AUGUST 14 2024

Efficient design of complex-valued neural networks with application to the classification of transient acoustic signals

Vlad S. Paul ; Philip A. Nelson

J. Acoust. Soc. Am. 156, 1099–1110 (2024)
https://doi.org/10.1121/10.0028230

JASA ARTICLE

Efficient design of complex-valued neural networks with application to the classification of transient acoustic signals

Vlad S. Paul[a] and Philip A. Nelson
Institute of Sound and Vibration Research, University of Southampton, Southampton SO17 1BJ, United Kingdom

ABSTRACT:
A paper by the current authors Paul and Nelson [JASA Express Lett. **3**(9), 094802 (2023)] showed how the singular value decomposition (SVD) of the matrix of real weights in a neural network could be used to prune the network during training. The paper presented here shows that a similar approach can be used to reduce the training time and increase the implementation efficiency of complex-valued neural networks. Such networks have potential advantages compared to their real-valued counterparts, especially when the complex representation of the data is important, which is the often case in acoustic signal processing. In comparing the performance of networks having both real and complex elements, it is demonstrated that there are some advantages to the use of complex networks in the cases considered. The paper includes a derivation of the backpropagation algorithm, in matrix form, for training a complex-valued multilayer perceptron with an arbitrary number of layers. The matrix-based analysis enables the application of the SVD to the complex weight matrices in the network. The SVD-based pruning technique is applied to the problem of the classification of transient acoustic signals. It is shown how training times can be reduced, and implementation efficiency increased, while ensuring that such signals can be classified with remarkable accuracy.
https://doi.org/10.1121/10.0028230

(Received 3 February 2024; revised 2 July 2024; accepted 29 July 2024; published online 14 August 2024)

[Editor: Haiqiang Niu] Pages: 1099–1110

I. INTRODUCTION

The purpose of this paper is to show how complex-valued neural networks (CVNNs) can be efficiently designed by using a novel algorithm based on the singular value decomposition (SVD) of the weight matrices in the network. There has been growing interest in CVNNs with recent reviews (Bassey *et al.*, 2021; Lee *et al.*, 2022) showing the variety of tasks to which CVNNs have already been applied. Although the theory underlying the backpropagation algorithm using complex-valued data was first discussed in the early 1990s (Benvenuto and Piazza, 1992; Georgiou and Koutsougeras, 1992; Leung and Haykin, 1991), CVNNs have recently received increasing interest, with some of the most popular applications being MRI fingerprinting (Virtue *et al.*, 2017), wireless communications (Marseet and Sahin, 2017), and image processing (Cao *et al.*, 2019; Popa, 2017). Some work has been undertaken in the audio signal processing field, typically in the context of speech processing (Hayakawa *et al.*, 2018; Lee *et al.*, 2017) and source localization (Paul and Nelson, 2022; Tsuzuki *et al.*, 2013).

One of the main advantages of CVNNs, as noted by Hirose (2009), is that they can treat the real and imaginary parts of a number as a single component, thus keeping the relation between the magnitude and phase at a given frequency. This property is especially useful for applications involving the processing of acoustic signals, where the frequency-domain is often used as an input feature and where the real and imaginary parts are statistically dependent upon one another. This topic has been discussed in detail in Hirose (2011), where the author presented an analysis showing that a real-valued network, where the real and imaginary parts are concatenated into a single vector, is not equivalent to using the complex-valued number directly. While the addition of the real and imaginary parts separately is the same as the addition of the complex number, the multiplication is different, since the multiplication of complex numbers introduces an angle rotation and an amplitude attenuation/amplification. More about the convergence and merits of CVNNs can be found in Hirose (2009), Nitta (2003), and Zhang *et al.* (2014).

In addition to the properties of the CVNNs described above, complex-valued activation functions and learning rates can be used to enhance training, as discussed in Bassey *et al.* (2021), Scardapane *et al.* (2020), and Zhang and Mandic (2015). Despite their potential benefits, CVNNs have not been extensively used in acoustic signal processing, as shown in Bassey *et al.* (2021). One possible reason for the lack of popularity could be the increased computational cost needed during training, since, as discussed below, the gradients of complex-valued functions require the computation of more terms than for real-valued functions.

The work in this paper will present a technique to remove some of the computational cost during training, ideally without losing any performance. This is known as

[a]Email: vlad_paul_1995@yahoo.com

network pruning and has been intensively researched (Augasta and Kathirvalavakumar, 2013; Blalock *et al.*, 2020; Choudhary *et al.*, 2020), given that computational power has become a concern when employing high dimensional networks with a substantial number of neurons. The use of the SVD as a low-rank approximation technique for removing training parameters is only one of the available pruning techniques and Bermeitinger *et al.* (2019) discussed in detail its use on machine learning models. The use of the SVD in network models can be linked to an early investigation by Psichogios and Ungar (1994), where the SVD was employed to reduce overfitting and enhance generalization error. After the training was completed, Psichogios and Ungar (1994) discarded redundant singular values along with their corresponding hidden layer nodes.

With a focus on acoustics, Cai *et al.* (2014) and Xue *et al.* (2013) used SVD-based approaches to reduce the training parameters of feed-forward networks. In Xue *et al.* (2013), the authors replaced the matrix of weights **W** by two smaller matrices computed from the SVD matrices of **W** once during training. They evaluated the pruning technique on a LVCSR speech recognition task and showed a reduction of model size by 73% with less than 1% relative accuracy loss. In the work by Cai *et al.* (2014), the authors argued against the direct use of the SVD on the randomly initialized weight matrices, showing that the pruning performance can be improved if the SVD is applied only after the model has trained for a couple of iterations. In both of these examples, only real-valued networks (RVNNs) were considered. In a recent study, Singh and Plumbley (2022) investigated the use of a different pruning technique on a Convolutional Neural Network trained for acoustic scene classification. The authors remove filters with similar content, assuming that such filters yield similar responses and are thus redundant for the overall training process.

The technique presented here follows that presented previously (Paul and Nelson, 2023) in order to efficiently design real-valued multilayer perceptrons (MLPs). In this work, it is shown that such an approach can be successfully applied to complex-valued networks, even if here all operations are in the complex domain. Compared to other SVD-based pruning techniques (e.g., Psichogios and Ungar, 1994; Yang *et al.*, 2020), this approach discards weights as the learning progresses and does not need a full training of the model before reducing its dimensions. Other approaches (e.g., Cai *et al.*, 2014; Xue *et al.*, 2013) apply the SVD once at the beginning or during the training and do not change the dimension of the hidden layer when singular values are discarded. In the technique presented here, it is shown that removing singular values at several consecutive points during the training is beneficial. Furthermore, the resizing of the hidden layer ends up being an adaptive process that depends on the task and network structure, as will be discussed later.

The example used here consists of an MLP with two layers (a hidden layer and an output layer), and through the iterative discarding of singular values during training, it is shown that a network can be designed such that it can be implemented with high computational efficiency. The problem of classifying the complex spectra associated with some model transient signals is used to illustrate the method. The transient signals used are the impulse responses of some bandpass filters of the type used in the authors' previous work (Paul and Nelson, 2021b) on the classification of acoustic power spectra. By analogy with the previous work, the use of such models also enables a good estimate of the accuracy to which very similar transient signals can be classified. It is demonstrated that, using an appropriately trained network, two transient signals that can be barely distinguished using conventional spectral analysis can be classified accurately from single time history samples. The work presented also establishes the limits to classification accuracy determined by the level of noise added to the signals. The theoretical background will first be presented and the equations governing the behavior of the network will be derived from first principles.

II. THE COMPLEX-VALUED MLP (CMLP)

A. Background

The derivation of the complex-valued backpropagation algorithm requires an understanding of the theoretical basis for the use of complex numbers and their derivatives. The analysis presented here is based on the work of Kreutz-Delgado (2009) and Amin *et al.* (2011), both of whom draw on the fundamental work of Wirtinger (1927). A detailed discussion of complex numbers and their use for signal processing applications can be found in Adali *et al.* (2011). This paper also helpfully summarizes Wirtinger calculus, its derivative identities, and deals with other issues, such as the treatment of proper and circular complex numbers and how these can enhance performance of algorithms, such as the independent component analysis for source separation.

The following identities based on Kreutz-Delgado (2009), building on earlier work by Brandwood (1983), will be used for the derivation of the backpropagation algorithm. For a function $h(\mathbf{g})$, where $\mathbf{g}$ is a complex vector, the partial derivatives with respect to the complex vector $\mathbf{z}$ and its complex conjugate $\mathbf{z}^*$ are given by the chain rules

$$\frac{\partial h(\mathbf{g})}{\partial \mathbf{z}} = \frac{\partial h}{\partial \mathbf{g}}\frac{\partial \mathbf{g}}{\partial \mathbf{z}} + \frac{\partial h}{\partial \mathbf{g}^*}\frac{\partial \mathbf{g}^*}{\partial \mathbf{z}}, \tag{1}$$

$$\frac{\partial h(\mathbf{g})}{\partial \mathbf{z}^*} = \frac{\partial h}{\partial \mathbf{g}}\frac{\partial \mathbf{g}}{\partial \mathbf{z}^*} + \frac{\partial h}{\partial \mathbf{g}^*}\frac{\partial \mathbf{g}^*}{\partial \mathbf{z}^*}. \tag{2}$$

Furthermore, it also follows that the derivative of the complex conjugate of $h(\mathbf{g})$ is given by the following:

$$\frac{\partial h^*(\mathbf{g})}{\partial \mathbf{z}^*} = \frac{\partial h^*}{\partial \mathbf{g}}\frac{\partial \mathbf{g}}{\partial \mathbf{z}^*} + \frac{\partial h^*}{\partial \mathbf{g}^*}\frac{\partial \mathbf{g}^*}{\partial \mathbf{z}^*}. \tag{3}$$

Equations (2) and (3) can be combined into the form of a composite matrix, given by the following:

$$\begin{bmatrix} \dfrac{\partial h(\mathbf{g})}{\partial \mathbf{z}^*} \\ \dfrac{\partial h^*(\mathbf{g})}{\partial \mathbf{z}^*} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial h}{\partial \mathbf{g}} & \dfrac{\partial h}{\partial \mathbf{g}^*} \\ \dfrac{\partial h^*}{\partial \mathbf{g}} & \dfrac{\partial h^*}{\partial \mathbf{g}^*} \end{bmatrix} \begin{bmatrix} \dfrac{\partial \mathbf{g}}{\partial \mathbf{z}^*} \\ \dfrac{\partial \mathbf{g}^*}{\partial \mathbf{z}^*} \end{bmatrix}. \tag{4}$$

B. Forward propagation in the cMLP

Using matrix notation (capital and bold for matrices; and lowercase and bold for vectors), the forward propagation of a cMLP with two layers can be expressed as follows:

$$\mathbf{a}^{(2)} = \mathbf{W}^{(2)}\mathbf{x} + \mathbf{b}^{(2)}, \tag{5}$$

$$\mathbf{z}^{(2)} = h(\mathbf{a}^{(2)}), \tag{6}$$

$$\mathbf{a}^{(1)} = \mathbf{W}^{(1)}\mathbf{z}^{(2)} + \mathbf{b}^{(1)}, \tag{7}$$

$$\mathbf{z}^{(1)} = h(\mathbf{a}^{(1)}) = \hat{\mathbf{y}}, \tag{8}$$

where all variables are defined in the complex domain. Such a network is illustrated in Fig. 1. Thus, the neurons in the lth layer, (where $l = 1$, 2 in this case) produce a vector of complex outputs $\mathbf{z}^{(l)}$ that are related to the vector of complex inputs $\mathbf{a}^{(l)}$ by a complex activation function $h(\mathbf{a}^{(l)})$. Note that the layers are counted from the output backward, such that $l = 1$ defines the output layer. The vector $\mathbf{b}^{(l)}$ is the complex bias associated with the neurons in the lth layer. Since the task to be solved here involves a classification problem, the cross-entropy function extended for complex numbers can be defined (Cao *et al.*, 2019) as follows:

$$L = -\frac{1}{2}\frac{1}{K}\sum_{k=1}^{K}\left(\mathbb{R}(y_k)\log\left(\mathbb{R}(\hat{y}_k)\right) + \mathbb{I}(y_k)\log\left(\mathbb{I}(\hat{y}_k)\right)\right), \tag{9}$$

where $\mathbb{R}()$ and $\mathbb{I}()$ denote the real and imaginary parts of the kth estimated output $\hat{y}_k$ and target output y_k. Note that $\hat{y}_k = z_k^{(1)}$. This expression reduces to the cross-entropy function discussed in Paul and Nelson (2023) for the real-valued MLP case. For a classification task using complex-valued outputs, the target outputs $\mathbf{y}$ were defined as one-hot encoded vectors, where for the correct class, the target output was defined as $1 + 1j$, which is a straightforward transformation from the real case. This way, a phase term is enforced when estimating the output, which could improve the convergence due to the additional constraint. It should be noted that, in dealing with classification tasks, the activation function in the output layer is usually a softmax function. One of the existing approaches that extend the softmax function to complex-valued data can be defined (Cao *et al.*, 2019) as follows:

$$\text{softmax}(z) = \text{softmax}(\mathbb{R}(z)) + j\cdot\text{softmax}(\mathbb{I}(z)). \tag{10}$$

Since the softmax function is applied to the real and imaginary parts separately, its output can be directly compared to a target output, for example, of $1 + 1j$.

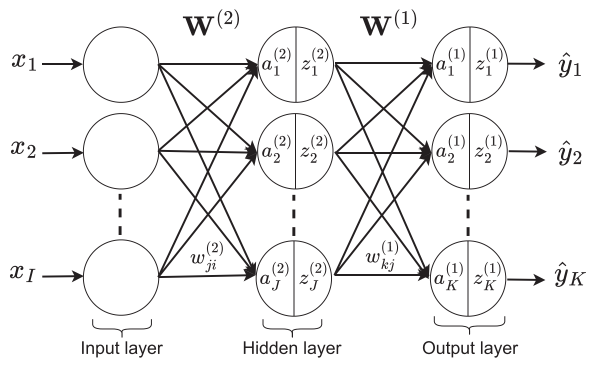

FIG. 1. MLP model with two layers.

C. Backpropagation in the cMLP

Although the work presented here will focus on the use of a two-layer MLP (where $l = 1$, 2), it is simple enough to present the derivation of the backpropagation algorithm for the general case of a number of layers (where $l = 1$, 2, 3,…,l_{max}). Thus, assume that the MLP consists of a number of layers of neurons where the variables are designated starting with the output layer and the layers are designated from $l = 1$ at the output to $l = l_{max}$ at the input.

The complex gradients describing the dependence of the loss function on the matrix of weights $\mathbf{W}^{(l)}$ can be evaluated by using the *vec* operator that sequentially orders the columns of a matrix into a single vector. This gives composite vectors of weights $\mathbf{w}^{(l)} = vec(\mathbf{W}^{(l)})$. First, the gradient of the network outputs with respect to the weight vector $\mathbf{w}^{(l)}$ is computed, and then one can work sequentially through the other layers. When dealing with complex-valued networks, it follows from the Wirtinger calculus (Amin *et al.*, 2011) that the gradient of the loss function with respect to the weights $\mathbf{w}^{(l)}$ can be written as the product

$$\frac{\partial L}{\partial \mathbf{w}^{(l)*}} = \begin{bmatrix} \dfrac{\partial L}{\partial \mathbf{z}^{(1)}} & \left(\dfrac{\partial L}{\partial \mathbf{z}^{(1)}}\right)^* \end{bmatrix} \begin{bmatrix} \dfrac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{w}^{(l)*}} \\ \dfrac{\partial \mathbf{z}^{(1)*}}{\partial \mathbf{w}^{(l)*}} \end{bmatrix}. \tag{11}$$

It is shown in Appendix A that the composite row vector containing the terms $\partial L/\partial \mathbf{z}^{(1)}$ and $(\partial L/\partial \mathbf{z}^{(1)})^*$ can be written as $\tilde{\mathbf{d}} = [\mathbf{d}^{\mathrm{H}} \quad \mathbf{d}^{\mathrm{T}}]$, where elements of the vector $\mathbf{d}$ will depend on whether the network is aimed at either a regression or a classification task. The composite matrix on the right side of the above equation can be written by using the identity in Eq. (4) above such that

$$\begin{bmatrix} \dfrac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{w}^{(l)*}} \\ \dfrac{\partial \mathbf{z}^{(1)*}}{\partial \mathbf{w}^{(l)*}} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{a}^{(1)}} & \dfrac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{a}^{(1)*}} \\ \left(\dfrac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{a}^{(1)*}}\right)^* & \left(\dfrac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{a}^{(1)}}\right)^* \end{bmatrix} \begin{bmatrix} \dfrac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{w}^{(l)*}} \\ \dfrac{\partial \mathbf{a}^{(1)*}}{\partial \mathbf{w}^{(l)*}} \end{bmatrix}. \tag{12}$$

Furthermore, the matrix on the right side of this equation can also be expressed by using the identity in Eq. (4), which then leads to

$$\begin{bmatrix} \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{w}^{(l)*}} \\ \frac{\partial \mathbf{a}^{(1)*}}{\partial \mathbf{w}^{(l)*}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(2)}} & \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(2)*}} \\ \left(\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(2)*}}\right)^* & \left(\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(2)}}\right)^* \end{bmatrix} \begin{bmatrix} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{w}^{(l)*}} \\ \frac{\partial \mathbf{z}^{(2)*}}{\partial \mathbf{w}^{(l)*}} \end{bmatrix}. \quad (13)$$

The matrices of partial derivatives can be written compactly as $\partial \mathbf{z}^{(1)}/\partial \mathbf{a}^{(1)} = \mathbf{H}^{(1)}$, $\partial \mathbf{z}^{(1)}/\partial \mathbf{a}^{(1)*} = \hat{\mathbf{H}}^{(1)}$, where each element in the matrices can be computed for a classification task as shown in Appendix B. It can also be shown (see Appendix C) that $\partial \mathbf{a}^{(1)}/\partial \mathbf{z}^{(2)} = \mathbf{W}^{(1)}$ and that $\partial \mathbf{a}^{(1)}/\partial \mathbf{z}^{(2)*} = \mathbf{0}$, so that the combination of the above two relationships results in

$$\begin{bmatrix} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{w}^{(l)*}} \\ \frac{\partial \mathbf{z}^{(1)*}}{\partial \mathbf{w}^{(l)*}} \end{bmatrix} = \begin{bmatrix} \mathbf{H}^{(1)} & \hat{\mathbf{H}}^{(1)} \\ \hat{\mathbf{H}}^{(1)*} & \mathbf{H}^{(1)*} \end{bmatrix} \begin{bmatrix} \mathbf{W}^{(1)} & 0 \\ 0 & \mathbf{W}^{(1)*} \end{bmatrix} \begin{bmatrix} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{w}^{(l)*}} \\ \frac{\partial \mathbf{z}^{(2)*}}{\partial \mathbf{w}^{(l)*}} \end{bmatrix}. \quad (14)$$

The two composite matrices from above can be rewritten in a more compact form using the composite matrix notations $\tilde{\mathbf{H}}^{(1)}$ and $\tilde{\mathbf{W}}^{(1)}$, respectively.

Now note that one can evaluate the partial derivatives of $\mathbf{z}^{(2)}$ with respect to $\mathbf{w}^{(l)}$ by using identical reasoning. It then follows that

$$\begin{bmatrix} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{w}^{(l)*}} \\ \frac{\partial \mathbf{z}^{(2)*}}{\partial \mathbf{w}^{(l)*}} \end{bmatrix} = \begin{bmatrix} \mathbf{H}^{(2)} & \hat{\mathbf{H}}^{(2)} \\ \hat{\mathbf{H}}^{(2)*} & \mathbf{H}^{(2)*} \end{bmatrix} \begin{bmatrix} \mathbf{W}^{(2)} & 0 \\ 0 & \mathbf{W}^{(2)*} \end{bmatrix} \begin{bmatrix} \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{w}^{(l)*}} \\ \frac{\partial \mathbf{z}^{(3)*}}{\partial \mathbf{w}^{(l)*}} \end{bmatrix}, \quad (15)$$

where again the composite matrices can be substituted by $\tilde{\mathbf{H}}^{(2)}$ and $\tilde{\mathbf{W}}^{(2)}$. This process can be repeated until one reaches the final layer l_{max}, which comprises the input to the network. The gradient at any single layer can be written using the composite matrix notation as

$$\begin{bmatrix} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{w}^{(l)*}} \\ \frac{\partial \mathbf{z}^{(1)*}}{\partial \mathbf{w}^{(l)*}} \end{bmatrix} = \tilde{\mathbf{H}}^{(1)} \tilde{\mathbf{W}}^{(1)} \cdots \tilde{\mathbf{H}}^{(l-1)} \tilde{\mathbf{W}}^{(l-1)} \begin{bmatrix} \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{w}^{(l)*}} \\ \frac{\partial \mathbf{z}^{(l)*}}{\partial \mathbf{w}^{(l)*}} \end{bmatrix}. \quad (16)$$

Finally, it can be shown (see Appendix D) that $\partial \mathbf{a}^{(l)}/\partial \mathbf{w}^{(l)*} = 0$ and $\partial \mathbf{a}^{(l)*}/\partial \mathbf{w}^{(l)*} = \mathbf{z}^{(l+1)\mathrm{H}} \otimes \mathbf{I}^{(1)}$, where $\otimes$ denotes the Kronecker product and $\mathbf{I}^{(l)}$ is the identity matrix of dimensions equal to the length of the vector $\mathbf{z}^{(l+1)}$. It therefore follows that

$$\begin{bmatrix} \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{w}^{(l)*}} \\ \frac{\partial \mathbf{z}^{(l)*}}{\partial \mathbf{w}^{(l)*}} \end{bmatrix} = \begin{bmatrix} \mathbf{H}^{(l)} & \hat{\mathbf{H}}^{(l)} \\ \hat{\mathbf{H}}^{(l)*} & \mathbf{H}^{(l)*} \end{bmatrix} \begin{bmatrix} \mathbf{0} \\ \mathbf{z}^{(l+1)\mathrm{H}} \otimes \mathbf{I}^{(l)} \end{bmatrix}. \quad (17)$$

The net result for the gradient at the lth layer is then given by the product of composite matrices

$$\begin{bmatrix} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{w}^{(l)*}} \\ \frac{\partial \mathbf{z}^{(1)*}}{\partial \mathbf{w}^{(l)*}} \end{bmatrix} = \tilde{\mathbf{H}}^{(1)} \tilde{\mathbf{W}}^{(1)} \cdots \tilde{\mathbf{H}}^{(l-1)} \tilde{\mathbf{W}}^{(l-1)} \tilde{\mathbf{H}}^{(l)} \begin{bmatrix} \mathbf{0} \\ \mathbf{z}^{(l+1)\mathrm{H}} \otimes \mathbf{I}^{(l)} \end{bmatrix}. \quad (18)$$

Writing the product of the composite matrices on the right of this equation as $\tilde{\mathbf{B}}^{(l)}$ then shows that the gradient of the loss function with respect to the lth weight vector $\mathbf{w}^{(l)}$ can be written as

$$\frac{\partial L}{\partial \mathbf{w}^{(l)*}} = \tilde{\mathbf{d}}\tilde{\mathbf{B}}^{(l)} \begin{bmatrix} \mathbf{0} \\ \mathbf{z}^{(l+1)\mathrm{H}} \otimes \mathbf{I}^{(l)} \end{bmatrix}. \quad (19)$$

Using the same reasoning as that described in Paul and Nelson (2021a), one can rewrite the gradient with respect to the matrix of weights $\mathbf{W}^{(l)}$ as

$$\frac{\partial L}{\partial \mathbf{W}^{(l)}} = \left[\mathbf{0} \quad \mathbf{I}^{(l)}\right] \left[\tilde{\mathbf{d}}\tilde{\mathbf{B}}^{(l)}\right]^{\mathrm{T}} \mathbf{z}^{(l+1)\mathrm{H}}. \quad (20)$$

The gradient with respect to the bias vector $\mathbf{b}^{(l)}$ is identical to Eq. (20), but with the term $\mathbf{z}^{(l+1)\mathrm{H}}$ omitted. Note that if $l = l_{max}$, then $\mathbf{z}^{(l+1)\mathrm{H}} = \mathbf{x}^{\mathrm{H}}$. In the work that follows, the above equations were used as the basis of code written in MATLAB, the matrix formulation given enabling a clear understanding of the performance of the algorithms.

D. SVD-based pruning with reduction in hidden layer dimensions

The aim of the pruning technique is to reduce the dimensions of the hidden layer based on the number of singular values discarded iteratively during training. It has been found by Paul and Nelson (2021b) that a good method for choosing the iterations at which to discard singular values (called here discarding points) is to space them logarithmically, using more discarding points at the start of training and fewer as training progresses. The equation used for determining the discarding points is given by the following:

$$d(n) = d(n-1) \cdot 10^{(b-a)/(N-1)}, \quad (21)$$

where N is the total number of discarding points, n is the iteration index, and a and b define, respectively, the lower and higher bounds of the sequence of discarding points. If the lower bound is defined to be 3 for example, the value of a in the equation above would be $a = \log_{10}(3)$. The value of the index τ at which discarding takes place during training is given by the nearest integer value of $d(n)$.

The following notation will be used to derive the update equations for the new set of matrices. At the nth discarding point, before removing any singular values, the SVD of $\mathbf{W}_{n-1}^{(2)}$ is given by $\mathbf{W}_{n-1}^{(2)} = \mathbf{U}_{n-1}\mathbf{\Sigma}_{n-1}\mathbf{V}_{n-1}^{\mathrm{H}}$. After small singular values have been removed, the SVD matrices are replaced by $\mathbf{U}_n\mathbf{\Sigma}_n\mathbf{V}_n^{\mathrm{H}}$, respectively. Following the same notation, assuming the nth discarding point, the hidden layer

$\mathbf{a}_{n-1}^{(2)}$ can be multiplied by $\mathbf{U}_n^{\mathrm{H}}$ every time singular values are discarded and the new hidden layer with fewer neurons is denoted as $\mathbf{a}_n^{(2)}$. Following this, the forward propagation for the hidden layer becomes $\mathbf{U}_n^{\mathrm{H}}\mathbf{a}_{n-1}^{(2)} = (\mathbf{U}_n^{\mathrm{H}}\mathbf{U}_n\boldsymbol{\Sigma}_n\mathbf{V}_n^{\mathrm{H}})\mathbf{x} + \mathbf{U}_n^{\mathrm{H}}\mathbf{b}_{n-1}^{(2)}$. Since $\mathbf{U}_n^{\mathrm{H}}\mathbf{U}_n = \mathbf{I}$, the identity matrix, and by multiplying the remaining matrices $\boldsymbol{\Sigma}_n\mathbf{V}_n^{\mathrm{H}} = \mathbf{W}_n^{(2)}$, the forward propagation for cMLP-SVD is given by the following:

$$\mathbf{a}_n^{(2)} = \mathbf{W}_n^{(2)}\mathbf{x} + \mathbf{b}_n^{(2)}, \tag{22}$$

$$\mathbf{z}_n^{(2)} = h(\mathbf{a}_n^{(2)}), \tag{23}$$

$$\mathbf{a}^{(1)} = \mathbf{W}_n^{(1)}\mathbf{z}_n^{(2)} + \mathbf{b}^{(1)}, \tag{24}$$

$$\mathbf{z}^{(1)} = h(\mathbf{a}^{(1)}) = \hat{\mathbf{y}}, \tag{25}$$

where $\mathbf{a}_n^{(2)} = \mathbf{U}_n^{\mathrm{H}}\mathbf{a}_{n-1}^{(2)}$ and $\mathbf{b}_n^{(2)} = \mathbf{U}_n^{\mathrm{H}}\mathbf{b}_{n-1}^{(2)}$. The size of both $\mathbf{a}_n^{(2)}, \mathbf{b}_n^{(2)}$ depends on the dimensions of $\mathbf{U}_n^{\mathrm{H}}$, which changes depending on how many singular values are discarded in $\boldsymbol{\Sigma}_n$. Note that since the number of neurons in $\mathbf{z}_n^{(2)} = h(\mathbf{a}_n^{(2)})$ can change after every discarded singular value, the dimensions of $\mathbf{W}_n^{(1)}$ also have to be adapted. This can be done by removing as many last columns of $\mathbf{W}_n^{(1)}$ as singular values were removed at that stage. It has been found that this process yields good results; however, it is not impossible that other approaches might prove to be more effective. Finally, since the forward propagation of cMLP-SVD is identical to that from the cMLP, the gradient equations are also identical to Eq. (20), with the observation that the dimensions of the gradients will depend on the number of singular values discarded.

III. RESULTS

A. Description of classification task

Previous work by the current authors (Paul and Nelson, 2021b, 2023) explored the ability of real-valued MLPs to discern acoustic spectra that may be challenging to differentiate using traditional power spectral analysis. Here, however, the objective is to evaluate the extent to which complex MLPs, once taught, can identify small differences between transient acoustic signals. The model used here is based on that used in the previous paper (Paul and Nelson, 2021b). In this case, the transient signals considered are based on the impulse responses of the bandpass filters used in the previous work. A unit impulse signal is passed through bandpass filters having different center frequencies and bandwidths in order to generate impulse responses of very similar bandpass filters. Once the impulse responses have been generated, different white noise signals having a Gaussian distribution were added to the signals based on a signal-to-noise ratio (SNR) in order to investigate the limits of the cMLP to discriminate between small changes in the complex spectra associated with the model transient acoustic signals.

The impulse responses were generated using a filter bandwidth of 200 Hz and a difference in center frequencies of 20 Hz. The network models were trained for two different classification tasks. For the first task, five different classes of impulse responses were generated using center frequencies between 900 and 980 Hz. For the second task, the number of classes was increased from five to ten, the center frequencies being between 800 and 980 Hz. The reason for generating two datasets is that the structure of the network is changing, increasing the number of output neurons from five to ten. The change of structure is expected to influence the number of discarded singular values during training. The length of the impulse responses was defined to be 512 samples. The signal was long enough to capture the ringing of the response for these particular bandpass filters, which decreases by more than 60 dB. The time histories of around 20-ms-long at a sampling frequency of $fs = 24$ kHz were then transformed into the frequency domain. For a fast Fourier transform (FFT) length (N) of 512 samples, the spectral resolution can be calculated as the fraction fs/N, which for the cases presented here was around 47 Hz. In other words, signals generated from bandpass filters with a difference in center frequency smaller than 47 Hz will have closely related spectra and their differences will be difficult to detect by inspection only, especially if white noise is added. Figure 2 shows a comparison between the moduli of the complex spectra of two impulse responses with 10 dB SNR added white noise, where in the upper plot the bandpass filters have a difference in center frequencies of

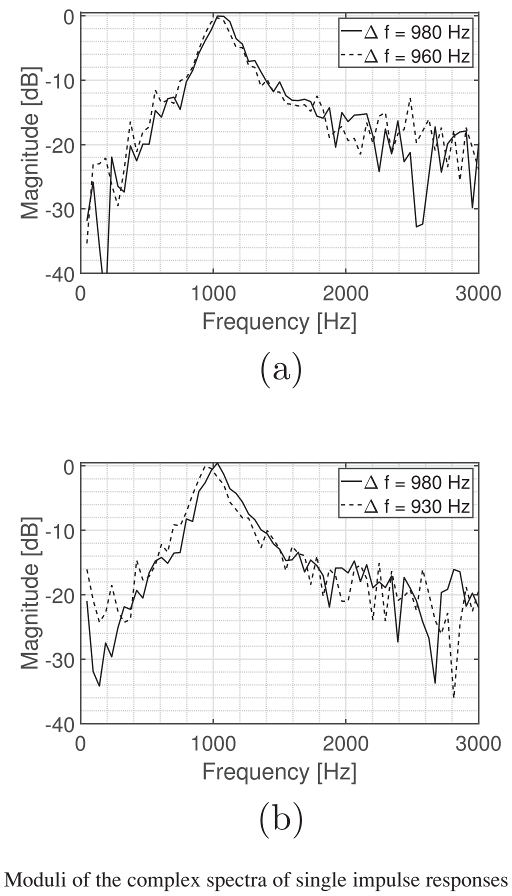

FIG. 2. Moduli of the complex spectra of single impulse responses of bandpass filters with a spacing between center frequencies Δf of: (a) 20 Hz and (b) 50 Hz using a bandwidth of $B = 200$ Hz. White noise was added to each impulse response using a 10 dB SNR.

$\Delta f = 20\,\text{Hz}$, while the lower plot, $\Delta f = 50\,\text{Hz}$. The training dataset consisted of 500 signals in each class, where each class contained the impulse response of one bandpass filter with white noise snippets having the same duration as the impulse response to which it was added. The input into the network was the complex FFT of the noisy impulse response, as shown in Fig. 2. The size of the input layer was 257 samples long, corresponding to the positive frequencies in the FFT spectrum.

B. Network parameters

The 500 samples in each class were split into 60% training, 20% validation, and 20% test datasets and a batch size of 32 was used during the training. Both network architectures were trained using the Adam optimizer (Kingma and Ba, 2014), where the only difference from the real-valued approach is that here the gradients are complex. A real-valued learning rate of 0.002 was chosen, since with a complex-valued learning rate, the training was observed to be less smooth. The use of a complex-valued learning rate is worthy of further investigation, as discussed by Zhang and Mandic (2015). The network architectures had one hidden layer with 50 neurons and an output layer with five or ten neurons corresponding to the different classes. The activation function used in the hidden layer was the complex cardioid function (Virtue *et al.*, 2017), which reduces to rectified linear unit if the numbers are real. The activation function in the output layer was the softmax function applied individually to the real and imaginary values of the estimated output. Since the estimated output is complex valued, the classification accuracy is computed by comparing the moduli of the K elements in the estimated vector $\hat{\mathbf{y}}$ with those in the target vector $\mathbf{y}$. If, for example, the maximum of the moduli of $\hat{\mathbf{y}}$ is at index $k = 1$ and the correct label $1 + 1j$ in the target output vector is also at $k = 1$, the classification is considered to be correct.

The training was stopped after 150 iterations. The results shown below were computed by averaging the performance of ten trials. The results were averaged, since network weights are initialized with random numbers and therefore the performance can differ slightly between trials. For the cMLP-SVD approach, a discarding threshold of 0.2 was chosen empirically, which means that all singular values with magnitude smaller than 20% of the largest singular value were removed at each discarding point. Based on Eq. (21), three discarding points were defined starting with iteration 3 and stopping at one-fourth of the total number of iterations.

The performance of the cMLP and cMLP-SVD models is compared to that of their real-valued counterparts proposed by Paul and Nelson (2023), denoted here as rMLP and rMLP-SVD. For the RVNNs, the real and imaginary parts of the FFT spectrum are concatenated into a one-dimensional vector. The hidden layer of the RVNNs is therefore also doubled to account for the concatenation and the output layers are kept the same for both network types. The discarding threshold for the rMLP-SVD was set to 0.2, keeping the threshold the same for both RVNNs and CVNNs.

The pruning approaches are further compared with a benchmark method adapted from Han *et al.* (2015) (denoted here as cMLP pruned), which is one of the most popular magnitude-based pruning techniques. The method prunes weights in an unstructured manner by replacing them with zeros if their magnitude is smaller than a threshold. For the work presented here, the modulus of the weight was chosen as a threshold. Using this approach, the structure of the network is not changed; however, the weight matrices become sparse and need less storage and fewer computations. By setting weights to zero, the model ignores certain connections during training and can focus on the more important connections. To enable a fair comparison with the SVD approaches, the same percentage of weights are set to zero as number of singular values are removed by the end of training. The pruning is performed once at the same iteration as the third (final) discarding point in the SVD method. This way, the pruned models have a significant number of iterations to fine-tune the weights and to converge. All the network models have been trained on a MacBook Pro M3 Max chip with 14-core CPU and 30-core GPU.

C. Comparison of performance between the cMLP and cMLP-SVD

Table I shows the test accuracy for the first dataset, together with the averaged training time for the full training and the number of floating point operations (FLOPs) needed to compute a forward propagation after the network models finished training. The number of FLOPs can be computed for multiplications and additions of matrices and vectors by evaluating their dimensions (Golub and Van Loan, 2013). A complex-valued operation requires more FLOPs than a real operation. For example, a multiplication of two complex numbers needs four real multiplications and two additions. Note that the number of FLOPs is a rough estimate of the number of operations needed by the network model, since the operations required, for example by the activation functions, are not included.

All network architectures are able to classify a single time history corresponding to a noisy impulse response with an accuracy of 66% or above, depending on the SNR value. The CVNNs have a better performance than their real-valued counterparts on average, and the reasons for this behavior will be discussed in Sec. IV. The training time of the cMLP models is slightly higher than that of the rMLPs due to the complex multiplications in the forward and backward propagation. The SVD-based networks finish training with a similar number of neurons in the hidden layer, even if the rMLP-SVD model starts with twice the number of neurons in the hidden layer.

The proposed pruning technique is outperforming the benchmark method adapted from Han *et al.* (2015) in all three cases. The main reason for this is that the proposed SVD-based pruning technique is more robust to the different

TABLE I. Comparison between the cMLP, cMLP-SVD, rMLP, and rMLP-SVD for a classification task of noisy impulse responses using three different SNR values and five output classes. The networks have 257 neurons as input (514 for the real networks), 50 neurons in the hidden layer (100 for the real networks), and five neurons in the output layer. The values in boldface represent the highest accuracy, the lowest training time, and the smallest number of FLOPs for each training task.

	Accuracy (%)			Training time (s)			Remaining neurons			Number of FLOPs		
SNR (dB)	1	5	10	1	5	10	1	5	10	1	5	10
cMLP	71	93	**100**	25	25	25	50	50	50	**104 910**	104 910	104 910
cMLP-SVD	**75**	**94**	**100**	**7**	**7**	**5**	5	5	4	**10 500**	10 500	8402
cMLP pruned	74	84	92	26	25	25	50	50	50	10 800	10 800	8664
rMLP	66	88	**100**	18	19	17	100	100	100	**103 905**	103 905	103 905
rMLP-SVD	67	85	88	11	10	9	14	5	5	14 551	**5200**	**5200**

datasets and training procedures. This will be discussed further in Sec. IV. Interestingly, as the SNR value becomes higher and the classification task becomes easier to solve, the SVD models discard more neurons in the hidden layer by the end of training. This automatically leads to a reduced training time. Due to the removal of neurons in the hidden layer, the number of FLOPs is also drastically reduced compared to the basic MLP networks for a forward propagation of the trained model. The rMLP-SVD approach needs overall the smallest number of FLOPs; however, the performance is lower on average and the rMLP-SVD models are less robust to the pruning of neurons, as will be discussed in Sec. IV.

Figure 3 shows the behavior of the validation accuracy of the five models for the middle task (SNR 5 dB). The performance is also compared to a cMLP network that starts training with five neurons in the hidden layer, in order to investigate the need of the pruning algorithm (see the curve labeled cMLP5 in Fig. 3). The discarding points where the network is changing its structure can be seen at the iterations where the validation accuracy drops significantly. Note that every time singular values are discarded, a new weight matrix with different values is computed.

With the new weight matrices and hidden layer dimensions, the networks learn quickly and the validation accuracy recovers in a couple of iterations for the tasks

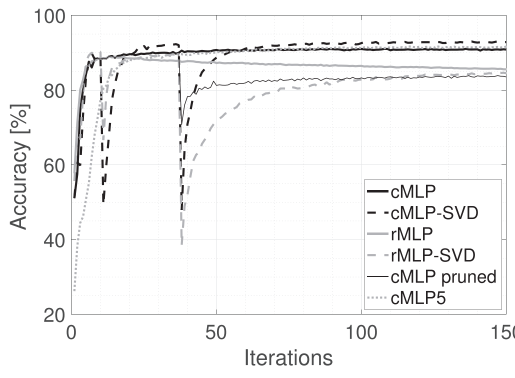

FIG. 3. Validation accuracy of all six networks for the task of classifying noisy impulse responses with an SNR of 5 dB and five output classes. The two thin lines correspond to the benchmark method (cMLP pruned) and the network model with only few initial neurons in the hidden layer.

investigated here. The benchmark method is not able to fully recover after a large number of weights are set to zero; therefore, on average over ten trials, it achieves a lower accuracy than the cMLP-SVD method. The validation accuracy shows a couple of interesting behaviors. First, overfitting occurs for both rMLP and cMLP models. This is shown by the validation accuracy which for the rMLP achieves its maximum at an early stage and starts decreasing as training progresses. The cMLP model is able to reduce this effect and can generalize better. A similar observation was found, for example, by Grinstein and Naylor (2022). A second important observation is that, due to the pruning of the models, both rMLP-SVD and cMLP-SVD models can enhance performance and reduce overfitting. This observation has been found in other work, such as Shmalo *et al.* (2023) and has been discussed in detail in Hirose (2009). A brief discussion about this behavior is mentioned in Sec. IV. A final important factor to note is that training directly the cMLP with five neurons in the hidden layer leads to a slightly lower validation and test accuracy on average compared to the cMLP-SVD. While the difference in performance is not significant, the advantage of using the SVD-based pruning method is that it offers a more robust training behavior on average and gives the user an estimate of the number of neurons that are needed by the model during training. Averaged over the three scenarios discussed in Table I, the cMLP-SVD had a 2%–3% better accuracy than the network models that started with only four or five neurons in the hidden layer.

Moving on to the second scenario, where the number of output classes is increased to ten, Table II shows the performance of the five network models. In this case, the CVNNs have a similar performance to the RVNNs, but do not outperform them as clearly as in the previous scenario. However, the same pattern occurs that the SVD-based approaches outperform the regular cMLP and rMLP models. For the SNR = 1 dB case, the benchmark method outperforms the proposed SVD-based approach, suggesting that the cMLP-SVD model was not able to learn the right patterns after singular values have been discarded. For the other two datasets, the benchmark method shows less robustness to the pruning of weight connections and achieves on average a lower performance.

TABLE II. Comparison between the cMLP, cMLP-SVD, rMLP, and rMLP-SVD for a classification task of noisy impulse responses using three different SNR values and ten output classes. The networks have 257 neurons as input (514 for the real networks), 50 neurons in the hidden layer (100 for the real networks), and ten neurons in the output layer. The values in boldface represent the highest accuracy, the lowest training time, and the smallest number of FLOPs for each training task.

	Accuracy (%)			Training time (s)			Remaining neurons			Number of FLOPs		
SNR (dB)	1	5	10	1	5	10	1	5	10	1	5	10
cMLP	66	89	**100**	57	57	56	50	50	50	106 920	106 920	106 920
cMLP-SVD	71	**94**	**100**	23	21	**13**	9	7	3	19 262	14 986	**6434**
cMLP pruned	**77**	91	92	57	55	58	50	50	50	19 344	15 072	6528
rMLP	65	91	**100**	36	36	36	100	100	100	104 910	104 910	104 910
rMLP-SVD	74	93	86	**21**	**20**	18	11	8	7	**11 549**	**8 402**	7353

The training time is drastically reduced due to the discarding of singular values. This is shown in Fig. 4, where the remaining number of singular values after every discarding point are compared for the two network models for the same task of classifying five or ten impulse responses with a 1 dB SNR. Increasing the number of output classes leads to a faster reduction of neurons in the hidden layer as training progresses but does not necessarily lead to a bigger reduction in training time. As shown in Fig. 4, the SVD models with ten output classes discard more neurons at the first and maybe second discarding points, but fewer at the third discarding point. This behavior can be observed for all scenarios with different SNR values investigated here. These results suggest that the chosen discarding threshold of 0.2 was large, so that it made the SVD models finish training with a similar number of neurons in the hidden layer, regardless of the structure of the network. This observation is particularly valuable as it describes the varying behavior of the SVD approach across different cMLP models. Note that the reduction in the number of network parameters resulted from the pruning may not reveal the "effective dimensionality" of the network, since, as shown by Maddox *et al.* (2020), "simple parameter counting can be a misleading proxy for model complexity and generalization performance."

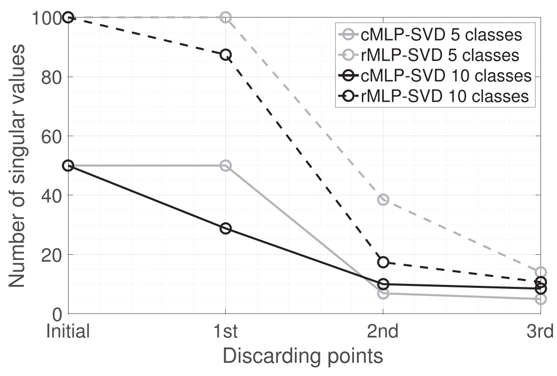

FIG. 4. Remaining number of singular values after every discarding point for the cMLP-SVD, rMLP-SVD when trained on five or ten classes with 1 dB SNR.

IV. DISCUSSION

A. Performance improvements

The pruning method proposed here for cMLPs shows excellent performance, similarly to the work presented in Paul and Nelson (2023) for the real-valued models. The detailed investigation of multiple scenarios with different datasets shows the robustness of the cMLP-SVD approach for the scenarios investigated here. The method can achieve the same or even better accuracy than the regular cMLP in less training time and needs fewer FLOPs. The pruning approach can adapt to different network structures and removes neurons in the hidden layer based on an empirically defined threshold. The multiple discarding points spaced logarithmically show the importance of discarding neurons several times during the training.

Compared to the benchmark method adapted from Han *et al.* (2015), the proposed SVD-based technique shows a higher classification accuracy in most of the cases. The benchmark method is not always able to fully recover after a large number of weights are set to zero; and therefore, on average over ten trials, it achieves a lower accuracy than the cMLP-SVD method. Since the method is setting weights to zero, but is not changing the size of the model, one would need to perform sparse matrix multiplications in order to save training time and FLOPs. The advantage of the proposed SVD technique is that it changes the network structure as training progresses and finishes training with a smaller number of neurons in the hidden layer. At least for the scenarios investigated here, evaluating the singular values and removing those that are small seem to be a more robust way to eliminate neurons and weight connections. Of course, both the cMLP-SVD and the benchmark methods can be developed further and improvements have been already proposed for unstructured weight pruning based on a threshold (Cheng *et al.*, 2023).

Note that the benchmark technique has been used with prior information from the cMLP-SVD approach. For example, if the cMLP-SVD model finished training with five neurons in the hidden layer, the same percentage of weight connections in the cMLP was set to zero when implementing the benchmark method. However, if one would blindly select a percentage for setting weight connections to zero, the classification accuracy would be slightly different. For example, for the SNR = 1 dB and ten output classes, a pruning percentage of 50% leads to a classification accuracy of 70% on average, which is slightly lower than that of the cMLP-SVD. Similarly, if the pruning percentage is increased to 70%, the classification accuracy increases to 75%. One of the reasons for a better performance when more weight connections are removed in the benchmark technique is that the model complexity is reduced and the chance of overfitting is smaller. However, if too many weight connections are set to zero, the chance increases for the models to fail to learn the right patterns after the pruning. This behavior can be observed, especially for the easier tasks (SNR = 5 dB and SNR = 10 dB), where the benchmark method performs worse than the cMLP-SVD. Interestingly, if the discarding threshold of the cMLP-SVD is increased to 0.3, the performance for the SNR = 1 dB and ten output classes is increased from 71% to 81%. However, if the benchmark technique (cMLP pruned) is implemented by eliminating as many weight connections as the cMLP-SVD (94%), the accuracy decreases from 77% to 65%.

B. Reduction of overfitting effect

While the main purpose of the presented technique is to reduce the size of the model in an efficient way, the fact that pruning the network models can reduce overfitting and enhance performance is worth discussing further. The proposed pruning method has a regularizing effect on the model in the sense that it discards small singular values. It ignores certain neurons during training due to the discarding of singular values, thus leading to a reduced overfitting effect in the case investigated here. Compared to a classical approach of reducing overfitting, such as the dropout method (Hinton *et al.*, 2012), the SVD-based pruning technique permanently changes the network structure as training progresses and the final trained model consists of fewer parameters and a reduced complexity. During dropout, the training weights are kept the same (apart from those that are masked), while in the proposed pruning technique, all the values in the weight matrix are changed at every discarding point. The potential advantage of the pruning method to reduce overfitting and thus improve performance is task-dependent and depends strongly on the training process. If certain neurons that are considered redundant learn irrelevant information from the training dataset, discarding them will automatically reduce overfitting. This is indeed the case here, where due to the small dataset and large SNR values, the original network models were prone to overfit on the patterns in the training dataset. In order to investigate this behavior further, it would be helpful to undertake a systematic ablation study (see, e.g., Meyes *et al.*, 2019), which aims to better understand the inner representations of network models. Following such an approach, one can determine the importance of specific parts of the network model and which of these representations are redundant.

C. CVNNs outperform RVNNs

When compared to the real counterparts, the CVNNs outperform the RVNNs in most of the cases, being able to reduce overfitting and enhance the performance. Both networks need similar training times and number of FLOPs; however, due to real operations, the rMLP and rMLP-SVD models need slightly less FLOPs, if enough neurons have been removed during the pruning. One of the main reasons for the performance enhancement is that the complex multiplication reduces the degree of freedom in the CVNNs compared to a multiplication of the real and imaginary parts independently. As discussed in detail by Hirose (2011), reducing a "possibly harmful" part of the freedom during training can result in a better generalization, since the arbitrariness of the solution is reduced. Moreover, as noted above, the use of complex numbers during training imposes additional constraints on the network, which are beneficial in this case. For example, the use of a phase constraint in the target output $(1 + 1\mathrm{j})$ also demonstrates an improved convergence.

D. Removal of neurons using the SVD

Compared to other pruning techniques that remove neurons based on a magnitude threshold [see Cheng *et al.* (2023) for a comprehensive recent review], the SVD-based method creates a new matrix of weights using the information from the low-rank approximation. Therefore, the technique is not only removing redundant neurons, keeping all other weights the same, but rather rebuilds the matrix of weights and the hidden layer using less information that is considered to be more important for the training process. This is the main reason for the large drops in accuracy as training progresses and, depending on the task to be solved, the time needed to recover the accuracy will vary. During the simulations, it was found that the discarding threshold strongly depends on the training process. Based on the distribution of singular values of the matrix of weights at every discarding point, a larger or smaller number of singular values will be discarded. The main influential factors include the training data, weight initialization using random numbers, activation functions, learning rates, and network structures. The substantial dependence on these factors emphasizes the potential for implementing an adaptive process for the discarding threshold. Such an adaptive process would enable the network to autonomously determine the optimal discarding threshold based on the distribution of singular values at any given discarding point.

Finally, this work focused on the use of three discarding points during training, although this number can be varied, possibly allowing for the removal of more singular values or enhancing the performance. However, the computational time may lengthen with each additional discarding point, especially for large weight matrices, where the SVD can become computationally expensive. Future work could involve exploring the scalability of SVD-based pruning with larger networks containing multiple hidden layers or large weight matrices. Initial simulations on cMLP models with several hidden layers showed great potential, but further investigation is required if more general conclusions can be reached. In general, there will be a trade-off between the number of SVDs during training and the number of discarded singular values. If not enough singular values are discarded during the training, the training time might increase. However, even in this case, the advantage of having a pruned network model at the end of training that requires a smaller number of FLOPs when used on a device can be beneficial.

V. CONCLUSIONS

This paper has analyzed from first principles a cMLP with any number of hidden layers and non-holomorphic activation functions. The analysis presented enables the use of the SVD to observe the behavior during training of the singular values of the weight matrices in the network. It is shown how the removal of small singular values during training enables a reduction of the number of neurons in the hidden layer. The discarding of singular values is undertaken sequentially during the training and the time that is saved depends mostly on the size and shape of the MLP network. The proposed cMLP-SVD approach has been successfully applied to a classification task using transient model signals, where the network was trained to distinguish between very small changes in the signals. The effect of SNR on classification accuracy was also established. The performance was compared to the regular cMLP, and it has been shown that the cMLP-SVD can achieve the same or higher accuracy compared to the regular cMLP, using less training time and fewer FLOPs to implement. When compared to the real-valued network models, both cMLP and cMLP-SVD outperform their counterparts in most of the cases and are less prone to overfitting. The method proposed for the cases considered here is likely to be extended to other network architectures and multiple hidden layers, although additional investigations will be necessary.

ACKNOWLEDGMENTS

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC, UKRI) EP/R513325/1.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts of interest to disclose.

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

APPENDIX A

The cross-entropy loss for complex numbers for the kth neuron is given by

$$L_k = -\frac{1}{2}\left(y_{k_R}\log\left(z_{k_R}^{(1)}\right) + y_{k_I}\log\left(z_{k_I}^{(1)}\right)\right), \quad \text{(A1)}$$

where the subscripts R and I denote the real and imaginary parts of the complex number, respectively. Using the identities from Adali *et al.* (2011), it follows that

$$\frac{\partial L_k}{\partial z_k^{(1)}} = \frac{1}{2}\left(\frac{\partial L_k}{\partial z_{k_R}^{(1)}} - j\frac{\partial L_k}{\partial z_{k_I}^{(1)}}\right). \quad \text{(A2)}$$

Computing each gradient term individually shows that

$$\frac{\partial L_k}{\partial z_{k_R}^{(1)}} = -\frac{1}{2}\frac{y_{k_R}}{z_{k_R}^{(1)}} \quad \text{and} \quad \frac{\partial L_k}{\partial z_{k_I}^{(1)}} = -\frac{1}{2}\frac{y_{k_I}}{z_{k_I}^{(1)}}, \quad \text{(A3)}$$

and the final gradient expression can be written as $(-1/4)d_k^*$, where $d_k^* = y_{k_R}/z_{k_R}^{(1)} - jy_{k_I}/z_{k_I}^{(1)}$. If one computes the total error L_k with respect to the vector of outputs $\mathbf{z}^{(1)}$, the expression becomes $\partial L_k/\partial \mathbf{z}^{(1)} = (-1/4)[d_1^*, d_2^*, \ldots, d_K^*] = \mathbf{d}^{\mathrm{H}}$. Similarly, $\partial L_k/\partial \mathbf{z}^{(1)*} = (-1/4)[d_1, d_2, \ldots, d_K] = \mathbf{d}^{\mathrm{T}}$.

APPENDIX B

The terms $\partial \mathbf{z}^{(1)}/\partial \mathbf{a}^{(1)}$ and $\partial \mathbf{z}^{(1)}/\partial \mathbf{a}^{(1)*}$ can be computed as follows. Assuming the mth output of the softmax function $z_m^{(1)}$ and due to the nth input into the function $a_n^{(1)}$, the derivative is given by

$$\frac{\partial z_m^{(1)}}{\partial a_n^{(1)}} = \frac{\partial}{\partial a_n^{(1)}}\left(\mathrm{soft}(\mathbb{R}\{a_m^{(1)}\}) + j\mathrm{soft}(\mathbb{I}\{a_m^{(1)}\})\right). \quad \text{(B1)}$$

By denoting $\mathbb{R}\{a_m^{(1)}\}$ as $a_{m_R}^{(1)}$ and $\mathbb{I}\{a_m^{(1)}\}$ as $a_{m_I}^{(1)}$, we can compute the derivative of $z_m^{(1)}$ with respect to $\mathbb{R}\{a_n^{(1)}\} = a_{n_R}^{(1)}$ and $\mathbb{I}\{a_n^{(1)}\} = a_{n_I}^{(1)}$, following the identities in Adali *et al.* (2011). First, for the real part $a_{n_R}^{(1)}$, it follows that

$$\frac{\partial z_m^{(1)}}{\partial a_{n_R}^{(1)}} = \frac{\partial}{\partial a_{n_R}^{(1)}}\left(\frac{e^{a_{m_R}^{(1)}}}{\sum_{k=1}^{K} e^{a_{k_R}^{(1)}}} + j\frac{e^{a_{m_I}^{(1)}}}{\sum_{k=1}^{K} e^{a_{k_I}^{(1)}}}\right), \quad \text{(B2)}$$

where the sum is over all K output neurons. Since the derivative of the imaginary part with respect to $a_{n_R}^{(1)}$ is zero, if one applies the quotient rule, the resulting expression for the two cases $m = n, m \neq n$ is very similar to the real-valued softmax derivative and is given by

$$\frac{\partial z_m^{(1)}}{\partial a_{n_R}^{(1)}} = \begin{cases} z_{m_R}^{(1)}(1 - z_{m_R}^{(1)}) & \text{if } m = n, \\ z_{m_R}^{(1)} z_{n_R}^{(1)} & \text{if } m \neq n. \end{cases} \quad \text{(B3)}$$

Similarly, the partial derivative of $z_m^{(1)}$ with respect to the imaginary part $a_{n_I}^{(1)}$ is given by

$$\frac{\partial z_m^{(1)}}{\partial a_{n_I}^{(1)}} = \begin{cases} j \cdot z_{m_I}^{(1)}(1 - z_{m_I}^{(1)}) & \text{if } m = n, \\ -j \cdot z_{m_I}^{(1)} z_{n_I}^{(1)} & \text{if } m \neq n. \end{cases} \quad \text{(B4)}$$

Using these results shows that

$$\frac{\partial z_m^{(1)}}{\partial a_n^{(1)}} = \frac{1}{2}\left(\frac{\partial z_m^{(1)}}{\partial a_{n_I}^{(1)}} - j\frac{\partial z_m^{(1)}}{\partial a_{n_I}^{(1)}}\right) \tag{B5}$$

$$= \begin{cases} \frac{1}{2}\left[z_{m_R}^{(1)}(1-z_{m_R}^{(1)}) + z_{m_I}^{(1)}(1-z_{m_I}^{(1)})\right], & m=n, \\ \frac{1}{2}\left[-z_{m_R}^{(1)}z_{n_R}^{(1)} - z_{m_I}^{(1)}z_{n_I}^{(1)}\right], & m\neq n. \end{cases} \tag{B6}$$

The derivative with respect to the complex conjugate $a_n^{(1)*}$ is given by

$$\frac{\partial z_m^{(1)}}{\partial a_n^{(1)*}} = \frac{1}{2}\left(\frac{\partial z_m^{(1)}}{\partial a_{n_I}^{(1)}} - j\frac{\partial z_m^{(1)}}{\partial a_{n_I}^{(1)}}\right) \tag{B7}$$

$$= \begin{cases} \frac{1}{2}\left[z_{m_R}^{(1)}(1-z_{m_R}^{(1)}) - z_{m_I}^{(1)}(1-z_{m_I}^{(1)})\right], & m=n, \\ \frac{1}{2}\left[-z_{m_R}^{(1)}z_{n_R}^{(1)} + z_{m_I}^{(1)}z_{n_I}^{(1)}\right], & m\neq n. \end{cases} \tag{B8}$$

The matrices of derivatives $\mathbf{H}^{(1)}$ and $\hat{\mathbf{H}}^{(1)}$ can be computed by placing all $m=n$ derivatives on the diagonal and all $m \neq n$ derivatives on the off diagonal positions.

APPENDIX C

Since $\mathbf{a}^{(1)} = \mathbf{W}^{(1)}\mathbf{z}^{(2)} + \mathbf{b}^{(1)}$, by omitting the bias term for simplicity, the derivative of the mth term $a_m^{(1)}$ with respect to $z_m^{(2)}$ is given by

$$\frac{\partial a_m^{(1)}}{\partial z_m^{(2)}} = \frac{1}{2}\left(\frac{\partial a_m^{(1)}}{\partial z_{m_R}^{(2)}} - j\frac{\partial a_m^{(1)}}{\partial z_{m_I}^{(2)}}\right). \tag{C1}$$

After some algebra, it follows that $\partial a_m^{(1)}/\partial z_m^{(2)} = w_m^{(1)}$. Similarly, it can be shown that $\partial a_m^{(1)}/\partial z_m^{(2)*} = 0$. It follows that $\partial \mathbf{a}^{(1)}/\partial \mathbf{z}^{(2)} = \mathbf{W}^{(1)}$, and that $\partial \mathbf{a}^{(1)}/\partial \mathbf{z}^{(2)*} = \mathbf{0}$.

APPENDIX D

Applying the identities from Adali *et al*. (2011), the mth element of the derivative can be written as

$$\frac{\partial a_m^{(1)}}{\partial w_m^{(1)}} = \frac{1}{2}\left(\frac{\partial a_m^{(1)}}{\partial w_{m_R}^{(1)}} - j\frac{\partial a_m^{(1)}}{\partial w_{m_I}^{(1)}}\right), \tag{D1}$$

where $w_R^{(1)} = \mathbb{R}\{w^{(1)}\}$ and $w_I^{(1)} = \mathbb{I}\{w^{(1)}\}$. Expanding $a_m^{(1)}$ and omitting some algebra, it can be shown that $\partial a_m^{(1)}/\partial w_m^{(1)} = z_m^{(2)}$. Similarly,

$$\frac{\partial a_m^{(1)}}{\partial w_{m_R}^{(1)*}} = \frac{1}{2}\left(\frac{\partial a_m^{(1)}}{\partial w_{m_R}^{(1)}} + j\frac{\partial a_m^{(1)}}{\partial w_{m_I}^{(1)}}\right), \tag{D2}$$

and the expression can be simplified to $\partial a_m^{(1)}/\partial w_m^{(1)*} = 0$. It follows that the partial derivatives $\partial \mathbf{a}^{(1)}/\partial \mathbf{w}^{(1)*} = 0$ and $\partial \mathbf{a}^{(1)*}/\partial \mathbf{w}^{(1)*} = \mathbf{z}^{(2)\mathrm{H}} \otimes \mathbf{I}^{(1)}$.

Adali, T., Schreier, P. J., and Scharf, L. L. (**2011**). "Complex-valued signal processing: The proper way to deal with impropriety," IEEE Trans. Signal Process. **59**(11), 5101–5125.

Amin, M. F., Amin, M. I., Al-Nuaimi, A. Y. H., and Murase, K. (**2011**). "Wirtinger calculus based gradient descent and Levenberg-Marquardt learning algorithms in complex-valued neural networks," in *Lecture Notes in Computer Science* (Springer, Berlin), Chap. 66, pp. 550–559.

Augasta, M., and Kathirvalavakumar, T. (**2013**). "Pruning algorithms of neural networks: A comparative study," Open Comput. Sci. **3**(3), 105–115.

Bassey, J., Qian, L., and Li, X. (**2021**). "A survey of complex-valued neural networks," arXiv:2101.12249.

Benvenuto, N., and Piazza, F. (**1992**). "On the complex backpropagation algorithm," IEEE Trans. Signal Process. **40**(4), 967–969.

Bermeitinger, B., Hrycej, T., and Handschuh, S. (**2019**). "Singular value decomposition and neural networks," in *Artificial Neural Networks and Machine Learning–ICANN 2019: Deep Learning: 28th International Conference on Artificial Neural Networks*, September 17–19, Munich, pp. 153–164.

Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., and Guttag, J. (**2020**). "What is the state of neural network pruning?," arXiv:2003.03033.

Brandwood, D. (**1983**). "A complex gradient operator and its application in adaptive array theory," IEE Proc., Part H: Microwave Opt. Antennas. **130**(1), 11–16.

Cai, C., Ke, D., Xu, Y., and Su, K. (**2014**). "Fast learning of deep neural networks via singular value decomposition," in *PRICAI 2014: Trends in Artificial Intelligence: 13th Pacific Rim International Conference on Artificial Intelligence*, December 1-5, Gold Coast, Australia, pp. 820–826.

Cao, Y., Wu, Y., Zhang, P., Liang, W., and Li, M. (**2019**). "Pixel-wise polsar image classification via a novel complex-valued deep fully convolutional network," Remote Sens. **11**(22), 2653.

Cheng, H., Zhang, M., and Shi, J. Q. (**2023**). "A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations," arXiv:2308.06767.

Choudhary, T., Mishra, V., Goswami, A., and Sarangapani, J. (**2020**). "A comprehensive survey on model compression and acceleration," Artif. Intell. Rev. **53**, 5113–5155.

Georgiou, G., and Koutsougeras, C. (**1992**). "Complex domain backpropagation," IEEE Trans. Circuits Syst. II **39**(5), 330–334.

Golub, G. H., and Van Loan, C. F. (**2013**). *Matrix Computations* (JHU Press, Baltimore, MD).

Grinstein, E., and Naylor, P. A. (**2022**). "Deep complex-valued convolutional-recurrent networks for single source DOA estimation," in *2022 International Workshop on Acoustic Signal Enhancement (IWAENC)*, Bamberg, Germany, pp. 1–5.

Han, S., Pool, J., Tran, J., and Dally, W. J. (**2015**). "Learning both weights and connections for efficient neural networks," in *Proceedings of the 28th International Conference. on. Neural Information Processing Systems*, Montreal, QC, Canada, December 2015, pp. 1135–1143.

Hayakawa, D., Masuko, T., and Fujimura, H. (**2018**). "Applying complex-valued neural networks to acoustic modeling for speech recognition," in *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, Honolulu, HI (IEEE, New York), pp. 1725–1731.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (**2012**). "Improving neural networks by preventing co-adaptation of feature detectors," arXiv:1207.0580.

Hirose, A. (**2009**). "Complex-valued neural networks: The merits and their origins," in *Proceedings of the International Joint Conference on Neural Networks*, pp. 1237–1244.

Hirose, A. (**2011**). "Nature of complex number and complex-valued neural networks," Front. Electr. Electron. Eng. China **6**(1), 171–180.

Kingma, D. P., and Ba, J. (**2014**). "Adam: A method for stochastic optimization," arXiv:1412.6980.

Kreutz-Delgado, K. (**2009**). "The complex gradient operator and the CR-calculus," arXiv:0906.4835.

Lee, C., Hasegawa, H., and Gao, S. (**2022**). "Complex-valued neural networks: A comprehensive survey," IEEE/CAA J. Autom. Sin. **9**(8), 1406–1426.

Lee, Y.-S., Wang, C.-Y., Wang, S.-F., Wang, J.-C., and Wu, C.-H. (**2017**). "Fully complex deep neural network for phase-incorporating monaural source separation," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, New Orleans, LA (IEEE, New York), pp. 281–285.

Leung, H., and Haykin, S. (**1991**). "The complex backpropagation algorithm," IEEE Trans. Signal Process. **39**(9), 2101–2104.

Maddox, W. J., Benton, G., and Wilson, A. G. (**2020**). "Rethinking parameter counting in deep models: Effective dimensionality revisited," arXiv:2003.02139.

Marseet, A., and Sahin, F. (**2017**). "Application of complex-valued convolutional neural network for next generation wireless networks," in *2017 IEEE Western New York Image and Signal Processing Workshop (WNYISPW)*, Rochester, NY (IEEE, New York), pp. 1–5.

Meyes, R., Lu, M., de Puiseau, C. W., and Meisen, T. (**2019**). "Ablation studies in artificial neural networks," arXiv:1901.08644.

Nitta, T. (**2003**). "Solving the XOR problem and the detection of symmetry using a single complex-valued neuron," Neural Netw. **16**(8), 1101–1105.

Paul, V., and Nelson, P. A. (**2021a**). "Complex valued neural networks for audio signal processing," in *Proceedings of the Institute of Acoustics Vol. 43. Pt. 2*, Milton Keynes, UK, available at https://eprints.soton.ac.uk/457352/.

Paul, V. S., and Nelson, P. A. (**2021b**). "Matrix analysis for fast learning of neural networks with application to the classification of acoustic spectra," J. Acoust. Soc. Am. **149**(6), 4119–4133.

Paul, V., and Nelson, P. A. (**2022**). "Analysis of complex-valued neural networks for audio source localisation," in *Proceedings of the Institute of Acoustics Vol. 44. Pt. 3*, Milton Keynes, UK, available at http://eprints.soton.ac.uk/id/eprint/476879.

Paul, V. S., and Nelson, P. A. (**2023**). "Efficient design of neural networks for the classification of acoustic spectra," JASA Express Lett. **3**(9), 094802.

Popa, C.-A. (**2017**). "Complex-valued convolutional neural networks for real-valued image classification," in *2017 International Joint Conference on Neural Networks (IJCNN), Anchorage*, AK, pp. 816–822.

Psichogios, D. C., and Ungar, L. H. (**1994**). "SVD-NET: An algorithm that automatically selects network structure," IEEE Trans. Neural Netw. **5**(3), 513–515.

Scardapane, S., Van Vaerenbergh, S., Hussain, A., and Uncini, A. (**2020**). "Complex-valued neural networks with nonparametric activation functions," IEEE Trans. Emerg. Top. Comput. Intell. **4**(2), 140–150.

Shmalo, Y., Jenkins, J., and Krupchytskyi, O. (**2023**). "Deep learning weight pruning with RMT-SVD: Increasing accuracy and reducing overfitting," arXiv:2303.08986.

Singh, A., and Plumbley, M. D. (**2022**). "A passive similarity based CNN filter pruning for efficient acoustic scene classification," arXiv:2203.15751.

Tsuzuki, H., Kugler, M., Kuroyanagi, S., and Iwata, A. (**2013**). "An approach for sound source localization by complex-valued neural network," IEICE Trans. Inf. Syst. **E96.D**(10), 2257–2265.

Virtue, P., Yu, S. X., and Lustig, M. (**2017**). "Better than real: Complex-valued neural nets for MRI fingerprinting," in *Proceedings of the IEEE International Conference on Image Processing*, pp. 3953–3957.

Wirtinger, W. (**1927**). "Zur formalen theorie der funktionen von mehr complexen veränderlichen," ("On the formal theory of functions from more complex systems"), Math. Ann. **97**, 357–375.

Xue, J., Li, J., and Gong, Y. (**2013**). "Restructuring of deep neural network acoustic models with singular value decomposition," in *Proceedings of Interspeech*, Lyon, France, pp. 2365–2369.

Yang, H., Tang, M., Wen, W., Yan, F., Hu, D., Li, A., Li, H., and Chen, Y. (**2020**). "Learning low-rank deep neural networks via singular vector orthogonality regularization and singular value sparsification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, Seattle, WA (IEEE, New York), pp. 2899–2908.

Zhang, H., Liu, X., Xu, D., and Zhang, Y. (**2014**). "Convergence analysis of fully complex backpropagation algorithm based on Wirtinger calculus," Cogn. Neurodyn. **8**(3), 261–266.

Zhang, H., and Mandic, D. P. (**2016**). "Is a complex-valued stepsize advantageous in complex-valued gradient learning algorithms?," IEEE Trans. Neural Netw. Learn. Syst. **27**(12), 2730–2735.