Noncommutative C^*-algebra Net: Learning Neural Networks with Powerful Product Structure in C^*-algebra

Anonymous authors
Paper under double-blind review

Abstract

We propose a new generalization of neural networks with noncommutative C^*-algebra. An important feature of C^*-algebras is their noncommutative structure of products, but the existing C^*-algebra net frameworks have only considered commutative C^*-algebras. We show that this noncommutative structure of C^*-algebras induces powerful effects in learning neural networks. Our framework has a wide range of applications, such as learning multiple related neural networks simultaneously with interactions and learning invariant features with respect to group actions. We also show the validity of our framework numerically, which illustrates its potential power.

1 Introduction

Generalization of the parameter space of neural networks from real numbers to others has attracted researchers for decades. Although using real-valued parameters is standard and straightforward for real-valued data, it may be more suitable to adopt parameters of complex numbers (Hirose, 1992; Nishikawa et al., 2005; Amin et al., 2008; Yadav et al., 2005; Trabelsi et al., 2018; Lee et al., 2022) or quaternion numbers Nitta (1995); Arena et al. (1997); Zhu et al. (2018); Gaudet & Maida (2018) for data in signal processing, computer vision, and robotics domains, among others. Clifford-algebra, the generalization of these numbers, allows more flexible geometrical processing of data, and thus is applied to neural networks Pearson & Bisset (1994); Buchholz (2005); Buchholz & Sommer (2008) to handle rich geometric relationships in data (Rivera-Rovelo et al., 2010; Zang et al., 2022; Brandstetter et al., 2022). Different from these approaches focusing on the geometric perspective of parameter values, another direction of generalization is to use function-valued parameters Rossi & Conan-Guez (2005); Thind et al. (2022), which broadens the applications of neural networks to functional data.

Hashimoto et al. (2022) proposed generalizing neural network parameters to C^*-algebra, which is called C^*-algebra net. They showed that multiple related neural networks can be continuously combined into a single C^*-algebra net. For example, networks for the same task with different training datasets or different initial parameters can be combined continuously, which enables the construction of infinitely many networks and efficient learning using shared information among them. Such interaction among networks is also applicable to learning from related tasks, such as ensemble learning (Dong et al., 2020; Ganaie et al., 2022) and multitask learning (Zhang & Yang, 2022). However, because the product structure in the C^*-algebra that Hashimoto et al. (2021) focused on is commutative, they needed specially designed loss functions to induce the interaction.

In this paper, we propose a new generalization of neural networks with noncommutative C^*-algebra, which also generalizes the framework of Hashimoto et al. (2022). C^*-algebra is a generalization of the space of complex numbers Murphy (1990); Hashimoto et al. (2021). Typical examples of C^*-algebras include the space of diagonal matrices and the space of (not necessarily diagonal) matrices. An important feature of C^*-algebra is the product structure. Analogous to the case of complex numbers, for any two elements a and b in a C^*-algebra, we can calculate the product $a \cdot b$. However, unlike the case of complex numbers, the product is not always commutative, i.e., $a \cdot b = b \cdot a$ is not always satisfied. For example, whereas the product of two diagonal matrices commutes, the product of two nondiagonal matrices does not commute. In the case of diagonal matrices, the product is just described by the product of each diagonal element, and there are no interactions with the other elements. On the other hand, the product of two nondiagonal matrices is described by the sum of the products between different elements in the matrices, which induces interactions with the other elements.

For neural networks, the interactions induced by the noncommutative C^*-algebras correspond to interactions among multiple neural networks. Because the interactions are encoded in the structure of the network, we do not need to design loss functions to make the networks interact. Instead, the interactions required for them are implicitly and automatically learned through the noncommutative product structure in C^*-algebras. Therefore, our framework, which takes advantage of noncommutative C^*-algebras enables us to go beyond existing frameworks of neural networks.

Our framework of the noncommutative C^*-algebra net is general and has a wide range of applications, not limited to the above case of matrices. For example, by setting the C^*-algebra as a group C^*-algebra, we can construct a group equivariant neural network. In this case, we can naturally generalize real-valued parameters of neural networks to those inducing group convolution. We can also set the C^*-algebra as the C^*-algebra of bounded linear operators on a function space, which can be applied to analyzing functional data. Our main contributions are summarized as follows:

- We generalize the commutative C^*-algebra net proposed by Hashimoto et al. (2022) to noncommutative C^*-algebra, which enables us to take advantage of the noncommutative product structure in the C^*-algebra when learning neural networks.
- We show a wide range of applications of our framework, inducing interactions among networks and learning invariant features with respect to group actions.
- We empirically illustrate the validity of noncommutative C^*-algebra nets, including interactions among neural networks.

We emphasize that C^*-algebra is a powerful tool for neural networks, and our work provides a lot of important perspectives about its application.

2 Background

In this section, we review the mathematical background of C^*-algebra required for this paper and the existing C^*-algebra net. For more theoretical details of the C^*-algebra, see, for example, Murphy (1990).

2.1 C^*-algebra

C^*-algebra is a generalization of the space of complex values. It has structures of the product, involution *, and norm.

Definition 1 (C^*-algebra) *A set $\mathcal{A}$ is called a C^*-algebra if it satisfies the following conditions:*

1. *$\mathcal{A}$ is an algebra over $\mathbb{C}$ and equipped with a bijection $(\cdot)^* : \mathcal{A} \to \mathcal{A}$ that satisfies the following conditions for $\alpha, \beta \in \mathbb{C}$ and $c, d \in \mathcal{A}$:*
    - $(\alpha c + \beta d)^* = \overline{\alpha} c^* + \overline{\beta} d^*$,
    - $(cd)^* = d^* c^*$,
    - $(c^*)^* = c$.
2. *$\mathcal{A}$ is a normed space with $\|\cdot\|$, and for $c, d \in \mathcal{A}$, $\|cd\| \le \|c\|\,\|d\|$ holds. In addition, $\mathcal{A}$ is complete with respect to $\|\cdot\|$.*
3. *For $c \in \mathcal{A}$, $\|c^* c\| = \|c\|^2$ holds.*

The product structure in C^*-algebras can be both commutative and noncommutative.

Example 1 (Commutative C^*-algebra) *Let $\mathcal{A}$ be the space of continuous functions on a compact Hausdorff space $\mathcal{Z}$. We can regard $\mathcal{A}$ as a C^*-algebra by setting*

- *Product: Pointwise product of two functions, i.e., for $a_1, a_2 \in \mathcal{A}$, $a_1a_2(z) = a_1(z)a_2(z)$.*
- *Involution: Pointwise complex conjugate, i.e., for $a \in \mathcal{A}$, $a^*(z) = \overline{a(z)}$.*
- *Norm: Sup norm, i.e., for $a \in \mathcal{A}$, $\|a\| = \sup_{z\in\mathcal{Z}} |a(z)|$.*

In this case, the product in $\mathcal{A}$ is commutative.

Example 2 (Noncommutative C^*-algebra) *Let $\mathcal{A}$ be the space of bounded linear operators on a Hilbert space $\mathcal{H}$, which is denoted by $\mathcal{B}(\mathcal{H})$. We can regard $\mathcal{A}$ as a C^*-algebra by setting*

- *Product: Composition of two operators,*
- *Involution: Adjoint of an operator,*
- *Norm: Operator norm of an operator, i.e., for $a \in \mathcal{A}$, $\|a\| = \sup_{v\in\mathcal{H},\|v\|_{\mathcal{H}}=1} \|av\|_{\mathcal{H}}$.*

Here, $\|\cdot\|_{\mathcal{H}}$ is the norm in $\mathcal{H}$. In this case, the product in $\mathcal{A}$ is noncommutative. Note that if $\mathcal{H}$ is a d-dimensional space for a finite natural number d, then elements in $\mathcal{A}$ are d by d matrices.

Example 3 (Group C^*-algebra) *The group C^*-algebra on a group G, which is denoted as $C^*(G)$, is the set of maps from G to $\mathbb{C}$ equipped with the following product, involution, and norm:*

- *Product: $(a \cdot b)(g) = \int_G a(h)b(h^{-1}g)\mathrm{d}\lambda(h)$ for $g \in G$,*
- *Involution: $a^*(g) = \Delta(g^{-1})\overline{a(g^{-1})}$ for $g \in G$,*
- *Norm: $\|a\| = \sup_{[\pi]\in\hat{G}} \|\pi(a)\|$,*

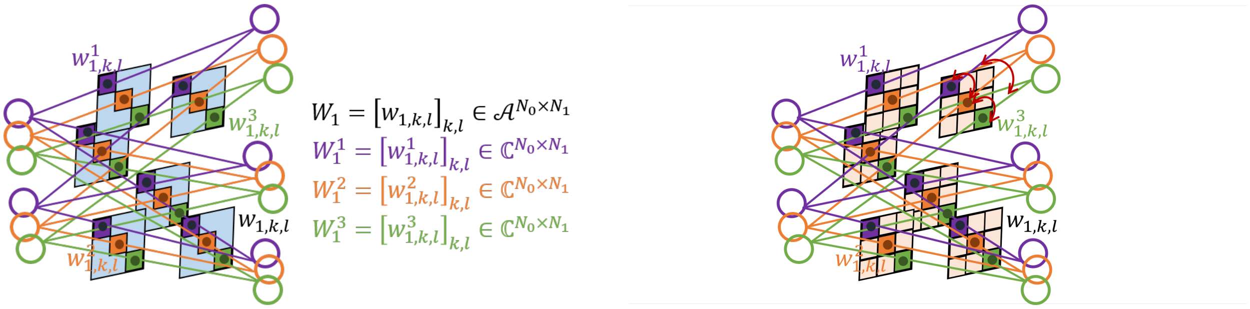
where $\Delta(g)$ is a positive number satisfying $\lambda(Eg) = \Delta(g)\lambda(E)$ for the Haar measure λ on G. In addition, $\hat{G}$ is the set of equivalence classes of irreducible unitary representations of G. Note that if G is discrete, then λ is the counting measure on G. In this paper, we focus mainly on the product structure of $C^*(G)$. For details of the Haar measure and representations of groups, see Kirillov (1976). If $G = \mathbb{Z}/p\mathbb{Z}$, then $C^*(G)$ is C^*-isomorphic to the C^*-algebra of circulant matrices (Hashimoto et al., 2023). Note also that if G is noncommutative, then $C^*(G)$ can also be noncommutative.

2.2 C^*-algebra net

Hashimoto et al. (2022) proposed generalizing real-valued neural network parameters to commutative C^*-algebra-valued ones. Here, we briefly review the existing (commutative) C^*-algebra net. Let $\mathcal{A} = C(\mathcal{Z})$, the commutative C^*-algebra of continuous functions on a compact Hausdorff space $\mathcal{Z}$. Let H be the depth of the network and $N_0, \ldots, N_H$ be the width of each layer. For $i = 1, \ldots, H$, set $W_i : \mathcal{A}^{N_{i-1}} \to \mathcal{A}^{N_i}$ as an Affine transformation defined with an $N_i \times N_{i-1}$ $\mathcal{A}$-valued matrix and an $\mathcal{A}$-valued bias vector in $\mathcal{A}^{N_i}$. In addition, set a nonlinear activation function $\sigma_i : \mathcal{A}^{N_i} \to \mathcal{A}^{N_i}$. The commutative C^*-algebra net $f : \mathcal{A}^{N_0} \to \mathcal{A}^{N_H}$ is defined as

$$f = \sigma_H \circ W_H \circ \cdots \circ \sigma_1 \circ W_1. \tag{1}$$

By generalizing neural network parameters to functions, we can combine multiple standard (real-valued) neural networks continuously, which enables them to learn efficiently. We show an example of commutative C^*-nets below. To simplify the notation, we focus on the case where the network does not have biases. However, the same arguments are valid for the case where the network has biases.


$w^1_{1,k,l}$
$w^3_{1,k,l}$
$w_{1,k,l}$
$w^2_{1,k,l}$
$W_1 = [w_{1,k,l}]_{k,l} \in \mathcal{A}^{N_0 \times N_1}$
$W^1_1 = [w^1_{1,k,l}]_{k,l} \in \mathbb{C}^{N_0 \times N_1}$
$W^2_1 = [w^2_{1,k,l}]_{k,l} \in \mathbb{C}^{N_0 \times N_1}$
$W^3_1 = [w^3_{1,k,l}]_{k,l} \in \mathbb{C}^{N_0 \times N_1}$
$w^1_{1,k,l}$
$w^3_{1,k,l}$
$w_{1,k,l}$
$w^2_{1,k,l}$

(a) C^*-algebra net over diagonal matrices (commutative). Each diagonal element corresponds to an individual sub-model.

(b) C^*-algebra net over nondiagonal matrices (noncommutative). The nondiagonal elements induce the interactions among multiple sub-models.

Figure 1: Difference between commutative and noncommutative C^*-algebra nets.

2.2.1 The case of diagonal matrices

If $\mathcal{Z}$ is a finite set, then $\mathcal{A} = \{a \in \mathbb{C}^{d\times d} \mid a \text{ is a diagonal matrix}\}$. The C^*-algebra net f on $\mathcal{A}$ corresponds to d separate real or complex-valued sub-models. Indeed, denote by x^j the vector composed of the jth diagonal elements of $x \in \mathcal{A}^N$, which is defined as the vector in $\mathbb{C}^N$ whose kth element is the jth diagonal element of the $\mathcal{A}$-valued kth element of x. Assume the activation function $\sigma_i : \mathcal{A}^N \to \mathcal{A}^N$ is defined as $\sigma_i(x)^j = \tilde{\sigma}_i(x^j)$ for some $\tilde{\sigma}_i : \mathbb{C}^N \to \mathbb{C}^N$. Since the jth diagonal element of a_1a_2 for $a_1, a_2 \in \mathcal{A}$ is the product of the jth element of a_1 and a_2, we have

$$f(x)^j = \tilde{\sigma}_H \circ W_H^j \circ \cdots \circ \tilde{\sigma}_1 \circ W_1^j, \tag{2}$$

where $W_i^j \in \mathbb{C}^{N_i \times N_{i-1}}$ is the matrix whose (k,l)-entry is the jth diagonal of the (k,l)-entry of $W_i \in \mathcal{A}^{N_i \times N_{i-1}}$. Figure 1 (a) schematically shows the C^*-algebra net over diagonal matrices.

3 Noncommutative C^*-algebra Net

Although the existing C^*-algebra net provides a framework for applying C^*-algebra to neural networks, it focuses on commutative C^*-algebras, whose product structure is simple. Therefore, we generalize the existing commutative C^*-algebra net to noncommutative C^*-algebra. Since the product structures in noncommutative C^*-algebras are more complicated than those in commutative C^*-algebras, they enable neural networks to learn features of data more efficiently. For example, if we focus on the C^*-algebra of matrices, then the neural network parameters describe interactions between multiple real-valued sub-models (see Section 3.1.1).

Let $\mathcal{A}$ be a general C^*-algebra and consider the network f in the same form as Equation (1). We emphasize that in our framework, the choice of $\mathcal{A}$ is not restricted to a commutative C^*-algebra. We list examples of $\mathcal{A}$ and their validity for learning neural networks below.

3.1 Examples of C^*-algebras for neural networks

As mentioned in the previous section, we focus on the case where the network does not have biases for simplification in this subsection.

3.1.1 Nondiagonal matrices

Let $\mathcal{A} = \mathbb{C}^{d\times d}$. Note that $\mathcal{A}$ is a noncommutative C^*-algebra. In this case, unlike the network (2), the jth diagonal element of $a_1a_2a_3$ for $a_1, a_2, a_3 \in \mathcal{A}$ depends not only on the jth diagonal element of a_2, but also the other diagonal elements of a_2. Thus, $f(x)^j$ depends not only on the sub-model corresponding to jth diagonal element discussed in Section 2.2.1, but also on other sub-models. The nondiagonal elements in $\mathcal{A}$ induce interactions between d real or complex-valued sub-models. In practice, to regard the nondiagonal elements as factors of interactions, their values should be small compared to the diagonal elements. We will see the effect of the nondiagonal elements in $\mathcal{A}$ numerically in Section 4.1. Figure 1 (b) schematically shows the C^*-algebra net over nondiagonal matrices.

3.1.2 Block diagonal matrices

Let $\mathcal{A} = \{a \in \mathbb{C}^{d\times d} \mid a = \operatorname{diag}(\mathbf{a}_1, \ldots, \mathbf{a}_m),\ \mathbf{a}_i \in \mathbb{C}^{d_i \times d_i}\}$. The product of two block diagonal matrices $a = \operatorname{diag}(\mathbf{a}_1, \ldots, \mathbf{a}_m)$ and $b = \operatorname{diag}(\mathbf{b}_1, \ldots, \mathbf{b}_m)$ can be written as

$$ab = \operatorname{diag}(\mathbf{a}_1\mathbf{b}_1, \ldots, \mathbf{a}_m\mathbf{b}_m).$$

In a similar manner to Section 2.2.1, we denote by $\mathbf{x}^j$ the N by d_j matrix composed of the jth diagonal blocks of $x \in \mathcal{A}^N$. Assume the activation function $\sigma_i : \mathcal{A}^N \to \mathcal{A}^N$ is defined as $\sigma_i(x) = \operatorname{diag}(\tilde{\boldsymbol{\sigma}}_i^1(\mathbf{x}^1), \ldots, \tilde{\boldsymbol{\sigma}}_i^m(\mathbf{x}^m))$ for some $\tilde{\boldsymbol{\sigma}}_{i,j} : \mathbb{C}^{N\times d_j} \to \mathbb{C}^{N\times d_j}$. Then, we have

$$\mathbf{f}(\mathbf{x})^j = \tilde{\boldsymbol{\sigma}}_H^j \circ \mathbf{W}_H^j \circ \cdots \circ \tilde{\boldsymbol{\sigma}}_1^j \circ \mathbf{W}_1^j, \tag{3}$$

where $\mathbf{W}_i^j \in (\mathbb{C}^{d_j\times d_j})^{N_i \times N_{i-1}}$ is the block matrix whose (k, l)-entry is the jth block diagonal of the (k, l)-entry of $W_i \in \mathcal{A}^{N_i\times N_{i-1}}$. In this case, we have m groups of sub-models, each of which is composed of interacting d_j sub-models mentioned in Section 3.1.1. Indeed, the block diagonal case generalizes the diagonal and nondiagonal cases stated in Sections 2.2.1 and 3.1.1. If $d_j = 1$ for all $j = 1, \ldots, m$, then the network (3) is reduced to the network (2) with diagonal matrices. If $m = 1$ and $d_1 = d$, then the network (3) is reduced to the network with d by d nondiagonal matrices.

3.1.3 Circulant matrices

Let $\mathcal{A} = \{a \in \mathbb{C}^{d\times d} \mid a \text{ is a circulant matrix}\}$. Here, a circulant matrix a is the matrix represented as

$$a = \begin{bmatrix} a_1 & a_d & \cdots & a_2 \\ a_2 & a_1 & \cdots & a_3 \\ & \ddots & \ddots & \\ a_d & a_{d-1} & \cdots & a_1 \end{bmatrix}$$

for $a_1, \ldots, a_d \in \mathbb{C}$. Note that in this case, $\mathcal{A}$ is commutative. Circulant matrices are diagonalized by the discrete Fourier matrix as follows (Davis, 1979). We denote by F the discrete Fourier transform matrix, whose (i, j)-entry is $\omega^{(i-1)(j-1)}/\sqrt{p}$, where $\omega = \mathrm{e}^{2\pi\sqrt{-1}/d}$.

Lemma 1 *Any circulant matrix a is decomposed as $a = F\Lambda_a F^*$, where*

$$\Lambda_a = \operatorname{diag}\left(\sum_{i=1}^{d} a_i\omega^{(i-1)\cdot 0}, \ldots, \sum_{i=1}^{d} a_i\omega^{(i-1)(d-1)}\right).$$

Since $ab = F\Lambda_a\Lambda_b F^*$ for $a, b \in \mathcal{A}$, the product of a and b corresponds to the multiplication of each Fourier component of a and b. Assume the activation function $\sigma_i : \mathcal{A}^N \to \mathcal{A}^N$ is defined such that $(F^*\sigma_i(x)F)^j$ equals to $\hat{\tilde{\sigma}}_i((FxF^*)^j)$ for some $\hat{\tilde{\sigma}}_i : \mathbb{C}^N \to \mathbb{C}^N$. Then, we obtain the network

$$(F^*f(x)F)^j = \hat{\tilde{\sigma}}_H \circ \hat{W}_H^j \circ \cdots \circ \hat{\tilde{\sigma}}_1 \circ \hat{W}_1^j, \tag{4}$$

where $\hat{W}_i^j \in \mathbb{C}^{N_i\times N_{i-1}}$ is the matrix whose (k, l)-entry is $(Fw_{i,k,l}F^*)^j$, where $w_{i,k,l}$ is the the (k, l)-entry of $W_i \in \mathcal{A}^{N_i\times N_{i-1}}$. The jth sub-model of the network (4) corresponds to the network of the jth Fourier component.

Remark 1 *The jth sub-model of the network (4) does not interact with those of other Fourier components than the jth component. This fact corresponds to the fact that $\mathcal{A}$ is commutative in this case. Analogous to the case in Section 3.1.1, if we set $\mathcal{A}$ as noncirculant matrices, then we obtain interactions between sub-models corresponding to different Fourier components.*

3.1.4 Group C^*-algebra on a symmetric group

Let G be the symmetric group on the set $\{1,\ldots,d\}$ and let $\mathcal{A}=C^*(G)$. Note that since G is noncommutative, $C^*(G)$ is also noncommutative. Then, the output $f(x)\in\mathcal{A}^{N_H}$ is the $\mathbb{C}^{N_H}$-valued map on G. Using the product structure introduced in Example 3, we can construct a network that takes the permutation of data into account. Indeed, an element $w\in\mathcal{A}$ of a weight matrix $W\in\mathcal{A}^{N_{i-1}\times N_i}$ is a function on G. Thus, $w(g)$ describes the weight corresponding to the permutation $g\in G$. Since the product of $x\in C^*(G)$ and w is defined as $wx(g)=\sum_{h\in G}w(h)x(h^{-1}g)$, by applying W, all the weights corresponding to the permutations affect the input. For example, let $z\in\mathbb{R}^d$ and set $x\in C^*(G)$ as $x(g)=g\cdot z$, where $g\cdot z$ is the action of g on z, i.e., the permutation of z with respect to g. Then, we can input all the patterns of permutations of z simultaneously, and by virtue of the product structure in $C^*(G)$, the network is learned with the interaction among these permutations. Regarding the output, if the network is learned so that the outputs y become constant functions on G, i.e., $y(g)=c$ for some constant c, then it means that c is invariant with respect to g, i.e., invariant with respect to the permutation. We will numerically investigate the application of the group C^*-algebra net to permutation invariant problems in Section 4.2.

Remark 2 *If the activation function σ is defined as $\sigma(x)(g)=\sigma(x(g))$, i.e., applied elementwisely to x, then the network f is permutation equivariant. That is, even if the input $x(g)$ is replaced by $x(gh)$ for some $h\in G$, the output $f(x)(g)$ is replaced by $f(x)(gh)$. This is because the product in $C^*(G)$ is defined as a convolution. This feature of the convolution has been studied for group equivariant neural networks (Lenssen et al., 2018; Cohen et al., 2019; Sannai et al., 2021; Sonoda et al., 2022). The above setting of the C^*-algebra net provides us with a design of group equivariant networks from the perspective of C^*-algebra.*

3.1.5 Bounded linear operators on a Hilbert space

For functional data, we can also set $\mathcal{A}$ as an infinite-dimensional space. Using infinite-dimensional C^*-algebra for analyzing functional data has been proposed (Hashimoto et al., 2021). We can also adopt this idea for neural networks. Let $\mathcal{A}=\mathcal{B}(L^2(\Omega))$ for a measure space Ω. Set $\mathcal{A}_0=\{a\in\mathcal{A}\mid a \text{ is a multiplication operator}\}$. Here, a multiplication operator a is a linear operator that is defined as $av=v\cdot u$ for some $u\in L^\infty(\Omega)$. The space $\mathcal{A}_0$ is a generalization of the space of diagonal matrices to the infinite-dimensional space. If we restrict elements of weight matrices to $\mathcal{A}_0$, then we obtain infinitely many sub-models without interactions. Since outputs are in $\mathcal{A}_0^{N_H}$, we can obtain functional data as outputs. Similar to the case of matrices (see Section 3.1.1), by setting elements of weight matrices as elements in $\mathcal{A}$, we can take advantage of interactions among infinitely many sub-models.

3.2 Approximation of functions with interactions by C^*-algebra net

We observe what kind of functions the C^*-algebra net can approximate. We focus on the case of $\mathcal{A}=\mathbb{C}^{d\times d}$. Consider a shallow network $f:\mathcal{A}^{N_0}\to\mathcal{A}$ defined as $f(x)=W_2^*\sigma(W_1x+b)$, where $W_1\in\mathcal{A}^{N_1\times N_0}$, $W_2\in\mathcal{A}^{N_1}$, and $b\in\mathcal{A}^{N_1}$. Let $\tilde{f}:\mathcal{A}^{N_0}\to\mathcal{A}$ be the function in the form of $\tilde{f}(x)=[\sum_{j=1}^d f_{kj}(x^l)]_{kl}$, where $f_{kj}:\mathbb{C}^{N_0d}\to\mathbb{R}$. Here, we abuse the notation and denote by $x^l\in\mathbb{C}^{N_0d}$ the lth column of x regarded as an N_0d by d matrix. Assume f_{kj} is represented as

$$f_{kj}(x)=\int_{\mathbb{R}}\int_{\mathbb{R}^{N_0d}}T_{kj}(w,b)\sigma(w^*x+b)\mathrm{d}w\,\mathrm{d}b \tag{5}$$

for some $T_{kj}:\mathbb{R}^{N_0d}\times\mathbb{R}\to\mathbb{R}$. By the theory of the ridgelet transform, such T_{kj} exists for most realistic settings (Candès, 1999; Sonoda & Murata, 2017). For example, if f_{kj} and its Fourier transform is in $L^1(\mathbb{R}^{N_0d})$ and σ is the ReLU function, then f_{kj} has a representation of Equation (5). We discretize Equation (5) by replacing the Lebesgue measures with $\sum_{i=1}^{N_1}\delta_{w_{ij}}$ and $\sum_{i=1}^{N_1}\delta_{b_{ij}}$, where δ_w is the Dirac measure centered at w. Then, the (k,l)-entry of $\tilde{f}(x)$ is written as

$$\sum_{j=1}^{d}\sum_{i=1}^{N_1}T_{kj}(w_{ij},b_{ij})\sigma(w_{ij}^*x^l+b_{ij}).$$

Setting the i-th element of $W_2 \in \mathcal{A}^{N_1}$ as $[T_{kj}(w_{ij}, b_{ij})]_{kj}$, the (i,m)-entry of $W_1 \in \mathcal{A}^{N_1 \times N_0}$ as $[(w_{i,j})_{md+l}]_{jl}$, the ith element of $b \in \mathcal{A}^{N_1}$ as $[b_j]_{jl}$, we obtain

$$\sum_{j=1}^{d}\sum_{i=1}^{N_1} T_{kj}(w_{ij}, b_{ij})\sigma(w_{ij}^* x^l + b_{ij}) = (W_2^k)^* \sigma(W_1 x^l + b^l),$$

which is the (k,l)-entry of $f(x)$.

Remark 3 *As we discussed in Sections 2.2.1 and 3.1.1, a C^*-algebra net over matrices can be regarded as d interacting sub-models. The above argument insists the lth column of $f(x)$ and $\tilde{f}(x)$ depends only on x^l. Thus, in this case, if we input data x^l corresponding to the lth sub-model, then the output is obtained as the lth column of the $\mathcal{A}$-valued output $f(x)$. On the other hand, the weight matrices W_1 and W_2 and the bias b are used commonly in providing the outputs for any sub-model, i.e., W_1, W_2, and b are learned using data corresponding to all the sub-models. Therefore, W_1, W_2, and b induce interactions among the sub-models.*

4 Experiments

In this section, we numerically demonstrate the abilities of noncommutative C^*-algebra nets using nondiagonal C^*-algebra nets over matrices and group C^*-algebra nets. We use C^*-algebra-valued multi-layered perceptrons (MLPs) to simplify the experiments. However, they can be naturally extended to other neural networks, such as convolutional neural networks.

The models were implemented with `JAX` (Bradbury et al., 2018). Experiments were conducted on an AMD EPYC 7543 CPU and an NVIDIA A-100 GPU. See Appendix A.1 for additional information on experiments.

4.1 C^*-algebra nets over matrices

In a noncommutative C^*-algebra net over matrices consisting of nondiagonal-matrix parameters, each sub-model is expected to interact with others and thus improve performance compared with its commutative counterpart consisting of diagonal matrices. We demonstrate the effectiveness of such interaction using image classification and neural implicit representation (NIR) tasks.

See Section 3.1.1 for the notations. When training the jth sub-model ($j = 1, 2, \ldots, d$), an original N_0-dimensional input data point $\boldsymbol{x} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N_0}] \in \mathbb{R}^{N_0}$ is converted to its corresponding representation $x \in \mathcal{A}^{N_0} = \mathbb{R}^{N_0 \times d \times d}$ such that $x_{i,j,j} = \boldsymbol{x}_i$ for $i = 1, 2, \ldots, N_0$ and 0 otherwise. The loss to its N_H-dimensional output of a C^*-algebra net $y \in \mathcal{A}^{N_H}$ and the target $t \in \mathcal{A}^{N_H}$ is computed as $\ell(y_{:,j,j}, t_{:,j,j}) + \frac{1}{2}\sum_{k,(l \neq j)}(y_{k,j,l}^2 + y_{k,l,j}^2)$, where ℓ is a certain loss function; we use the mean squared error (MSE) for image classification and the Huber loss for NIR. The second and third terms suppress the nondiagonal elements of the outputs to 0. In both examples, we use leaky-ReLU as an activation function and apply it only to the diagonal elements of pre-activations.

4.1.1 Image classification

We conduct experiments of image classification tasks using MNIST (Le Cun et al., 1998), Kuzushiji-MNIST (Clanuwat et al., 2018), and Fashion-MNIST (Xiao et al., 2017), which are composed of 10-class 28 × 28 gray-scale images. Each sub-model is trained on a mutually exclusive subset sampled from the original training data and then evaluated on the entire test data. Each subset is sampled to be balanced, i.e., each class has the same number of training samples. As a baseline, we use a commutative C^*-algebra net over diagonal matrices, which consists of the same sub-models but cannot interact with other sub-models. Both noncommutative and commutative models share hyperparameters: the number of layers was set to 4, the hidden size was set to 128, and the models were trained for 30 epochs.

Table 1 shows the average test accuracy over sub-models. As can be seen, a noncommutative C^*-algebra net consistently outperforms its commutative counterpart, which is significant, particularly when the number of sub-models is 40. Note that when the number of sub-models is 40, the size of the training dataset for each sub-model is 40 times smaller than the original one, and thus, the commutative C^*-algebra net fails to learn.

Table 1: Average test accuracy over sub-models of commutative and noncommutative C^*-algebra nets over matrices on test datasets. Interactions between sub-models that the noncommutative C^*-algebra net improves performance significantly when the number of sub-models is 40.

Dataset	# sub-models	Commutative C^*-algebra net (baseline)	Noncommutative C^*-algebra net
MNIST	5	0.963 ± 0.003	0.970 ± 0.002
	10	0.937 ± 0.004	0.956 ± 0.002
	20	0.898 ± 0.007	0.937 ± 0.002
	40	0.605 ± 0.007	0.906 ± 0.004
Kuzushiji-MNIST	5	0.837 ± 0.003	0.859 ± 0.003
	10	0.766 ± 0.008	0.815 ± 0.007
	20	0.674 ± 0.011	0.758 ± 0.007
	40	0.453 ± 0.026	0.682 ± 0.008
Fashion-MNIST	5	0.862 ± 0.001	0.868 ± 0.002
	10	0.839 ± 0.003	0.852 ± 0.004
	20	0.790 ± 0.010	0.832 ± 0.005
	40	0.650 ± 0.018	0.810 ± 0.005

Table 2: Average test accuracy over five sub-models simultaneously trained on the three datasets. The (fully) noncommutative C^*-algebra net outperforms block-diagonal the noncommutative C^*-algebra net on Kuzushiji-MNIST, and Fashion-MNIST, indicating that the interaction can leverage related tasks.

Dataset	Commutative	Block-diagonal	Noncommutative
MNIST	0.956 ± 0.002	0.969 ± 0.002	0.970 ± 0.002
Kuzushiji-MNIST	0.745 ± 0.004	0.778 ± 0.006	0.796 ± 0.008
Fashion-MNIST	0.768 ± 0.007	0.807 ± 0.006	0.822 ± 0.002

Nevertheless, the noncommutative C^*-algebra net retains performance mostly. These results suggest that sub-models share knowledge through interaction.

Additionally, Table 2 illustrates that related tasks help performance improvement through interaction. Specifically, we prepare five sub-models per dataset, one of MNIST, Kuzushiji-MNIST, and Fashion-MNIST, and train a total of 15 sub-models simultaneously. In addition to the commutative C^*-algebra net, where sub-models have no interaction, and the noncommutative C^*-algebra net, where each sub-model can interact with any other sub-models, we use a block-diagonal noncommutative C^*-algebra net (see Section 3.1.2), where each sub-model can only interact with other sub-models trained on the same dataset. Table 2 shows that the fully noncommutative C^*-algebra net surpasses the block-diagonal one on Kuzushiji-MNIST and Fashion-MNIST, implying that not only intra-task interaction but also inter-task interaction helps performance gain. Note that each dataset is subsampled so that every class has the same number of samples, so it is not possible to compare the values of Tables 1 and 2.

4.1.2 Neural implicit representation

In the next experiment, we use a C^*-algebra net over matrices to learn implicit representations of 2D images that map each pixel coordinate to its RGB colors (Sitzmann et al., 2020; Xie et al., 2022). Specifically, an input coordinate in $[0,1]^2$ is transformed into a random Fourier features in $[-1,1]^{320}$ and then converted into its C^*-algebraic representation over matrices as an input to a C^*-algebra net over matrices. Similar to the image classification task, we compare noncommutative NIRs with commutative NIRs, using the following hyperparameters: the number of layers to 6 and the hidden dimension to 256. These NIRs learn 128×128-pixel images of ukiyo-e pictures from The Metropolitan Museum of Art[1] and photographs of cats from the AFHQ dataset (Choi et al., 2020).

---

[1] `https://www.metmuseum.org/art/the-collection`

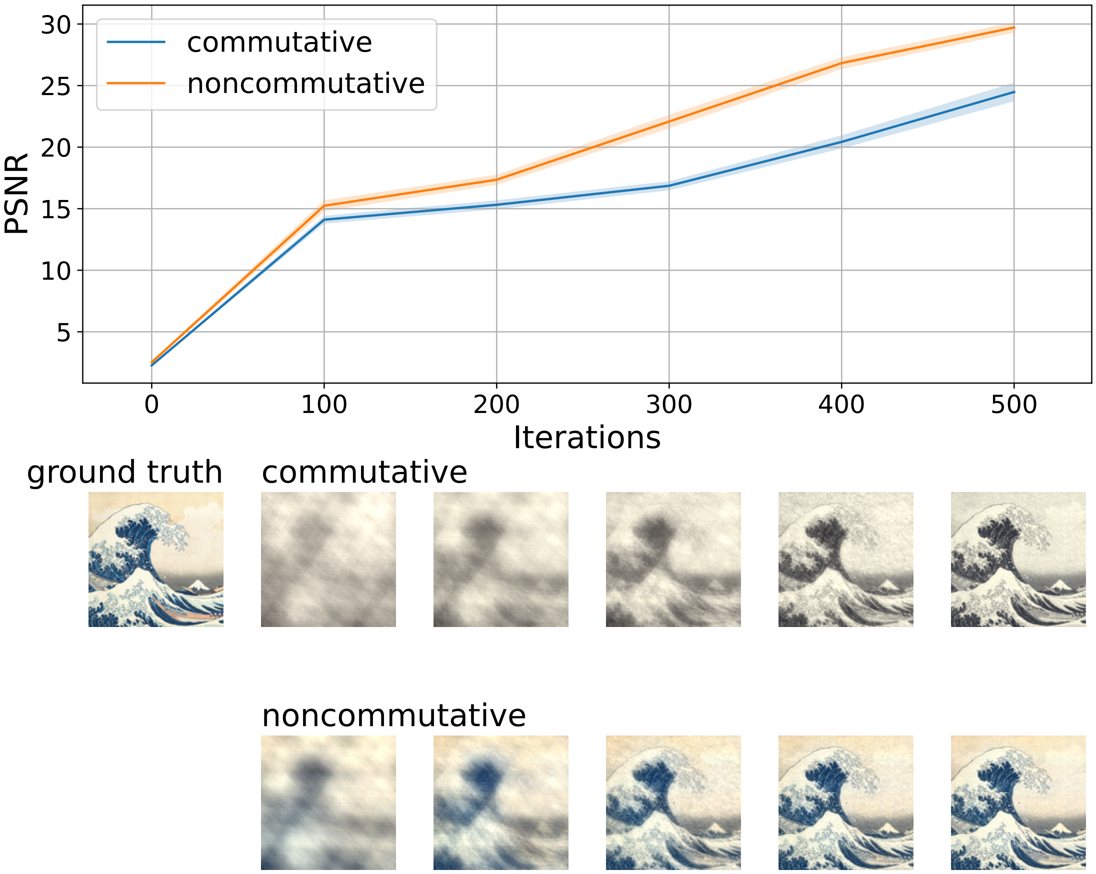


Figure 2: Average PSNR of implicit representations of the image below (top) and reconstructions of the ground truth image at every 100 iterations (bottom). The noncommutative C^*-algebra net learns the geometry and colors of the image faster than its commutative counterpart.

Figure 2 (top) shows the curves of the average PSNR (Peak Signal-to-Noise Ratio) of sub-models corresponding to the image below. Both commutative and noncommutative C^*-algebra nets consist of five sub-models trained on five ukiyo-e pictures (see also Figure 6). PSNR, the quality measure, of the noncommutative NIR grows faster, and correspondingly, it learns the details of ground truth images faster than its commutative version (Figure 2 bottom). Noticeably, the noncommutative representations reproduce colors even at the early stage of learning, although the commutative ones remain monochrome after 500 iterations of training. Along with the similar trends observed in the pictures of cats (Figure 3), these results further emphasize the effectiveness of the interaction. Longer-term results are presented in Figure 7.

This INR for 2D images can be extended to represent 3D models. Figure 4 shows synthesized views of 3D implicit representations using the same C^*-algebra MLPs trained on three 3D chairs from the ShapeNet dataset (Chang et al., 2015). The presented poses are unseen during training. Again, the noncommutative INR reconstructs the chair models with less noisy artifacts, indicating that interaction helps efficient learning. See Appendices A.1 and A.2 for details and results.

4.2 Group C^*-algebra nets

As another experimental example of C^*-algebra nets, we showcase group C^*-algebra nets, which we introduced in Section 3.1.4. The group C^*-algebra nets take functions on a symmetric group as input and return functions on the group as output.

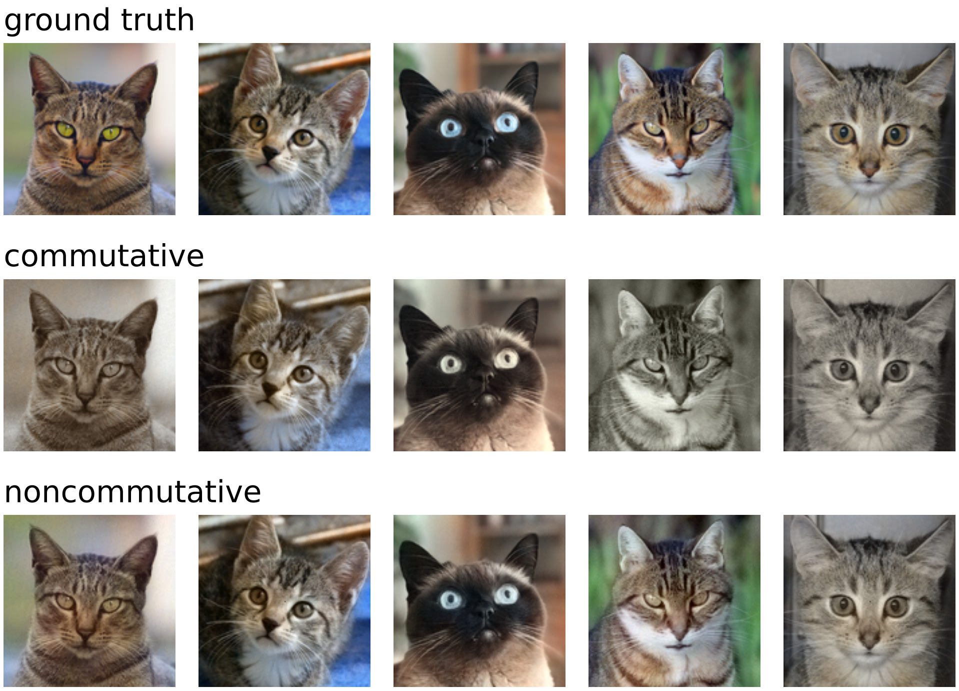


Figure 3: Ground truth images and their implicit representations of commutative and noncommutative C^*-algebra nets after 500 iterations of training. The noncommutative C^*-algebra net reproduces colors more faithfully.

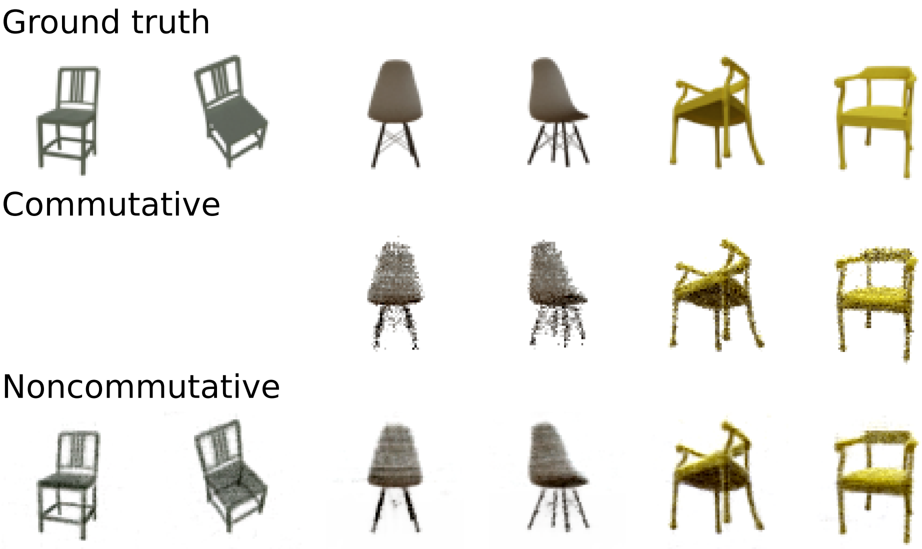


Figure 4: Synthesized views of 3D implicit representations of commutative and noncommutative C^*-algebra nets after 5000 iterations of training. The noncommutative C^*-algebra net can produce finer details. Note that the commutative C^*-algebra net could not synthesize the chair on the left.

Refer to Section 3.1.4 for notations. A group C^*-algebra net is trained on data $\{(x, y) \in \mathcal{A}^{N_0} \times \mathcal{A}^{N_H}\}$, where x and y are N_0- and N_H-dimensional vector-valued functions. Practically, such functions may be represented as real tensors, e.g., $x \in \mathbb{C}^{N_0 \times \#G}$, where $\#G$ is the size of G. Using product between functions explained in

Table 3: Average test accuracy of a DeepSet model and a group C^*-algebra net on test data of the sum-of-digits task after 100 epochs of training. The group C^*-algebra net can learn from fewer data.

Dataset size	DeepSet	Group C^*-algebra net
1k	0.413 ± 0.031	0.777 ± 0.009
5k	0.807 ± 0.031	0.921 ± 0.002
10k	0.878 ± 0.009	0.944 ± 0.005
50k	0.904 ± 0.007	0.971 ± 0.001

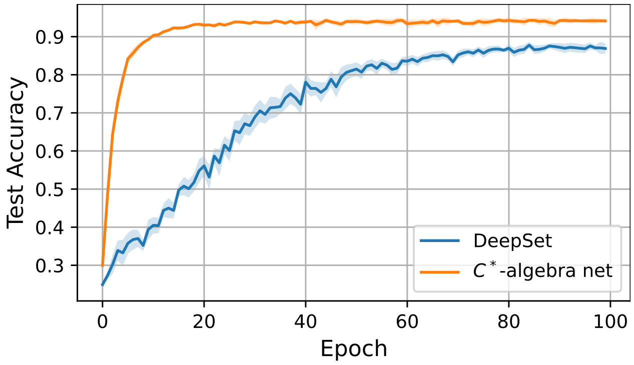


Figure 5: Average test accuracy curves of a DeepSet model and a group C^*-algebra net trained on 10k data of the sum-of-digits task. The group C^*-algebra net can learn more efficiently and effectively.

Section 3.1.4 and element-wise addition, a linear layer, and consequently, an MLP, on $\mathcal{A}$ can be constructed. Following the C^*-algebra nets over matrices, we use leaky ReLU for activations.

One of the simplest tasks for the group C^*-algebra nets is to learn permutation-invariant representations, e.g., predicting the sum of given d digits. In this case, x is a function that outputs permutations of features of d digits, and $y(g)$ is an identity function that returns their sum for all $g \in G$. In this experiment, we use features of MNIST digits of a pre-trained CNN in 32-dimensional vectors. Digits are selected so that their sum is less than 10 to simplify the problem, and the model is trained to classify the sum of given digits using cross-entropy loss. We set the number of layers to 4 and the hidden dimension to 32. For comparison, we prepare a permutation-invariant DeepSet model (Zaheer et al., 2017), which uses sum pooling for permutation invariance, containing the same number of parameters as floating-point numbers with the group C^*net.

Table 3 displays the results of the task with various training dataset sizes when $d = 3$. What stands out in the table is that the group C^*-algebra net consistently outperforms the DeepSet model by large margins, especially when the number of training data is limited. Additionally, as shown in Figure 5, the group C^*-algebra net converges much faster than the DeepSet model. These results suggest that the inductive biases implanted by the product structure in the group C^*-algebra net are effective.

5 Conclusion and Discussion

In this paper, we have generalized the space of neural network parameters to noncommutative C^*-algebras. Their rich product structures bring powerful properties to neural networks. For example, a C^*-algebra net over nondiagonal matrices enables its sub-models to interact, and a group C^*-algebra net learns permutation-equivariant features. We have empirically demonstrated the validity of these properties in various tasks, image classification, neural implicit representation, and sum-of-digits tasks.

Another important and interesting research direction is the application of infinite-dimensional C^*-algebras. In this paper, we focused mainly on finite-dimensional C^*-algebras. We showed that the product structure in C^*-algebras is a powerful tool for neural networks, for example, learning with interactions and group equivariance (or invariance) even for the finite-dimensional case. Infinite-dimensional C^*-algebra allows us to analyze functional data. Practical applications of our framework to functional data with infinite-dimensional C^*-algebras are also our future work.

Our framework with noncommutative C^*-algebras has a wide range of applications, and we believe that our framework opens up a new approach to learning neural networks.

References

Md. Faijul Amin, Md. Monirul Islam, and Kazuyuki Murase. Single-layered complex-valued neural networks and their ensembles for real-valued classification problems. In *IJCNN*, 2008.

Paolo Arena, Luigi Fortuna, Giovanni Muscato, and Maria Gabriella Xibilia. Multilayer perceptrons to approximate quaternion valued functions. *Neural Networks*, 10(2):335–342, 1997.

Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, John Quan, George Papamakarios, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Luyu Wang, Wojciech Stokowiec, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL `http://github.com/deepmind`.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K Gupta. Clifford neural layers for pde modeling. *arXiv*, 2022.

Sven Buchholz. *A Theory of Neural Computation with Clifford Algebras.* PhD thesis, 2005.

Sven Buchholz and Gerald Sommer. On clifford neurons and clifford multi-layer perceptrons. *Neural Networks*, 21(7):925–935, 2008.

Emmanuel J. Candès. Harmonic analysis of neural networks. *Applied and Computational Harmonic Analysis*, 6(2):197–218, 1999.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical report, Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. Stargan v2: Diverse image synthesis for multiple domains. In *CVPR*, 2020.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv*, 2018.

Taco S Cohen, Mario Geiger, and Maurice Weiler. A general theory of equivariant CNNs on homogeneous spaces. In *NeurIPS*, 2019.

Philip J. Davis. *Circulant Matrices.* Wiley, 1979.

Xibin Dong, Zhiwen Yu, Wenming Cao, Yifan Shi, and Qianli Ma. A survey on ensemble learning. *Frontiers of Computer Science*, 14:241–258, 2020.

M.A. Ganaie, Minghui Hu, A.K. Malik, M. Tanveer, and P.N. Suganthan. Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115:105151, 2022.

Chase J Gaudet and Anthony S Maida. Deep quaternion networks. In *IJCNN*, pp. 1–8. IEEE, 2018.

Yuka Hashimoto, Isao Ishikawa, Masahiro Ikeda, Fuyuta Komura, Takeshi Katsura, and Yoshinobu Kawahara. Reproducing kernel Hilbert C^*-module and kernel mean embeddings. *Journal of Machine Learning Research*, 22(267):1–56, 2021.

Yuka Hashimoto, Zhao Wang, and Tomoko Matsui. C^*-algebra net: a new approach generalizing neural network parameters to C^*-algebra. In *ICML*, 2022.

Yuka Hashimoto, Masahiro Ikeda, and Hachem Kadri. Learning in RKHM: a C^*-algebraic twist for kernel machines. In *AISTATS*, 2023.

A Hirose. Continuous complex-valued back-propagation learning. *Electronics Letters*, 28:1854–1855, 1992.

Patrick Kidger and Cristian Garcia. Equinox: neural networks in JAX via callable PyTrees and filtered transformations. *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.

Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. In *ICLR*.

Aleksandr A. Kirillov. *Elements of the Theory of Representations.* Springer, 1976.

Yann Le Cun, Leon Bottou, Yoshua Bengio, and Patrij Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

ChiYan Lee, Hideyuki Hasegawa, and Shangce Gao. Complex-valued neural networks: A comprehensive survey. *IEEE/CAA Journal of Automatica Sinica*, 9(8):1406–1426, 2022.

Jan Eric Lenssen, Matthias Fey, and Pascal Libuschewski. Group equivariant capsule networks. In *NeurIPS*, 2018.

Gerard J. Murphy. *C^*-Algebras and Operator Theory.* Academic Press, 1990.

I. Nishikawa, K. Sakakibara, T. Iritani, and Y. Kuroe. 2 types of complex-valued hopfield networks and the application to a traffic signal control. In *IJCNN*, 2005.

Tohru Nitta. A quaternary version of the back-propagation algorithm. In *ICNN*, volume 5, pp. 2753–2756, 1995.

J.K. Pearson and D.L. Bisset. Neural networks in the clifford domain. In *ICNN*, 1994.

Jorge Rivera-Rovelo, Eduardo Bayro-Corrochano, and Ruediger Dillmann. Geometric neural computing for 2d contour and 3d surface reconstruction. In *Geometric Algebra Computing*, pp. 191–209. Springer, 2010.

Fabrice Rossi and Brieuc Conan-Guez. Functional multi-layer perceptron: a non-linear tool for functional data analysis. *Neural Networks*, 18(1):45–60, 2005.

Akiyoshi Sannai, Masaaki Imaizumi, and Makoto Kawano. Improved generalization bounds of group invariant / equivariant deep networks via quotient feature spaces. In *UAI*, 2021.

Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *NeurIPS*, 2020.

Sho Sonoda and Noboru Murata. Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, 43(2):233–268, 2017.

Sho Sonoda, Isao Ishikawa, and Masahiro Ikeda. Universality of group convolutional neural networks based on ridgelet analysis on groups. In *NeurIPS*, 2022.

Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P. Srinivasan, Jonathan T. Barron, and Ren Ng. Learned Initializations for Optimizing Coordinate-Based Neural Representations. In *CVPR*, 2021.

Barinder Thind, Kevin Multani, and Jiguo Cao. Deep learning with functional inputs. *Journal of Computational and Graphical Statistics*, 0(0):1–10, 2022.

Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, Joao Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J Pal. Deep complex networks. In *ICLR*, 2018.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv*, 2017.

Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. *Computer Graphics Forum*, 2022.

A. Yadav, D. Mishra, S. Ray, R.N. Yadav, and P.K. Kalra. Representation of complex-valued neural networks: a real-valued approach. In *ICISIP*, 2005.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *NeurIPS*, 2017.

Di Zang, Xihao Chen, Juntao Lei, Zengqiang Wang, Junqi Zhang, Jiujun Cheng, and Keshuang Tang. A multi-channel geometric algebra residual network for traffic data prediction. *IET Intelligent Transport Systems*, 16(11):1549–1560, 2022.

Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609, 2022.

Xuanyu Zhu, Yi Xu, Hongteng Xu, and Changjian Chen. Quaternion convolutional neural networks. In *ECCV*, 2018.

A Appendix

A.1 Implementation details

We implemented C^*-algebra nets using `JAX` (Bradbury et al., 2018) with `equinox` (Kidger & Garcia, 2021) and `optax` (Babuschkin et al., 2020). Throughout the experiments, we used the Adam optimizer (Kingma & Ba) with a learning rate of 1.0×10^{-4}, except for the 3D NIR experiment, where Adam's initial learning rate was set to 1.0×10^{-3}. We set the batch size to 32, except for the 2D NIR, where each batch consisted of all pixels, and 3D NIR, where a batch size of 4 was used. Listings 1 and 2 illustrate linear layers of C^*-algebra nets using NumPy, equivalent to the implementations used in the experiments in Sections 4.1 and 4.2.

The implementation of 3D neural implicit representation (Section 4.1.2) is based on a simple NeRF-like model and its renderer in Tancik et al. (2021). For training, 25 views of each 3D chair from the ShapeNet dataset (Chang et al., 2015) are adopted with their 64×64 pixel reference images. The same C^*-algebra MLPs with the 2D experiments were used, except for the hyperparameters: the number of layers of four and the hidden dimensional size of 128.

The permutation-invariant DeepSet model used in Section 4.2 processes each data sample with a four-layer MLP with hyperbolic tangent activation, sum-pooling, and a linear classifier. Although we tried leaky ReLU activation as the group C^*-algebra net, this setting yielded sub-optimal results. The hidden dimension of the MLP was set to 84 to match the number of floating-point-number parameters equal to that of the group C^*-algebra net.

A.2 Additional results

Figures 6 and 7 present the additional figures of 2D INRs (Section 4.1.2). Figure 6 is an ukiyo-e counterpart of Figure 3 in the main text. Again, the noncommutative C^*-algebra net learns color details faster than

```python
import numpy as np

def matrix_valued_linear(weight: Array,
                         bias: Array,
                         input: Array
                         ) -> Array:
    """
    weight: Array of shape {output_dim}x{input_dim}x{dim_matrix}x{dim_matrix}
    bias: Array of shape {output_dim}x{dim_matrix}x{dim_matrix}
    input: Array of shape {input_dim}x{dim_matrix}x{dim_matrix}
    out: Array of shape {output_dim}x{dim_matrix}x{dim_matrix}
    """

    out = []
    for _weight, b in zip(weight, bias):
        tmp = np.zeros(weight.shape[2:])
        for w, i in zip(_weight, input):
            tmp += w @ i + b
        out.append(tmp)
    return np.array(out)
```

Listing 1: Numpy implementation of a linear layer of a C^*-algebra net over matrices used in Section 4.1.

the commutative one. Figure 7 shows average PSNR curves over three NIRs of the image initialized with different random states for 5,000 iterations. Although it is not as effective as the beginning stage, the noncommutative C^*-algebra net still outperforms the commutative one after the convergence.

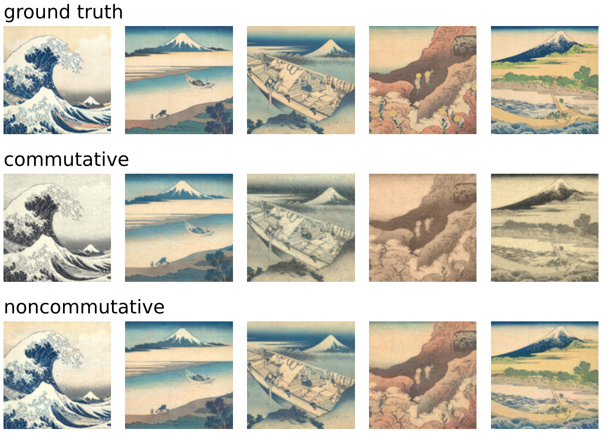


Figure 6: Ground truth images and their implicit representations of commutative and noncommutative C^*-algebra nets after 500 iterations of training.

Table 4 and Figure 8 show the additional results of 3D INRs (Section 4.1.2). Table 4 presents the average PSNR of synthesized views. As can be noticed from the synthesized views in Figures 4 and 8, the noncommutative C^*-algebra net produces less noisy output, resulting in a higher PSNR.

Commutative	Noncommutative
18.40 ± 4.30	25.22 ± 1.45

Table 4: Average PSNR over synthesized views. The specified poses of the views are unseen during training.

```python
import dataclasses
import numpy as np

@dataclasses.dataclass
class Permutation:
    # helper class to handle permutation
    value: np.ndarray

    def inverse(self) -> Permutation:
        return Permutation(self.value.argsort())

    def __mul__(self, perm: Permutation) -> Permutation:
        return Permutation(self.value[perm.value])

    def __eq__(self, other: Permutation) -> bool:
        return np.all(self.value == other.value)

    @staticmethod
    def create_hashtable(set_size: int) -> Array:
        perms = [Permutation(np.array(p)) for p in permutations(range(set_size))]
        map = {v: i for i, v in enumerate(perms)}
        out = []
        for perm in perms:
            out.append([map[perm.inverse() * _perm] for _perm in perms])
        return np.array(out)

def group_linear(weight: Array,
                 bias: Array,
                 input: Array
                 ) -> Array:
    """
    weight: {output_dim}x{input_dim}x{group_size}
    bias: {output_dim}x{group_size}
    input: {input_dim}x{group_size}
    out: {output_dim}x{group_size}
    """

    hashtable = Permutation.create_hashtable(set_size)  # {group_size}x{group_size}
    g = np.arange(hashtable.shape[0])
    out = []
    for _weight in weight:
        tmp0 = []
        for y in g:
            tmp1 = []
            for w, f in zip(_weight, input):
                tmp2 = []
                for x in g:
                    tmp2.append(w[x] * f[hashtable[x, y]])
                tmp1.append(sum(tmp2))
            tmp0.append(sum(tmp1))
        out.append(tmp0)
    return np.array(out) + bias
```

Listing 2: Numpy implementation of a group C^*-algebra linear layer used in Section 4.2.

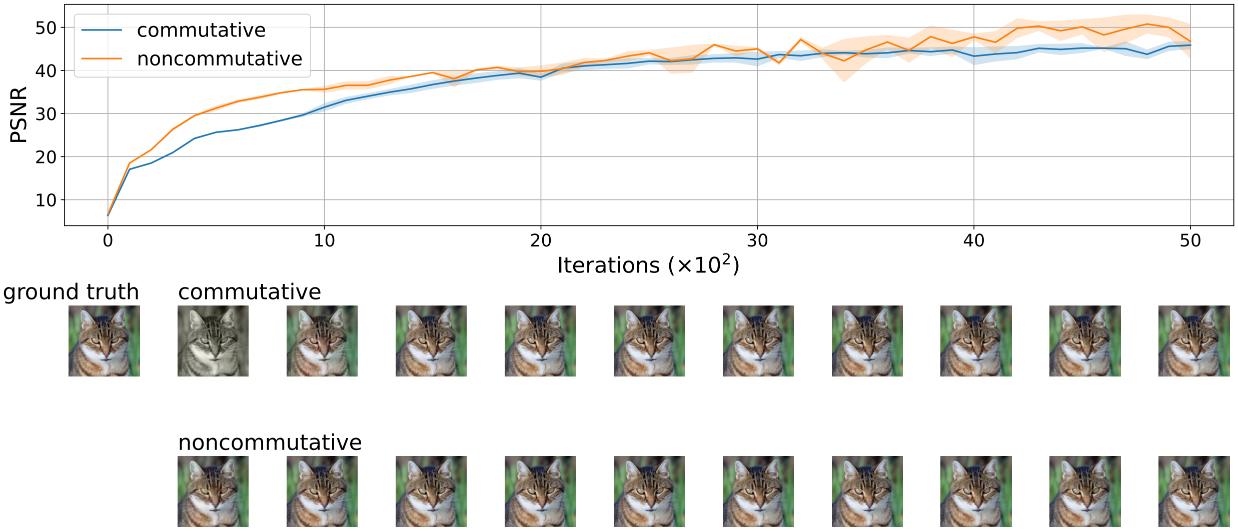


Figure 7: Average PSNR over implicit representations of the image of commutative and noncommutative C^*-algebra nets trained on five cat pictures (top) and reconstructions of the ground truth image at every 500 iterations (bottom).

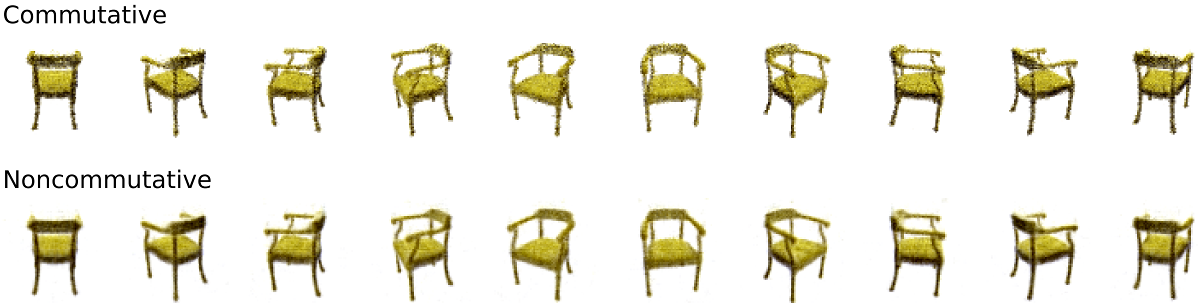


Figure 8: Synthesized views of implicit representations of a chair.