Backpropagation with N-D Vector-Valued Neurons Using Arbitrary Bilinear Products

Zhe-Cheng Fan, Tak-Shing T. Chan, *Member, IEEE,* Yi-Hsuan Yang, *Senior Member,* and Jyh-Shing R. Jang, *Member, IEEE*

***Abstract*—Vector-valued neural learning has emerged as a promising direction in deep learning recently. Traditionally, training data for neural networks (NNs) are formulated as a vector of scalars; however, its performance may not be optimal since associations among adjacent scalars are not modeled. In this paper, we propose a new vector neural architecture called the Arbitrary BIlinear Product Neural Network (ABIPNN), which processes information as vectors in each neuron, and the feed-forward projections are defined using arbitrary bilinear products. Such bilinear products can include circular convolution, seven-dimensional vector product, skew circular convolution, reversed-time circular convolution, or other new products not seen in previous work. As a proof-of-concept, we apply our proposed network to multispectral image denoising and singing voice separation. Experimental results show that ABIPNN obtains substantial improvements when compared to conventional NNs, suggesting that associations are learned during training.**

***Index Terms*—Vector neural learning, vector neural network, bilinear products, vector products, backpropagation.**

I. Introduction

VECTOR-valued neurons have received much attention lately in different scientific fields, such as communication systems, biological processing, image processing, and audio signal processing [1]–[5]. Each training sample of these applications can be represented as a multidimensional vector, which can be processed directly by vector-valued neurons. These aforementioned works have shown that vector-valued neurons have good performance in learning, association, and generalization. In the meantime, an increasing number of datasets [6]–[8] provide multidimensional data suitable for vector-valued neural learning.

In real-valued neural network (NN) learning with multidimensional data, the input is concatenated from a set of vectors and reformulated as a long vector. A neuron takes only one real value as its input and a network is configured to use as many neurons as the dimension of the long vector. But this configuration may not achieve satisfactory performance for multidimensional problems since associations within each vector are not learned. Therefore, multidimensional vector neurons have received some attention in the literature and have been proposed to address the associations among different dimensions. A vector-valued neuron accepts and represents information as a vector and elements in each vector are processed together as a single unit.

There are several approaches to extend real-valued neurons to vector-valued ones. Two-dimensional complex-valued NNs [9] have been proven to have orthogonal decision boundaries [10], [11] and the ability to solve exclusive-or (XOR) problem [12] using only a single neuron. Another extension to two dimensions is the hyperbolic neural network [13], in which all parameters are hyperbolic numbers. Decision boundaries of hyperbolic neurons are investigated in [14]. An alternative hyperbolic backpropagation algorithm [15] is developed using Wirtinger calculus. Three-dimensional neurons have been proposed in two ways. The first is based on the vector product [16] [17], in which inputs, outputs, weights and biases are all three-dimensional vectors. The second [18] is similar to the first but the weights are now three-dimensional orthogonal matrices. Four-dimensional hypercomplex-valued neurons [16], [19]–[21] are proposed by using the quaternion algebra. Eight-dimensional octonion-valued neurons [22] represent a generalization of quaternion NNs. More recently, deep complex networks [5] have been introduced using complex convolution and complex batch normalization.

However, the dimensionality of a vector-valued neuron should not be constrained to a particular number. For instance, in the task of multispectral image denoising [23], different bands of images are stacked into a tensor. Nonlinear mapping between the noisy tensor and the clean tensor is then performed. For singing voice separation [24], the data structure of a temporal-frequency matrix input is flattened into a high-dimensional vector in the traditional way, and then reshaped as a matrix so that each neuron represents and receives a vector. For EEG-based emotion recognition [7], [25], the human brain wave is filtered into five main frequency bands. Given the above observations, it seems important that the dimensionality should be an arbitrary N. Thus, a good extension of the real-valued neuron is the N-dimensional real-valued neuron [16], [26], [27], which was proposed to have N-dimensional vector inputs and outputs, but N-dimensional orthogonal matrix weights. Nevertheless, we observe that this formulation does not address the case of multiple neurons. Other architectures have also been advanced to extend real-valued neurons. For example, Clifford algebra has been employed to build vector-valued NNs with dimensionality 2^N [28]–[31]; however, we note that many datasets do not have power-of-two dimensionalities. Matrix-valued NNs [32] have been proposed in which the inputs, outputs, weights, and biases are $N \times N$ square matrices. But it is not always easy to formulate the inputs and outputs like this. Finally, for multiway classification, the tensor-factorized NN [33] integrates Tucker decomposition with neural learning, though its efficiency in regression problems is yet unproven. A similar concept would be the vector neuron models of associative memory [34], which are vector formalisms of Potts-glass NN [35]–[37]

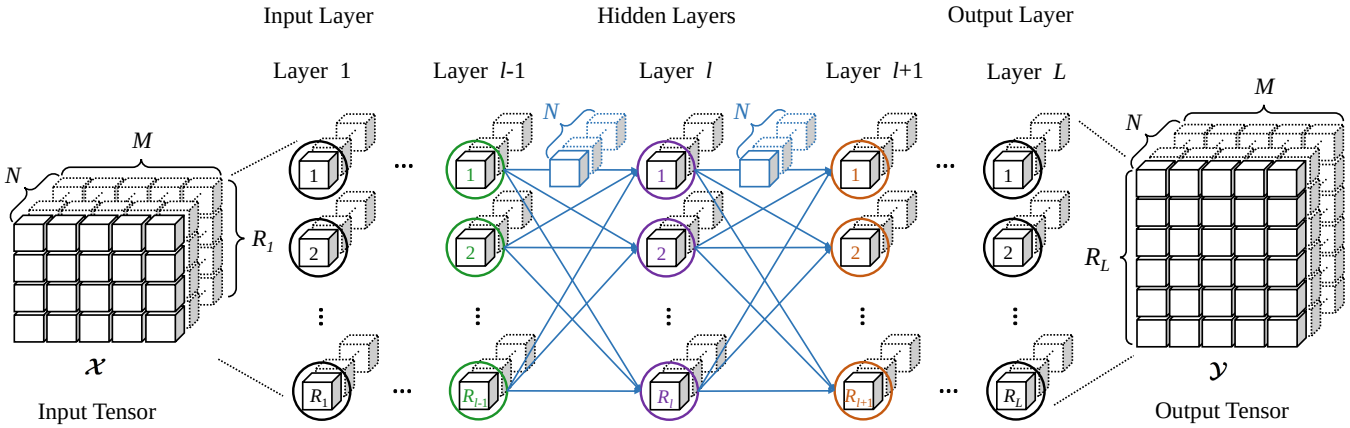


Fig. 1: Architecture of ABIPNN (best viewed in color), where the inputs, outputs, weights, and biases are all N-dimensional vectors. Cubes in the architecture represent scalars. Blue lines between hidden layers represent weights. M represents the number of training data patterns. R_l represents the dimension at layer l, where $1 \leq l \leq L$. Among them, R_1 and R_L are respectively the dimensions of the input and output layers. From R_2 to R_{L-1} are the dimensions of the hidden layers. $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$ respectively represent the input and output tensors. When N equals one, each neuron represents a scalar, depicted as a cube with solid lines. This architecture can be viewed as a conventional NN. When $N > 1$, each neuron represents a vector, depicted as the concatenation of cubes connected by the dotted lines.

and parametrical NN (PNN) [38], but this concept does not belong to supervised learning since there are no inputs and corresponding targets for training.

In light of the above observations, we propose a new vector-valued NN architecture consisting of N-dimensional vector-valued neurons, where N is an arbitrary positive integer (as shown in Fig. 1). For each neuron, the inputs, outputs, weights, and biases are all N-dimensional vectors. Our key observation is that the products used by the aforementioned vector-valued NNs (such as the vector product [17] and quaternion multiplication [19]) are bilinear products (defined in Section II). This prompts us to propose a general form of bilinear neurons to model the associations between the vector elements. We call it the Arbitrary BIlinear Product Neural Network (ABIPNN). When N equals one, each neuron represents a scalar and the architecture performs matrix multiplications like a conventional deep neural network (DNN). When N is larger than one, each neuron represents a vector and the architecture uses bilinear products for multiplications. In this way, the proposed architecture not only allows for using vectors of arbitrary dimensionality in vector-valued NNs, but also all the vector-valued products that are bilinear.

The remainder of this paper is organized as follows. Section II derives the feedforward and backpropagation processes using the proposed bilinear neurons. Section III compares the performance between ABIPNN and conventional NNs in singing voice separation and multispectral image denoising. We conclude the paper in Section IV.

II. Vector Neural Learning

The idea of ABIPNN is to extend the data type of each neuron from scalars to vectors by replacing multiplications with arbitrary bilinear products. Such products can be used to

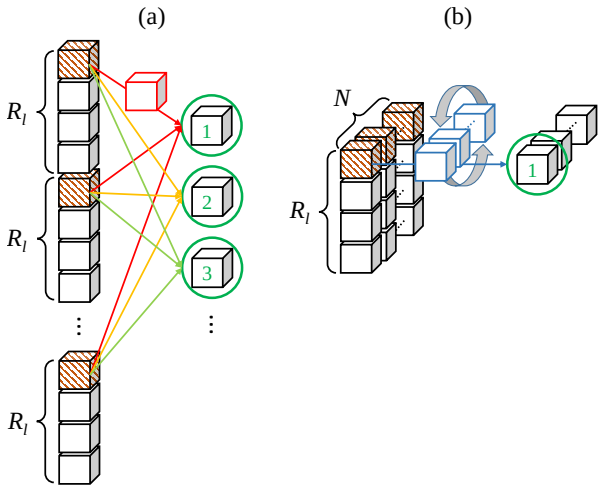


Fig. 2: Illustration of the operations employed by the (a) conventional NNs and (b) ABIPNNs; best viewed in color. In (a), vectors are concatenated together as a multidimensional input. Each weight stands for a scalar, depicted as a cube. Take the first entry of each vector for example, the associations between the brown cubes are not learned since each hidden neuron receives information by using its group of weights, which are depicted in the same color. Hidden neurons do not share identical weights when performing learning. In (b), vectors are stacked into a $R_l \times 1 \times N$ tensor (or lateral matrix) and brown cubes are concatenated together. Each neuron represents a vector and receives information by using the weight vector, depicted as blue cubes. For example, when we leverage circular convolution as the bilinear product in an ABIPNN, the weight vector can be deemed as a linear kernel mask which captures the associations between brown cubes by rotating itself in the learning process.

learn the associations between vector elements in the input (see Fig. 2). In the following, tensors are represented by bold uppercase calligraphic letters, matrices by bold uppercase letters, and vectors by bold lowercase letters. Matrix slices of tensors are represented as matrices and vector slices of matrices are represented as vectors. The notational conventions used in this paper are summarized in Table I.

TABLE I: NOTATIONAL CONVENTIONS

$\boldsymbol{\mathcal{X}}$	Input tensor
$\mathbf{X}_m$=$\boldsymbol{\mathcal{X}}_{:m:}$	Lateral slice from input tensor $\boldsymbol{\mathcal{X}}$
$\mathbf{x}$	Vector from input tensor $\boldsymbol{\mathcal{X}}$
$\boldsymbol{\mathcal{Y}}$	Output tensor
$\mathbf{Y}_m$=$\boldsymbol{\mathcal{Y}}_{:m:}$	Lateral slice from output tensor $\boldsymbol{\mathcal{X}}$
$\boldsymbol{\mathcal{W}}^l$	Weight tensor in layer l
$\mathbf{w}_{ij}^l = \boldsymbol{\mathcal{W}}_{ij:}^l$	Vector from tensor $\boldsymbol{\mathcal{W}}^l$
$\boldsymbol{\mathcal{B}}^l$	Bias tensor in layer l
$\mathbf{b}_i^l = \boldsymbol{\mathcal{B}}_{im:}^l$	Vector from tensor $\boldsymbol{\mathcal{B}}^l$
$\boldsymbol{\mathcal{Z}}^l$	Input hidden tensor in layer l
$\mathbf{z}_i^l = \boldsymbol{\mathcal{Z}}_{im:}^l$	Vector from tensor $\boldsymbol{\mathcal{Z}}^l$
$\boldsymbol{\mathcal{A}}^l$	Output hidden tensor in layer l
$\mathbf{a}_i^l = \boldsymbol{\mathcal{A}}_{im:}^l$	Vector from tensor $\boldsymbol{\mathcal{A}}^l$
$\boldsymbol{\mathcal{D}}^l$	Local gradient tensor in layer l
$\mathbf{d}_i^l = \boldsymbol{\mathcal{D}}_{km:}^l$	Vector from tensor $\boldsymbol{\mathcal{D}}^l$
$\bullet\colon \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^N$	Arbitrary bilinear product
$[\cdot]_\bullet$	Matrix representation of $\bullet$ (Eq. 3)
$[\cdot]_\bullet^\dagger$	Transmuted representation of $\bullet$ (Eq. 5)

A. Generalizing N-D Vector Neurons with Bilinear Products

To begin with, we identify a generalization of all the products used in previous vector-valued NN literature, including vector product [17], quaternion multiplication [19], octonion multiplication [22], and so on. We have been able to confirm that all of these products are bilinear (defined below). Motivated by this important observation, we will extend the vector-valued NN [26] to use any kind of bilinear product between two N-dimensional vectors.

Assume an N-dimensional space with standard basis $\{\mathbf{e}_1,\ \mathbf{e}_2,\ \ldots,\ \mathbf{e}_N\}$ as column vectors. Let $\mathbf{p} = \sum_{n=1}^N p_n \mathbf{e}_n = [p_1,\ p_2,\ \ldots,\ p_N]^T$ and $\mathbf{q} = \sum_{n=1}^N q_n \mathbf{e}_n = [q_1,\ q_2,\ \ldots,\ q_N]^T$ be two vectors in this space. A product $\bullet\colon \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^N$ is bilinear if and only if $\mathbf{p} \bullet \mathbf{q}$ is linear when we hold one of $\mathbf{p}$ or $\mathbf{q}$ fixed. If $\bullet$ is bilinear, we have:

$$\mathbf{p} \bullet \mathbf{q} = \mathbf{p} \bullet \left(\sum_{n=1}^N q_n \mathbf{e}_n \right), \tag{1}$$

where we have expanded the second term. Due to bilinearity,

$$\begin{aligned} \mathbf{p} \bullet \mathbf{q} &= \sum_{n=1}^N (\mathbf{p} \bullet \mathbf{e}_n)\, q_n \\ &= \begin{bmatrix} \mathbf{p} \bullet \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix},\ \mathbf{p} \bullet \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix},\ \ldots,\ \mathbf{p} \bullet \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_N \end{bmatrix} \\ &= [\mathbf{p} \bullet \mathbf{e}_1,\ \mathbf{p} \bullet \mathbf{e}_2,\ \ldots,\ \mathbf{p} \bullet \mathbf{e}_N]\, \mathbf{q}. \end{aligned} \tag{2}$$

If we let $[\mathbf{p}]_\bullet = [\mathbf{p} \bullet \mathbf{e}_1,\ \mathbf{p} \bullet \mathbf{e}_2,\ \ldots,\ \mathbf{p} \bullet \mathbf{e}_N]$, then:

$$\mathbf{p} \bullet \mathbf{q} = [\mathbf{p}]_\bullet\, \mathbf{q}. \tag{3}$$

Similarly, if we expand the first term, we can write:

$$\begin{aligned} \mathbf{p} \bullet \mathbf{q} &= \left(\sum_{n=1}^N p_n \mathbf{e}_n \right) \bullet \mathbf{q} = \sum_{n=1}^N p_n\, (\mathbf{e}_n \bullet \mathbf{q}) \\ &= [\mathbf{e}_1 \bullet \mathbf{q},\ \mathbf{e}_2 \bullet \mathbf{q},\ \ldots,\ \mathbf{e}_N \bullet \mathbf{q}]\, \mathbf{p}. \end{aligned} \tag{4}$$

Let $[\mathbf{q}]_\bullet^\dagger = [\mathbf{e}_1 \bullet \mathbf{q},\ \mathbf{e}_2 \bullet \mathbf{q},\ \ldots,\ \mathbf{e}_N \bullet \mathbf{q}]$, we also have:

$$\mathbf{p} \bullet \mathbf{q} = [\mathbf{q}]_\bullet^\dagger\, \mathbf{p}. \tag{5}$$

Eqs. 3 and 5 are the two possible matrix representations of the bilinear product $\bullet$. Here we call $[\mathbf{p}]_\bullet$ its matrix representation and $[\mathbf{q}]_\bullet^\dagger$ its transmuted representation (the term *transmuted* is originally used for quaternions [39] but here we extend it for general bilinear products). In what follows, the feedforward and backpropagation processes will be described by using the above notations for bilinear products and their representations. For convenience, in the rest of this paper, if $\mathbf{p}$ or $\mathbf{q}$ are not column vectors, then $\mathbf{p} \bullet \mathbf{q}$ will implicitly reshape them so that the above equations make sense.

B. Feedforward Process in a Bilinear Product Neuron

As before, we assume the inputs, outputs, weights, and biases are N-dimensional vectors. Suppose we have an L-layer vector-valued NN. For hidden layer l ($1 \le l \le L$), the input vector $\mathbf{z}_i^l$ of the hidden neuron i is defined as:

$$\mathbf{z}_i^l = \sum_{j=1}^{R_{l-1}} \mathbf{w}_{ij}^l \bullet \mathbf{a}_j^{l-1} + \mathbf{b}_i^l\ , \tag{6}$$

where $\bullet$ denotes an arbitrary bilinear product, $\mathbf{w}_{ij}^l$ stands for the weight vector connecting a neuron j ($1 \le j \le R_{l-1}$) in layer $l-1$ to a neuron i ($1 \le i \le R_l$) in layer l, and $\mathbf{w}_{ij}^l = \sum_{n=1}^N w_{ijn}^l \mathbf{e}_n = \left[w_{ij1}^l,\ w_{ij2}^l, \ldots, w_{ijN}^l\right]^T \in \mathbb{R}^N$. The output vector of neuron j in layer $l-1$ is $\mathbf{a}_j^{l-1} = \sum_{n=1}^N a_{jn}^{l-1} \mathbf{e}_n = \left[a_{j1}^{l-1},\ a_{j2}^{l-1}, \ldots,\ a_{jN}^{l-1}\right]^T \in \mathbb{R}^N$, the bias vector of neuron i in layer l is $\mathbf{b}_i^l = \sum_{n=1}^N b_{in}^l \mathbf{e}_n = \left[b_{i1}^l,\ b_{i2}^l, \ldots,\ b_{iN}^l\right]^T \in \mathbb{R}^N$. As a result, $\mathbf{z}_i^l = \sum_{n=1}^N z_{in}^l \mathbf{e}_n = \left[z_{i1}^l,\ z_{i2}^l, \ldots,\ z_{iN}^l\right]^T \in \mathbb{R}^N$. When l equals 1, the vector $\mathbf{z}$ is simply the input vector $\mathbf{x}$. After $\mathbf{z}_i^l$ is calculated, the output vector $\mathbf{a}_i^l$ of the hidden neuron i is as follows:

$$\mathbf{a}_i^l = \begin{bmatrix} a_{i1}^l \\ a_{i2}^l \\ \vdots \\ a_{iN}^l \end{bmatrix} = \phi(\mathbf{z}_i^l) = \begin{bmatrix} \phi(z_{i1}^l) \\ \phi(z_{i2}^l) \\ \vdots \\ \phi(z_{iN}^l) \end{bmatrix}, \tag{7}$$

where ϕ could be any differentiable activation function. Then the output vector of neuron g ($1 \le g \le R_L$) in output layer

L is defined as:

$$\mathbf{y}_g^L = \begin{bmatrix} y_{g1}^L \\ y_{g2}^L \\ \vdots \\ y_{gN}^L \end{bmatrix} = \phi(\mathbf{z}_g^L) = \begin{bmatrix} \phi(z_{g1}^L) \\ \phi(z_{g2}^L) \\ \vdots \\ \phi(z_{gN}^L) \end{bmatrix}, \tag{8}$$

where $\mathbf{z}_g^L$ is the input vector and $\mathbf{y}_g^L$ is the output vector of a neuron g. The objective of the training process is to estimate the parameters that minimize the cost function, defined as:

$$C(\mathbf{\Theta}) = \sum_{m=1}^{M} loss(\mathbf{Y}_m, f(\mathbf{X}_m; \mathbf{\Theta})) \ , \tag{9}$$

where M stands for the number of training data patterns, $\mathbf{\Theta}$ is the set of all training parameters (weights and biases), and $\mathbf{Y}_m$ is the training label related to the input $\boldsymbol{X}_m$. The output $f(\mathbf{X}_m; \mathbf{\Theta})$ is the predicted version of $\mathbf{Y}_m$, made up of $\mathbf{y}_g^L$ mentioned in Eq. 9. The $loss(\mathbf{Y}_m, f(\mathbf{X}_m; \mathbf{\Theta}))$ function measures the difference between the predicted results and the training labels.

C. Backpropagation Algorithm in a Bilinear Product Neuron

The bilinear product is also employed in the process of backpropagation learning. Before we present the error backpropagation process (Algorithm 1), we need to first derive:

1) The gradients of the biases $\mathbf{b}_i^l$.
2) The gradients of the weights $\mathbf{w}_{ij}^l$.
3) The backpropagation of local gradients $\mathbf{d}_k^{l+1} \rightarrow \mathbf{d}_i^l$.

Following Eq. 3, the bilinear product can be viewed as matrix-vector multiplication. Hence, Eq. 6 can be rewritten as

$$\begin{aligned} \mathbf{z}_i^l &= \sum_{j=1}^{R_{l-1}} \mathbf{w}_{ij}^l \bullet \mathbf{a}_j^{l-1} + \mathbf{b}_i^l = \sum_{j=1}^{R_{l-1}} \left[\mathbf{w}_{ij}^l\right]_\bullet \mathbf{a}_j^{l-1} + \mathbf{b}_i^l \\ &= \sum_{j=1}^{R_{l-1}} \left[\mathbf{w}_{ij}^l \bullet \mathbf{e}_1,\ \mathbf{w}_{ij}^l \bullet \mathbf{e}_2,\ \ldots, \mathbf{w}_{ij}^l \bullet \mathbf{e}_N\right] \mathbf{a}_j^{l-1} + \mathbf{b}_i^l \\ &= \sum_{j=1}^{R_{l-1}} \left(a_{j1}^{l-1} \mathbf{w}_{ij}^l \bullet \mathbf{e}_1 + \cdots + a_{jN}^{l-1} \mathbf{w}_{ij}^l \bullet \mathbf{e}_N\right) + \mathbf{b}_i^l. \end{aligned} \tag{10}$$

Here, $\mathbf{w}_{ij}^l$ can be formulated as the summation of scalar-vector multiplications:

$$\begin{aligned} \mathbf{z}_i^l = \sum_{j=1}^{R_{l-1}} &\left(a_{j1}^{l-1} \left(w_{ij1}^l \mathbf{e}_1 + \cdots + w_{ijN}^l \mathbf{e}_N\right) \bullet \mathbf{e}_1 \right. \\ &+ a_{j2}^{l-1} \left(w_{ij1}^l \mathbf{e}_1 + \cdots + w_{ijN}^l \mathbf{e}_N\right) \bullet \mathbf{e}_2 + \ldots \\ &\left. + a_{jN}^{l-1} \left(w_{ij1}^l \mathbf{e}_1 + \cdots + w_{ijN}^l \mathbf{e}_N\right) \bullet \mathbf{e}_N\right) + \mathbf{b}_i^l. \end{aligned} \tag{11}$$

To estimate the first-order partial derivatives of a vector-valued function, we apply the concept of Jacobian matrix, which is a matrix containing all the partial derivatives. For the vector-valued function $f : \mathbb{R}^N \rightarrow \mathbb{R}^M$, the Jacobian matrix is defined as $\left[\frac{\partial f}{\partial \boldsymbol{x}}\right]_{ij} = \frac{\partial}{\partial x_j} f(\boldsymbol{x})_i \in \mathbb{R}^{N \times M}$. The partial derivative of C with respect to $\mathbf{b}_i^l$ is as follows:

$$\begin{aligned} \frac{\partial C}{\partial \mathbf{b}_i^l} &= \left[\frac{\partial C}{\partial b_{i1}^l},\ \frac{\partial C}{\partial b_{i2}^l},\ \ldots,\ \frac{\partial C}{\partial b_{iN}^l}\right] \\ &= \begin{bmatrix} \frac{\partial C}{\partial z_{i1}^l}\frac{\partial z_{i1}^l}{\partial b_{i1}^l} + \frac{\partial C}{\partial z_{i2}^l}\frac{\partial z_{i2}^l}{\partial b_{i1}^l} + \cdots + \frac{\partial C}{\partial z_{iN}^l}\frac{\partial z_{iN}^l}{\partial b_{i1}^l} \\ \vdots \\ \frac{\partial C}{\partial z_{i1}^l}\frac{\partial z_{i1}^l}{\partial b_{iN}^l} + \frac{\partial C}{\partial z_{i2}^l}\frac{\partial z_{i2}^l}{\partial b_{iN}^l} + \cdots + \frac{\partial C}{\partial z_{iN}^l}\frac{\partial z_{iN}^l}{\partial b_{iN}^l} \end{bmatrix}^T, \end{aligned} \tag{12}$$

where $\frac{\partial C}{\partial \mathbf{b}_i^l}$ stands for the gradient vector of $\mathbf{b}_i^l$ belonging to neuron i in layer l. The terms $\frac{\partial C}{\partial z_{in}^l}$ $(1 \leq n \leq N)$ are extracted into an N-dimensional local gradient vector $\frac{\partial C}{\partial \mathbf{z}_i^l}$ and we call it $\mathbf{d}_i^l$ for convenience:

$$\begin{aligned} \frac{\partial C}{\partial \mathbf{z}_i^l} &= \left[\frac{\partial C}{\partial z_{i1}^l},\ \frac{\partial C}{\partial z_{i2}^l},\ \ldots,\ \frac{\partial C}{\partial z_{iN}^l}\right] \\ &\triangleq \left[d_{i1}^l,\ d_{i2}^l,\ \ldots,\ d_{iN}^l\right] = \mathbf{d}_i^l. \end{aligned} \tag{13}$$

The terms $\left\{\frac{\partial z_{i1}^l}{\partial b_{in}^l},\ \frac{\partial z_{i2}^l}{\partial b_{in}^l},\ \ldots,\ \frac{\partial z_{iN}^l}{\partial b_{in}^l}\right\}$ $(1 \leq n \leq N)$ can be obtained by differentiating Eq. 11:

$$\begin{aligned} \frac{\partial \mathbf{z}_i^l}{\partial b_{in}^l} &= \left[\frac{\partial z_{i1}^l}{\partial b_{in}^l},\ \frac{\partial z_{i2}^l}{\partial b_{in}^l},\ \ldots,\ \frac{\partial z_{iN}^l}{\partial b_{in}^l}\right]^T \\ &= \left[\frac{\partial b_{i1}^l}{\partial b_{in}^l},\ \frac{\partial b_{i2}^l}{\partial b_{in}^l},\ \ldots,\ \frac{\partial b_{iN}^l}{\partial b_{in}^l}\right]^T \\ &= \underbrace{\left[0,\ \ldots,\ 1,\ \ldots,\ 0\right]^T}_{\text{the } n\text{-th entry is 1}}. \end{aligned} \tag{14}$$

By combining Eqs. 13 and 14, the derivative $\frac{\partial C}{\partial b_{in}^l}$ can be formulated as a dot product:

$$\frac{\partial C}{\partial b_{in}^l} = \mathbf{d}_i^l \left[\frac{\partial \mathbf{z}_i^l}{\partial b_{in}^l}\right] = \underbrace{\left[d_{i1}^l,\ d_{i2}^l,\ \ldots,\ d_{iN}^l\right]}_{1 \times N} \underbrace{\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}}_{N \times 1} = d_{in}^l. \tag{15}$$

Then, each entry of $\frac{\partial C}{\partial \mathbf{b}_i^l}$ can be estimated by the same procedure from Eqs. 14 and 15. After each entry is estimated, Eq. 12 can be derived as follows:

$$\frac{\partial C}{\partial \mathbf{b}_i^l} = \left[\frac{\partial C}{\partial b_{i1}^l},\ \ldots,\ \frac{\partial C}{\partial b_{iN}^l}\right] = \left[d_{i1}^l,\ \ldots,\ d_{iN}^l\right] = \mathbf{d}_i^l. \tag{16}$$

As a result, it is evident that the derivative of $\mathbf{b}_i^l$ is completely equal to local gradient vector $\mathbf{d}_i^l$.

Next, we derive the partial derivative of C with respect to

$\mathbf{w}_{ij}^l$ as follows:

$$\begin{aligned}\frac{\partial C}{\partial \mathbf{w}_{ij}^l} &= \left[\frac{\partial C}{\partial w_{ij1}^l}, \frac{\partial C}{\partial w_{ij2}^l}, \ \ldots, \ \frac{\partial C}{\partial w_{ijN}^l}\right] \\ &= \begin{bmatrix} \frac{\partial C}{\partial z_{i1}^l}\frac{\partial z_{i1}^l}{\partial w_{ij1}^l} + \frac{\partial C}{\partial z_{i2}^l}\frac{\partial z_{i2}^l}{\partial w_{ij1}^l} + \cdots + \frac{\partial C}{\partial z_{iN}^l}\frac{\partial z_{iN}^l}{\partial w_{ij1}^l} \\ \frac{\partial C}{\partial z_{i1}^l}\frac{\partial z_{i1}^l}{\partial w_{ij2}^l} + \frac{\partial C}{\partial z_{i2}^l}\frac{\partial z_{i2}^l}{\partial w_{ij2}^l} + \cdots + \frac{\partial C}{\partial z_{iN}^l}\frac{\partial z_{iN}^l}{\partial w_{ij2}^l} \\ \vdots \\ \frac{\partial C}{\partial z_{i1}^l}\frac{\partial z_{i1}^l}{\partial w_{ijN}^l} + \frac{\partial C}{\partial z_{i2}^l}\frac{\partial z_{i2}^l}{\partial w_{ijN}^l} + \cdots + \frac{\partial C}{\partial z_{iN}^l}\frac{\partial z_{iN}^l}{\partial w_{ijN}^l} \end{bmatrix}^T \end{aligned} \tag{17}$$

where $\frac{\partial C}{\partial \mathbf{w}_{ij}^l}$ stands for the gradient vector of $\mathbf{w}_{ij}^l$ connecting neuron j to neuron i in layer l and it consists of N elements. From Eq. 13, here the terms $\left\{\frac{\partial C}{\partial z_{i1}^l}, \frac{\partial C}{\partial z_{i2}^l}, \ \ldots, \ \frac{\partial C}{\partial z_{iN}^l}\right\}$ are extracted as the local gradient vector $\mathbf{d}_i^l$. Likewise, we can obtain the terms $\left\{\frac{\partial z_{i1}^l}{\partial w_{ijn}^l}, \frac{\partial z_{i2}^l}{\partial w_{ijn}^l}, \ \ldots, \ \frac{\partial z_{iN}^l}{\partial w_{ijn}^l}\right\}$ $(1 \leq n \leq N)$ by differentiating Eq. 11:

$$\begin{aligned}\frac{\partial \mathbf{z}_i^l}{\partial w_{ijn}^l} &= \left[\frac{\partial z_{i1}^l}{\partial w_{ijn}^l}, \frac{\partial z_{i2}^l}{\partial w_{ijn}^l}, \ \ldots, \ \frac{\partial z_{iN}^l}{\partial w_{ijn}^l}\right]^T \\ &= a_{j1}^{l-1}\mathbf{e}_n \bullet \mathbf{e}_1 + a_{j2}^{l-1}\mathbf{e}_n \bullet \mathbf{e}_2 + \cdots + a_{jN}^{l-1}\mathbf{e}_n \bullet \mathbf{e}_N \\ &= \left[\mathbf{e}_n \bullet \mathbf{e}_1, \mathbf{e}_n \bullet \mathbf{e}_2, \ldots, \mathbf{e}_n \bullet \mathbf{e}_N\right]\left[a_{j1}^{l-1}, \ldots a_{jN}^{l-1}\right]^T \\ &= \left[\mathbf{e}_n \bullet \mathbf{e}_1, \mathbf{e}_n \bullet \mathbf{e}_2, \ldots, \mathbf{e}_n \bullet \mathbf{e}_N\right]\mathbf{a}_j^{l-1}, \end{aligned} \tag{18}$$

which is formulated as a matrix-vector multiplication. Each column in the matrix is the bilinear product of two standard bases. Different bilinear products contribute to different results. Then, the derivative $\frac{\partial C}{\partial w_{ijn}^l}$ can be formulated as a dot product by combining Eqs. 13 and 18:

$$\begin{aligned}\frac{\partial C}{\partial w_{ijn}^l} &= \mathbf{d}_i^l\left[\frac{\partial \mathbf{z}_i^l}{\partial w_{ijn}^l}\right] \\ &= \underbrace{\left[d_{i1}^l, \ d_{i2}^l, \ \ldots, \ d_{iN}^l\right]}_{1\times N} \\ &\qquad \left[\underbrace{\left[\mathbf{e}_n \bullet \mathbf{e}_1, \ \mathbf{e}_n \bullet \mathbf{e}_2, \ \ldots, \ \mathbf{e}_n \bullet \mathbf{e}_N\right]}_{N\times N}\underbrace{\mathbf{a}_j^{l-1}}_{N\times 1}\right] \\ &= \mathbf{d}_i^l\left[\sum_{h=1}^{N} a_{jh}^{l-1}\mathbf{e}_n \bullet \mathbf{e}_h\right]. \end{aligned} \tag{19}$$

Each entry of $\frac{\partial C}{\partial \mathbf{w}_{ij}^l}$ can be estimated by the same procedure from Eqs. 18 and 19. After each entry is estimated, Eq. 17 can be rewritten as:

$$\begin{aligned}\frac{\partial C}{\partial \mathbf{w}_{ij}^l} &= \left[\frac{\partial C}{\partial w_{ij1}^l}, \frac{\partial C}{\partial w_{ij2}^l}, \ \ldots, \ \frac{\partial C}{\partial w_{ijN}^l}\right] \\ &= \left[\mathbf{d}_i^l\left[\sum_{h=1}^{N} a_{jh}^{l-1}\mathbf{e}_1 \bullet \mathbf{e}_h\right], \ldots, \mathbf{d}_i^l\left[\sum_{h=1}^{N} a_{jh}^{l-1}\mathbf{e}_N \bullet \mathbf{e}_h\right]\right] \\ &= \mathbf{d}_i^l\left[\mathbf{e}_1 \bullet \left[\sum_{h=1}^{N} a_{jh}^{l-1}\mathbf{e}_h\right], \ \ldots, \mathbf{e}_N \bullet \left[\sum_{h=1}^{N} a_{jh}^{l-1}\mathbf{e}_h\right]\right], \end{aligned} \tag{20}$$

and referring to Eq. 4, we can rewrite Eq. 20 into:

$$\begin{aligned}\frac{\partial C}{\partial \mathbf{w}_{ij}^l} &= \mathbf{d}_i^l\left[\mathbf{e}_1 \bullet \mathbf{a}_j^{l-1}, \mathbf{e}_2 \bullet \mathbf{a}_j^{l-1}, \ \ldots, \ \mathbf{e}_N \bullet \mathbf{a}_j^{l-1}\right] \\ &= \mathbf{d}_i^l\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger, \end{aligned} \tag{21}$$

which is a vector-matrix multiplication with the transmuted representation.

After the derivative $\frac{\partial C}{\partial \mathbf{w}_{ij}^l}$ is calculated, we derive the local gradient vector $\mathbf{d}_i^l$ from layer $l+1$ by:

$$\begin{aligned}\mathbf{d}_i^l = \frac{\partial C}{\partial \mathbf{z}_i^l} &= \sum_{k=1}^{R_{l+1}} \frac{\partial C}{\partial \mathbf{z}_k^{l+1}}\frac{\partial \mathbf{z}_k^{l+1}}{\partial \mathbf{a}_i^l}\frac{\partial \mathbf{a}_i^l}{\partial \mathbf{z}_i^l} \\ &= \sum_{k=1}^{R_{l+1}} \frac{\partial C}{\partial \mathbf{z}_k^{l+1}}\frac{\partial \mathbf{z}_k^{l+1}}{\partial \mathbf{a}_i^l}\dot{\phi}(\mathbf{z}_i^l), \end{aligned} \tag{22}$$

which is a vector-matrix-matrix multiplication, where $\mathbf{z}_k^{l+1}$ stands for an N-dimensional input vector of neuron k in layer $l+1$ and $\mathbf{a}_i^l$ stands for an N-dimensional output vector of neuron i in layer l. The vector $\frac{\partial C}{\partial \mathbf{z}_k^{l+1}}$ is regarded as an N-dimensional local gradient vector, defined as $\mathbf{d}_k^{l+1}$ in layer $l+1$. Referring to Eqs. 12 to 16, we can see that $\mathbf{d}_k^{l+1}$ can be regarded as the derivative of $\mathbf{b}_k^{l+1}$. The matrix $\dot{\phi}(\mathbf{z}_i^l)$ represents the derivative of the activation function, which is an $N \times N$ diagonal matrix:

$$\begin{aligned}\dot{\phi}(\mathbf{z}_i^l) &= \frac{\partial \mathbf{a}_i^l}{\partial \mathbf{z}_i^l} \\ &= \begin{bmatrix} \frac{\partial a_{i1}^l}{\partial z_{i1}^l} & 0 & \ldots & 0 \\ 0 & \frac{\partial a_{i2}^l}{\partial z_{i2}^l} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \frac{\partial a_{iN}^l}{\partial z_{iN}^l} \end{bmatrix} \\ &= \begin{bmatrix} \dot{\phi}(z_{i1}^l) & 0 & \ldots & 0 \\ 0 & \dot{\phi}(z_{i2}^l) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \dot{\phi}(z_{iN}^l) \end{bmatrix}, \end{aligned} \tag{23}$$

where $\dot{\phi}$ can be an arbitrary differentiable activation function.

Next, the matrix $\frac{\partial \mathbf{z}_k^{l+1}}{\partial \mathbf{a}_i^l}$ is also a $N \times N$ square matrix:

$$\frac{\partial \mathbf{z}_k^{l+1}}{\partial \mathbf{a}_i^l} = \begin{bmatrix} \frac{\partial z_{k1}^{l+1}}{\partial a_{i1}^l} & \frac{\partial z_{k1}^{l+1}}{\partial a_{i2}^l} & \cdots & \frac{\partial z_{k1}^{l+1}}{\partial a_{iN}^l} \\ \frac{\partial z_{k2}^{l+1}}{\partial a_{i1}^l} & \frac{\partial z_{k2}^{l+1}}{\partial a_{i2}^l} & \cdots & \frac{\partial z_{k2}^{l+1}}{\partial a_{iN}^l} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial z_{kN}^{l+1}}{\partial a_{i1}^l} & \frac{\partial z_{kN}^{l+1}}{\partial a_{i2}^l} & \cdots & \frac{\partial z_{kN}^{l+1}}{\partial a_{iN}^l} \end{bmatrix}. \tag{24}$$

We calculate the above derivatives columnwise. The calculations are based on the feedforward process of $\mathbf{z}_k^{l+1}$:

$$\begin{aligned} \mathbf{z}_k^{l+1} &= \sum_{i=1}^{R_l} \mathbf{w}_{ki}^{l+1} \bullet \mathbf{a}_i^l + \mathbf{b}_k^{l+1} \\ &= \sum_{i=1}^{R_l} \mathbf{w}_{ki}^{l+1} \bullet \left[a_{i1}^l \mathbf{e}_1 + a_{i2}^l \mathbf{e}_2 + \cdots + a_{iN}^l \mathbf{e}_N\right] + \mathbf{b}_k^{l+1}. \end{aligned} \tag{25}$$

Then, the derivative of the n-th column of Eq. 24 is derived by differentiating Eq. 25:

$$\begin{aligned} \frac{\partial \mathbf{z}_k^{l+1}}{\partial a_{in}^l} &= \left[\frac{\partial z_{k1}^{l+1}}{\partial a_{in}^l}, \frac{\partial z_{k2}^{l+1}}{\partial a_{in}^l}, \ldots, \frac{\partial z_{kN}^{l+1}}{\partial a_{in}^l}\right]^T \\ &= \mathbf{w}_{ki}^{l+1} \bullet \mathbf{e}_n \\ &= \left(w_{ki1}^{l+1} \mathbf{e}_1 + w_{ki2}^{l+1} \mathbf{e}_2 + \cdots + w_{kiN}^{l+1} \mathbf{e}_N\right) \bullet \mathbf{e}_n \\ &= \sum_{h=1}^{N} w_{kih}^{l+1} \mathbf{e}_h \bullet \mathbf{e}_n \\ &= \underbrace{[\mathbf{e}_1 \bullet \mathbf{e}_n,\ \mathbf{e}_2 \bullet \mathbf{e}_n,\ \ldots\ , \mathbf{e}_N \bullet \mathbf{e}_n]}_{N \times N} \underbrace{\mathbf{w}_{ki}^{l+1}}_{N \times 1}, \end{aligned} \tag{26}$$

resulting in a column vector. Since each column of the matrix $\frac{\partial \mathbf{z}_k^{l+1}}{\partial \mathbf{a}_i^l}$ can be estimated by the same procedure mentioned above, the derivatives of the matrix can be derived as:

$$\begin{aligned} \frac{\partial \mathbf{z}_k^{l+1}}{\partial \mathbf{a}_i^l} &= \left[\left[\sum_{h=1}^{N} w_{kih}^{l+1} \mathbf{e}_h \bullet \mathbf{e}_1\right], \left[\sum_{h=1}^{N} w_{kih}^{l+1} \mathbf{e}_h \bullet \mathbf{e}_2\right], \right. \\ &\qquad \left. \ldots, \left[\sum_{h=1}^{N} w_{kih}^{l+1} \mathbf{e}_h \bullet \mathbf{e}_N\right]\right] \\ &= \left[\mathbf{w}_{ki}^{l+1} \bullet \mathbf{e}_1,\ \mathbf{w}_{ki}^{l+1} \bullet \mathbf{e}_2,\ \ldots,\ \mathbf{w}_{ki}^{l+1} \bullet \mathbf{e}_N\right] \\ &= \left[\mathbf{w}_{ki}^{l+1}\right]_\bullet, \end{aligned} \tag{27}$$

in which the matrix is made up of N vectors. After estimating the matrix $\frac{\partial \mathbf{z}_k^{l+1}}{\partial \mathbf{a}_i^l}$, Eq. 22 can be rewritten as follows:

$$\mathbf{d}_i^l = \sum_{k=1}^{R_{l+1}} \mathbf{d}_k^{l+1} \left[\mathbf{w}_{ki}^{l+1}\right]_\bullet \dot{\phi}(\mathbf{z}_i^l), \tag{28}$$

which becomes a vector-matrix-vector multiplication. It is evident that local gradient vector $\mathbf{d}_i^l$ in layer l is backpropagated from $\mathbf{d}_k^{l+1}$ in layer $l+1$. From this we see that the local gradient can be inferred from the output layer. In the output layer L, the local gradient vector $\mathbf{d}_g^L$ is defined as:

$$\begin{aligned} \frac{\partial C}{\partial \mathbf{z}_g^L} &= \left[\frac{\partial C}{\partial z_{g1}^L}, \frac{\partial C}{\partial z_{g2}^L}, \ \ldots, \frac{\partial C}{\partial z_{gN}^L}\right] \\ &= \left[\frac{\partial C}{\partial y_{g1}^L} \frac{\partial y_{g1}^L}{\partial z_{g1}^L}, \frac{\partial C}{\partial y_{g2}^L} \frac{\partial y_{g2}^L}{\partial z_{g2}^L}, \ \ldots, \frac{\partial C}{\partial y_{gN}^L} \frac{\partial y_{gN}^L}{\partial z_{gN}^L}\right] \\ &= \left[\frac{\partial C}{\partial y_{g1}^L} \dot{\phi}(z_{g1}^L), \frac{\partial C}{\partial y_{g2}^L} \dot{\phi}(z_{g2}^L), \ \ldots, \frac{\partial C}{\partial y_{gN}^L} \dot{\phi}(z_{gN}^L)\right], \end{aligned} \tag{29}$$

where $\frac{\partial C}{\partial y_{gn}^L}$ $(1 \le n \le N)$ is the derivative of the loss function used in the feedforward process, and $\dot{\phi}(z_{gn}^L)$ is the derivative of the activation function. The above process is summarized in Algorithm 1.

Algorithm 1 Arbitrary Bilinear Product Backpropagation

Input: Training inputs $\{\mathbf{X}_m\}_{m=1}^M$
Training targets $\{\mathbf{Y}_m\}_{m=1}^M$
Bilinear product $\bullet$
Activation function ϕ
Output: Parameters $\mathbf{\Theta} = \left\{\boldsymbol{\mathcal{W}}^l, \boldsymbol{\mathcal{B}}^l\right\}_{l=1}^L$

```
while not converged do
    for each minibatch ∈ {X}_{m=1}^M do
        Compute {Z^l, A^l}_{l=1}^L with Eqs. 6–8 (feedforward)
        Compute {D^l}_{l=1}^L with Eqs. 28 and 29 (backprop)
        Update {W^l}_{l=1}^L with the gradients from Eq. 21
        Update {B^l}_{l=1}^L with the gradients from Eq. 16
    end for
end while
```

D. Example: Circular Convolution as The Bilinear Product

In the above, we generalized the vector NNs using arbitrary bilinear products. This algorithm can be realized to train model parameters for scalar neurons or vector neurons from training data based on matrices or tensors. In this subsection, we derive $\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger$ and $\left[\mathbf{w}_{ki}^{l+1}\right]_\bullet$ for circular convolution. The derivation is described between layer $l-1$ and layer l.

Suppose $\mathbf{w}_{ij}^l \bullet \mathbf{a}_{ij}^{l-1}$ stands for the usual circular convolution, $\mathbf{w}_{ij}^l = \left[w_{ij1}^l, w_{ij2}^l, \ldots, w_{ijN}^l\right]^T \in \mathbb{R}^N$ and $\mathbf{a}_j^{l-1} = \left[a_{j1}^{l-1}, a_{j2}^{l-1}, \ldots, a_{jN}^{l-1}\right]^T \in \mathbb{R}^N$ are both N-dimensional vectors, and the matrix representation of $\mathbf{w}_{ij}^l \bullet \mathbf{a}_{ij}^{l-1}$ is:

$$\left[\mathbf{w}_{ij}^l\right]_\bullet = \begin{bmatrix} w_{ij1}^l & w_{ijN}^l & w_{ij(N-1)}^l & \cdots & w_{ij2}^l \\ w_{ij2}^l & w_{ij1}^l & w_{ijN}^l & \cdots & w_{ij3}^l \\ w_{ij3}^l & w_{ij2}^l & w_{ij1}^l & \cdots & w_{ij4}^l \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{ijN}^l & w_{ij(N-1)}^l & w_{ij(N-2)}^l & \cdots & w_{ij1}^l \end{bmatrix}, \tag{30}$$

where the weight vector is formulated as an $N \times N$ square matrix with w_{ij1}^l on the main diagonal. The matrix $\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger$

for Eq. 21 is extended as follows:

$$\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger = \begin{bmatrix} a_{j1}^{l-1} & a_{jN}^{l-1} & a_{j(N-1)}^{l-1} & \cdots & a_{j2}^{l-1} \\ a_{j2}^{l-1} & a_{j1}^{l-1} & a_{jN}^{l-1} & \cdots & a_{j3}^{l-1} \\ a_{j3}^{l-1} & a_{j2}^{l-1} & a_{j1}^{l-1} & \cdots & a_{j4}^{l-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{jN}^{l-1} & a_{j(N-1)}^{l-1} & a_{j(N-2)}^{l-1} & \cdots & a_{j1}^{l-1} \end{bmatrix}, \quad (31)$$

where the permutation of elements in the matrix $\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger$ is identical to the matrix $\left[\mathbf{w}_{ij}^{l}\right]_\bullet$. The matrix $\left[\mathbf{w}_{ki}^{l+1}\right]_\bullet$ for circular convolution is extended as:

$$\left[\mathbf{w}_{ki}^{l+1}\right]_\bullet = \begin{bmatrix} w_{ki1}^{l+1} & w_{kiN}^{l+1} & w_{ki(N-1)}^{l+1} & \cdots & w_{ki2}^{l+1} \\ w_{ki2}^{l+1} & w_{ki1}^{l+1} & w_{kiN}^{l+1} & \cdots & w_{ki3}^{l+1} \\ w_{ki3}^{l+1} & w_{ki2}^{l+1} & w_{ki1}^{l+1} & \cdots & w_{ki4}^{l+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{kiN}^{l+1} & w_{ki(N-1)}^{l+1} & w_{ki(N-2)}^{l+1} & \cdots & w_{ki1}^{l+1} \end{bmatrix}. \quad (32)$$

Note that when N equals two, this architecture becomes the hyperbolic NN [13]. When N is larger than two, this product is also known as polar complex multiplication [40], [41]. Detail investigations of more bilinear products are described in Appendix A.

E. Complexity Analysis

We compare below the computational cost of ABIPNN and DNN by estimating both the number of parameters and the number of multiplication operations. ABIPNN preserves the three-way tensor structure all the way from the inputs to the the outputs, while DNN uses the two-way matrix structure. Let $K = \sum_{l=2}^{L} R_l R_{l-1}$. The total number of parameters of an ABIPNN model is determined by:

$$\begin{aligned} &N\left(R_2R_1\right) + N\left(R_3R_2\right) + \cdots + N\left(R_LR_{L-1}\right) \\ &= N\sum_{l=2}^{L} R_lR_{l-1} = NK. \end{aligned} \quad (33)$$

In contrast, to obtain the same internal dimensionality, the number of parameters needed by the corresponding DNN is:

$$\begin{aligned} &\left(NR_2\right)\left(NR_1\right) + \left(NR_3\right)\left(NR_2\right) + \cdots + \left(NR_L\right)\left(NR_{L-1}\right) \\ &= N^2\sum_{l=2}^{L} R_lR_{l-1} = N^2K. \end{aligned} \quad (34)$$

Obviously, the total number of parameters of DNN increases much faster than that of ABIPNN.

We next compare the time complexity in terms of the number of multiplication operations. To simplify the expression, the complexity analysis is based on processing M training inputs. The multiplications come from three processes: the feedforward (T_f), the backpropagation of local gradients (T_b), and the derivations of the weight matrix (T_w). For ABIPNN, the three terms are computed by applying Eqs. 6, 28 and 21 respectively, and they are:

$$T_f = \sum_{l=2}^{L} \underbrace{\left(R_{l-1}N^2\right)}_{Eq.\ 6} R_lM = \mathcal{O}\left(N^2KM\right), \quad (35)$$

$$T_b = \sum_{l=2}^{L} \underbrace{R_{l+1}\left(N^2+N^2\right)}_{Eq.\ 28} R_lM = \mathcal{O}\left(N^2KM\right), \quad (36)$$

$$T_w = \sum_{l=2}^{L} \underbrace{\left(N^2\right)}_{Eq.\ 21} R_lR_{l-1}M = \mathcal{O}\left(N^2KM\right). \quad (37)$$

In contrast, to get identical internal dimensionality, for DNN we have $T_f = \sum_{l=2}^{L}(NR_l)(NR_{l-1})M = \mathcal{O}(N^2KM)$, $T_b = \sum_{l=2}^{L}(NR_l)(NR_{l-1})M = \mathcal{O}(N^2KM)$, and $T_w = \sum_{l=2}^{L}(NR_l)M(NR_{l-1}) = \mathcal{O}(N^2KM)$. From the aforementioned discussion, we see that ABIPNN and DNN are comparable in terms of time complexity. This will be empirically demonstrated in the next section.

III. Experiments

To validate the effectiveness of ABIPNN, we consider two regression problems that require learning a nonlinear mapping between tensor inputs and tensor outputs. The first is *multispectral image denoising*, which aims to recover the original multispectral image from a noisy input, with N set to 10. The second is *blind singing voice separation*, which aims to separate the singing voice and the accompaniment from a monaural audio mixture, with N set to 1, 3, 5, 7.

We intend to empirically compare the performance of the conventional DNNs with ABIPNN. In the first experiment, we will investigate their performance using a similar number of parameters. In the second one, the complexity analysis in Section II will also be verified. For both DNNs and ABIPNN, we use the mean-square error (MSE) as the objective function, and Adam [42] for gradient optimizations. For ABIPNN, we employ circular convolution as our bilinear product. The experiments are performed in Python on a personal computer equipped with an NVIDIA GeForce GTX 1080 Ti GPU and a memory of 64 GB RAM. For reproducible research, we will make the code of the experiments publicly available at https://github.com/zcfan-tw/vectorNNtoolbox.

A. Experiment on Multispectral Image Denoising

Multispectral (a.k.a. hyperspectral) imaging systems are usually employed to solve broadband color problems. A multispectral image is composed of a collection of monospectral (or monochrome) images, each of which is captured with a specific wavelength. These monospectral images can be considered as different *bands* of the multispectral image. As different spectral bands may exhibit some mutual associations, we can leverage such associations to enhance the accuracy of image processing applications.

In multispectral image denoising, we are given a noisy version of a multispectral image with N bands, and are asked

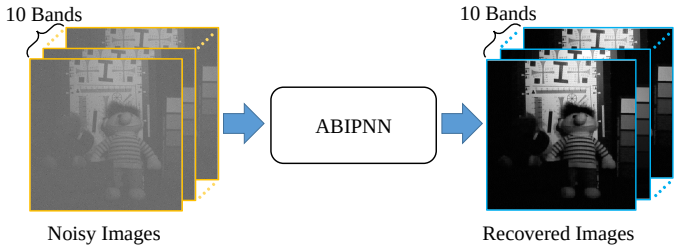


Fig. 3: Illustration of multispectral image denoising using ABIPNN. Noisy images are contaminated by Gaussian noise. We take the last 10 bands for the experiments.

to recover the clean version (also N bands). In a recent work presented by Zhang *et al.* [23], different supervised multidimensional dictionary learning methods were evaluated on the Columbia multispectral image database [6] for the denoising task. By following their settings, we can compare the performance of our models with these prior arts. These methods include K-TSVD, K-SVD [43], 3D K-SVD [44], LRTA [45], DNMDL [46] and PARAFAC [47].

The database [6] contains 32 real-world scenes and each scene contains 31 monochrome images of size 512×512, captured by varying the wavelength of a camera from 400 nm to 700 nm with a step size of 10 nm. Following Zhang *et al.* [23], we consider images in the "chart and stuffed toy" scene, resize each image to 205×205 and take images of the last 10 bands (i.e., starting from 600 nm), making $N = 10$. Moreover, we divide each image into $8 \times 8 \times 10$ overlapping tensor patches with a hop size of one. We randomly take 10 000 tensor patches as the training data and the rest for testing. From the training data, 1000 tensor patches are held out as the validation data. The maximal number of training epochs is set to 3000. We also stop the training process if the validation MSE does not decrease for 100 epochs.

For the noise model, we randomly select a certain number of pixels from each band of an image and add to the pixels Gaussian noise with specific *sigma value*. We refer to the ratio of pixels corrupted per band as the *sparsity* level of the noise. In our experiments, we vary the sigma value from 100 to 200, and the sparsity level from 5% to 15%, to simulate different degrees of corruption. The goal is to recover the corrupted images, as illustrated in Fig. 3. As for the objective function in model training, we compute the MSE between the recovered and the original versions of the patches in the training set, across all the 10 bands. As in Zhang *et al.* [23], we measure the performance of denoising in terms of the peak signal-to-noise ratio (PSNR). We calculate the PSNR for each test patch and report the average result.

In our implementation, the ABIPNN consists of 3 hidden layers with 512 neurons in each layer (i.e., $R_2 = \cdots = R_{L-1} = 512$ in Fig. 1). Both the input and output layers have $8 \times 8 = 64$ dimensions (i.e., $R_1 = R_L = 64$ in Fig. 1)and the topology is denoted as 64-512-512-512-64. In ABIPNN, each neuron represents a vector with size $N = 10$. Hence, for each patch the 10 bands are processed at the same time. The activation function is sigmoid with a learning rate of 5×10^{-4}. To compare the performance of ABIPNN with conventional DNNs using a similar number of parameters, we consider the following two variants of DNNs as the baselines. The first variant, *DNN-concat*, simply concatenates the 10 bands as a single vector to process them jointly, making $R_1 = R_L = 640$. There are 3 hidden layers with $1,450$ neurons in each layer, so the topology is 640-1450-1450-1450-640, where each neuron represents a scalar. The second variant, *DNN-parallel*, processes the 10 bands separately using 10 DNNs, each with a 64-512-512-512-64 topology. In other words, each DNN is trained for denoising a specific band.

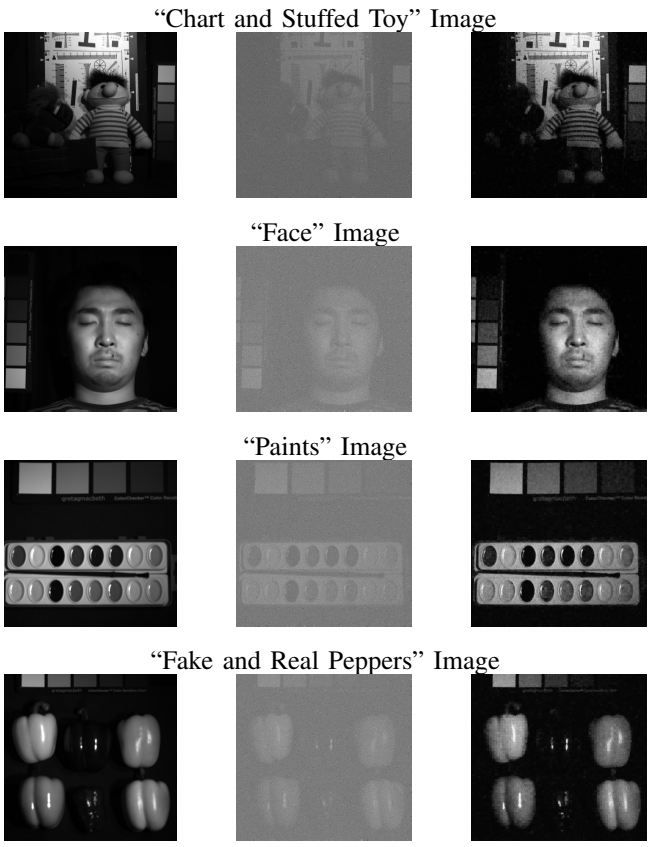
TABLE II: PSNR (IN DB) OBTAINED BY DIFFERENT METHODS FOR MULTISPECTRAL IMAGE DENOISING, UNDER DIFFERENT SPARSITY AND SIGMA VALUES

Sparsity	5%	10%	15%	10%	10%
Sigma	100	100	100	150	200
Referenced from [23]					
Noisy Image	20.96	18.18	16.35	14.75	12.10
K-SVD [43]	22.73	22.60	22.49	22.38	22.20
3DK-SVD [44]	22.61	22.53	22.47	22.41	22.20
LRTA [45]	23.54	26.84	26.65	23.90	22.03
DNMDL [46]	24.07	23.73	25.16	17.89	16.83
PARAFAC [47]	27.07	26.86	26.72	26.13	25.24
K-TSVD [23]	27.19	26.98	26.79	26.18	25.44
Our Experiments					
Noisy Image	20.92	18.16	16.35	14.64	12.10
DNN-concat	25.06	24.80	24.93	24.59	24.03
DNN-parallel	30.18	28.88	28.06	27.17	25.88
ABIPNN	33.92	32.47	31.74	31.01	29.55

Table II shows the experimental results. In the upper part of the table, we cite the PSNR values reported in [23], and in the lower part we report the results of our own implementation.[1] Because we follow their experimental settings, the PSNR values for the noisy images (i.e., before denoising) reported in [23] are close to what we observe in our implementation. Besides, Table II also shows that ABIPNN outperforms all the other methods, including the two DNN baselines, by a large margin across different values of sigma and sparsity. The PSNRs are improved from 12.10–20.92 dB to 29.55–33.92 dB. This result suggests the effectiveness of a neural network for this task. Fig. 4 demonstrates the original, noisy, and denoised versions of three images by using ABPINN.

Among the three NN-based methods, DNN-concat performs

[1]Each result we reported is the average of 10 simulations and the variance of each result is low. For example, when the sparsity is 5% and the sigma is 100, the variance of DNN-concat, DNN-paralled, and ABIPNN is 0.003, 0.014, and 0.011, respectively.



(a) Original (b) Noisy (c) Recovered

Fig. 4: Denoised images at the 700 nm band using the proposed ABIPNN method. The sparsity of the noisy pixels is 10% and the sigma value of the additive Gaussian noise is set to 200.

the worst, and is even inferior to that of PARAFAC [47] and K-TSVD [23], two non-deep learning based methods. This suggests that concatenating inputs from different bands does not make it easy for an NN to learn the association between bands. In contrast, using multidimensional vector neurons, ABIPNN learns the relations between bands by computing the circular convolution of two vectors: one coming from a hidden node and the other coming from a weight tensor. The vector coming from a weight tensor can be regarded as a linear kernel that captures interactions across bands, which may contribute to enhanced results in denoising.

Fig. 5(a) displays the changes in MSE values as a function of the training epochs in the training procedure for ABIPNN and DNN-concat. We find that the MSE values converge to a certain value for both methods, but ABIPNN converges much faster. We conjecture that this is because the error propagation in ABIPNN is not only optimized for each dimension (i.e., band) but also between different dimensions. It is also known that gradient variance reduction helps achieve better convergence in stochastic gradient descent (SGD) [48], [49]. Fig. 5(b) shows the changes in gradient variance during SGD training. Here ABIPNN provides lower variance at convergence (after 2000 epochs).

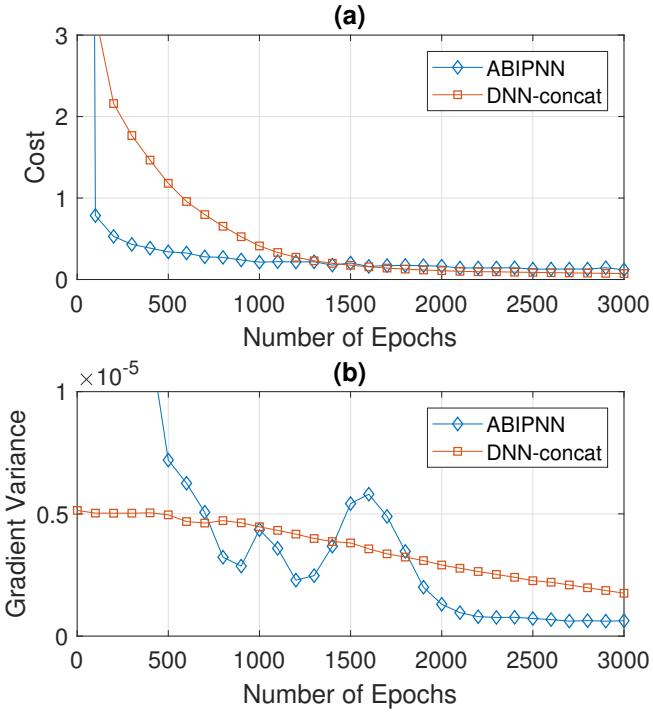


Fig. 5: (a) Mean square error with Adam and (b) gradient variance with SGD in the training procedure of ABIPNN and DNN-concat, for the case when the sparsity of the noisy pixels is 5% and the sigma value is set to 100.

B. Experiment on Blind Singing Voice Separation

Separating the singing voice and the accompaniment from a monaural audio mixture is challenging, because there are more unknowns than equations. Unsupervised methods such as robust principal component analysis [50]–[52] assume no labeled data are available and rely on assumptions on the characteristics of the sources for separation. For example, the spectrogram of the accompaniment part is assumed to have lower rank than that of the vocal part. If we are given the original sources of some audio mixtures, we can take these clean sources as the supervisory signal and employ supervised methods such as non-negative matrix factorization (NMF) [53] for source separation. Naturally, supervised methods usually outperform unsupervised methods, as the former can learn from the pairs of mixtures and sources. Recently, NN-based methods have been introduced to this task [54], [55], showing better result than non-NN based methods such as NMF. This can be done by taking the spectrogram of an audio mixture as the input and requiring the network to reproduce the spectrograms of the corresponding two sources at the output. NNs work better because they can learn a nonlinear mapping between the input and the outputs.

A spectrogram is a 2-D time-frequency representation. It is computed by the short time Fourier transform (STFT), which divides a given time signal into short segments of equal length and then computes the Fourier transform separately on each short segment. We call each short segment a ‘frame,’ and the

result of Fourier transform per frame as a 'spectrum.' The spectrogram considers only the magnitude part of STFT.

A naïve DNN approach for source separation, referred to as *DNN-simple* below, takes the spectrum of each frame of the mixture as input, and estimates the spectra of that frame for the vocal and the accompaniment parts respectively. This is done frame-by-frame, finally leading to the estimated (recovered) spectrograms $\tilde{\mathbf{Y}}_\mathbf{1}$ and $\tilde{\mathbf{Y}}_\mathbf{2}$ of the two sources. Then, we use the Weiner filter to compute the following soft time-frequency mask to smooth the source separation results.

$$\mathbf{m}(f) = \frac{|\tilde{\mathbf{Y}}_\mathbf{1}(f)|}{|\tilde{\mathbf{Y}}_\mathbf{1}(f)| + |\tilde{\mathbf{Y}}_\mathbf{2}(f)|}, \tag{38}$$

where $f = 1, 2, ..., F$ denotes different frequency bins. The estimated spectra $\tilde{\mathbf{s}}_\mathbf{1}$ and $\tilde{\mathbf{s}}_\mathbf{2}$, respectively corresponding to vocals and accompaniments, are produced by:

$$\begin{aligned} \tilde{\mathbf{s}}_\mathbf{1}(f) &= \mathbf{m}(f)\mathbf{z}(f), \\ \tilde{\mathbf{s}}_\mathbf{2}(f) &= (1 - \mathbf{m}(f))\mathbf{z}(f), \end{aligned} \tag{39}$$

where $\mathbf{z}(f)$ is the magnitude spectra of the input mixture. The estimated spectra $\tilde{\mathbf{s}}_\mathbf{1}$ and $\tilde{\mathbf{s}}_\mathbf{2}$ are finally transformed back to the time-domain by the inverse short time Fourier transform (ISTFT), assuming that the mixture and the two sources share the same phase. Parameters of the NN are learned by using the MSE between the estimated spectra and the groundtruth source spectra.

We can improve the performance of DNN-simple by adding the *temporal context* of each frame to the input [56]. Specifically, in addition to the current frame, we add the spectra of the previous-f and subsequent-f frames to compose a real-valued matrix. For a conventional NN, we can take the vectorized version of the matrix (which amounts to concatenating the spectra of these $2f + 1$ frames) as the input. We refer to this method as *DNN-concat*. Alternatively, we can view the $2f+1$ frames as different dimensions and use a vector NN to model the interaction between different frames. Please see Fig. 6 for an illustration.

Because ABIPNN can deal with input of arbitrary dimensions, in principle f can take any values. In this experiment, we compare the performance of ABIPNN with the classic vector product neural network (VPNN) [17] (i.e., where $f = 1$ and only two neighboring frames are considered), and the conventional DNNs (i.e., where $f = 1$–3). The major difference between ABIPNN (with $N = 3$) and VPNN is that the former uses circular convolutions.

In our experiments, we use the Demixing Secret Database (DSD100), which was used in the Signal Separation Evaluation Campaign (SiSEC) in 2016 [57], [58]. It is made up of 100 full-track professionally-produced music recordings of different styles. It can be used for training and testing for source separation algorithms, because it includes both the stereophonic mixtures and the original stereo sources. The database is divided into a development set and a test set by the organizers of SiSEC 2016, each consisting of 50 songs. The duration of the songs ranges from 2'22" to 7'20", and the average duration is 4'10". All the songs are sampled at 44 100 Hz. To reduce computational cost, all songs are downsampled

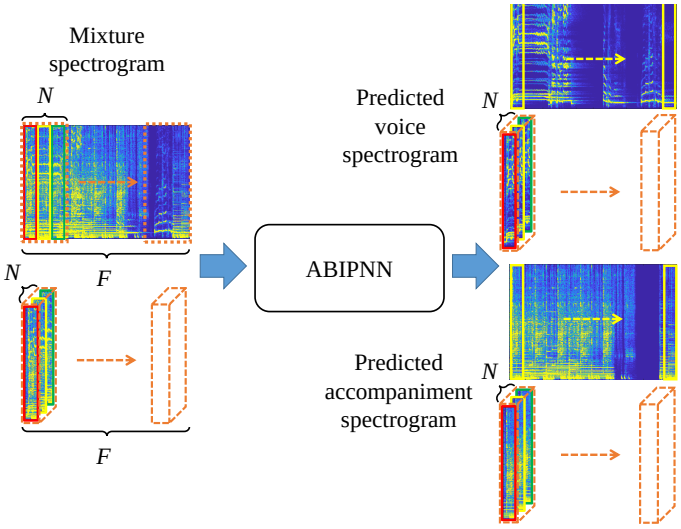


Fig. 6: Illustration of singing voice separation using ABIPNN (best seen in color), where F represents the number of frames and N the number of temporal context frames. Here, $N = 3$. We use red to indicate the previous frames, yellow for the current frames, and green for the subsequent frames. After training, we extract the second dimension (i.e., the yellow one) for soft-time frequency masking.

to 8000 Hz. For each song, we compute STFT with a 1024-point window and a 512-point hop size. We set the activation function to the rectified linear unit (ReLU) and the learning rate to 3×10^{-5} with exponentially decay reduced by 10% every 300 epochs.

Then we map each t-f unit of the magnitude spectrum to an N-dimensional vector to serve as the input to ABIPNN. Among the training frames, 5% of the tensor frames are randomly picked as the validation data. The training epoch is set to 1000 and the training process is stopped if the MSE of the validation set does not decrease for 20 epochs. The performance is measured in terms of source to distortion ratio (SDR), source to interferences ratio (SIR), and source to artifact ratio (SAR), as calculated by the Blind Source Separation (BSS) Eval Toolbox v3.0 [59].

Table III shows some details of the evaluated methods and their results. Each model has three hidden layers. In order to obtain the same internal dimensionality corresponding to real-valued neurons of DNN-concat and vector-valued neurons of ABIPNN, we use N times more neurons per layer for DNN-concat than ABIPNN.

In Table III, the two vector NNs (i.e., VPNN and ABIPNN) indeed outperform the conventional DNN methods (i.e., DNN-simple and DNN-concat), demonstrating the effectiveness of considering the interactions between frames. DNN-concat$_1$, VPNN and ABIPNN all outperform DNN-simple by a great margin in SDR. And, in terms of SAR, we see VPNN and ABIPNN perform much better than DNN-concat$_1$, despite that DNN-concat$_1$ has more parameters. This shows that vector neurons can take better advantage of the information provided by the temporal context.

Furthermore, we also compare the computation times of

TABLE III: COMPARISON OF SDR, SIR, SAR VALUES AND COMPUTATION TIME (SECONDS PER EPOCH) OBTAINED BY DIFFERENT METHODS FOR SINGING VOICE SEPARATION OVER THE DSD100 DATA SET, UNDER DIFFERENT VALUES OF N AND NUMBER OF NEURONS PER LAYER.

Model	N	Neurons per layer	Number of params	Time (second per epoch)	SDR		SIR		SAR	
					Vocal	Accomp.	Vocal	Accomp.	Vocal	Accomp.
DNN-simple	1	513	1.31M	4.2	4.37	9.98	8.38	12.89	5.81	13.89
DNN-concat$_1$	3	1539	8.68M	7.1	4.63	10.25	8.29	13.34	6.17	14.14
VPNN [24]	3	513	3.95M	9.1	4.64	10.17	8.37	12.97	6.59	14.57
ABIPNN$_1$	3	513	3.95M	9.2	4.67	10.08	8.39	12.76	6.67	14.24
DNN-concat$_2$	5	2565	22.36M	14.1	4.68	10.19	8.86	12.53	6.78	15.07
ABIPNN$_2$	5	513	6.59M	17.5	4.97	10.50	9.45	13.63	6.88	14.78
DNN-concat$_3$	7	3591	42.37M	23.3	4.73	10.30	**10.67**	12.45	6.28	**15.19**
ABIPNN$_3$	7	513	9.22M	30.6	**5.13**	**10.70**	9.41	**13.93**	**7.20**	14.87

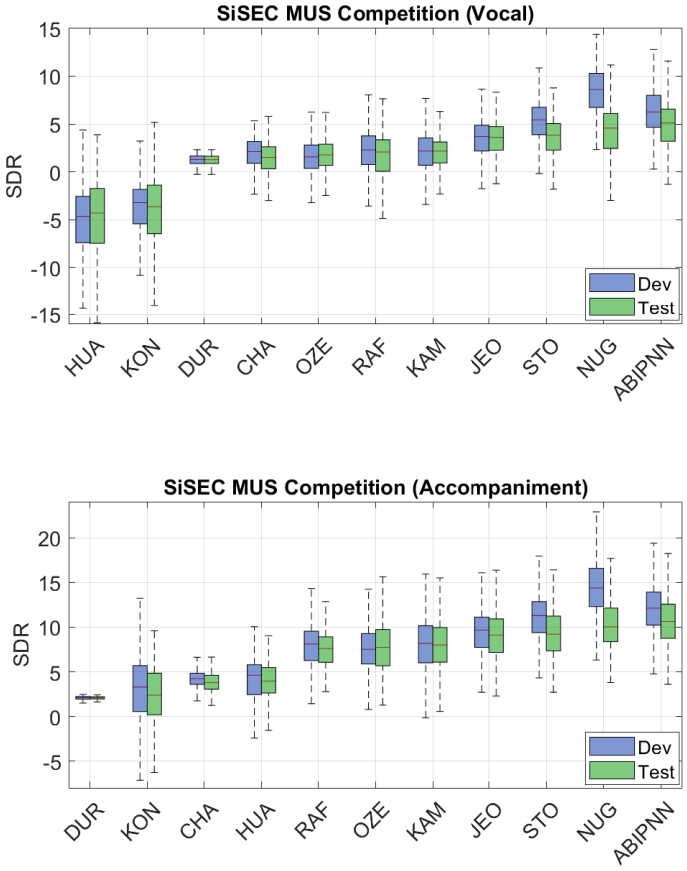


Fig. 7: Vocal and accompaniment results (in SDR) for the development part and test part of the DSD100 dataset. The methods are sorted by the median SDR for the test part. For the result of ABIPNN, the value of N is set to 7. Please note that we only consider methods that did not use any data augmentation here, for fair comparison.

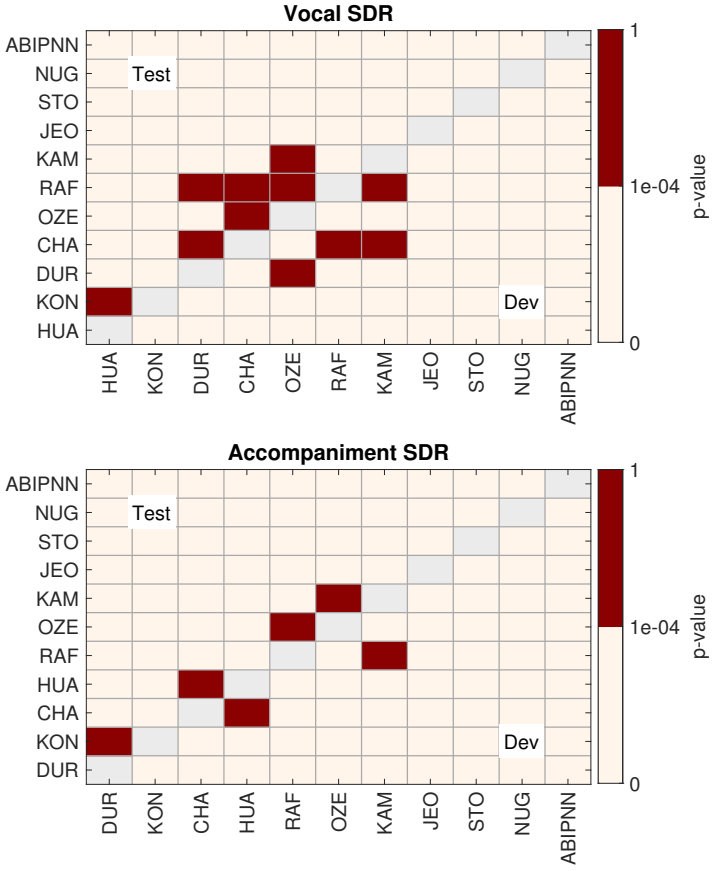


Fig. 8: Wilcoxon signed-rank test results for SDR vocals and accompaniments. The upper triangle represents the comparison of the test set and the lower triangle is for the development part. For the result of ABIPNN, the value of N is set to 7. Values of $p-$value $> 1e-04$ indicate no significant difference between the two group results.

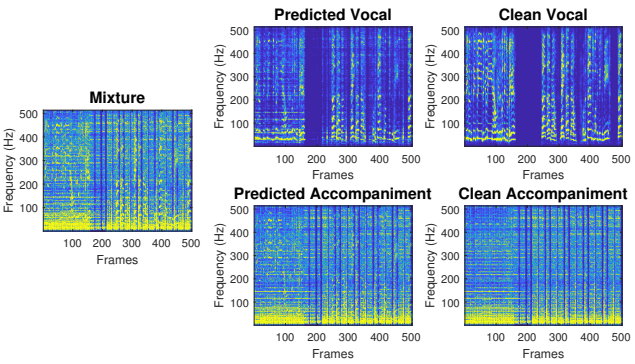
DNN-concat, VPNN and ABIPNN with an NVIDIA 1080 Ti GPU. It is obvious that the computation time of ABIPNN is comparable with DNN-concat given the same value of N. While we can only use $N = 3$ for VPNN, we can use larger N for ABIPNN. From Table III, we see that the performance difference between ABIPNN and DNN-concat increases as N goes larger, despite that the latter architecture has more total parameters. With $N = 7$, the SDR obtained by ABIPNN reaches 5.13 dB for the vocal part, which outperforms DNN-simple by 0.76 dB. As can be seen later from Fig. 7, such a performance gap is remarkable.

In Fig. 7, we compare the median SDR of the vocal and accompaniment parts of ABIPNN with the methods that have been evaluated for SiSEC 2016 [58], including NUG [60], STO [61], OZE [62], KON [63], KAM [64], JEO [65], HUA [50], DUR [66], CHA [67], and RAF [68]. Among them, CHA is based on CNN, and KON, STO, NUG are based on DNNs. For fair comparison, we only select those submissions that are not trained with augmentation data. Moreover, we show the vocal and accompaniment SDR here instead of other metrics, for saving space (following [58]) and because SDR is usually considered to be more important than the other metrics. It is also a convention in SiSEC to show the result for both the development and the test sets. From Fig. 7, we see that ABIPNN outperforms all the other methods. From the report [58], our results of ABIPNN are comparable to UHL [69], which applies data augmentation and a complicated NN architecture.

In order to show the effect of different methods on SDR values, we follow [70] and apply the Wilcoxon signed-rank test (two-tailed and Bonferroni corrected) for pairwise comparisons. From Fig. 8, we can see that the group of results from ABIPNN has significant differences over other methods.

Finally, Fig. 9 shows the spectrograms of the input mixture, the separation results by ABIPNN, and the original sources, for two songs randomly picked from the test set of DSD100. We see that the separation results (marked as "predicted vocal" or "predicted accompaniment") resemble the original sources.

(a) 009 - Bobby Nobody - Stitch Up



(b) 039 - Swinging Steaks - Lost My Way

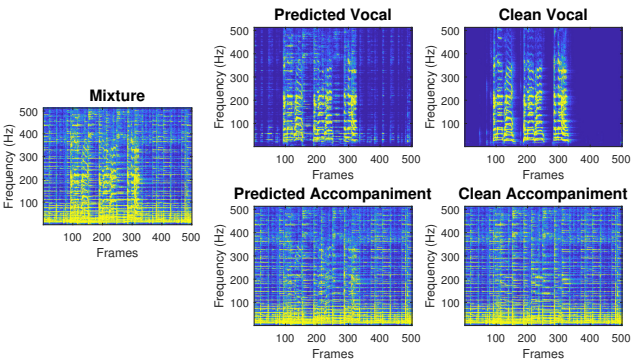


Fig. 9: Examples of singing voice separation employing ABIPNN on the test part of the DSD100 dataset.

IV. Conclusion

This paper proposes a novel vector-valued neuron which employs arbitrary bilinear products for feedforward and backpropagation. The proposed architecture generalizes and extends all existing vector-valued neurons and is useful for datasets where each training sample is a multidimensional vector. Through bilinear products, the vector neural network captures the associations among different entries in the same position in each vector. The model can be trained efficiently by using vector error backpropagation through the Adam algorithm. Experimental results on multispectral denoising and singing voice separation show that our proposed model performs better than conventional NNs. Future work involves three directions. First, we will apply the technique of vector neural learning with bilinear product on convolutional neural network (CNN) to deal with spatial data such as images. Next, we will compare the performance of vector neural learning on CNN to other deep models with classification or regression tasks. Finally, we observe that the training time of the proposed network grows quadratically as a function of the dimensionality N. Future work has to be done to make the training more scalable to higher values of N.

Appendix A
Bilinear Products for ABIPNN

In Section II-D, we use circular convolution as the bilinear product in ABIPNN and briefly derive the results of matrices $\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger$ and $\left[\mathbf{w}_{ki}^{l+1}\right]_\bullet$ used in the backpropagation process. In this section, we will derive some more bilinear products in detail. The feedforward connection is still described between layer $l-1$ and layer l.

A. Vector Product

Here $\mathbf{w}_{ij}^l \bullet \mathbf{a}_j^{l-1}$ stands for the usual vector product, $\mathbf{w}_{ij}^l = \left[w_{ij1}^l, w_{ij2}^l, w_{ij3}^l\right]^T \in \mathbb{R}^3$, and the input $\mathbf{a}_j^{l-1} = \left[a_{j1}^{l-1}, a_{j2}^{l-1}, a_{j3}^{l-1}\right]^T \in \mathbb{R}^3$ are all three-dimensional vectors $(N = 3)$. The matrix representation of $\mathbf{w}_{ij}^l \bullet \mathbf{a}_j^{l-1}$ is:

$$\left[\mathbf{w}_{ij}^l\right]_\bullet = \begin{bmatrix} 0 & -w_{ij3}^l & w_{ij2}^l \\ w_{ij3}^l & 0 & -w_{ij1}^l \\ -w_{ij2}^l & w_{ij1}^l & 0 \end{bmatrix}, \tag{40}$$

where the weight vector is formulated as a 3×3 square matrix with zeros on the main diagonal. When calculating $\frac{\partial C}{\partial \mathbf{w}_{ij}^l}$, the matrix $\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger$ in Eq. 21 can be extended as follows:

$$\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger = \begin{bmatrix} 0 & a_{j3}^{l-1} & -a_{j2}^{l-1} \\ -a_{j3}^{l-1} & 0 & a_{j1}^{l-1} \\ a_{j2}^{l-1} & -a_{j1}^{l-1} & 0 \end{bmatrix}. \tag{41}$$

ABIPNN with this product is identical to the three-dimensional vector product neural network [17].

B. Quaternion Multiplication

Quaternions are four-dimensional numbers $a+bi+cj+dk$ with the multiplication rule $i^2=j^2=k^2=ijk=-1$. Here $\mathbf{w}_{ij}^l \bullet \mathbf{a}_j^{l-1}$ stands for quaternion multiplication, the weight $\mathbf{w}_{ij}^l = \left[w_{ij1}^l, w_{ij2}^l, w_{ij3}^l, w_{ij4}^l\right]^T \in \mathbb{R}^4$ and the input $\mathbf{a}_j^{l-1} = \left[a_{j1}^{l-1}, a_{j2}^{l-1}, a_{j3}^{l-1}, a_{j4}^{l-1}\right]^T \in \mathbb{R}^4$ are all four-dimensional vectors ($N=4$). The matrix representation of $\mathbf{w}_{ij}^l \bullet \mathbf{a}_j^{l-1}$ is:

$$\left[\mathbf{w}_{ij}^l\right]_\bullet = \begin{bmatrix} w_{ij1}^l & -w_{ij2}^l & -w_{ij3}^l & -w_{ij4}^l \\ w_{ij2}^l & w_{ij1}^l & -w_{ij4}^l & w_{ij3}^l \\ w_{ij3}^l & w_{ij4}^l & w_{ij1}^l & -w_{ij2}^l \\ w_{ij4}^l & -w_{ij3}^l & w_{ij2}^l & w_{ij1}^l \end{bmatrix}, \tag{42}$$

where the weight vector is formulated as a 4×4 square matrix with w_{ij1}^l on the main diagonal. The matrix $\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger$ for the calculation of $\frac{\partial C}{\partial \mathbf{w}_{ij}^l}$ in Eq. 21 can be extended as follows:

$$\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger = \begin{bmatrix} a_{j1}^{l-1} & -a_{j2}^{l-1} & -a_{j3}^{l-1} & -a_{j4}^{l-1} \\ a_{j2}^{l-1} & a_{j1}^{l-1} & a_{j4}^{l-1} & -a_{j3}^{l-1} \\ a_{j3}^{l-1} & -a_{j4}^{l-1} & a_{j1}^{l-1} & a_{j2}^{l-1} \\ a_{j4}^{l-1} & a_{j3}^{l-1} & -a_{j2}^{l-1} & a_{j1}^{l-1} \end{bmatrix}. \tag{43}$$

With quaternion multiplication, the ABIPNN is equivalent to the quaternion-valued neural network [19].

C. Seven-Dimensional Vector Product

The seven-dimensional vector product is defined as per [71]. Here $\mathbf{w}_{ij}^l \bullet \mathbf{a}_j^{l-1}$ stands for the seven-dimensional vector product, $\mathbf{w}_{ij}^l = \left[w_{ij1}^l, w_{ij2}^l, \ldots, w_{ij7}^l\right]^T \in \mathbb{R}^7$ and $\mathbf{a}_j^{l-1} = \left[a_{j1}^{l-1}, a_{j2}^{l-1}, \ldots, a_{j7}^{l-1}\right]^T \in \mathbb{R}^7$ are both seven-dimensional vectors ($N=7$), and the matrix representation of $\mathbf{w}_{ij}^l \bullet \mathbf{a}_j^l$ is:

$$\left[\mathbf{w}_{ij}^l\right]_\bullet = \begin{bmatrix} 0 & -w_{ij4}^l & -w_{ij7}^l & w_{ij2}^l & -w_{ij6}^l & w_{ij5}^l & w_{ij3}^l \\ w_{ij4}^l & 0 & -w_{ij5}^l & -w_{ij1}^l & w_{ij3}^l & -w_{ij7}^l & w_{ij6}^l \\ w_{ij7}^l & w_{ij5}^l & 0 & -w_{ij6}^l & -w_{ij2}^l & w_{ij4}^l & -w_{ij1}^l \\ -w_{ij2}^l & w_{ij1}^l & w_{ij6}^l & 0 & -w_{ij7}^l & -w_{ij3}^l & w_{ij5}^l \\ w_{ij6}^l & -w_{ij3}^l & w_{ij2}^l & w_{ij7}^l & 0 & -w_{ij1}^l & -w_{ij4}^l \\ -w_{ij5}^l & w_{ij7}^l & -w_{ij4}^l & w_{ij3}^l & w_{ij1}^l & 0 & -w_{ij2}^l \\ -w_{ij3}^l & -w_{ij6}^l & w_{ij1}^l & -w_{ij5}^l & w_{ij4}^l & w_{ij2}^l & 0 \end{bmatrix}, \tag{44}$$

where the weight vector is formulated as a 7×7 square matrix in a similar way as the vector product one. The matrix $\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger$ in Eq. 21 becomes:

$$\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger = \begin{bmatrix} 0 & a_{j4}^{l-1} & a_{j7}^{l-1} & -a_{j2}^{l-1} & a_{j6}^{l-1} & -a_{j5}^{l-1} & -a_{j3}^{l-1} \\ -a_{j4}^{l-1} & 0 & a_{i5}^{l-1} & a_{j1}^{l-1} & -a_{j3}^{l-1} & a_{j7}^{l-1} & -a_{j6}^{l-1} \\ -a_{j7}^{l-1} & -a_{j5}^{l-1} & 0 & a_{j6}^{l-1} & a_{j2}^{l-1} & -a_{j4}^{l-1} & a_{j1}^{l-1} \\ a_{j2}^{l-1} & -a_{j1}^{l-1} & -a_{j6}^{l-1} & 0 & a_{j7}^{l-1} & a_{j3}^{l-1} & -a_{j5}^{l-1} \\ -a_{j6}^{l-1} & a_{j3}^{l-1} & -a_{j2}^{l-1} & -a_{j7}^{l-1} & 0 & a_{j1}^{l-1} & a_{j4}^{l-1} \\ a_{j5}^{l-1} & -a_{j7}^{l-1} & a_{j4}^{l-1} & -a_{j3}^{l-1} & -a_{j1}^{l-1} & 0 & a_{j2}^{l-1} \\ a_{j3}^{l-1} & a_{j6}^{l-1} & -a_{j1}^{l-1} & a_{j5}^{l-1} & -a_{j4}^{l-1} & -a_{j2}^{l-1} & 0 \end{bmatrix}, \tag{45}$$

where the signs are flipped when compared to the matrix $\left[\mathbf{w}_{ij}^l\right]_\bullet$ above.

D. Skew Circular Convolution

The skew-circular convolution is obtained by replacing the circulant matrix in Eq. 30 by a skew-circulant one [72]. The matrix representation of $\mathbf{w}_{ij}^l \bullet \mathbf{a}_j^{l-1}$ then becomes:

$$\left[\mathbf{w}_{ij}^l\right]_\bullet = \begin{bmatrix} w_{ij1}^l & -w_{ijN}^l & -w_{ij(N-1)}^l & \cdots & -w_{ij2}^l \\ w_{ij2}^l & w_{ij1}^l & -w_{ijN}^l & \cdots & -w_{ij3}^l \\ w_{ij3}^l & w_{ij2}^l & w_{ij1}^l & \cdots & -w_{ij4}^l \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{ijN}^l & w_{ij(N-1)}^l & w_{ij(N-2)}^l & \cdots & w_{ij1}^l \end{bmatrix}, \tag{46}$$

where the weight vector is formulated as an $N\times N$ square matrix with w_{ij1}^l on the main diagonal and the upper triangular part of the weight matrix is multiplied by minus one. When

we compute $\frac{\partial C}{\partial \mathbf{w}_{ij}^l}$ in Eq. 21, $\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger$ is extended as follows:

$$\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger = \begin{bmatrix} a_{j1}^{l-1} & -a_{jN}^{l-1} & -a_{j(N-1)}^{l-1} & \dots & -a_{j2}^{l-1} \\ a_{j2}^{l-1} & a_{j1}^{l-1} & -a_{jN}^{l-1} & \dots & -a_{j3}^{l-1} \\ a_{j3}^{l-1} & a_{j2}^{l-1} & a_{j1}^{l-1} & \dots & -a_{j4}^{l-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{jN}^{l-1} & a_{j(N-1)}^{l-1} & a_{j(N-2)}^{l-1} & \dots & a_{j1}^{l-1} \end{bmatrix}. \tag{47}$$

If $N = 2$, then ABIPNN becomes a complex-valued neural network [9]. For even $N \geq 2$, the skew circular convolution is also known as planar complex multiplication [40].

E. Reverse-Time Circular Convolution

The reversed-time circular convolution is obtained by flipping the circulant matrix upside down (see also [73]). The matrix representation of $\mathbf{w}_{ij}^l \bullet \mathbf{a}_j^{l-1}$ becomes:

$$\left[\mathbf{w}_{ij}^l\right]_\bullet = \begin{bmatrix} w_{ijN}^l & w_{ij(N-1)}^l & w_{ij(N-2)}^l & \dots & w_{ij1}^l \\ w_{ij(N-1)}^l & w_{ij(N-2)}^l & w_{ij(N-3)}^l & \dots & w_{ijN}^l \\ w_{ij(N-2)}^l & w_{ij(N-3)}^l & w_{ij(N-4)}^l & \dots & w_{ij(N-1)}^l \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{ij1}^l & w_{ijN}^l & w_{ij(N-1)}^l & \dots & w_{ij2}^l \end{bmatrix}, \tag{48}$$

where the weight vector is formulated as an $N \times N$ square matrix which is the same as the weight matrix for circular convolution rotated by 90 degrees. The matrix $\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger$ in Eq. 21 is extended as follows:

$$\left[\mathbf{a}_j^{l-1}\right]_\bullet^\dagger = \begin{bmatrix} a_{jN}^{l-1} & a_{j(N-1)}^{l-1} & a_{j(N-2)}^{l-1} & \dots & a_{j1}^{l-1} \\ a_{j(N-1)}^{l-1} & a_{j(N-2)}^{l-1} & a_{j(N-3)}^{l-1} & \dots & a_{jN}^{l-1} \\ a_{j(N-2)}^{l-1} & a_{2(N-3)}^{l-1} & a_{j(N-4)}^{l-1} & \dots & a_{j(N-1)}^{l-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{j1}^{l-1} & a_{jN}^{l-1} & a_{j(N-1)}^{l-1} & \dots & a_{j2}^{l-1} \end{bmatrix}. \tag{49}$$

REFERENCES

[1] D. P. Mandic and V. S. L. Goh, *Complex Valued Nonlinear Adaptive Filters: Noncircularity, Widely Linear and Neural Models*. New York, NY: Wiley, 2009.

[2] T. Nitta, *Complex-Valued Neural Networks: Utilizing High-Dimensional Parameters*. Hershey, PA: Information Science Reference, 2009.

[3] A. Hirose, *Complex-Valued Neural Networks*. Berlin: Springer, 2013.

[4] ——, *Complex-Valued Neural Networks: Advances and Applications*. New York: John Wiley and Sons, 2013.

[5] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J.-F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal, "Deep complex networks," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2018.

[6] F. Yasuma, T. Mitsunaga, D. Iso, and S. K. Nayar, "Generalized assorted pixel camera: postcapture control of resolution, dynamic range, and spectrum," *IEEE Trans. Image Process.*, vol. 19, no. 9, pp. 2241–2253, Mar. 2010, [Online]. Available: http://www1.cs.columbia.edu/CAVE/databases/multispectral/.

[7] S. Koelstra, C. Mühl, M. Soleymani, J.-S. Lee, A. Yazdani, T. Ebrahimi, T. Pun, A. Nijholt, and I. Patras, "DEAP: A database for emotion analysis; using physiological signals," *IEEE Trans. Affective Comput.*, vol. 3, no. 1, pp. 18–31, 2012.

[8] E. Vincent, S. Watanabe, J. Barker, and R. Marxer, "The third CHiME speech separation and recognition challenge: Dataset, task and baselines," in *Proc. IEEE Workshop Automat. Speech Recognition and Understanding (ASRU)*, 2015, pp. 504–511.

[9] T. Nitta, "An extension of the back-propagation algorithm to complex numbers," *Neural Netw.*, vol. 10, no. 8, pp. 1391–1415, 1997.

[10] ——, "An analysis of the fundamental structure of complex-valued neurons," *Neural Process. Lett.*, vol. 12, no. 3, pp. 239–246, 2000.

[11] ——, "Orthogonality of decision boundaries in complex-valued neural networks," *Neural Comput.*, vol. 16, no. 1, pp. 73–97, Jan. 2004.

[12] ——, "Solving the XOR problem and the detection of symmetry using a single complex-valued neuron," *Neural Netw.*, vol. 16, no. 8, pp. 1101–1105, 2003.

[13] S. Buchholz and G. Sommer, "A hyperbolic multilayer perceptron," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2000, pp. 129–133.

[14] T. Nitta and S. Buchholz, "On the decision boundaries of hyperbolic neurons," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2008, pp. 2974–2980.

[15] T. Nitta and Y. Kuroe, "Hyperbolic gradient operator and hyperbolic back-propagation learning algorithms," *IEEE Trans. Neural Netw. Learn. Syst.*, no. 99, pp. 1–14, Mar. 2017.

[16] B. K. Tripathi, *High Dimensional Neurocomputing*. India: Springer, New Delhi, 2015.

[17] T. Nitta, "A backpropagation algorithm for neural networks based an 3D vector product," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 1993, pp. 589–592.

[18] ——, "Three-dimensional vector valued neural network and its generalization ability," in *Neural Information Processing–Letters and Reviews*, 2006, pp. 237–242.

[19] P. Arena, L. Fortuna, L. Occhipinti, and M. Xibilia, "Neural networks for quaternion-valued function approximation," in *Proc. IEEE Int. Symp. Circuits and Syst.*, 1994, pp. 307–310.

[20] P. Arena, L. Fortuna, G. Muscato, and M. Xibilia, "MLP in quaternion algebra," in *Neural Networks in Multidimensional Domains*. London: Springer-Verlag London, 1998, vol. 234, pp. 49–75.

[21] T. Nitta, "A quaternary version of the back-propagation algorithm," in *Proc. IEEE Int. Conf. Neural Netw.*, 1995, pp. 2753–2756.

[22] C.-A. Popa, "Octonion-valued neural networks," in *Proc. Int. Conf. Artificial Neural Netw.*, 2016, pp. 435–443.

[23] Z. Zhang and S. Aeron, "Denoising and completion of 3D data via multidimensional dictionary learning," in *Proc. Int. Joint Conf. Artificial Intelligence (IJCAI)*, 2016, pp. 2371–2377.

[24] Z.-C. Fan, T.-S. T. Chan, Y.-H. Yang, and J.-S. R. Jang, "Music signal process using vector product neural network," in *Proc. Int. Workshop Deep Learning for Music*, 2017.

[25] S. Jirayucharoensak, S. Pan-Ngum, and P. Israsena, "EEG-based emotion recognition using deep learning network with principal component based covariate shift adaptation," *The Scientific World J.*, vol. 2014, 2014.

[26] T. Nitta, "N-dimensional vector neuron," in *Proc. Int. Joint Conf. Artificial Intelligence (IJCAI)*, 2007, pp. 2–7.

[27] ——, "N-dimensional vector neuron and its application to the N-bit parity problem," in *Complex-valued neural networks: Advances and applications*. New York: John Wiley and Sons, 2013, pp. 59–74.

[28] J. K. Pearson and D. L. Bisset, "Back propagation in a Clifford algebra." in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 2, 1992, pp. 413–416.

[29] ——, "Neural networks in the Clifford domain," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 3, 1994, pp. 1465–1469.

[30] E. J. Bayro-Corrochano, "Geometric neural computing," *IEEE Trans. Neural Netw.*, vol. 12, no. 5, pp. 968–986, 2001.

[31] S. Buchholz and G. Sommer, "Clifford algebra multilayer perceptrons," in *Geometric Computing with Clifford Algebras: Theoretical Foundations and Applications in Computer Vision and Robotics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 315–334.

[32] C.-A. Popa, "Matrix-valued neural networks," in *Mendel 2015*. Cham: Springer International Publishing, 2015, pp. 245–255.

[33] J.-T. Chien and Y.-T. Bao, "Tensor-factorized neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, no. 99, pp. 1–14, Apr. 2017.

[34] B. V. Kryzhanovsky, L. B. Litinskii, and A. L. Mikaelian, "Vector-neuron models of associative memory," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, vol. 2. IEEE, 2004, pp. 909–914.

[35] I. Kanter, "Potts-glass models of neural networks," *Physical Review A*, vol. 37, no. 7, p. 2739, 1988.

[36] D. Bollé, P. Dupont, and J. van Mourik, "Stability properties of Potts neural networks with biased patterns and low loading," *Journal of Physics A: Mathematical and General*, vol. 24, no. 5, p. 1065, 1991.
[37] D. Bollé, P. Dupont, and J. Huyghebaert, "Thermodynamic properties of the Q-state Potts-glass neural network," *Phys. Rew. A*, vol. 45, no. 6, p. 4194, 1992.
[38] B. V. Kryzhanovsky, L. B. Litinskii, and A. Fonarev, "An effective associative memory for pattern recognition," in *Int. Symposium Intelligent Data Anal.* Springer, 2003, pp. 179–186.
[39] T. A. Ell, "On systems of linear quaternion functions," *arXiv preprint arXiv:math/0702084*, 2007.
[40] S. Olariu, *Complex Numbers in N Dimensions*. Amsterdam: Elsevier, 2002.
[41] T.-S. T. Chan and Y.-H. Yang, "Polar n-complex and n-bicomplex singular value decomposition and principal component pursuit," *IEEE Trans. Signal Process.*, vol. 64, no. 24, pp. 6533–6544, 2016.
[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[43] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006.
[44] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Trans. Image Process.*, vol. 15, no. 12, pp. 3736–3745, Dec. 2006.
[45] N. Renard, S. Bourennane, and J. Blanc-Talon, "Denoising and dimensionality reduction using multilinear tools for hyperspectral images," *IEEE Trans. Geosci. Remote Sens.*, vol. 5, no. 2, pp. 138–142, Apr. 2008.
[46] Y. Peng, D. Meng, Z. Xu, C. Gao, Y. Yang, and B. Zhang, "Decomposable nonlocal tensor dictionary learning for multispectral image denoising," in *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, 2014, pp. 2949–2956.
[47] X. Liu, S. Bourennane, and C. Fossati, "Denoising of hyperspectral images using the parafac model and statistical performance analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 10, pp. 3717–3724, Oct. 2012.
[48] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances Neural Info. Process. Systems (NIPS)*, 2013, pp. 315–323.
[49] S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. J. Smola, "On variance reduction in stochastic gradient descent and its asynchronous variants," in *Advances Neural Info. Process. Systems (NIPS)*, 2015, pp. 2647–2655.
[50] P.-S. Huang, S. D. Chen, P. Smaragdis, and M. Hasegawa-Johnson, "Singing-voice separation from monaural recordings using robust principal component analysis," in *Proc. IEEE Int. Conf. Acoust., Speech and Signal Process. (ICASSP)*, 2012, pp. 57–60.
[51] Y.-H. Yang, "Low-rank representation of both singing voice and music accompaniment via learned dictionaries," in *Proc. Int. Soc. Music Info. Retrieval Conf. (ISMIR)*, 2013, pp. 427–432.
[52] T.-S. Chan, T.-C. Yeh, Z.-C. Fan, H.-W. Chen, L. Su, Y.-H. Yang, and R. Jang, "Vocal activity informed singing voice separation with the iKala dataset," in *Proc. IEEE Int. Conf. Acoust., Speech and Signal Process. (ICASSP)*, 2015, pp. 718–722.
[53] D. D. Lee and H. S. Seung, "Learning the parts of objects by nonnegative matrix factorization," *Nature*, vol. 401, pp. 788–791, 1999.
[54] P.-S. Huang, M. Kim, M. Hasegawa-Johnson, and P. Smaragdis, "Singing-voice separation from monaural recordings using deep recurrent neural networks," in *Proc. Int. Soc. Music Info. Retrieval Conf. (ISMIR)*, 2014, pp. 477–482.
[55] J. R. Hershey, Z. Chen, J. L. Roux, and S. Watanabe, "Deep clustering: Discriminative embeddings for segmentation and separation," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2016, pp. 31–35.
[56] X.-L. Zhang and D. Wang, "A deep ensemble learning method for monaural speech separation," *IEEE Trans. Audio, Speech, Language Process.*, vol. 24, no. 5, pp. 967–977, 2016.
[57] "SiSEC MUS Homepage," 2016, [Online] https://sisec.inria.fr/sisec-2016/2016-professionally-produced-music-recordings/.
[58] A. Liutkus, F.-R. Stöter, Z. Rafii, D. Kitamura, B. Rivet, N. Ito, N. Ono, and J. Fontecave, "The 2016 signal separation evaluation campaign," in *Proc. Int. Conf. Latent Variable Anal. Signal Separation*. Springer, 2017, pp. 323–332, [Online]. Available: https://www.sisec17.audiolabs-erlangen.de/.
[59] E. Vincent, R. Gribonval, and C. Fevotte, "Performance measurement in blind audio source separation," *IEEE Trans. Audio, Speech, Language Process.*, vol. 14, no. 4, pp. 1462–1469, July 2006.
[60] A. A. Nugraha, A. Liutkus, and E. Vincent, "Multichannel audio source separation with deep neural networks," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 24, no. 9, pp. 1652–1664, 2016.
[61] F.-R. Stöter, A. Liutkus, R. Badeau, B. Edler, and P. Magron, "Common fate model for unison source separation," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2016, pp. 126–130.
[62] A. Ozerov, E. Vincent, and F. Bimbot, "A general flexible framework for the handling of prior information in audio source separation," *IEEE Trans. Audio, Speech, Language Process.*, vol. 20, no. 4, pp. 1118–1133, 2012.
[63] P.-S. Huang, M. Kim, M. Hasegawa-Johnson, and P. Smaragdis, "Joint optimization of masks and deep recurrent neural networks for monaural source separation," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 23, no. 12, pp. 2136–2147, 2015.
[64] A. Liutkus, D. Fitzgerald, and Z. Rafii, "Scalable audio separation with light kernel additive modelling," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2015, pp. 76–80.
[65] I.-Y. Jeong and K. Lee, "Singing voice separation using rpca with weighted l_ {1} -norm," in *Proc. Int. Conf. Latent Variable Anal. Signal Separation*, 2017, pp. 553–562.
[66] J. L. Durrieu, B. David, and G. Richard, "A musically motivated mid-level representation for pitch estimation and musical audio source separation," *IEEE J. Sel. Topics Signal Process.*, vol. 5, no. 6, pp. 1180–1191, Oct. 2011.
[67] P. Chandna, M. Miron, J. Janer, and E. Gómez, "Monoaural audio source separation using deep convolutional neural networks," in *Proc. Int. Conf. Latent Variable Anal. Signal Separation*. Springer, 2017, pp. 258–266.
[68] Z. Rafii and B. Pardo, "REpeating Pattern Extraction Technique (REPET): A simple method for music/voice separation," *IEEE Trans. Audio, Speech, Language Process.*, vol. 21, no. 1, pp. 73–84, Jan 2013.
[69] S. Uhlich, M. Porcu, F. Giron, M. Enenkl, T. Kemp, N. Takahashi, and Y. Mitsufuji, "Improving music source separation based on deep neural networks through data augmentation and network blending," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2017, pp. 261–265.
[70] A. J. Simpson, G. Roma, E. M. Grais, R. D. Mason, C. Hummersone, A. Liutkus, and M. D. Plumbley, "Evaluation of audio source separation models using hypothesis-driven non-parametric statistical methods," in *2016 24th European Signal Processing Conference (EUSIPCO)*. IEEE, 2016, pp. 1763–1767.
[71] P. Lounesto, *Clifford Algebras and Spinors*. Cambridge: Cambridge University Press, 2001.
[72] P. J. Davis, *Circulant Matrices*. American Mathematical Society; 2 edition, 2012.
[73] S. R. Powell and P. M. Chau, "Time reversed filtering in real-time," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1990, pp. 1239–1243.

Zhe-Cheng Fan was born in 1987. He received the M.S. degree in computer science and information engineering from National Taiwan Normal University and the Ph.D. degree in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 2012 and 2019, respectively. His research interests include machine learning, deep learning, audio signal processing, and audio source separation.

Tak-Shing T. Chan (M'15) received the Ph.D. degree from the University of London in 2008. From 2006 to 2008, he was a Scientific Programmer at the University of Sheffield. In 2011, he worked as a Research Associate at the Hong Kong Polytechnic University. From 2014 to 2018, he was a Postdoctoral Fellow at Academia Sinica, Taiwan. Since April 2019, he is appointed as a Research Officer at Swansea University. His research interests include signal processing, cognitive informatics, distributed computing, pattern recognition, hypercomplex analysis, and topological data analysis.

Yi-Hsuan Yang (M'11–SM'17) is an Associate Research Fellow/Professor with Academia Sinica. He is also the Chief Music Scientist at the Taiwan AI Labs. He received his Ph.D. degree in communication engineering from National Taiwan University in 2010. His research interests include music information retrieval, affective computing, and machine learning. He was an Associate Editor for the *IEEE Transactions on Affective Computing* and the *IEEE Transactions on Multimedia*, both from 2016 to 2019.

Jyh-Shing Roger Jang (M'93) received his Ph.D. from EECS Department at UC Berkeley, where he studied fuzzy logic and neural networks with Lotfi Zadeh, the father of fuzzy logic. As of Aug. 2019, Google Scholar shows over 15,000 citations for Dr. Jang's seminal paper on ANFIS published in 1993. After obtaining his Ph.D., he joined the MathWorks to coauthor the Fuzzy Logic Toolbox (for MATLAB). He has since cultivated a keen interest in implementing industrial software for machine learning. He was a professor in the CS Dept. of National Tsing Hua Univ., Taiwan, from 1995 to 2012. Since August 2012, he has been a professor in the CSIE Dept. of National Taiwan Univ. (NTU), Taiwan. He has published one book entitled Neuro-Fuzzy and Soft Computing by Prentice Hall. He has also maintained toolboxes for Machine Learning and Speech/Audio Processing. He was the general chair of ISMIR (International Society for Music Information Retrieval) Conference, Taipei, 2014 and was a general co-chair of ISMIR Conference, Suzhou, 2017. He is currently serving as the director for FinTech Center at NTU. His research interests include machine learning in practice, with wide applications to speech recognition/assessment/synthesis, music analysis/retrieval, image classification, medical/healthcare data analytics, and FinTech.