weight kernel [9], and in this paper, we mainly focus on the finite rank parameterization of the weight kernel. Under this parameterization, GDNN is equivalent to a traditional DNN with the Integral Activation Transform (IAT), which serves as a multivariate ($\mathbb{R}^d \to \mathbb{R}^d$) nonlinear activation layer. In IAT, the input vector is first transformed into a state function by treating its elements as coefficients of some predetermined basis functions. Then, a pointwise activation is applied to the state function. Finally, the output vector is obtained by integrating the activated state function with another set of basis functions. We also show that by choosing rectangular functions as the basis functions, the IAT simplifies to the standard element-wise activation.

When the pointwise nonlinear activation is ReLU, we obtain a variant termed IAT-ReLU. Intriguingly, it is observed that both IAT-ReLU and element-wise ReLU exhibit a common characteristic where the activation matrix in the forward map is identical to the gradient. For example, applying ReLU element-wise to an input vector $\boldsymbol{z} = [z_1, ..., z_d]^T \in \mathbb{R}^d$ can be expressed as $\phi(\boldsymbol{z}) = \text{diag}([\mathbf{1}_{z_1>0}(\boldsymbol{z}), ..., \mathbf{1}_{z_d>0}(\boldsymbol{z})])\boldsymbol{z}$, and the activation matrix $\text{diag}([\mathbf{1}_{z_1>0}(\boldsymbol{z}), ..., \mathbf{1}_{z_d>0}(\boldsymbol{z})])$ in the forward map is also the gradient. This connection can be further elucidated through Euler's theorem for homogeneous functions, as both ReLU and IAT-ReLU are classified as homogeneous functions.

At first, it might seem unsurprising that IAT-ReLU becomes equivalent to ReLU when employing rectangular basis functions. However, our findings indicate that this equivalence persists regardless of the basis functions chosen, effectively expanding the scope for developing ReLU-like activation functions beyond the limitations of rectangular basis functions. In IAT-ReLU, the activation matrix is continuously determined by the activation pattern $\mathcal{D}(\boldsymbol{z})$, a compact subset of the interval $[-1, 1]$. This activation pattern is defined as $\mathcal{D}(\boldsymbol{z}) = \{s \in [-1, 1] : \boldsymbol{p}^T(s)\boldsymbol{z} \geq 0\}$, where $\boldsymbol{z}$ represents the input vector and $\boldsymbol{p}(s) = [p_1(s), \ldots, p_d(s)]^T$ is a set of selected basis functions. With rectangular basis functions, the activation pattern comprises fixed sub-intervals, each aligning with the support of a rectangular basis function. The inclusion of these intervals depends on the sign of $\boldsymbol{z}$, leading to a piecewise constant activation pattern, which also makes the gradient of ReLU discontinuous. Conversely, when using continuous basis functions, the activation pattern in IAT-ReLU is not confined to predefined sub-intervals. Instead, it can form infinitely many subsets within the domain $[-1, 1]$. Furthermore, we demonstrate that the activation pattern can vary continuously with the input $\boldsymbol{z}$, presenting multiple advantages:

- A continuous pattern leads to a continuous gradient, making the IAT-ReLU smooth and facilitating more stable training compared to ReLU, which is non-smooth and has a discontinuous gradient.
- The gradient can directly flow through the activation pattern, enabling the direct training of the activation pattern itself. In contrast, ReLU lacks gradient flow through the pattern as it is determined by the sign of $\boldsymbol{z}$.
- With a continuous pattern, it becomes possible to decouple the activation pattern from the model parameters while still maintaining a continuous function. In the case of ReLU, decoupling the activation pattern results in a discontinuous model.

Moreover, the selection of basis functions with high total variation can increase the variability of activation patterns in domain $[-1, 1]$. This is analogous to increasing the number of linear pieces in the ReLU network, where each linear piece is defined as the collection of inputs with the same activation pattern[10], and it is believed that by having more linear pieces the ReLU network will have more expressive power[11, 12].

We summarize the main contributions of this work as follows:

- We introduce the concept of GDNN and establish an equivalence between IAT and GDNN when using a finite rank parameterization for the weight kernel.
- We derive explicit expressions for the forward and backward computations of IAT-ReLU, as well as numerical methods to approximate them.
- We propose a generalize the activation pattern $\mathcal{D}(\boldsymbol{z})$ and analyze the continuity and variation of $\mathcal{D}(\boldsymbol{z})$ under different choices of basis functions.
- We provide numerical examples to demonstrate that IAT-ReLU has higher practical expressive power compared to ReLU, and show the impact of basis function selection and discretization on IAT-ReLU.

The rest of this paper is organized as follows: Section 1.1 offers an overview of related literature. Section 2 details the construction of the GDNN as an extension of the traditional DNN. In Section 3, the IAT is introduced and its association with GDNN is explored. Section 4 analyzes and compares IAT-ReLU with the scalar ReLU, focusing on aspects such as smoothness, activation patterns, and the decoupling of the activation patterns. Section 5 presents a range of numerical experiments to assess the influence of basis function choices, and discretization strategy in terms of expressive power. The paper concludes in Section 6 with final remarks and perspectives.

1.1. Related works: Infinitely Wide Neural Network: At initialization, the behavior of a neural network (NN) in the limit of infinite width is equivalent to that of a Gaussian process (GP). This correspondence was initially noted in [13] for NNs with a single hidden layer and was later extended to multilayer NNs in [14, 15]. Additionally, [16] demonstrated that the training dynamics of infinite-width NNs can also be described by a kernel called the Neural Tangent Kernel (NTK), which was later extended to convolutional neural networks by [17]. After discretization, the proposed GDNN can be transformed into a DNN of arbitrary width. However, it is important to note that the notion of width in the discretized GDNN differs significantly from the concept of infinitely wide neural networks in kernel regimes. In the discretized GDNN, the width is derived from the evaluation of the weight integral kernel function on a 2D mesh grid, which leads to non-i.i.d. samples. This is in contrast to the GP regime, which relies on the Central Limit Theorem and assumes i.i.d. parameters. Similarly, the NTK regime is based on overparameterization, whereas the integral kernel in the GDNN is governed by only moderate-sized parameters. Therefore, the discretized GDNN does not align with either the GP regime or the NTK regime.

Integral transform in DNN: The concept of using integral transforms to represent hidden layers in DNNs has been explored in the context of Neural Operator [9, 18], but there are notable distinctions between their approach and ours. Firstly, their focus is primarily on operator learning, specifically mapping PDE parameters to PDE solutions, whereas our work centers around utilizing integral transforms to analyze the expressive power of DNNs. Secondly, they primarily study different ways of characterizing the integral transform and find that Fourier Neural Operator (FNO) [18] works best. In contrast, we focus on the finite rank parameterization of the integral kernel and derive the equivalent IAT as an activation layer, which is now only characterized by the choice of basis functions. Lastly, our use of a 1D hidden state function provides greater analytical tractability. For instance, with the ReLU activation function, we are able to compute the integral transforms analytically, enhancing the interpretability of the model.

Expressive power of ReLU network: The piecewise linear behavior of ReLU networks has been extensively studied, with research investigating convergence[19, 20, 21, 22, 23], generalization ability [24, 25, 26, 22, 27, 28, 29], and expressive power [24, 26, 30, 12]. The linear regions in ReLU networks correspond to inputs with the same activation pattern, and estimating the maximum number of linear pieces has been a focus of a series of works [31, 32, 11, 33, 34, 35]. Empirical evidence [10, 36], however, suggests that the practical number of linear regions is often lower than the theoretical maximum and exhibits minimal changes during training. In our proposed IAT-ReLU network, which becomes a piecewise linear function after discretization, we have control over the number of linear pieces through the choice of discretization. Analytically evaluating IAT-ReLU leads to infinitely many linear regions. Through numerical experiments, we have observed two effects of the number of linear pieces controlled by the mesh size. Firstly, it enhances solution smoothness, thereby improving performance in tasks that require smoothness, such as function fitting. Secondly, a larger number of linear pieces promotes gradient continuity and increases training stability.

2. Generalized Deep Neural Network (GDNN)

In this section, we present the GDNN as a generalization of the traditional DNN, conceptualized by treating the network width as a continuous variable. Subsection 2.1 details the construction of the GDNN, and in subsection 2.2, we provide a parameterization framework for the GDNN.

2.1. The Construction of GDNN.

We begin by constructing the GDNN as a continuous extension of the traditional DNN, from the perspective of width. This concept is illustrated through a basic 3-layer NN, comprising an input layer, an output layer, and one fully connected (FC) layer. These components encapsulate all necessary elements, and deeper structures can be easily achieved by stacking additional FC layers without further complications.

Consider $\boldsymbol{x} \in \mathbb{R}^{d_0}$ as the model input and $\boldsymbol{y} \in \mathbb{R}^{d_3}$ as the NN output. The traditional NN structure is outlined as:

(1a) Input layer: $\boldsymbol{z}^1 = W^0\boldsymbol{x} + \boldsymbol{b}^0$

(1b) FC layer: $\boldsymbol{u}^1 = \sigma(\boldsymbol{z}^1),$

(1c) $\boldsymbol{z}^2 = W^1\boldsymbol{u}^1 + \boldsymbol{b}^1,$

(1d) Output layer: $\boldsymbol{u}^2 = \sigma(\boldsymbol{z}^2),$

(1e) $\boldsymbol{y} = W^2\boldsymbol{u}^2 + \boldsymbol{b}^2,$

where $W^n \in \mathbb{R}^{d_n \times d_{n-1}}$ for $i = 0, 1, 2$ are weight matrices, $\boldsymbol{b}^n \in \mathbb{R}^{d_{n+1}}$, for $i = 0, 1, 2$ are bias vectors and $\sigma(\cdot)$ is the nonlinear activation function.

Inspired by Neural ODEs [8], which conceptualize depth as a continuous variable, our approach uniquely considers network width as a continuous variable. In line with this, the vectors of hidden neurons in the traditional DNN are replaced by state functions, defined within the interval $[-1, 1]$. To accommodate for this change, the GDNN employs integral transforms instead of the affine transformations found in the traditional DNNs. In the input layer of GDNN, the input vector $\boldsymbol{x}$ is used to expand the input weight functions $\boldsymbol{w}^0(s^1)$, creating a continuous state function $z^1(s^1)$ over the interval $[-1, 1]$. The FC layer then applies a pointwise activation $\sigma(\cdot)$ to $z^1(s^1)$, resulting in the activated state function $u^1(s^1)$. This function is then integrated with the weight kernel $w^1(s^2, s^1)$ over s^1, generating the next state function $z^2(s^2)$. In the output layer, the activated state function $u^2(s^2)$ integrates with the output weight functions $\boldsymbol{w}^2(s^2)$, to yield the final output vector $\boldsymbol{y}$. Additionally, bias functions $b^0(s^1)$ and $b^1(s^2)$ are incorporated into the states $z^1(s^1)$ and $z^2(s^2)$, respectively. The complete GDNN structure is thus expressed as follows:

(2a) Input layer: $z^1(s^1) = \boldsymbol{x}^T\boldsymbol{w}^0(s^1) + b^0(s^1)$

(2b) FC layer: $u^1(s^1) = \sigma(z^1(s^1)),$

(2c) $$z^2(s^2) = \int_{-1}^{1} w^1(s^2, s^1)u^1(s^1)ds^1 + b^1(s^2),$$

(2d) Output layer: $u^2(s^2) = \sigma(z^2(s^2)),$

(2e) $$\boldsymbol{y} = \int_{-1}^{1} \boldsymbol{w}^2(s^2)u^2(s^2)ds^2 + \boldsymbol{b}^2,$$

where $z^1, z^2 : [-1, 1] \to \mathbb{R}$ are the state functions, $u^1, u^2 : [-1, 1] \to \mathbb{R}$ are the corresponding activated state function, $\boldsymbol{w}^0 : [-1, 1] \to \mathbb{R}^{d_0}$ is the input weight function, $w^1 : [-1, 1] \times [-1, 1] \to \mathbb{R}$ are weight kernel, $\boldsymbol{w}^2 : [-1, 1] \to \mathbb{R}^{d_3}$ is the output weight function, $b^0, b^1 : [-1, 1] \to \mathbb{R}$ are bias functions and $\boldsymbol{b}^2$ is the same bias vector in Eqs. (1).

DNN as a Special Case of GDNN: By selecting the 2D weight kernel $w^1(\cdot, \cdot)$, the 1D bias functions $b^0(\cdot)$ and $b^1(\cdot)$, and the 1D input/output weight functions $\boldsymbol{w}^2(\cdot)$ and $\boldsymbol{w}^0(\cdot)$ in Eqs. (2) as piecewise constant (PC) functions, as illustrated in the left column of Figure1, we can exactly compute the integral transform in the GDNN. In such scenarios, this integral transform is equivalent to an affine transformation, simplifying the GDNN to a traditional DNN. In other words, the traditional DNN can be perceived as a particular instance of the GDNN, realized by selecting the generalized weight kernels, bias functions, and weight functions as PC functions.

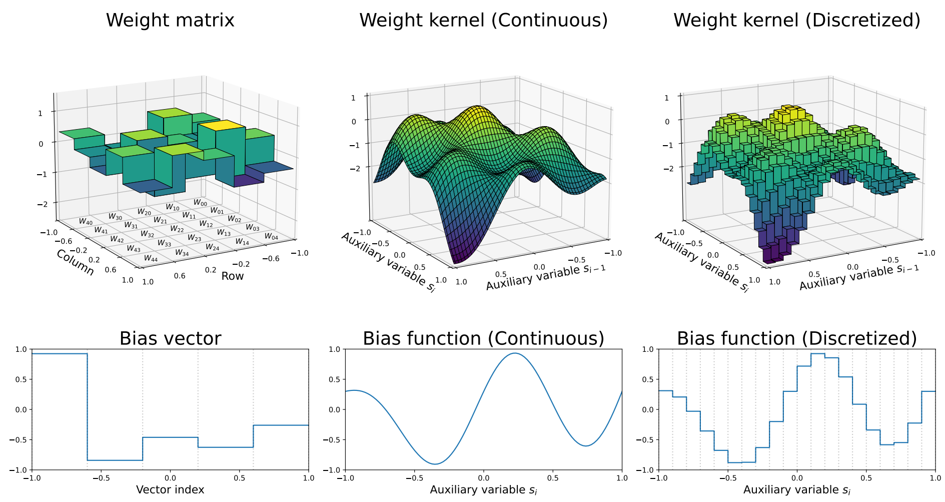

FIGURE 1. The visualization for the weight kernels and bias functions. Left column: The PC weight kernel and bias function, with the height of each segment controlled by the corresponding weight matrix and bias vector values. Middle column: The continuous weight kernel and bias function, as a generalization of the PC weight kernel and bias function. Right column: The discretized version of the continuous weight kernel and bias function, represented in the form of PC functions.

GDNN as an Infinitely Wide DNN: In cases where the weight kernel and bias functions are not piecewise constant, exact computation of the GDNN becomes generally infeasible, necessitating an appropriate discretization approach. In our framework, discretized weight matrices and bias vectors are derived by sampling the weight integral kernels and bias functions at the predetermined mesh points, resulting in matrices in $\mathbb{R}^{M\times M}$ and vectors in $\mathbb{R}^{M}$, with M being the mesh size. After the discretization, integral transforms in GDNN become affine transformations, rendering the discretized GDNN analogous to a traditional DNN equipped with such affine transformations. Notably, the size of these affine transformations depends on the mesh size M, contrasting with traditional DNNs where affine transformations correspond to the fixed number of neurons in each layer. As M tends towards infinity, the discretized GDNN progressively mirrors a finitely wide DNN, ultimately converging to the GDNN in its analytical form. This convergence can also be understood as the approximation of a continuous function using PC functions, as illustrated in the right column of Figure 1.

2.2. Parameterization of GDNN. In the GDNN framework, much like in traditional DNNs, the trainable parameters include the weight functions/kernels, $\boldsymbol{w}^0(\cdot)$,

$w^1(\cdot,\cdot)$, $\boldsymbol{w}^2(\cdot)$, and bias functions/vectors $b^0(\cdot)$, $b^1(\cdot)$, $\boldsymbol{b}^2$. However, the direct training of these infinite-dimensional functions is impractical, and it is necessary to adopt a suitable parameterization for effective training. In our approach, we use 1D basis functions to construct and parameterize all the continuous 1D weight and bias functions. Additionally, the 2D weight integral kernels are formed by directly multiplying these 1D basis functions, a technique known as finite rank parameterization for the integral kernel. The detailed parameterization scheme is as follows:

$$\text{Input layer:}\quad \boldsymbol{w}^0(s^1) = (W^0)^T\boldsymbol{p}^1(s^1) \tag{3a}$$

$$b^0(s^1) = (\boldsymbol{b}^0)^T\boldsymbol{p}^1(s^1), \tag{3b}$$

$$\text{FC layer:}\quad w^1(s^2,s^1) = (\boldsymbol{p}^2(s^2))^T W^1\boldsymbol{q}^1(s^1), \tag{3c}$$

$$b^1(s^2) = (\boldsymbol{b}^1)^T\boldsymbol{p}^2(s^2), \tag{3d}$$

$$\text{Output layer:}\quad \boldsymbol{w}^2(s^2) = W^2\boldsymbol{q}^2(s^2), \tag{3e}$$

where $\{W^0, W^1, W^2, \boldsymbol{b}^1, \boldsymbol{b}^2\}$ are parameters from the same space as those in the DNN described in Eq. (1), and $\boldsymbol{p}^n, \boldsymbol{q}^n : \mathbb{R} \to \mathbb{R}^{d_n}$ for $n = 1, 2$ are the selected collections of 1D input and output basis functions, respectively. In subsequent sections, we demonstrate that this parameterized GDNN is equivalent to a DNN equipped with a special Integral Activation Transform as its activation layers.

3. Integral Activation Transform (IAT)

In this section, we first give the definition of the IAT in Subsection 3.1, then establish its connection with the GDNN in Subsection 3.2, and finally, present the numerical approximation of the IAT in Subsection 3.3.

3.1. Integral Activation Transform (IAT).

Let $\boldsymbol{p}(\cdot) = [p_1(\cdot), ..., p_{d_1}(\cdot)]^T$ and $\boldsymbol{q}(\cdot) = [q_1(\cdot), ..., q_{d_2}(\cdot)]^T$ denote two sets of basis functions, serving as the input and output basis functions for the IAT respectively, where $p_1, ..., p_{d_1}, q_1, ..., q_{d_2} : [-1, 1] \to \mathbb{R}$ are all 1D basis functions. We construct the IAT $\mathcal{I}^{\sigma}_{\boldsymbol{p},\boldsymbol{q}} : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ as a nonlinear activation layer in the following manner:

$$\boldsymbol{u} = \mathcal{I}^{\sigma}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z}) = \int_{-1}^{1} \boldsymbol{q}(s)\sigma(\boldsymbol{z}^T\boldsymbol{p}(s))ds, \tag{4}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ represents the nonlinear activation function, with $\boldsymbol{z} \in \mathbb{R}^{d_1}$ as the input and $\boldsymbol{u} \in \mathbb{R}^{d_2}$ as the output.

Contrary to the typical activation functions in DNNs, which are applied to each component of the input $\boldsymbol{z}$ to directly generate the components of the output $\boldsymbol{u}$, the IAT adopts an alternative approach, operating activation in a function space defined over $[-1, 1]$. In the IAT, the input vector $\boldsymbol{z}$ is initially mapped to a state function $\boldsymbol{z}^T\boldsymbol{p}(s)$ defined on $[-1, 1]$, with $\boldsymbol{z}$ serving as the coefficients for the input basis $\boldsymbol{p}(s)$. Then, the pointwise activation function σ is applied to the state function to introduce the nonlinearity. Finally, this activated state function $\sigma(\boldsymbol{z}^T\boldsymbol{p}(s))$ is integrated with the output basis $\boldsymbol{q}(s)$ to produce the output vector $\boldsymbol{u}$. The conceptual diagram of IAT is depicted in Figure 2.

In IAT, there is flexibility in selecting the input basis $\boldsymbol{p}(\cdot)$, output basis $\boldsymbol{q}(\cdot)$, and activation function $\sigma(\cdot)$. The input basis $\boldsymbol{p}(\cdot)$ plays a crucial role in forming the state function $\boldsymbol{z}^T\boldsymbol{p}(s)$, upon which the nonlinear activation is applied. The choice

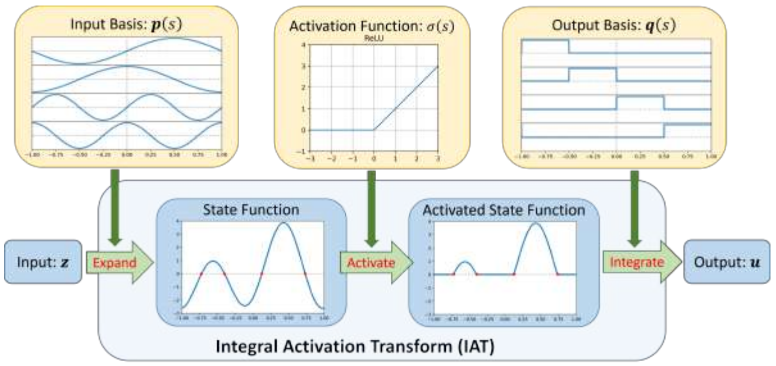
Input Basis: $\boldsymbol{p}(s)$
Activation Function: $\sigma(s)$
ReLU
Output Basis: $\boldsymbol{q}(s)$
State Function
Activated State Function
Input: $\boldsymbol{z}$
Expand
Activate
Integrate
Output: $\boldsymbol{u}$
Integral Activation Transform (IAT)

FIGURE 2. The schematic for IAT with ReLU activation: First, the input vector $\boldsymbol{z}$ is mapped to the state function $\boldsymbol{z}^T\boldsymbol{p}(s)$ by treating $\boldsymbol{z}$ as coefficients of the input basis $\boldsymbol{p}(s)$. Then, a pointwise activation function σ is applied to the state function to introduce nonlinearity. Finally, the activated state function $\sigma(\boldsymbol{z}^T\boldsymbol{p}(s))$ is integrated with the output basis $\boldsymbol{q}(s)$ to produce the output vector $\boldsymbol{u}$. In the case of ReLU activation, the red dots represent the roots of the state function, which determine the segments of the state function that are set to zero.

of input basis significantly influences the interaction between the nonlinearity and the input vector $\boldsymbol{z}$. Conversely, the output basis $\boldsymbol{q}(\cdot)$ is in charge of extracting pertinent information from the activated state function, and different choices here can yield distinct IAT behaviors, a topic that will be explored in Section 5.1.

A specific basis function worthy of mention is the rectangular function, depicted as the output basis $\boldsymbol{q}(s)$ in Figure 2. Rectangular functions represent the simplest form of PC functions. Selecting these rectangular functions for both $\boldsymbol{p}(\cdot)$ and $\boldsymbol{q}(\cdot)$ can lead to PC weight kernels and bias functions shown in Figure 1. Consequently, this choice can simplify the IAT to a traditional element-wise activation function, described as:

$$\mathcal{I}^{\sigma}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z}) = \sigma(\boldsymbol{z}), \tag{5}$$

where $\sigma(\cdot)$ is individually applied to each component of $\boldsymbol{z}$.

3.2. Connection to GDNN. This subsection demonstrates the equivalence of the GDNN as described in Eqs. (2) with its parameterization in Eqs. (3) to a traditional DNN equipped with the IAT, defined in Eq. (4). To proceed, we also assume that all considered basis functions $\boldsymbol{p}^n(\cdot), \boldsymbol{q}^n(\cdot), n = 1, 2$ are linear independent within each collection.

Firstly, in the input layer, the state function $z^1(s^1)$ lies within the span of $\boldsymbol{p}^1(s^1)$, and can be expressed as

$$z^1(s^1) = \left(\boldsymbol{p}^1(s^1)\right)^T \boldsymbol{z}^1,$$

where $\boldsymbol{z}^1$ represents the coefficients for $\boldsymbol{p}^1(s^1)$. By substituting Eqs. (3a,3b) into Eq. (2a) we have

$$\begin{aligned} z^1(s^1) &= \left((W^0)^T\boldsymbol{p}^1(s^1)\right)^T\boldsymbol{x} + \left(\boldsymbol{p}^1(s^1)\right)^T\boldsymbol{b}^0 \\ &= \left(\boldsymbol{p}^1(s^1)\right)^T\left(W^0\boldsymbol{x}+\boldsymbol{b}^0\right). \end{aligned}$$

By linear Independence of $\boldsymbol{p}^1(s^1)$, we obtain

$$\boldsymbol{z}^1 = W^0\boldsymbol{x}+\boldsymbol{b}^0. \tag{6}$$

In the FC layer, $z^2(s^2)$ is within the span of $\boldsymbol{p}^2(s^2)$ and can be represented as

$$z^2(s^2) = \left(\boldsymbol{p}^2(s^2)\right)^T\boldsymbol{z}^2.$$

By substituting Eqs. (3c,3d) into Eq. (2c), we have

$$\begin{aligned} z^2(s^2) &= \int_{-1}^{1}\left(\boldsymbol{p}^2(s^2)\right)^T W^1\boldsymbol{q}^1(s^1)u^1(s^1)ds^1 + \left(\boldsymbol{p}^2(s^2)\right)^T\boldsymbol{b}^1 \\ &= \left(\boldsymbol{p}^2(s^2)\right)^T\left(W^1\int_{-1}^{1}\boldsymbol{q}^1(s^1)u^1(s^1)ds^1+\boldsymbol{b}^1\right) \\ &= \left(\boldsymbol{p}^2(s^2)\right)^T\left(W^1\int_{-1}^{1}\boldsymbol{q}^1(s^1)\sigma((\boldsymbol{p}^1(s^1))^T\boldsymbol{z}^1)ds^1+\boldsymbol{b}^1\right) \\ &= \left(\boldsymbol{p}^2(s^2)\right)^T\left(W^1\mathcal{I}^{\sigma}_{\boldsymbol{p}^1,\boldsymbol{q}^1}(\boldsymbol{z}^1)+\boldsymbol{b}^1\right) \\ &= \left(\boldsymbol{p}^2(s^2)\right)^T\left(W^1\boldsymbol{u}^1+\boldsymbol{b}^1\right). \end{aligned}$$

Following the linear independence of $\boldsymbol{p}^2(s^2)$, we obtain

$$\boldsymbol{z}^2 = W^1\boldsymbol{u}^1+\boldsymbol{b}^1, \tag{7}$$

where

$$\boldsymbol{u}^1 = \mathcal{I}^{\sigma}_{\boldsymbol{p}^1,\boldsymbol{q}^1}(\boldsymbol{z}^1). \tag{8}$$

For the output layer, substituting Eq. (3e) into Eq. (2e) yields

$$\begin{aligned} \boldsymbol{y} &= \int_{-1}^{1} W^2\boldsymbol{q}^2(s^2)u^2(s^2)ds^2+\boldsymbol{b}^2 \\ &= W^2\int_{-1}^{1}\boldsymbol{q}^2(s^2)\sigma(z^2(s^2))ds^2+\boldsymbol{b}^2 \\ &= W^2\int_{-1}^{1}\boldsymbol{q}^2(s^2)\sigma((\boldsymbol{p}^2(s^2))^T\boldsymbol{z}^2)ds^2+\boldsymbol{b}^2 \\ &= W^2\int_{-1}^{1}\boldsymbol{q}^2(s^2)\sigma((\boldsymbol{p}^2(s^2))^T\boldsymbol{z}^2)ds^2+\boldsymbol{b}^2 \\ &= W^2\mathcal{I}^{\sigma}_{\boldsymbol{p}^2,\boldsymbol{q}^2}(\boldsymbol{z}^2)+\boldsymbol{b}^2 \\ &= W^2\boldsymbol{u}^2+\boldsymbol{b}^2. \end{aligned}$$

This gives the last piece of the derivation, and we have

$$\boldsymbol{y} = W^2\boldsymbol{u}^2+\boldsymbol{b}^2, \tag{9}$$

where

$$\boldsymbol{u}^2 = \mathcal{I}^{\sigma}_{\boldsymbol{p}^2,\boldsymbol{q}^2}(\boldsymbol{z}^2). \tag{10}$$

Combining the above derivations in Eqs. (6,7,8,9,10), the GDNN as described in Eqs. (2) with its parameterization in Eqs. (3) can be reformulated as follows:

$$\text{Input layer:} \quad \boldsymbol{z}^1 = W^0\boldsymbol{x} + \boldsymbol{b}^0 \tag{11a}$$

$$\text{FC layer:} \quad \boldsymbol{u}^1 = \mathcal{I}^{\sigma}_{\boldsymbol{p}^1,\boldsymbol{q}^1}(\boldsymbol{z}^1), \tag{11b}$$

$$\boldsymbol{z}^2 = W^1\boldsymbol{u}^1 + \boldsymbol{b}^1, \tag{11c}$$

$$\text{Output layer:} \quad \boldsymbol{u}^2 = \mathcal{I}^{\sigma}_{\boldsymbol{p}^2,\boldsymbol{q}^2}(\boldsymbol{z}^2), \tag{11d}$$

$$\boldsymbol{y} = W^2\boldsymbol{u}^2 + \boldsymbol{b}^2, \tag{11e}$$

which is equivalent to the classical DNN in Eqs. (1) with the element-wise activation function $\sigma(\cdot)$ replaced by IAT $\mathcal{I}^{\sigma}_{\boldsymbol{p}^n,\boldsymbol{q}^n}$ for $n = 1, 2$.

3.3. The approximation of IAT. The IAT, $\mathcal{I}^{\sigma}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$, as defined in Eq. (4) comprises d_2 1D integrals over the interval $[-1, 1]$ and each integral is given by $\int_{-1}^{1} q_i(s)\sigma(\boldsymbol{z}^T\boldsymbol{p}(s))ds$, where $i = 1, \cdots, d_2$. Without making further assumptions on $\boldsymbol{q}(\cdot)$, $\boldsymbol{p}(\cdot)$, and $\sigma(\cdot)$, these integrals can be approximated using the midpoint rule with a uniform partition of $[-1, 1]$. This discretized version of IAT, denoted as $\hat{\mathcal{I}}^{\sigma}_{\boldsymbol{p},\boldsymbol{q}}$, is then expressed as:

$$\hat{\boldsymbol{u}} = \hat{\mathcal{I}}^{\sigma}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z}) = \frac{2Q\sigma(P^T\boldsymbol{z})}{M} \tag{12}$$

where M is the number of partition for the interval $[-1, 1]$ and $P \in \mathbb{R}^{d_1\times M}$ and $Q \in \mathbb{R}^{d_2\times M}$ represent the evaluation of basis functions $\boldsymbol{p}(\cdot)$ and $\boldsymbol{q}(\cdot)$ at the M chosen mesh points. The discretization of the IAT mirrors the discretization process of the weight kernel in GDNN, as discussed in Section 2.1. Specifically, the discretized weight kernel $w^1(\cdot,\cdot)$, as parameterized in Eq. (3c), is expressed as an M by M matrix $\hat{W}^1 = (P^2)^T W^1 Q^1$, where P^2 and Q^1 are matrices that correspond to the evaluations of the basis functions $\boldsymbol{p}^2(\cdot)$ and $\boldsymbol{q}^1(\cdot)$, respectively.

Furthermore, the gradient of the discretized IAT-ReLU, as defined in Eq. (12), can be computed as:

$$\nabla_{\boldsymbol{z}}\hat{\mathcal{I}}^{\sigma}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z}) = \frac{2Q\,\mathrm{diag}(\sigma'(P^T\boldsymbol{z}))\,P^T}{M}. \tag{13}$$

According to [37], when $\sigma(\boldsymbol{z}^T\boldsymbol{p}(s))$ is differentiable for all $\boldsymbol{z}$ and almost all $s \in [-1, 1]$, the Leibniz integral rule can be applied to calculate the gradient of $\mathcal{I}^{\sigma}_{\boldsymbol{p},\boldsymbol{q}}(\cdot)$. By interchanging the integration and differentiation, the gradient is given by:

$$\nabla_{\boldsymbol{z}}\mathcal{I}^{\sigma}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z}) = \int_{-1}^{1} \boldsymbol{q}(s)(\boldsymbol{p}(s))^T\sigma'(\boldsymbol{z}^T\boldsymbol{p}(s))ds. \tag{14}$$

This expression provides an analytical formula for the gradient, under the differentiability condition of $\sigma(\cdot)$. Notably, this analytical gradient aligns with the numerical gradient in Eq. (13). However, dealing with non-smooth activation functions like ReLU requires more careful consideration when interchanging integration and differentiation, a topic further explored in Section 4.

4. IAT-ReLU

In this section, we focus on a variant of the IAT using the ReLU as the activation function, referred to as IAT-ReLU and denoted by $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}$. In Subsection 4.1, we begin by establishing explicit formulas for the forward and backward computations in IAT-ReLU, demonstrating that IAT-ReLU generalizes the scalar ReLU. Subsequently, we investigate the activation patterns of IAT-ReLU in Subsection 4.2 and discuss its computational approach in Subsection 4.3. Finally, we explore decoupling the activation pattern from model parameters in Subsection 4.4.

4.1. IAT-ReLU as a generalization of ReLU. We begin by establishing that IAT-ReLU is a natural generalization of the scalar ReLU. For simplicity, we use $\phi(\cdot)$ to denote both scalar and element-wise ReLU, depending on the input type. When ReLU is applied element-wise to a vector $\boldsymbol{z} = [z_1, \ldots, z_d] \in \mathbb{R}^d$, the operation can be expressed in the linear form as follows:

$$\phi(\boldsymbol{z}) = [\mathbf{1}_{z_1 \geq 0}(\boldsymbol{z}) z_1, ..., \mathbf{1}_{z_d \geq 0}(\boldsymbol{z}) z_d]^T = S^{\phi}(\boldsymbol{z})\boldsymbol{z} \tag{15}$$

where $S^{\phi}(\boldsymbol{z}) = \mathrm{diag}([\mathbf{1}_{z_1 \geq 0}(\boldsymbol{z}), ..., \mathbf{1}_{z_d \geq 0}(\boldsymbol{z})]) \in \mathbb{R}^{d \times d}$ is called the activation matrix. This activation matrix is piecewise constant, implying that $\nabla_{\boldsymbol{z}} S^{\phi}(\boldsymbol{z}) = 0$ almost everywhere. And it follows that the gradient of the element-wise ReLU is also equivalent to its activation matrix, expressed as:

$$\nabla_{\boldsymbol{z}} \phi(\boldsymbol{z}) = S^{\phi}(\boldsymbol{z}). \tag{16}$$

We proceed to establish that IAT-ReLU exhibits similar properties to the scalar ReLU. For IAT-ReLU, we can rewrite its expression as follows:

$$\begin{aligned}
\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z}) &= \int_{-1}^{1} \boldsymbol{q}(s)\sigma(\boldsymbol{z}^T \boldsymbol{p}(s))ds \\
&= \int_{-1}^{1} \boldsymbol{q}(s)\mathbf{1}_{\boldsymbol{z}^T\boldsymbol{p}(s)>0}(s)(\boldsymbol{z}^T\boldsymbol{p}(s))ds \\
&= \int_{\{s\in[-1,1]:\boldsymbol{z}^T\boldsymbol{p}(s)>0\}} \boldsymbol{q}(s)(\boldsymbol{p}(s))^T ds\boldsymbol{z} \\
&= S(\mathcal{D}(\boldsymbol{z}))\boldsymbol{z}.
\end{aligned}$$

This implies that the IAT-ReLU has the same linear form as follows:

$$\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z}) = S(\mathcal{D}(\boldsymbol{z}))\boldsymbol{z}, \tag{17}$$

where $S(\mathcal{D}(\boldsymbol{z})) \in \mathbb{R}^{d\times d}$ is the activation matrix for IAT-ReLU. Its expression is given by

$$S(\mathcal{D}(\boldsymbol{z})) = \int_{\{s\in\mathcal{D}(\boldsymbol{z})\}} \boldsymbol{q}(s)\boldsymbol{p}^T(s)ds. \tag{18}$$

where $\mathcal{D}(\boldsymbol{z})$ is called the activation pattern and is defined as:

$$\mathcal{D}(\boldsymbol{z}) = \{s \in [-1, 1] : \boldsymbol{p}^T(s)\boldsymbol{z} > 0\}. \tag{19}$$

This formulation confirms that IAT-ReLU maintains the linear form analogous to the scalar ReLU.

We now derive the gradient of $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$. Let the state function be denoted as $f(s, \boldsymbol{z}) = \boldsymbol{z}^T\boldsymbol{p}(s)$ and we can first rewrite $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ as

$$\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z}) = \int_{\mathcal{D}(\boldsymbol{z})} \boldsymbol{q}(s)f(s, \boldsymbol{z})ds.$$

The dependence of $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ on the input $\boldsymbol{z}$ is through two paths: the state function $f(s, \boldsymbol{z})$ and the activation pattern $\mathcal{D}(\boldsymbol{z})$. The set $\mathcal{D}(\boldsymbol{z})$, a subset of $[-1, 1]$, consists of disjoint intervals characterized by boundary points $r_k, k = 1, \ldots, K$, at which

$f(\cdot,\boldsymbol{z})$ changes sign. Therefore, the gradient of $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ will have the general expression as follows:

$$\nabla_{\boldsymbol{z}}\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z}) = S(\mathcal{D}(\boldsymbol{z})) + \sum_{k=1}^{K} f(r_k,\boldsymbol{z})\boldsymbol{q}(r_k)(\nabla_{\boldsymbol{z}} r_k)^T \tag{20}$$

where $\nabla_{\boldsymbol{z}} r_k$ is the gradient of boundary point r_k with respect to the input $\boldsymbol{z}$. In the above expression, the first term represents the gradient through $f(s,\boldsymbol{z})$ and the second sum stands for the gradient on $\mathcal{D}(\boldsymbol{z})$ through each r_k.

While the dependence through $f(s,\boldsymbol{z})$ always exists, we only need to analyze the dependence through $\mathcal{D}(\boldsymbol{z})$ via r_k, and we identify two scenarios for such dependency. In the first scenario, $f(\cdot,\boldsymbol{z})$ intersects the x-axis at r_k, making r_k a root of $f(\cdot,\boldsymbol{z})$. Here, r_k has a nonzero gradient with respect to $\boldsymbol{z}$, i.e., $\nabla_{\boldsymbol{z}} r_k \neq 0$. However, $f(r_k,\boldsymbol{z}) = 0$ since r_k is the root, resulting in $f(r_k,\boldsymbol{z})\boldsymbol{q}(r_k)(\nabla_{\boldsymbol{z}} r_k)^T = 0$. In the second scenario, $f(\cdot,\boldsymbol{z})$ changes sign at r_k due to a jump discontinuity. From the definition of the $f(\cdot,\boldsymbol{z})$, the discontinuities in $f(\cdot,\boldsymbol{z})$ can only come from the discontinuities within basis $\boldsymbol{p}(\cdot)$, and are independent of the value of $\boldsymbol{z}$. Therefore, we have $\nabla_{\boldsymbol{z}} r_k = 0$, which leads to $f(r_k,\boldsymbol{z})\boldsymbol{q}(r_k)(\nabla_{\boldsymbol{z}} r_k)^T = 0$ despite $f(r_k,\boldsymbol{z}) \neq 0$. Combining the both cases, the gradient of $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ is simplified to:

$$\nabla_{\boldsymbol{z}}\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z}) = S(\mathcal{D}(\boldsymbol{z})). \tag{21}$$

This implies that the gradient of $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z}) = S(\mathcal{D}(\boldsymbol{z}))\boldsymbol{z}$ with respect to $\boldsymbol{z}$ equals $S(\mathcal{D}(\boldsymbol{z}))$, despite the non-zero gradient of $S(\mathcal{D}(\boldsymbol{z}))$ on $\boldsymbol{z}$.

By comparing Eq. (15) with Eq. (17), and Eq. (16) with Eq. (21), it becomes evident that both ReLU and IAT-ReLU exhibit a shared linear form, as characterized by their activation matrices, respectively. Additionally, their gradients are found to be equivalent to their activation matrices. The primary difference between them resides in the specific values taken by the activation matrix $S(\mathcal{D}(\boldsymbol{z}))$ as a function of $\boldsymbol{z}$. The forthcoming subsections will undertake a detailed investigation of the behavior of the activation matrix $S(\mathcal{D}(\boldsymbol{z}))$.

In addition, the gradient formulation in Eq. (21), derived for the non-smooth activation ReLU, aligns with the gradient in Eq. (14), obtained under the assumption of a smooth activation function. This alignment suggests that the numerical approximation in Eq. (12) remains consistent for IAT-ReLU as well. This consistency underscores the robustness of the IAT framework, capable of accommodating both smooth and non-smooth activation functions without compromising the integrity of its gradient approximation.

4.2. The activation pattern of IAT-ReLU. From Eq. (17), we observe that the nonlinearity within $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ is determined by the variation of $S(\mathcal{D}(\boldsymbol{z}))$. Specifically, when $S(\mathcal{D}(\boldsymbol{z}))$ remains locally constant, $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ behaves as a linear map. Further inspection of Eq. (18) reveals that the activation matrix $S(\mathcal{D}(\boldsymbol{z}))$ is dependent on the input $\boldsymbol{z}$ through the activation pattern $\mathcal{D}(\boldsymbol{z})$. It is important to note that the function $S(\cdot)$ itself is not directly related to $\boldsymbol{z}$. Hence, the subsequent subsections focus on analyzing the variation of $\mathcal{D}(\boldsymbol{z})$ as a function of $\boldsymbol{z}$. We will investigate two distinct types of variations in $\mathcal{D}(\boldsymbol{z})$: local variations and global variations.

4.2.1. The local variation of $\mathcal{D}(\boldsymbol{z})$. The local variation of $\mathcal{D}(\boldsymbol{z})$ concerns how $\mathcal{D}(\boldsymbol{z})$ changes relate to local variations in $\boldsymbol{z}$. Here, we specifically investigate two aspects of such local variation: the presence of local changes in $\mathcal{D}(\boldsymbol{z})$ as $\boldsymbol{z}$ varies locally and the continuity of such changes in $\boldsymbol{z}$.

As discussed above, we know that $\mathcal{D}(\boldsymbol{z})$ is a subset of $[-1, 1]$, and there are two scenarios on how $\mathcal{D}(\boldsymbol{z})$ is determined by $\boldsymbol{z}$:

- In the first scenario, $\mathcal{D}(\boldsymbol{z})$ is determined by the roots of $f(\cdot; \boldsymbol{z}) = \boldsymbol{z}^T \boldsymbol{p}(\cdot)$, which requires $f(\cdot; \boldsymbol{z})$ to be continuous, and this is achieved by choosing continuous input basis functions $\boldsymbol{p}(\cdot)$. Generally speaking, variations in the coefficients $\boldsymbol{z}$ induce perturbations in the roots of $f(\cdot; \boldsymbol{z})$, ultimately causing changes in $\mathcal{D}(\boldsymbol{z})$. Therefore, when $\mathcal{D}(\boldsymbol{z})$ determined by the roots of $f(\cdot; \boldsymbol{z})$, there are local changes in $\mathcal{D}(\boldsymbol{z})$ as $\boldsymbol{z}$ changes. In addition, if the r_k is a single root of $f(\cdot; \boldsymbol{z})$, the gradient is given by

$$\frac{\partial r_k}{\partial \boldsymbol{z}} = -\frac{\boldsymbol{p}(r_k)}{\boldsymbol{z}^T \boldsymbol{p}'(r_k)}. \tag{22}$$

  If $\boldsymbol{p}(\cdot)$ and $\boldsymbol{p}'(\cdot)$ are both continuous and given that $\mathcal{D}(\boldsymbol{z})$ is solely determined by single roots, then $\mathcal{D}(\boldsymbol{z})$ will be continuous in $\boldsymbol{z}$. In addition, if $\boldsymbol{p}(\cdot)$ are analytic functions, the continuity of $\mathcal{D}(\boldsymbol{z})$ in $\boldsymbol{z}$ is ensured, regardless of the root types, owing to the fact that the roots of analytic function are continuous functions of their coefficients.

- In the second scenario, $\mathcal{D}(\boldsymbol{z})$ is determined by the discontinuity of $f(\cdot; \boldsymbol{z})$, which comes from the discontinuities in $\boldsymbol{p}(\cdot)$. The rectangular basis functions, illustrated as the output basis in Figure 2, exemplify this situation. Since the discontinuities in $\boldsymbol{p}(s)$ do not change, $\mathcal{D}(\boldsymbol{z})$ is merely selecting the sub-intervals on which $f(\cdot; \boldsymbol{z})$ is positive, and the sub-intervals are defined by the fixed discontinuities in $\boldsymbol{p}(s)$. Consequently, $\mathcal{D}(\boldsymbol{z})$ is confined to a finite set of possible choices. As $\boldsymbol{z}$ changes, based on the active discontinuities that separate the positive and negative part of $f(\cdot; \boldsymbol{z})$, $\mathcal{D}(\boldsymbol{z})$ either remains unchanged or undergoes an abrupt transition to another configuration. Thus, when $\mathcal{D}(\boldsymbol{z})$ is shaped by the discontinuity of $f(\cdot; \boldsymbol{z})$, local changes in $\mathcal{D}(\boldsymbol{z})$ from $\boldsymbol{z}$ are absent except for jumps. This stands in stark contrast to the first scenario, where $\mathcal{D}(\boldsymbol{z})$ can undergo continuous changes as $\boldsymbol{z}$ varies. Additionally, in this second scenario, it is obvious that $\mathcal{D}(\boldsymbol{z})$ lacks continuity in $\boldsymbol{z}$.

From the integral definition in Eq. (18), it can be seen that $S(\cdot)$ is absolute continuity with respect to $\mathcal{D}(\boldsymbol{z})$. The aforementioned properties of $\mathcal{D}(\boldsymbol{z})$ can be directly extended to characterize the activation matrix $S(\mathcal{D}(\boldsymbol{z}))$, which governs the gradient $\nabla_{\boldsymbol{z}} \mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$, as expressed in Eq. (21).

Regarding the mapping $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ itself, the continuity of $\mathcal{D}(\boldsymbol{z})$ dictates the smoothness of $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$, since the continuity of $\mathcal{D}(\boldsymbol{z})$ decides the continuity of $S(\mathcal{D}(\boldsymbol{z}))$ in $\boldsymbol{z}$. Therefore, a continuous $\mathcal{D}(\boldsymbol{z})$ yields a smooth $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$, and vise versa.

In the context of training a NN with $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$, two major consequences arise. The first one is the plateau phenomenon observed during training, characterized by an initial rapid loss reduction, followed by an extended stagnation period before another rapid decrease. This behavior is commonly observed in NNs with ReLU activation in the literature. The plateau is a consequence of $S(\mathcal{D}(\boldsymbol{z}))$ being piecewise constant, which, in turn, results from the piecewise constant nature of $\mathcal{D}(\boldsymbol{z})$. If

$\mathcal{D}(\boldsymbol{z})$ remains static during training, the nonlinearity $S(\mathcal{D}(\boldsymbol{z}))$ remains unchanged as the model parameters are updated, effectively making the NN behave like a linear model. Gradient descent can rapidly minimize the loss within the subspace defined by the fixed $S(\mathcal{D}(\boldsymbol{z}))$ [38], leading to a quick initial loss decrease followed by a prolonged stagnation period until a more favorable $\mathcal{D}(\boldsymbol{z})$ emerges, if at all. Continuous changes in $\mathcal{D}(\boldsymbol{z})$ as the model parameters are updated prevent gradient descent from saturating within a fixed subspace, reducing the likelihood of the model getting stuck in a local minimum and thereby enhancing overall trainability.

The second consequence is training stability, a property derived from the continuity of the gradient $S(\mathcal{D}(\boldsymbol{z}))$, which is ultimately determined by the continuity of $\mathcal{D}(\boldsymbol{z})$. When the gradient $S(\mathcal{D}(\boldsymbol{z}))$ is discontinuous, slight changes in the parameters could lead to significant jumps in the gradient, leading to unstable or divergent training dynamics. This instability is often manifested by a highly fluctuating loss curve. Common strategies to address this issue involve using a smaller learning rate, which, however, slows down training, or overparameterizing the model to control the expected changes in the activation pattern, aiming to stabilize the training process [19]. In contrast, the use of analytic $\boldsymbol{p}(\cdot)$ to achieve a continuous $\mathcal{D}(\boldsymbol{z})$ ensures that the gradient remains continuous with respect to the model parameters, eliminating unexpected jumps. This continuity in the gradient facilitates a much smoother decay of the loss during the training process compared to scenarios with a discontinuous gradient.

4.2.2. The global variation of $\mathcal{D}(\boldsymbol{z})$. The global variation of $\mathcal{D}(\boldsymbol{z})$ assesses how much $\mathcal{D}(\boldsymbol{z})$ varies across the input domain. In Eq. (17), the nonlinearity of $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ is determined by changes in the activation matrix $S(\mathcal{D}(\boldsymbol{z}))$ via $\mathcal{D}(\boldsymbol{z})$. Specifically, if a ReLU network exhibits a constant activation pattern in a region, the model reduces to a linear map in that region, limiting its expressive power. Many researchers posit that the number of activation patterns (linear pieces) in a ReLU network contributes to its expressiveness [32]. To enhance a ReLU network's expressiveness, diverse activation patterns are preferred across the input space. This implies that as the input varies, the activation pattern should change to introduce more linear pieces. With a greater number of distinct activation patterns corresponding to various input space regions, the network achieves enhanced expressive capacity to model intricate relationships and capture diverse data patterns.

Conversely, with the selection of a continuous input basis $\boldsymbol{p}(\cdot)$, IAT-ReLU can manifest a continuous activation pattern, akin to finitely many linear pieces. Each linear piece represents a straight line passing through the origin, as the positive part of $f(\cdot;\boldsymbol{z})$ remains unchanged when the input $\boldsymbol{z}$ is scaled to $a\boldsymbol{z}$, where a is a non-zero constant. However, an infinite number of linear pieces does not necessarily imply increased expressive power if the activation pattern only exhibits slight changes across the input domain. Therefore, choosing an appropriate $\boldsymbol{p}(\cdot)$ is crucial to ensure that the activation pattern $\mathcal{D}(\boldsymbol{z})$ explores a diverse range of subsets in the interval $[-1, 1]$ as $\boldsymbol{z}$ varies. As mentioned earlier, the activation pattern $\mathcal{D}(\boldsymbol{z})$ is determined by the roots of $f(\cdot;\boldsymbol{z})$. In selecting basis functions, two strategies can be considered to enhance the variation of $\mathcal{D}(\boldsymbol{z})$ by increasing the diversity of roots in $f(\cdot;\boldsymbol{z})$ as $\boldsymbol{z}$ varies.

- Use basis functions with zero integral: Using input basis functions $\boldsymbol{p}(\cdot)$ each with zero integral ensures that the state function $f(\cdot,\boldsymbol{z})$ also possesses a zero integral. This guarantees that $f(\cdot,\boldsymbol{z})$ intersects the x-axis, leading to at least one root or two roots if $\boldsymbol{p}(\cdot)$ are also cyclic. This condition prevents situations where the $f(\cdot, \boldsymbol{z})$ lies entirely above or below zero. Additionally, each $\boldsymbol{z}$ will activate approximately half of $[-1, 1]$, resulting in a more balanced distribution of $\mathcal{D}(\boldsymbol{z})$ as a subset of $[-1, 1]$.

- Use basis functions with large total variations: Another strategy is to select basis functions characterized by large total variations, with typical examples being global basis functions. These functions are sensitive to small changes in any component of the input $\boldsymbol{z}$, inducing substantial variations in the state function $f(\cdot; \boldsymbol{z})$ as well as its roots. As a result, the activation pattern will demonstrate increased diversity across the input space, potentially granting the model greater nonlinearity to capture diverse patterns in the data and enhance its expressive power.

4.3. The approximation of IAT-ReLU. The gradient of $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ in Eq. (21) is equivalent to the gradient of IAT for smooth activation action in Eq. (14). Therefore, the discretization method used in Eq. (12) can also be applied to $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$, and we denote this discretized version as $\hat{\mathcal{I}}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$. Under this discretization, the activation pattern $\mathcal{D}(\boldsymbol{z}) \subset [-1, 1]$ is represented by the discretized activation pattern $\hat{\mathcal{D}}(\boldsymbol{z}) \in \mathbb{R}^M$, which captures the sign of $f(\cdot, \boldsymbol{z})$ on the M chosen evaluation points. As a result, $\hat{\mathcal{D}}(\boldsymbol{z})$ remains piecewise constant, as very small perturbations in $\boldsymbol{z}$ may not change the sign of $f(\cdot, \boldsymbol{z})$ on the evaluation points if the evaluation points are not dense enough. Therefore, $\hat{\mathcal{I}}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ remains a piecewise linear function. However, by choosing a large number of evaluation points M, we can make each linear piece smaller and make $\hat{\mathcal{I}}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ smoother while still being piecewise linear. As M approaches infinity, we obtain infinitely many linear pieces, and $\hat{\mathcal{I}}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ converges to $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ which is a smooth function. An example of the above discussion is demonstrated in Figure 3. For $M = 10$ and $M = 20$, the locations where the activation pattern changes are marked by the green dashed lines and we consider $M = 10,000$ as an example of the continuous activation pattern. As we increase the mesh size M, each linear piece becomes smaller, and the fitted solution becomes smoother while still being a piecewise linear function. It becomes a truly smooth function when the activation pattern is continuous. For discrete patterns ($M = 10$ and 20), the decoupled model exhibits discontinuities where the activation pattern changes. On the other hand, the continuous activation pattern still provides a continuous solution even after we decouple the activation pattern from the parameters that generate them.

Alternatively, to circumvent the reliance on numerical integration techniques, we can approximate the IAT-ReLU computation nearly exactly by analytically solving the integral in Eq. (4). This approach involves accurately finding the roots of the state function, which are illustrated as the red dots in Figure 2. This can be accomplished either through root-finding algorithms like bisection or Newton's method, or by employing specific classes of basis functions. For instance, when using piecewise linear basis functions, the state function becomes piecewise linear, allowing for the roots to be analytically determined for each linear segment. Once these roots are identified, the integral can be computed analytically by applying the fundamental theorem of calculus. It is worth noting that the root-finding procedure can be performed off the computation graph during forward propagation since gradients from $\boldsymbol{z}$ to roots are multiplied by zero in the gradient of $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$. By achieving high-accuracy root finding, we can obtain a continuous activation pattern

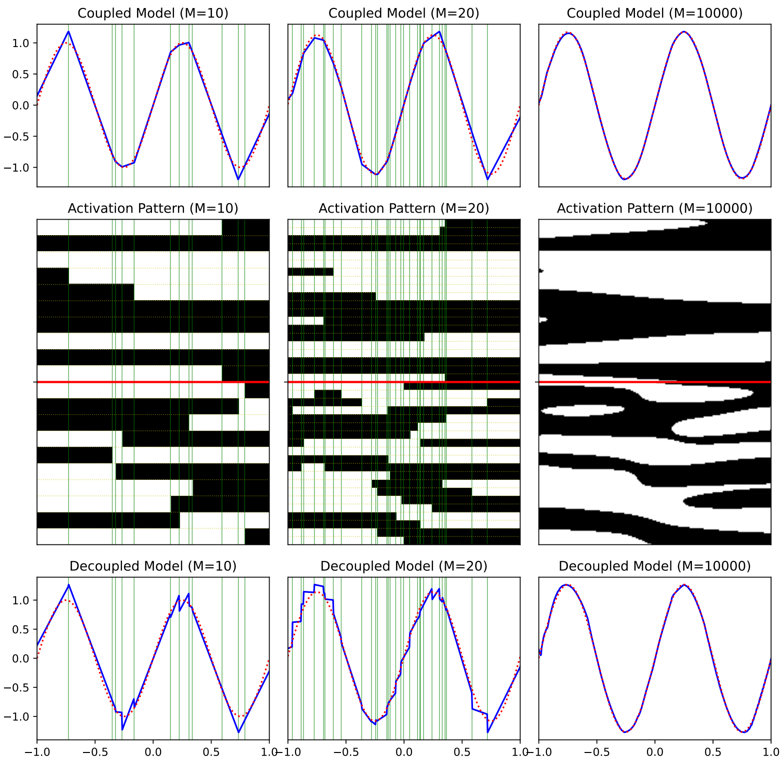

FIGURE 3. Activation pattern for 3-layer IAT-ReLU network in Eqs. (11) with 10 neurons in each layer for fitting $\sin(2\pi x)$ in $[-1, 1]$. Top row: The fitted solution for original IAT-ReLU the model. Middle row: The activation patterns at the two hidden layers. The x-axis corresponds to the input space, while the y-axis represents the two activation pattern spaces, with the first hidden layer positioned above the red line and the second hidden layer below the red line. Bottom row: The fitted solution for the decoupled model, where the activation pattern is taken from the trained model in the middle row. From left to right, we show the results for M=10,20, and 10000, respectively.

$\mathcal{D}(\boldsymbol{z})$ as a subset of $[-1, 1]$, resulting in a smooth $\mathcal{I}^{\mathrm{ReLU}}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$. However, this approach requires more computational resources for the root-finding procedure compared to the computational cost of the two extra matrix multiplications involved in Eq. (12).

Remark 1 (Higher order derivative)**.** *For applications that require higher-order derivatives, such as Physics-Informed Neural Networks (PINN), it is necessary to compute* $\mathcal{I}^{ReLU}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ *analytically instead of using the discretized version* $\hat{\mathcal{I}}^{ReLU}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$*, as the latter has zero second-order derivatives everywhere, similar to the ReLU activation. To capture higher-order derivatives, it is important to preserve the gradient between* $\boldsymbol{z}$ *and the roots* r_k *when computing* $\mathcal{I}^{ReLU}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ *analytically. This can be achieved by numerically inserting the value of the gradient from Eq.* (22)

into the roots r_k obtained from the root-finding algorithms. Alternatively, if the basis functions $\boldsymbol{p}(\cdot)$ are piecewise linear, it is possible to derive an explicit expression for the root in terms of $\boldsymbol{z}$. However, this specific aspect is left as future work. In the scope of this paper, we focus on tasks that do not require higher-order derivatives, such as classification and regression.

Remark 2 (Scaling of the basis functions)**.** *In addition to choosing the shape of the basis functions $\boldsymbol{p}(\cdot)$ and $\boldsymbol{q}(\cdot)$, it is also important to select a proper scaling constant for them. It is worth noting that $\mathcal{I}^{ReLU}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ exhibits homogeneity with respect to the scaling of $\boldsymbol{p}(\cdot)$, $\boldsymbol{q}(\cdot)$, and $\boldsymbol{z}$. Let a_1, a_2 and a_3 be three scalars, the homogeneity of $\mathcal{I}^{ReLU}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z})$ means*

$$\mathcal{I}^{ReLU}_{a_1\boldsymbol{p},a_2\boldsymbol{q}}(a_3\boldsymbol{z}) = a_1a_2a_3\mathcal{I}^{ReLU}_{\boldsymbol{p},\boldsymbol{q}}(\boldsymbol{z}). \tag{23}$$

This equation shows that scaling $\boldsymbol{p}(\cdot)$, $\boldsymbol{q}(\cdot)$, and $\boldsymbol{z}$ does not change the overall nonlinearity of the function. One simple trick that can be used to avoid the scaling issue is to apply batch normalization after the IAT-ReLU layer. Batch normalization can help standardize the IAT-ReLU and make them more robust to scaling variations.

4.4. The decoupled IAT-ReLU network. When considering the IAT-ReLU as an activation layer in a DNN, an interesting aspect is to decouple the activation matrices from the hidden states that generate them. For the purpose of demonstration, let's consider the same 3-layer DNN described in Eqs. (1). Using the linear form from Eq. (17), we can write the DNN in a similar deep linear form as follows:

$$\boldsymbol{y} = W^2(S(D^2)(W^1(S(D^1)(W^0\boldsymbol{x} + b^0) + b^1)) + b^2 \tag{24}$$

where $D^1 = \mathcal{D}(\boldsymbol{z}^1)$ and $D^2 = \mathcal{D}(\boldsymbol{z}^2)$ is the activation patterns of hidden states $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$, respectively. We can also write Eq. (24) in a more general form as follows:

$$\boldsymbol{y} = F(\boldsymbol{x}, \mathbf{W}, \boldsymbol{\mathcal{D}}(\boldsymbol{x}, \mathbf{W})) \tag{25}$$

where $\mathbf{W} = [W^0, b^0, W^1, b^1, W^2, b^2]$ represents all the parameters of the DNN, and $\boldsymbol{\mathcal{D}}(\boldsymbol{x}, \mathbf{W}) = [\mathcal{D}^1(\boldsymbol{x}, \mathbf{W}), \mathcal{D}^2(\boldsymbol{x}, \mathbf{W})]$ represents all the activation patterns as a function of the model input $\boldsymbol{x}$ and the parameter $\mathbf{W}$.

Indeed, we can observe that the nonlinearity in the model is solely determined by the activation pattern $\boldsymbol{\mathcal{D}}(\boldsymbol{x}, \mathbf{W})$. When $\boldsymbol{\mathcal{D}}(\boldsymbol{x}, \mathbf{W})$ remains constant with respect to $\boldsymbol{x}$ and $\mathbf{W}$, the model output $\boldsymbol{y}$ exhibits a linear relationship with the input $\boldsymbol{x}$ and each parameter pair (W_n, b^n), where $n = 0, 1, 2$. Thus, the $\mathbf{W}$ in the second argument of $F(\cdot,\cdot,\cdot)$ controls the linearity of the mapping from $\boldsymbol{x}$ to $\boldsymbol{y}$, while the $\mathbf{W}$ in $\boldsymbol{\mathcal{D}}(\boldsymbol{x}, \mathbf{W})$ controls the nonlinearity from $\boldsymbol{x}$ to $\boldsymbol{y}$.

For instance, in the case of a ReLU DNN, $\boldsymbol{\mathcal{D}}(\boldsymbol{x}, \mathbf{W})$ characterizes each linear piece, and the $\mathbf{W}$ in the second position of $F(\cdot,\cdot,\cdot)$ determines the linear mapping within each piece. We also believe that using the same $\mathbf{W}$ to control both the linear and nonlinear aspects of the model simultaneously makes the optimization for $\mathbf{W}$ challenging. This is in contrast to the traditional approach where the mesh structure, which can be viewed as the linear pieces represented by $\boldsymbol{\mathcal{D}}(\boldsymbol{x}, \mathbf{W})$, is first determined, and subsequent computations are performed within each linear piece, which is governed by the $\mathbf{W}$ in the second position of $F(\cdot,\cdot,\cdot)$.

Therefore, a natural idea is to decouple the linearity and nonlinearity of the model into two collections of parameters. In other words, we can rewrite Eq. (25) as follows:

$$\boldsymbol{y} = F(\boldsymbol{x}, \mathbf{W}_1, \boldsymbol{\mathcal{D}}(\boldsymbol{x}, \mathbf{W}_2)) \tag{26}$$

where $\mathbf{W}_1$ is the parameters controlling the linearity and $\mathbf{W}_2$ represents the parameters associated with the nonlinearity through the activation pattern.

This idea has been explored in the work of [39] for ReLU networks. However, decoupling the activation matrix in ReLU networks poses two major problems. First, there is the issue of discontinuity. If the activation pattern jumps when the hidden state is not zero, it introduces a discontinuity in the model. For coupled ReLU, this issue is avoided because the activation pattern only changes when the state is at zero. The second problem is the inability to learn the parameters $\mathbf{W}_2$ that generate a good activation pattern. Since there is no gradient flowing through the discrete pattern, it becomes impossible to optimize and update the parameters through gradient-based optimizers. As a result, these parameters can only be randomly initialized, limiting the model's ability to learn an effective activation pattern based on the data.

For IAT-ReLU with continuous input basis $\boldsymbol{p}(\cdot)$, we do not have the aforementioned problems. Firstly, the activation pattern is a continuous function of the model input, and after decoupling, the resulting model will still be continuous, which is demonstrated in Figure 3. This ensures a continuous mapping between the input and output. Secondly, for continuous $\boldsymbol{p}(\cdot)$, the pattern is determined by the roots and these roots have non-zero gradients flowing through them. This makes active learning of the activation pattern through gradient-based optimizer possible, in contrast to the previous case with ReLU where the pattern has zero gradient everywhere. Thirdly, another advantage is that all activation patterns reside in the same space $[-1, 1]$, regardless of the network width. This enables the transfer of patterns between networks with different widths, providing more flexibility. In the case of ReLU, the pattern is an $\mathbb{R}^d$ vector, and it needs to match the width of another network, limiting its transferability.

5. Numerical experiments

In this section, we conduct numerical experiments to assess the expressive power of IAT-ReLU. We focus on two tasks: fitting a high-frequency function and random label memorization, which are shown in Figure 4. Both tasks require a high degree of expressive power to achieve good performance. The first task involves approximating a periodic function using piecewise linear functions, which typically require a large number of linear pieces. The second task evaluates the model's ability to memorize and recall random labels. While smoothness is not crucial for label memorization, the task still demands a high level of expressive power. By examining the performance of IAT-ReLU on these tasks, we can gain insights into its ability to handle complex functions and exhibit exceptional expressive power.

5.1. Choice of the basis functions. When selecting the basis function for IAT-ReLU applied to DNN, we consider four properties: continuity, smoothness, global variation, and zero integral. Based on these properties, six candidate basis functions are included in our numerical experiments as visualized in Figure 5, and they are listed as follows. The Rectangular (Rect) basis functions are local, discontinuous, and exhibit a piecewise constant activation pattern. Choosing the Rect basis as both input and output basis reduces IAT-ReLU to scalar ReLU. The Piecewise Linear (PWL) basis serves as the continuous counterpart to the Rect basis, featuring locally varying activation patterns. Piecewise Quadratic (PWQ) basis functions are

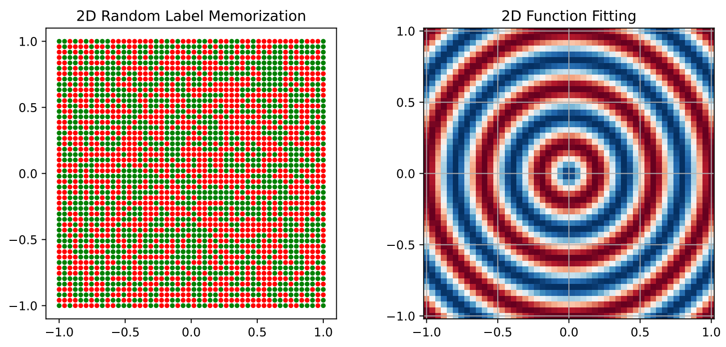

FIGURE 4. Visualization of the test problems to demonstrate the expressive power. Left: Random label memorization, where color represents the label (-1 and 1) and labels are provided on 50 by 50 uniform mesh points in $[-1,1]^2$. Right: high-frequency function in 2D space. The target function is sampled on 50 by 50 uniform mesh points in $[-1,1]^2$.

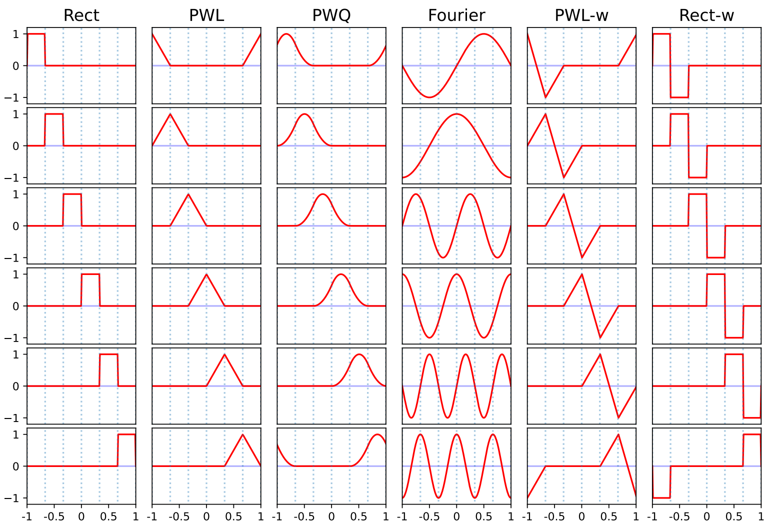

FIGURE 5. Visualization of the tested basis function with collection size $d = 6$ and discretization $M = 180$, where the collection size d also determines the number of partitions for all the piecewise functions. Each column corresponds to one collection of $\boldsymbol{p}(\cdot)$.

smooth versions of Rect basis. The Rectangular Wavelet (Rect-w) basis functions incorporate the zero integral property to enhance the global variation of the activation pattern. The Piecewise Linear Wavelet (PWL-w) is the continuous version of the Rect-w basis. Lastly, Fourier basis functions are smooth with global variation and zero integral, and their analytic nature ensures a globally continuous activation pattern. Notably, although PWL, PWQ, and PWL-w lack globally continuous activation patterns, their tendency for activation pattern jumps is minimal. Therefore, these basis functions can be considered to have an almost continuous activation pattern. The properties of these basis functions are summarized in Table 1.

Table 1. properties for basis.

	Continuous	Smooth	Global	Zero integral
Rect	✗	✗	✗	✗
PWL	✓	✗	✗	✗
PWQ	✓	✓	✗	✗
Fourier	✓	✓	✓	✓
PWL-w	✓	✗	✗	✓
Rect-w	✗	✗	✗	✓

Table 3 and Table 2 present the performance of IAT-ReLU for different combinations of basis functions. The performance is measured by mean-squared error (MSE) and prediction accuracy, respectively. Comparing the input basis function, we observe that the basis functions with zero integral (Rect-w, PWL-w, and Fourier basis), outperform the local input basis functions (Rect, PWL, and PWQ). This is because the zero integral basis functions exhibit an enriched activation pattern, leading to increased expressive power and improved trainability. Notably, the Fourier input basis demonstrates the best overall performance across both tasks. Regarding the output basis, the local basis functions (Rect, PWL, and PWQ) exhibit relatively better performance. The best performance in both tasks is achieved with the PWQ output basis. These results demonstrate the effectiveness of combining the input basis functions, based on properties from Section 4.2.2 for promoting the global variation of the activation pattern, with local output basis functions to extract the enriched features.

Table 2. The prediction accuracy (correctly predicted labels / the total number of points) for the random label memorization task. Top: Performance of IAT-ReLU under different choices of basis function with discretization M=500, where each row corresponds to the same input basis $\boldsymbol{p}(\cdot)$ and each column corresponds to the same output basis $\boldsymbol{q}(\cdot)$. The best 10 cases are highlighted in boldface. Bottom: Performance for traditional scalar activation functions.

Reulsts for IAT-ReLU

$\boldsymbol{p}(\cdot)$ \ $\boldsymbol{q}(\cdot)$	Rect	PWL	PWQ	Fourier	PWL-w	Rect-w
Rect	89.7%	90.5%	**92.1**%	89.4%	86.5%	86.1%
PWL	85.8%	87.3%	85.2%	**93.0**%	86.2%	86.5%
PWQ	85.5%	80.4%	80.5%	91.6%	86.7%	88.9%
Fourier	**93.2**%	**92.5**%	**94.1**%	89.5%	89.5%	88.4%
PWL-w	90.5%	**93.0**%	**92.3**%	**91.9**%	82.8%	86.3%
Rect-w	90.1%	**95.1**%	**98.3**%	89.2%	85.7%	82.7%

Scalar Activation

ReLU	Tanh	Sigmoid
90.4%	84.7%	83.9%

TABLE 3. MSE for 2D function fitting problem. Top: Performance of IAT-ReLU under different choices of basis function with discretization M=500, where each row corresponds to the same input basis $\boldsymbol{p}(\cdot)$ and each column corresponds to the same output basis $\boldsymbol{q}(\cdot)$.The best 10 cases are highlighted in boldface. Bottom: Performance for traditional scalar activation functions.

Reulsts for IAT-ReLU

$\boldsymbol{p}(\cdot)$ \ $\boldsymbol{q}(\cdot)$	Rect	PWL	PWQ	Fourier	PWL-w	Rect-w
Rect	8.8E-4	2.6E-3	1.8E-3	2.0E-3	1.5E-3	3.0E-3
PWL	1.7E-4	2.5E-4	**7.5E-5**	3.9E-4	2.5E-4	1.9E-4
PWQ	**1.6E-4**	**1.0E-4**	**6.7E-5**	5.2E-4	2.2E-4	2.4E-4
Fourier	**7.9E-5**	**7.0E-5**	**9.9E-5**	**1.0E-4**	**7.6E-5**	1.9E-4
PWL-w	2.7E-4	2.0E-4	**1.3E-4**	2.8E-4	3.9E-4	3.3E-4
Rect-w	1.1E-3	3.1E-3	3.1E-3	2.6E-3	3.0E-3	2.7E-3

Scalar Activation

ReLU	Tanh	Sigmoid
1.1E-3	4.1E-6	3.2E-6

5.2. Choosing discretization M. Another important aspect of IAT-ReLU is the discretization parameter, M, which controls the smoothness of the solution by determining the number of linear pieces. Figure 6 illustrates the performance of IAT-ReLU with different mesh ratios, where the discretization M is set as $M = d \times$ mesh ratio. To maintain consistent discretization error across different network widths d in our experiment, we use the mesh ratio to control the discretization error instead of M directly. This is because the network width d determines the number of basis functions in IAT-ReLU, and the discretization error depends on the complexity of the state function, which is largely controlled by the number of basis functions d. As a result, as d increases, the discretization M should also increase accordingly. Notably, for piecewise basis functions, the mesh ratio can also be viewed as the number of evaluation points per segment. From Figure 6, we observe two key effects of increasing the discretization. Firstly, for tasks that require smoothness, such as the function fitting task, higher discretization improves the final fitting quality. Conversely, for tasks that do not require smoothness, like the random label memorization task, increasing the discretization yields limited improvements in performance. Our results show that a mesh ratio of 2 yields the best performance. Deviations from this ratio lead to decreased performance, which we attribute to the complexity of the decision boundary in the randomly labeled dataset. Given the random label, the decision boundaries are highly irregular, which requires more linear pieces to accurately capture the intricate zigzag pattern in the decision boundary. However, as M increases further, the introduced smoothness becomes a constraint, preventing the network from making the sharp turns necessary to delineate the zigzag decision boundaries, which in turn reduces performance. Additionally, finer discretization results in more continuous gradients of the parameters. This is achieved by having more piecewise constant pieces in the solution. Therefore, increasing the discretization enhances training stability, even if the solution does not necessarily demand smoothness.

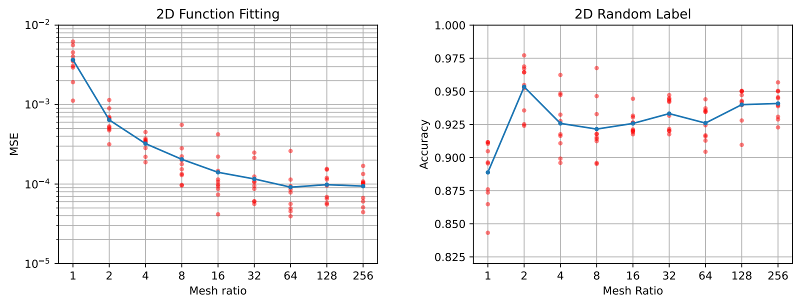

FIGURE 6. The performance of the IAT-ReLU network for different mesh ratios. The red dots in the graph represent outcomes from 10 independent runs of the test problems, while the blue curve indicates their average. The network structure comprises 10 fully FC layers, each with width d being 10. For IAT-ReLU, the discretization $M = d \times$ mesh ratio is used for the integral evaluation, and the Fourier and PWQ basis have been selected as the input and output basis functions, respectively. Left: MSE in the function fitting problem with different mesh ratios. Right: Accuracy in label memorization problem with different mesh ratios.

6. Conclusion

In this work, we propose the IAT as a means to increase the practical expressive power of DNNs. The IAT utilizes input and output basis functions to perform activation in the function space. By choosing different input/output basis functions and nonlinear activations, various versions of IAT can be obtained. In addition, we also showed the equivalence between the IAT and GDNN. In particular, when the nonlinear activation function is ReLU, we have IAT-ReLU, which serves as a generalization of ReLU. We demonstrate that IAT-ReLU and ReLU share the same forward and backward propagation structure. Moreover, by selecting appropriate basis functions (continuous with large total variation), IAT-ReLU exhibits continuous and enriched activation patterns, enhancing the expressive power of DNNs. Numerical experiments support the superiority of IAT-ReLU over ReLU and other activation functions. This study also suggests several potential avenues for future research.

Firstly, the idea of activation pattern pre-training and transfer can be explored. With the generalized activation pattern proposed in this study, it is possible to transfer a well-trained activation pattern to an untrained model of different sizes. By fixing the transferred activation pattern, the remaining optimization problem becomes simpler, since each pair of weight matrix and bias vector is linear to the model output. Following this direction, Transnet [40] is introduced, in which the activation pattern of shallow NN is carefully designed and pretrained for transferring. After transferring and fixing the activation pattern, the remaining optimization can be solved by a simple least square. Secondly, training DNNs with decoupled activation patterns can be considered. Instead of fixing the activation pattern, we can train another network solely responsible for providing the activation pattern, while the original network focuses on linear coefficients. This is feasible due to the non-zero gradient flowing through the activation pattern. This approach allows for separate optimization procedures for the activation pattern and linear coefficients. Thirdly, there is room for further optimization of basis functions. In this study, we only scratched the surface by considering six types of basis functions. It is worth exploring and designing additional basis functions to improve performance. In addition to designing new basis functions, we may also consider mixing basis functions of different properties, such as a mixture of local and global basis functions. Lastly, alternative optimization algorithms can be explored. IAT-ReLU represents a new class of activation functions that are 1-homogeneous and smooth when choosing continuous basis functions. Being 1-homogeneous implies that only the angle of the input matters in IAT-ReLU. Therefore, it is possible to investigate other optimization algorithms that focus on controlling the angle of the state vector by updating the parameters. These research directions have the potential to advance our understanding and utilization of IAT-ReLU, paving the way for further enhancements in the expressive power and performance of DNNs.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under the contract ERKJ387 at the Oak Ridge National Laboratory, which is operated by UT-Battelle, LLC, for the U.S. Department of Energy under Contract DE-AC05-00OR22725. The first author (FB) would also like to acknowledge the support from U.S. National Science Foundation through project DMS-2142672 and the support from the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Grant DE-SC0022297.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805, 2018.

[3] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational physics, vol. 378, pp. 686–707, 2019.

[4] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al., Highly accurate protein structure prediction with alphafold, Nature, vol. 596, no. 7873, pp. 583–589, 2021.

[5] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural networks, vol. 4, no. 2, pp. 251–257, 1991.

[6] S. S. Schoenholz, J. Gilmer, S. Ganguli, and J. Sohl-Dickstein, Deep information propagation, arXiv preprint arXiv:1611.01232, 2016.

[7] B. Hanin, Which neural net architectures give rise to exploding and vanishing gradients?, Advances in neural information processing systems, vol. 31, 2018.

[8] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, Neural ordinary differential equations, Advances in neural information processing systems, vol. 31, 2018.

[9] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar, Neural operator: Learning maps between function spaces, arXiv preprint arXiv:2108.08481, 2021.

[10] B. Hanin and D. Rolnick, Deep relu networks have surprisingly few activation patterns, Advances in neural information processing systems, vol. 32, 2019.

[11] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, Understanding deep neural networks with rectified linear units, arXiv preprint arXiv:1611.01491, 2016.

[12] I. Daubechies, R. DeVore, S. Foucart, B. Hanin, and G. Petrova, Nonlinear approximation and (deep) relu networks, Constructive Approximation, vol. 55, no. 1, pp. 127–172, 2022.

[13] R. M. Neal, Bayesian learning for neural networks, vol. 118. Springer Science & Business Media, 1996.

[14] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein, Deep neural networks as gaussian processes, arXiv preprint arXiv:1711.00165, 2017.

[15] A. G. d. G. Matthews, M. Rowland, J. Hron, R. E. Turner, and Z. Ghahramani, Gaussian process behaviour in wide deep neural networks, arXiv preprint arXiv:1804.11271, 2018.

[16] A. Jacot, F. Gabriel, and C. Hongler, Neural tangent kernel: Convergence and generalization in neural networks, Advances in neural information processing systems, vol. 31, 2018.

[17] S. Arora, S. S. Du, W. Hu, Z. Li, R. R. Salakhutdinov, and R. Wang, On exact computation with an infinitely wide neural net, Advances in neural information processing systems, vol. 32, 2019.

[18] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv preprint arXiv:2010.08895, 2020.

[19] Z. Allen-Zhu, Y. Li, and Z. Song, A convergence theory for deep learning via over-parameterization, in International Conference on Machine Learning, pp. 242–252, PMLR, 2019.

[20] Y. Li and Y. Yuan, Convergence analysis of two-layer neural networks with relu activation, Advances in neural information processing systems, vol. 30, 2017.

[21] S. S. Du, X. Zhai, B. Poczos, and A. Singh, Gradient descent provably optimizes over-parameterized neural networks, arXiv preprint arXiv:1810.02054, 2018.

[22] X. Zhang, Y. Yu, L. Wang, and Q. Gu, Learning one-hidden-layer relu networks via gradient descent, in The 22nd international conference on artificial intelligence and statistics, pp. 1524–1534, PMLR, 2019.

[23] D. Zou, Y. Cao, D. Zhou, and Q. Gu, Stochastic gradient descent optimizes over-parameterized deep relu networks, arXiv preprint arXiv:1811.08888, 2018.

[24] Z. Allen-Zhu, Y. Li, and Y. Liang, Learning and generalization in overparameterized neural networks, going beyond two layers, Advances in neural information processing systems, vol. 32, 2019.

[25] Y. Li and Y. Liang, Learning overparameterized neural networks via stochastic gradient descent on structured data, Advances in neural information processing systems, vol. 31, 2018.

[26] S. Arora, S. Du, W. Hu, Z. Li, and R. Wang, Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks, in International Conference on Machine Learning, pp. 322–332, PMLR, 2019.

[27] G. Zhang, J. Zhang, and J. Hinkle, Learning nonlinear level sets for dimensionality reduction in function approximation, in Advances in Neural Information Processing Systems (H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, pp. 13199–13208, Curran Associates, Inc., 2019.

[28] Y. Teng, Z. Wang, L. Ju, A. Gruber, and G. Zhang, Level set learning with pseudoreversible neural networks for nonlinear dimension reduction in function approximation, SIAM Journal on Scientific Computing, vol. 45, no. 3, pp. A1148–A1171, 2023.

[29] M. Yang, P. Wang, D. del Castillo-Negrete, Y. Cao, and G. Zhang, A pseudo-reversible normalizing flow for stochastic dynamical systems with various initial distributions, arXiv preprint arXiv:2306.05580, 2023.

[30] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, The expressive power of neural networks: A view from the width, Advances in neural information processing systems, vol. 30, 2017.

[31] R. Pascanu, G. Montufar, and Y. Bengio, On the number of response regions of deep feed forward networks with piece-wise linear activations, arXiv preprint arXiv:1312.6098, 2013.

[32] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, On the number of linear regions of deep neural networks, Advances in neural information processing systems, vol. 27, 2014.

[33] T. Serra, C. Tjandraatmadja, and S. Ramalingam, Bounding and counting linear regions of deep neural networks, in International Conference on Machine Learning, pp. 4558–4566, PMLR, 2018.

[34] T. Serra and S. Ramalingam, Empirical bounds on linear regions of deep rectifier networks, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 5628–5635, 2020.

[35] X. Xie, G. Zhang, and C. G. Webster, Non-intrusive inference reduced order model for fluids using deep multistep neural network, Mathematics, vol. 7, no. 8, 2019.
[36] B. Hanin and D. Rolnick, Complexity of linear regions in deep networks, in International Conference on Machine Learning, pp. 2596–2604, PMLR, 2019.
[37] G. B. Folland, Real analysis: modern techniques and their applications, vol. 40. John Wiley & Sons, 1999.
[38] M. Ainsworth and Y. Shin, Plateau phenomenon in gradient descent training of relu networks: Explanation, quantification, and avoidance, SIAM Journal on Scientific Computing, vol. 43, no. 5, pp. A3438–A3468, 2021.
[39] J. Fiat, E. Malach, and S. Shalev-Shwartz, Decoupling gating from linearity, arXiv preprint arXiv:1906.05032, 2019.
[40] Z. Zhang, F. Bao, L. Ju, and G. Zhang, Transnet: Transferable neural networks for partial differential equations, Journal of Scientific Computing, vol. 99, no. 2, 2024.

Computer Science and Mathematics Division, Oak Ridge National Lab, Oak Ridge, TN 37831.

E-mail: `zhangz2@ornl.gov`

Department of Mathematics, Florida State University, Tallahassee, FL 32306.

E-mail: `bao@math.fsu.edu`

Computer Science and Mathematics Division, Oak Ridge National Lab, Oak Ridge, TN 37831.

E-mail: `zhangg@ornl.gov`