Submitted for publication in a special issue on Machine Learning in Acoustics.

Matrix analysis for fast learning of neural networks with application to the classification of acoustic spectra

Vlad S. Paul[1] and Philip A. Nelson[1]

Institute of Sound and Vibration Research, University of Southampton, Southampton, SO17 1BJ, United Kingdom

(Dated: 16 April 2021)

Neural networks are increasingly being applied to problems in acoustics and audio signal processing. Large audio data sets are being generated for use in training machine learning algorithms and the reduction of training times is of increasing relevance. The work presented here begins by reformulating the analysis of the classical multilayer perceptron (MLP) to show the explicit dependence of network parameters on the properties of the weight matrices in the network. This analysis then allows the application of the singular value decomposition (SVD) to the weight matrices. An algorithm is presented which makes use of regular applications of the SVD to progressively reduce the dimensionality of the network. This results in significant reductions in network training times of up to 50 percent with very little or no loss in accuracy. The use of the algorithm is demonstrated by applying it to a number of acoustical classification problems that help quantify the extent to which closely related spectra can be distinguished by machine learning.

I. INTRODUCTION

The increasing availability of computational power has enabled the use of large training data sets in machine learning in a wide variety of applications, including in the field of acoustics (Bianco *et al.*, 2019). The work presented here aims to contribute to acoustical research in this field in three ways. The first is by providing an accessible matrix-based analysis of classical machine learning. The second is by using this analysis to show how the rate of learning can be substantially increased. The third is by seeking to answer a basic question in the application of machine learning to acoustics; the extent to which single samples of frequency spectra generated by different sources might be distinguished from one another, once a neural network has been trained with sufficient data. The answer to this question is provided by formulating a multiclass classification problem. This is tackled by simulating the generation of a number of spectra from different sources and teaching the network to recognise those sources. Tests are then undertaken of the extent to which the sources can be recognised when only a single sample of the source time history is used to derive an FFT-based spectral estimate. The results show that recognition of different sources can be achieved with remarkable accuracy. In particular it is demonstrated that, with adequate training, it is possible to distinguish between two or more spectra that might not easily be distinguished from one another by using conventional power spectral analysis.

Turning to the application of machine learning, there are now many software packages available to assist in the implementation and training of the neural networks that pro-

vide the foundation of many machine learning applications (examples include Tensor Flow (Abadi *et al.*, 2015), the MATLAB Deep Learning Toolbox (The MathWorks, 2020), PyTorch (Paszke *et al.*, 2017), Keras and Scikit-Learn (Géron, 2019)). One of the difficulties of such provision is that the inner workings of the networks are not necessarily readily available to the user and this can make it difficult to develop an understanding of the parameters that determine the performance of the network. The work described here begins with a rigorous analysis of the classical multilayer perceptron (MLP) (Rumelhart *et al.*, 1985, 1986), which forms the foundation for many network architectures that, for example, include recurrent neural networks (RNNs) (Rumelhart *et al.*, 1986), long short-term memory networks (LSTMs) (Hochreiter and Schmidhuber, 1997) and convolutional neural networks (CNNs) (LeCun *et al.*, 1989). One of the main contributions of this paper is to make explicit, in matrix form, the forward and backward propagation equations for an MLP with any number of hidden layers. Whilst there are now many good texts on neural networks and machine learning (see Aggarwal, 2018; Bishop, 1995, 2006; Goodfellow *et al.*, 2016; Graves, 2012; Nielsen, 2015), most descriptions of backpropagation rely on the classical analysis due to Rumelhart *et al.* (1985, 1986). In this paper, the equations of backpropagation are derived from first principles using matrix calculus. To the author's knowledge, this approach does not seem to have been widely used previously, but yields results that may prove useful to other researchers in acoustical machine learning.

In particular, the equations derived facilitate the application of the traditional tools of linear algebra for the further analysis of network parameters. This is demonstrated in the

work described below in which the singular value decomposition (SVD) is applied to the weight matrices in the network with the objective of reducing computational load through effectively reducing the number of weights that have to be updated. Although some work has been undertaken previously using such an approach (Cai *et al.*, 2014; Xue *et al.*, 2013), an algorithm is presented here that demonstrates the effective reduction in training time through the sequential application of the SVD to the weight matrices of an MLP without incurring significant losses in training error. In the work described by Cai *et al.* (2014) the authors discarded a number of small singular values at one point during training and replaced the weight matrix with two component matrices formed from the SVD. These were then trained using backpropagation. A similar approach was taken by Xue *et al.* (2013) who similarly observed significant reductions in training time. The connection between the SVD and neural networks has also been discussed by Bermeitinger *et al.* (2019) and a recent helpful review of other approaches to low-rank approximations in neural networks has been presented by Choudhary *et al.* (2020) . In the work presented here an algorithm is introduced that is shown to be highly effective in progressively discarding singular values as the network is trained. It is also likely that a similar approach could be used to good effect with other network architectures.

The equations governing forward and backward propagation through the network have been implemented using MATLAB and have been applied to the classification problem described above that aims to learn to distinguish between a number of typical acoustical spectra. In Section II the paper introduces the basic matrix analysis of the MLP and introduces the

modifications to the backpropagation algorithm that can be used to incorporate the SVD of the weight matrices. In Section III the algorithm is applied to a number of acoustical classification problems and the enhancement of training performance is demonstrated in Section IV before conclusions are drawn in Section V.

II. MATRIX ANALYSIS OF THE MULTILAYER PERCEPTRON

A. Forward propagation in the basic MLP

This section presents the analysis necessary to enable the use of the singular value decomposition (SVD) in the multilayer perceptron (MLP). An initial description of the forward and backward propagation in matrix form has been described previously in a recent paper (Paul and Nelson, 2020) but is developed considerably further here in order to make explicit the connection to classical backpropagation and its efficient implementation. Most importantly, the equations of backpropagation that enable the use of the SVD are also presented. Figure 1 illustrates the basic structure of the MLP. The input to the network is defined by the I-dimensional vector $\mathbf{x}$. The output of the K neurons in the output layer are denoted by the vector $\mathbf{z}^{(1)}$ whilst the J neurons in the hidden layer have outputs denoted by the vector $\mathbf{z}^{(2)}$. This choice of superscripts, working back from output to input, is made to facilitate the description of the backpropagation process as shown later in the paper. The estimated outputs of the network are denoted by the K-dimensional vector $\hat{\mathbf{y}}$ and the target outputs by the K-dimensional vector $\mathbf{y}$. The KxJ matrix relating the outputs of the neurons in the hidden layer to the inputs of the neurons in the output layer is denoted as $\mathbf{W}^{(1)}$. Similarly,

the JxI matrix relating the input vector to the inputs to the neurons in the hidden layer is denoted as $\mathbf{W}^{(2)}$. The activation relating the input of each neuron to their output will be denoted by $h(\cdot)$. The neurons in the output and hidden layers have inputs denoted respectively by the K- and J-dimensional vectors $\mathbf{a}^{(1)}$ and $\mathbf{a}^{(2)}$. Similarly, the vectors $\mathbf{b}^{(1)}$ and $\mathbf{b}^{(2)}$ define the bias added to the neuron inputs in the output and hidden layers.

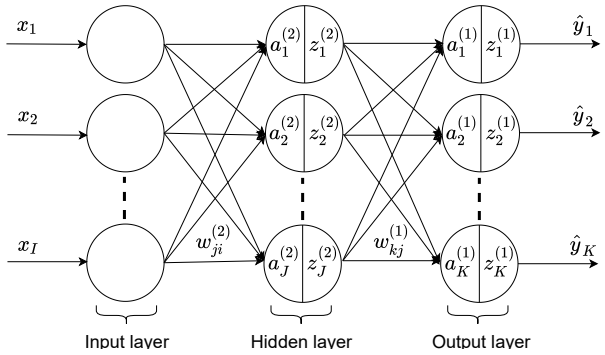


FIG. 1. MLP model with one hidden layer

The forward pass from the input to the output layer through the MLP architecture in Figure 1 can be written in matrix form as

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

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

$$\mathbf{a}^{(1)} = \mathbf{W}^{(1)}\mathbf{z}^{(2)} + \mathbf{b}^{(1)} \tag{3}$$

$$\mathbf{z}^{(1)} = h(\mathbf{a}^{(1)}) = \hat{\mathbf{y}}. \tag{4}$$

In general applications of neural networks, the non-linear activation function $h(\cdot)$ is usually chosen to be the sigmoid, the tanh, the ReLU, or the softmax function (Nielsen, 2015, p.121-126). In the simulations presented in this paper we will use the ReLU in the hidden layer and the softmax function in the output layer. Given that the objective is to train the

network to distinguish between a range of different frequency spectra, the natural loss (or cost) function for such a multiclass classification problem is the cross-entropy (Graves, 2012, p.15-16).The softmax function is often used in the output layer when undertaking multiclass classification tasks, usually in combination with the cross-entropy as the loss function. A detailed description of the the softmax function and its interpretation in identifying the probability of class membership is given by Aggarwal (2018, p. 68). The network is thus trained to minimise the cross-entropy loss function L_{E} given by the sum of the errors e_k where

$$e_k = -y_k \log \hat{y}_k \tag{5}$$

$$L_{\mathrm{E}} = \sum_{k=1}^{K} e_k. \tag{6}$$

The classical approach to reducing the cost function is to use the steepest descent algorithm (Cauchy, 1847) or it's variants to iteratively adjust the weights in the network. This leads to the idea of backpropagation (Rumelhart *et al.*, 1986).

B. Backpropagation through the output layer

The process of backpropagation will now be developed for the network architecture shown in Figure 1. Central to the analysis presented below is the chain rule of partial derivatives expressed in matrix form. Thus for example if a vector quantity $\mathbf{u}$ is a function of another vector quantity $\mathbf{v}$ such that $\mathbf{u} = f_1(\mathbf{v})$ and similarly that $\mathbf{v} = f_2(\mathbf{w})$, $\mathbf{w} = f_3(\mathbf{x})$ then the chain rule can be written in the form (Magnus and Neudecker, 2019, p.100-102)

$$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} = \frac{\partial \mathbf{u}}{\partial \mathbf{v}} \frac{\partial \mathbf{v}}{\partial \mathbf{w}} \frac{\partial \mathbf{w}}{\partial \mathbf{x}}. \tag{7}$$

Note that this relationship holds irrespective of the dimensions of the individual vectors, but it does assume that the functions involved are differentiable. Each of the three terms in the above equation constitutes a Jacobian matrix of partial derivatives, which, if $\mathbf{u}$ is a Mx1 vector and $\mathbf{x}$ is a Nx1 vector, is of form

$$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial u_1}{\partial x_1} & \frac{\partial u_1}{\partial x_2} & \cdots & \frac{\partial u_1}{\partial x_N} \\ \frac{\partial u_2}{\partial x_1} & \frac{\partial u_2}{\partial x_2} & \cdots & \frac{\partial u_2}{\partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial u_M}{\partial x_1} & \frac{\partial u_M}{\partial x_2} & \cdots & \frac{\partial u_M}{\partial x_N} \end{bmatrix}, \tag{8}$$

It is also very important to understand that such Jacobian matrices are ordered in a particular way. For example, if $\mathbf{u}$ is simply a scalar the matrix $\partial \mathbf{u}/\partial \mathbf{x}$ becomes a row vector. This is often known as "numerator notation". See for example Magnus and Neudecker (2019, p.191-210) or the paper by Magnus (2010) for a discussion of this point. Another helpful description is given by Caswell (2019). So for example, using numerator notation, and in what follows in this paper, the derivative of a scalar u with respect to a vector $\mathbf{x}$ is the row vector given by

$$\frac{\partial u}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial u}{\partial x_1} & \frac{\partial u}{\partial x_2} & \cdots & \frac{\partial u}{\partial x_N} \end{bmatrix} \tag{9}$$

Applying this approach to the network in Figure 1, it is simplest to first consider the dependence of the loss function L_{E} on the bias vector $\mathbf{b}^{(1)}$. The relevant partial derivatives can be written as

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{b}^{(1)}} = \frac{\partial L_{\mathrm{E}}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{b}^{(1)}}, \tag{10}$$

When the cross-entropy is used as the cost function, the first term in the chain rule in Eq. (10) results in a row vector $\mathbf{d}^{\mathrm{T}}$ whose k-th entry is given by

$$\frac{\partial L_{\mathrm{E}}}{\partial z_k^{(1)}} = \frac{\partial}{\partial z_k^{(1)}}\left(-y_k \log(z_k^{(1)})\right) = -\frac{y_k}{z_k^{(1)}} = d_k. \tag{11}$$

The second term $\partial \mathbf{z}^{(1)}/\partial \mathbf{a}^{(1)}$ clearly depends on the form of the activation function since $\mathbf{z}^{(1)} = h(\mathbf{a}^{(1)})$. The softmax function defines the relationship between the inputs $\mathbf{a}^{(1)}$ and outputs $\mathbf{z}^{(1)}$ of the neurons in the hidden layer through the relationship (Bridle, 1990)

$$\hat{\mathbf{y}} = \mathbf{z}^{(1)} = \mathrm{softmax}(\mathbf{a}^{(1)}) = \frac{\exp(\mathbf{a}^{(1)})}{\sum_{k=1}^{K} \exp(a_k^{(1)})}. \tag{12}$$

This function has the effect of rescaling a vector of values between 0 and 1, by assigning a value closer to one to the largest components in the vector whilst the lower values are reduced towards zero. The function also ensures that the sum of the components of the output vector add to unity. The derivative of the output $\mathbf{z}^{(1)}$ with respect to the vector $\mathbf{a}^{(1)}$ input to the softmax function can be written element-wise as

$$\frac{\partial z_j^{(1)}}{\partial a_k^{(1)}} = \frac{\partial}{\partial a_k^{(1)}}\left(\frac{e^{a_j^{(1)}}}{\sum_{k=1}^{K} e^{a_k^{(1)}}}\right), \tag{13}$$

where the indices j, k denote the elements of the vectors. Differentiating these terms results in two cases where

$$\frac{\partial z_j^{(1)}}{\partial a_k^{(1)}} = \begin{cases} z_k^{(1)}(1 - z_j^{(1)}) & \text{if} \quad j = k \\ -z_k^{(1)} z_j^{(1)} & \text{otherwise.} \end{cases} \tag{14}$$

The gradient term $\partial \mathbf{z}^{(1)}/\partial \mathbf{a}^{(1)}$ is therefore given by the Jacobian matrix $\mathbf{H}_{\mathrm{E}}^{(1)}$ whose diagonal terms have the form $z_k^{(1)}(1 - z_j^{(1)})$ with off-diagonal terms given by $-z_k^{(1)} z_j^{(1)}$. Finally it follows from Eq. (3) that the last term $\partial \mathbf{a}^{(1)}/\partial \mathbf{b}^{(1)}$ in Eq. (10) is simply the identity matrix

and Eq. (10) can therefore be written as

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{b}^{(1)}} = \mathbf{d}^{\mathrm{T}}\mathbf{H}_{\mathrm{E}}^{(1)}. \tag{15}$$

Now consider the derivative of the loss function with respect to the weight matrix $\mathbf{W}^{(1)}$. The chain rule is this case can be written in the form

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}^{(1)}} = \frac{\partial L_{\mathrm{E}}}{\partial \mathbf{z}^{(1)}}\frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{a}^{(1)}}\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{w}^{(1)}}, \tag{16}$$

where the matrix $\mathbf{W}^{(1)}$ has been transformed into a vector $\mathbf{w}^{(1)}$ by using the $vec(\cdot)$ operator (i.e. $\mathbf{w} = vec(\mathbf{W})$) which forms the vector from stacking successive columns of the matrix (Magnus and Neudecker, 2019, p. 34-36). The new vector $\mathbf{w}^{(1)}$ is of dimensions (KJ)x1 and can now be used in conjunction with the definitions of the partial derivative of a vector with respect to another vector. First note that the first two terms in the above expression are identical to those computed above when evaluating $\partial L_{\mathrm{E}}/\partial \mathbf{b}^{(1)}$. In order to calculate the last term in Eq. (16), Roth's theorem (Roth, 1934) is used. This states that

$$vec(\mathbf{ABC}) = (\mathbf{C}^{\mathrm{T}} \otimes \mathbf{A})vec(\mathbf{B}), \tag{17}$$

where $\otimes$ is the Kronecker product. This relationship can be used to rewrite the forward propagation for $\mathbf{a}^{(1)}$, given by

$$\mathbf{a}^{(1)} = \mathbf{W}^{(1)}\mathbf{z}^{(2)} + \mathbf{b}^{(1)} = \mathbf{I}^{(1)}\mathbf{W}^{(1)}\mathbf{z}^{(2)} + \mathbf{b}^{(1)}, \tag{18}$$

where $\mathbf{I}^{(1)}$ is a KxK identity matrix. Application of Roth's theorem then results in

$$vec(\mathbf{a}^{(1)}) = (\mathbf{z}^{(2)\mathrm{T}} \otimes \mathbf{I}^{(1)})vec(\mathbf{W}^{(1)}) + \mathbf{b}^{(1)}. \tag{19}$$

As described above, the $vec(\cdot)$ operator reorders the matrix $\mathbf{W}^{(1)}$ into the vector $\mathbf{w}^{(1)}$ and $vec(\cdot)$ of a vector is the vector itself. Thus the final expression is given by

$$\mathbf{a}^{(1)} = (\mathbf{z}^{(2)\mathrm{T}} \otimes \mathbf{I}^{(1)})\mathbf{w}^{(1)} + \mathbf{b}^{(1)}. \tag{20}$$

The last term $\partial \mathbf{a}^{(1)}/\partial \mathbf{w}^{(1)}$ in Eq. (16) can then be simplified to give

$$\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{w}^{(1)}} = \frac{\partial}{\partial \mathbf{w}^{(1)}} \left[(\mathbf{z}^{(2)\mathrm{T}} \otimes \mathbf{I}^{(1)})\mathbf{w}^{(1)} + \mathbf{b}^{(1)} \right] = \mathbf{z}^{(2)\mathrm{T}} \otimes \mathbf{I}^{(1)}. \tag{21}$$

Finally, the gradient of the loss function L_{E} with respect to the first set of weights is given by combining the results of Eq. (15) and (21) so that

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}^{(1)}} = \mathbf{d}^{\mathrm{T}} \mathbf{H}_{\mathrm{E}}^{(1)} (\mathbf{z}^{(2)\mathrm{T}} \otimes \mathbf{I}^{(1)}). \tag{22}$$

This relationship thus specifies a row vector whose elements are the derivatives of the loss function with respect to each element of the weight matrix $\mathbf{W}^{(1)}$. However, recalling that $\mathbf{w}^{(1)} = vec(\mathbf{W}^{(1)})$ it can be shown that this relationship can be cast into the form of a matrix that explicitly shows the derivatives of the loss function with respect to each element of the weight matrix. That is, again using numerator notation, the matrix of the partial derivatives $\partial L_{\mathrm{E}}/\partial \mathbf{W}^{(1)}$ is given by

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{W}^{(1)}} = \begin{bmatrix} \frac{\partial L_{\mathrm{E}}}{\partial w_{11}^{(1)}} & \frac{\partial L_{\mathrm{E}}}{\partial w_{12}^{(1)}} & \cdots & \frac{\partial L_{\mathrm{E}}}{\partial w_{1J}^{(1)}} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial L_{\mathrm{E}}}{\partial w_{K1}^{(1)}} & \frac{\partial L_{\mathrm{E}}}{\partial w_{K2}^{(1)}} & \cdots & \frac{\partial L_{\mathrm{E}}}{\partial w_{KJ}^{(1)}} \end{bmatrix}. \tag{23}$$

It is shown in the Appendix that, by taking the matrix transpose of Eq. (22) for $\partial L_{\mathrm{E}}/\partial \mathbf{w}^{(1)}$, the gradient of the loss function L_{E} with respect to the weight matrix $\mathbf{W}^{(1)}$ can be written in the form

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{W}^{(1)}} = \mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}} \mathbf{d}\mathbf{z}^{(2)\mathrm{T}}. \tag{24}$$

Similarly, taking the transpose of Eq. (15) that gives the gradient of the loss L_{E} with respect to the bias vector $\mathbf{b}^{(1)}$ results in

$$\left[\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{b}^{(1)}}\right]^{\mathrm{T}} = \mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}}\mathbf{d}, \tag{25}$$

which is now in the form of a column vector. To summarise, recall that $\mathbf{d}$ is the vector containing the errors d_k and the KxK matrix $\mathbf{H}_{\mathrm{E}}^{(1)}$ contains the derivatives $\partial z_j^{(1)}/\partial a_k^{(1)}$ of the non-linear activation functions. If one checks the dimensions, the fraction $\partial L_{\mathrm{E}}/\partial \mathbf{W}^{(1)}$ results in a KxJ matrix of gradients and $\partial L_{\mathrm{E}}/\partial \mathbf{b}^{(1)}$ is a Kx1 row vector.

Finally, it is worth noting that very often in applications of machine learning, the loss function chosen for minimisation in adjusting the parameters in the network is given by the sum of squared errors between the elements of the output and target vectors. Thus a loss function $L_{\mathrm{S}} = \sum_k^K e_k^2$ is defined having elements $e_k = \hat{y}_k - y_k$. In this case, the first term in Eq. (16) given by $\partial L_{\mathrm{S}}/\partial \mathbf{z}^{(1)}$ results in a row vector $\mathbf{e}^{\mathrm{T}}$ with entries $e_k = \hat{y}_k - y_k$, again using numerator notation. Also, if the sigmoid, tanh or ReLU functions are used in the output layer, assuming the general notation $h(\cdot)$, the gradient term $\partial \mathbf{z}^{(1)}/\partial \mathbf{a}^{(1)}$ results in a diagonal matrix $\mathbf{H}^{(1)}$ whose k-th term is given by $h'(a_k^{(1)})$ since the k-th element $\partial z_k^{(1)}/\partial a_k^{(1)}$ can be written as

$$\frac{\partial z_k^{(1)}}{\partial a_k^{(1)}} = \frac{\partial}{\partial a_k^{(1)}} h(a_k^{(1)}) = h'(a_k^{(1)}). \tag{26}$$

There is thus an important difference between $\mathbf{H}_{\mathrm{E}}^{(1)}$ and $\mathbf{H}^{(1)}$ when implementing backpropagation, since the softmax function involves all output neurons in generating every k-th output.

C. Backpropagation through the hidden layer

The next step in the analysis is to establish the dependence of the loss function on the bias vector $\mathbf{b}^{(2)}$ and the weight matrix $\mathbf{W}^{(2)}$. Again, it helps to start with the gradient with respect to the bias vector which can be written as

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{b}^{(2)}} = \frac{\partial L_{\mathrm{E}}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{b}^{(2)}}. \tag{27}$$

The first term in this expression has been calculated above and is simply given by the row vector $\mathbf{d}^{\mathrm{T}}$. The second term in Eq. (27) can be written as

$$\frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{z}^{(2)}} = \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(2)}}. \tag{28}$$

The first matrix of derivatives in this expression has been evaluated above in Eq. (26) and is given by $\mathbf{H}_{\mathrm{E}}^{(1)}$. The second matrix of derivatives is readily evaluated to give

$$\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(2)}} = \frac{\partial}{\partial \mathbf{z}^{(2)}} (\mathbf{W}^{(1)} \mathbf{z}^{(2)} + \mathbf{b}^{(1)}) = \mathbf{W}^{(1)}. \tag{29}$$

Finally, it follows from the relationship $\mathbf{z}^{(2)} = h(\mathbf{W}^{(2)} \mathbf{x} + \mathbf{b}^{(2)})$ that the third term in the chain rule of Eq. (27) is given by the diagonal matrix $\mathbf{H}^{(2)}$ of derivatives of the hidden layer activation functions. Therefore Eq. (27) can be written as

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{b}^{(2)}} = \mathbf{d}^{\mathrm{T}} \mathbf{H}_{\mathrm{E}}^{(1)} \mathbf{W}^{(1)} \mathbf{H}^{(2)}. \tag{30}$$

Now consider the derivative of the loss function with respect to the weight matrix where the chain rule can be written in the form

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}^{(2)}} = \frac{\partial L_{\mathrm{E}}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{w}^{(2)}}, \tag{31}$$

where again the elements of the weight matrix have been stacked into a vector using $\mathbf{w}^{(2)} = vec(\mathbf{W}^{(2)})$. The first two terms in this expression have already been evaluated above and

yield the product $\mathbf{d}^{\mathrm{T}}\mathbf{H}_{\mathrm{E}}^{(1)}\mathbf{W}^{(1)}$. The third term in Eq. (31) can be calculated from the relationship $\mathbf{z}^{(2)} = h(\mathbf{W}^{(2)}\mathbf{x} + \mathbf{b}^{(2)})$ by writing

$$\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{w}^{(2)}} = \frac{\partial h(\mathbf{a}^{(2)})}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{w}^{(2)}}. \tag{32}$$

The first term follows from Eq. (26) and is given by the matrix $\mathbf{H}^{(2)}$. Substituting the expression for $\mathbf{a}^{(2)}$ results in

$$\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{w}^{(2)}} = \mathbf{H}^{(2)} \frac{\partial}{\partial \mathbf{w}^{(2)}} (\mathbf{I}^{(2)}\mathbf{W}^{(2)}\mathbf{x} + \mathbf{b}^{(2)}), \tag{33}$$

where $\mathbf{I}^{(2)}$ is a JxJ identity matrix that has been used to pre-multiply the weight matrix. This allows the application of Roth's theorem in an analogous way to that used in Eq. (21) so that the expression can be written as

$$\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{w}^{(2)}} = \mathbf{H}^{(2)} \frac{\partial}{\partial \mathbf{w}^{(2)}} \left[(\mathbf{x}^{\mathrm{T}} \otimes \mathbf{I}^{(2)}) vec(\mathbf{W}^{(2)}) + \mathbf{b}^{(2)} \right] = \mathbf{H}^{(2)} (\mathbf{x}^{\mathrm{T}} \otimes \mathbf{I}^{(2)}). \tag{34}$$

The three terms from Eq. (31) used to evaluate $\partial L_{\mathrm{E}} / \partial \mathbf{w}^{(2)}$ can now be written as

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}^{(2)}} = \mathbf{d}^{\mathrm{T}} \mathbf{H}^{(1)} \mathbf{W}^{(1)} \mathbf{H}^{(2)} (\mathbf{x}^{\mathrm{T}} \otimes \mathbf{I}^{(2)}), \tag{35}$$

where the identity matrix in the above expression has the dimensions JxJ. Taking the matrix transpose of this result, and again using the manipulations described in the Appendix, the matrix of derivatives of the loss function with respect to the weights can be written as

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{W}^{(2)}} = \mathbf{H}^{(2)} \mathbf{W}^{(1)\mathrm{T}} \mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}} \mathbf{d}\mathbf{x}^{\mathrm{T}}. \tag{36}$$

Checking the dimensions resulting from the the multiplication of the terms in Eq. (36) shows the equation to be dimensionally correct. Similarly, taking the transpose of the expression for the derivative of the loss function with respect to the bias vector shows that

$$\left[\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{b}^{(2)}} \right]^{\mathrm{T}} = \mathbf{H}^{(2)} \mathbf{W}^{(1)\mathrm{T}} \mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}} \mathbf{d}. \tag{37}$$

The weight matrices can now be updated using the gradient descent algorithm or one of its variants. Thus the final update equations for the weight matrix $\mathbf{W}^{(1)}$ can be written as

$$\mathbf{W}^{(1)}(\tau+1) = \mathbf{W}^{(1)}(\tau) - \eta \frac{\partial L}{\partial \mathbf{W}^{(1)}}(\tau), \tag{38}$$

where τ corresponds to the iteration and η is the learning rate. Similar expressions follow for updating the second set of weights $\mathbf{W}^{(2)}$ and the bias vectors $\mathbf{b}^{(1)}$ and $\mathbf{b}^{(2)}$.

D. Backpropagation through any number of layers

The analysis presented above is readily extended to deal with multiple hidden layers. For example, assume that a second hidden layer is introduced between the input $\mathbf{x}$ and the hidden layer $\mathbf{z}^{(2)}$. The output of the neurons in this layer will be denoted by $\mathbf{z}^{(3)}$, where it is assumed that the layer consists of P neurons. A third matrix of weights $\mathbf{W}^{(3)}$ with dimensions $P\mathrm{x}I$ is then introduced that connects the inputs to the third layer of neurons. The dependence of the loss function on the weights in this matrix is given by the gradient $\partial L_{\mathrm{E}}/\partial \mathbf{w}^{(3)}$ and can be calculated by application of the chain rule. Similarly for the bias vector associated with the third layer of neurons. It therefore follows that

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}^{(3)}} = \frac{\partial L_{\mathrm{E}}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(3)}} \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{w}^{(3)}} \tag{39}$$

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{b}^{(3)}} = \frac{\partial L_{\mathrm{E}}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(3)}} \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{b}^{(3)}}. \tag{40}$$

The first two terms in the expression for $\partial L_{\mathrm{E}}/\partial \mathbf{w}^{(3)}$ have already been computed above, so that $\partial L_{\mathrm{E}}/\partial \mathbf{z}^{(1)} = \mathbf{d}^{\mathrm{T}}$ and $\partial \mathbf{z}^{(1)}/\partial \mathbf{z}^{(2)} = \mathbf{H}^{(1)}\mathbf{W}^{(1)}$. Application of the same processes as used above can be used to compute the next two terms. Firstly note that

$$\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(3)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(3)}} = \mathbf{H}^{(2)} \frac{\partial}{\partial \mathbf{z}^{(3)}} (\mathbf{W}^{(2)} \mathbf{z}^{(3)}), \tag{41}$$

and therefore it follows that

$$\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(3)}} = \mathbf{H}^{(2)}\mathbf{W}^{(2)}, \tag{42}$$

where $\mathbf{H}^{(2)}$ results from a similar calculation to that described by Eq. (26). Next, the last term for both gradients can be computed in the same way that shown in Eq. (34) so that

$$\frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{w}^{(3)}} = \mathbf{H}^{(3)}(\mathbf{x}^{\mathrm{T}} \otimes \mathbf{I}^{(3)}) \tag{43}$$

$$\frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{b}^{(3)}} = \mathbf{H}^{(3)}. \tag{44}$$

It therefore follows that the gradients with respect to the weights and biases is given in matrix form as

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}^{(3)}} = \mathbf{d}^{\mathrm{T}}\mathbf{H}_{\mathrm{E}}^{(1)}\mathbf{W}^{(1)}\mathbf{H}^{(2)}\mathbf{W}^{(2)}\mathbf{H}^{(3)}(\mathbf{x}^{\mathrm{T}} \otimes \mathbf{I}^{(3)}) \tag{45}$$

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{b}^{(3)}} = \mathbf{d}^{\mathrm{T}}\mathbf{H}_{\mathrm{E}}^{(1)}\mathbf{W}^{(1)}\mathbf{H}^{(2)}\mathbf{W}^{(2)}\mathbf{H}^{(3)}. \tag{46}$$

Again these expressions may be transposed as described above and in the Appendix to give an explicit expression for the matrix of derivatives of the loss function with respect to the elements of $\mathbf{W}^{(3)}$. The matrix of derivatives with respect to the weights is given by

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{W}^{(3)}} = \mathbf{H}^{(3)}\mathbf{W}^{(2)\mathrm{T}}\mathbf{H}^{(2)}\mathbf{W}^{(1)\mathrm{T}}\mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}}\mathbf{d}\mathbf{x}^{\mathrm{T}}. \tag{47}$$

It is readily shown that the extension of the network through the addition of further hidden layers leads to more general gradient equations having an obvious recurrent structure. Of particular importance are the equations of backpropagation in matrix form. Thus note that from the above analysis one may define $\boldsymbol{\delta}^{(1)} = \mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}}\mathbf{d}$ such that $\boldsymbol{\delta}^{(2)} = \mathbf{H}^{(2)}\mathbf{W}^{(1)\mathrm{T}}\boldsymbol{\delta}^{(1)}$ and $\boldsymbol{\delta}^{(3)} = \mathbf{H}^{(3)}\mathbf{W}^{(2)\mathrm{T}}\boldsymbol{\delta}^{(2)}$ and so forth. In general for an MLP architecture with an arbitrary number of layers one may write for the n-th layer (where $n > 1$ and counting back from the

output layer $n = 1$)

$$\boldsymbol{\delta}^{(n)} = \mathbf{H}^{(n)}\mathbf{W}^{(n-1)\mathrm{T}}\boldsymbol{\delta}^{(n-1)} \tag{48}$$

Similarly the update equations for the weights and biases in a network where each layer has an arbitrary number of neurons, can be written as

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{W}^{(n)}} = \boldsymbol{\delta}^{(n)}\mathbf{z}^{(n)\mathrm{T}} \tag{49}$$

$$\left[\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{b}^{(n)}}\right]^{\mathrm{T}} = \boldsymbol{\delta}^{(n)}, \tag{50}$$

where $\mathbf{z}^{(n)\mathrm{T}}$ is always the input to the set of weights preceding the n-th hidden layer. In other words, if $n = 1$, the input vector $\mathbf{z}^{(1)\mathrm{T}} = \mathbf{x}^{\mathrm{T}}$. An important special case occurs when the loss function for updating the network is chosen to be the sum of squared errors. As noted above, in that case the matrix $\mathbf{H}_{\mathrm{E}}^{(1)}$ is replaced by the diagonal matrix $\mathbf{H}^{(1)}$ such that $\mathbf{H}^{(1)} = \mathbf{H}^{(1)\mathrm{T}}$ and the vector $\mathbf{d}$ is replaced by the vector $\mathbf{e}$ of errors. In this case $\boldsymbol{\delta}^{(1)} = \mathbf{H}^{(1)\mathrm{T}}\mathbf{e}$ but otherwise the backpropagation relationships above still apply.

E. Application of the singular value decomposition

The singular value decomposition (SVD) can be used to approximate a matrix by another matrix of lower rank (Eckart and Young, 1936). The technique can be used with matrices of any rectangular form, such that a real matrix $\mathbf{W}$ of dimensions MxN can be decomposed into the product of three matrices given by

$$\mathbf{W} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}}, \tag{51}$$

where $\mathbf{U}$ is the MxM matrix whose columns are the left singular vectors of $\mathbf{W}$ and $\mathbf{V}^{\mathrm{T}}$ is the NxN matrix whose rows are the right singular vectors of $\mathbf{W}$. The matrix $\boldsymbol{\Sigma}$ is the

MxN diagonal matrix containing R nonzero singular values σ of $\mathbf{W}$, where $R = \text{rank}(\mathbf{W})$ and where $\sigma_1 > \sigma_2 ... > \sigma_R$ are the rank ordered singular values. The computation of this decomposition is a routine matter (Golub and van Loan, 2013). The advantage of the SVD in this context is that a weight matrix can be approximated by discarding (setting to zero) small singular values and this in turn leads to a reduction in the number of terms that have to be updated during backpropagation. The rationale for such an approach dates back to the Eckart-Young-Mirsky theorem (Eckart and Young, 1936) that shows that discarding small singular values leads to the minimisation of the Frobenius norm of the difference between the original matrix and its lower rank approximation (Klema and Laub, 1980; Xue *et al.*, 2013). The number of singular values discarded is clearly problem dependent and Xu (1998), for example, presents a review of methods that can be used to determine a discarding threshold of singular values in linear least squares problems. Discarding more singular values gives greater savings in computational time, although the degree to which this will be effective is also problem dependent. Of course by discarding singular values, information may be lost and this in turn may affect the training of the network. The previous work using the SVD (Cai *et al.*, 2014; Xue *et al.*, 2013) showed that it is possible to reduce training times without major sacrifices in accuracy provided an appropriate number of singular values are discarded. It should also be mentioned that there is also a connection between the SVD and the "autoencoder" network as discussed by Aggarwal (2018, p. 70-76) although in the case considered here the component matrices of the SVD simply replace the hidden layer and in principle could be used in any number of hidden layers in deeper networks. Here we demonstrate that the method is particularly well-suited to spectral classification problems

in acoustics, while providing a different technique of iteratively discarding singular values during training.

F. Forward and backward propagation of the MLP-SVD

The forward propagation through the basic MLP with one hidden layer described in Section II A can be rewritten using the SVD of the weight matrix $\mathbf{W}^{(2)}$ between the input and the hidden layer such that

$$\mathbf{a}^{(2)} = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}})\mathbf{x} + \mathbf{b}^{(2)} \tag{52}$$

$$\mathbf{z}^{(2)} = h(\mathbf{a}^{(2)}) \tag{53}$$

$$\mathbf{a}^{(1)} = \mathbf{W}^{(1)}\mathbf{z}^{(2)} + \mathbf{b}^{(1)} \tag{54}$$

$$\mathbf{z}^{(1)} = h(\mathbf{a}^{(1)}) = \hat{\mathbf{y}}, \tag{55}$$

where $\mathbf{U}$, $\mathbf{V}^{\mathrm{T}}$ and $\mathbf{\Sigma}$ define the SVD of the matrix $\mathbf{W}^{(2)}$. It is helpful to simplify these equations by making the substitutions $\mathbf{p} = \mathbf{V}^{\mathrm{T}}\mathbf{x}$ and $\mathbf{q} = \mathbf{\Sigma}\mathbf{p} = \mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}\mathbf{x}$. This enables the expression for $\mathbf{a}^{(2)}$ to be written in the form

$$\mathbf{a}^{(2)} = \mathbf{U}\mathbf{\Sigma}\mathbf{p} + \mathbf{b}^{(2)} = \mathbf{U}\mathbf{q} + \mathbf{b}^{(2)}. \tag{56}$$

Using the above definitions of the vectors $\mathbf{p}$ and $\mathbf{q}$, the interaction between the input vector and the output layer can be described by the diagram shown in Figure 2, where the vectors $\mathbf{p}$ and $\mathbf{q}$ have R elements and their dimensions are reduced with every discarded singular value of $\mathbf{\Sigma}$, even though the dimensions of $\mathbf{x}$ and $\mathbf{z}^{(2)}$ remain the same. It is now possible to apply the process of backpropagation to the MLP-SVD network in a similar way to that described in Sections II B and II C. The gradients of the loss function with respect to the

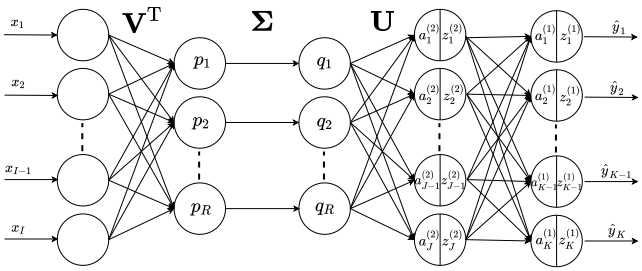


FIG. 2. MLP-SVD architecture with the $\mathbf{U}, \mathbf{\Sigma}$ and $\mathbf{V}^{\mathrm{T}}$ matrices between the input and the hidden layer. There are R connections between the vectors $\mathbf{p}$ and $\mathbf{q}$ each weighted by a singular value in the diagonal matrix $\mathbf{\Sigma}$.

matrices $\mathbf{U}$, $\mathbf{V}^{\mathrm{T}}$ and $\mathbf{\Sigma}$ can be evaluated by using an identical approach to that used above. Since the forward propagation between the hidden and the output layer remains the same as for the general MLP, the gradients of $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ are unchanged and are given by Eq. (24) and (25).

Turning to the gradients with respect to the elements of the matrices comprising the SVD, the gradient of the loss function with respect to the elements of the matrix $\mathbf{U}$ in vectorized form where $\mathbf{u} = vec(\mathbf{U})$ is given by:

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{u}} = \frac{\partial L_{\mathrm{E}}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{u}}. \tag{57}$$

The first two terms of this equation have been evaluated above so that

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{u}} = \mathbf{d}^{\mathrm{T}} \mathbf{H}^{(1)} \mathbf{W}^{(1)} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{u}}. \tag{58}$$

Further application of the chain rule then yields

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{u}} = \mathbf{d}^{\mathrm{T}} \mathbf{H}^{(1)} \mathbf{W}^{(1)} \frac{\partial h(\mathbf{a}^{(2)})}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{u}}. \tag{59}$$

Since the first of the partial derivatives is simply the diagonal matrix $\mathbf{H}^{(2)}$ of derivatives of activation functions and using the expression above for $\mathbf{a}^{(2)}$ shows that

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{u}} = \mathbf{d}^{\mathrm{T}}\mathbf{H}^{(1)}\mathbf{W}^{(1)}\mathbf{H}^{(2)}\frac{\partial}{\partial \mathbf{u}}\left(\mathbf{IUq}\right). \tag{60}$$

Application of Roth's theorem to the product $\mathbf{IUq}$ then shows that

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{u}} = \mathbf{d}^{\mathrm{T}}\mathbf{H}^{(1)}\mathbf{W}^{(1)}\mathbf{H}^{(2)}\frac{\partial}{\partial \mathbf{u}}\left[\left(\mathbf{q}^{\mathrm{T}}\otimes\mathbf{I}\right)vec(\mathbf{U})\right], \tag{61}$$

and evaluating the partial derivative results in

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{u}} = \mathbf{d}^{\mathrm{T}}\mathbf{H}^{(1)}\mathbf{W}^{(1)}\mathbf{H}^{(2)}\left(\mathbf{q}^{\mathrm{T}}\otimes\mathbf{I}\right), \tag{62}$$

where the vector $\mathbf{q}$ is computed from $\mathbf{q} = \mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}\mathbf{x}$. Following the same approach, writing $\boldsymbol{\sigma} = vec(\mathbf{\Sigma})$ and $\mathbf{v} = vec(\mathbf{V})$ the vectors of partial derivatives $\partial L_{\mathrm{E}}/\partial\boldsymbol{\sigma}$ and $\partial L_{\mathrm{E}}/\partial\mathbf{v}$ can be shown to be

$$\frac{\partial L_{\mathrm{E}}}{\partial \boldsymbol{\sigma}} = \mathbf{d}^{\mathrm{T}}\mathbf{H}^{(1)}\mathbf{W}^{(1)}\mathbf{H}^{(2)}\mathbf{U}\left(\mathbf{p}^{\mathrm{T}}\otimes\mathbf{I}\right) \tag{63}$$

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{v}} = \mathbf{d}^{\mathrm{T}}\mathbf{H}^{(1)}\mathbf{W}^{(1)}\mathbf{H}^{(2)}\mathbf{U}\mathbf{\Sigma}\left(\mathbf{x}^{\mathrm{T}}\otimes\mathbf{I}\right), \tag{64}$$

where the vector $\mathbf{p}$ is computed from $\mathbf{p} = \mathbf{V}^{\mathrm{T}}\mathbf{x}$. The gradient with respect to the bias $\mathbf{b}^{(2)}$ is given by the same equation as in Eq. (37). It follows directly from these relationships that the matrix elements of the SVD can be updated by using the backpropagation techniques developed in Section II D above. This results in the weight matrix update equations given by $\partial L_{\mathrm{E}}/\partial\mathbf{U} = \boldsymbol{\delta}^{(2)}\mathbf{q}^{\mathrm{T}}$, $\partial L_{\mathrm{E}}/\partial\mathbf{\Sigma} = \mathbf{U}^{\mathrm{T}}\boldsymbol{\delta}^{(2)}\mathbf{p}^{\mathrm{T}}$ and $\partial L_{\mathrm{E}}/\partial\mathbf{V} = \mathbf{\Sigma}\mathbf{U}^{\mathrm{T}}\boldsymbol{\delta}^{(2)}\mathbf{x}^{\mathrm{T}}$ where exactly as in the case of the single hidden layer described above, $\boldsymbol{\delta}^{(2)} = \mathbf{H}^{(2)}\mathbf{W}^{(1)\mathrm{T}}\boldsymbol{\delta}^{(1)}$ where $\boldsymbol{\delta}^{(1)} = \mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}}\mathbf{d}$, and use has been made of the relationship $\mathbf{\Sigma} = \mathbf{\Sigma}^{\mathrm{T}}$ since it is a diagonal matrix.

Replacing the matrix $\mathbf{W}^{(2)}$ with the three matrices $\mathbf{U}, \mathbf{\Sigma}$ and $\mathbf{V}^{\mathrm{T}}$ actually results in a higher

number of weight elements that have to be trained. However, discarding singular values also results in a reduction in the number of columns of $\mathbf{U}$ and rows of $\mathbf{V}^{\mathrm{T}}$ which then results in a steep decrease of number of weight elements to be trained. Figure 3 shows the reduction of the number of weights to be trained in the component SVD matrices as singular values are discarded. It is important to note that without discarding singular values, the number of weights that need to be trained using the $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^{\mathrm{T}}$ matrices is higher than simply using $\mathbf{W}^{(2)}$ alone. However, by discarding some of the singular values, the number of weights to train decreases dramatically as shown in Figure 3. For the examples shown for weight matrices $\mathbf{W}^{(2)}$ having (i) 10240 (ii) 5120 and (iii) 2560 elements, the number of weight elements to be trained reduces after discarding only (i) 2 (ii) 4 and (iii) 6 singular values respectively. It is clear from Figure 3 that the greatest influence of discarding is when weight matrix $\mathbf{W}^{(2)}$ initially contains more values. The dotted lines in Figure 3 represent the approach discussed in Cai *et al.* (2014) where the two matrices $\mathbf{\Sigma}$ and $\mathbf{V}^{\mathrm{T}}$ were multiplied to form a single weight matrix. It can be seen that the number of weight elements to be trained in this case is slightly lower, although the difference is not significant.

III. APPLICATIONS OF MLP-SVD

A. Implementation of the algorithm

The MLP-SVD algorithm has been developed with the aim of reducing significantly the training time of the MLP whilst minimising any reduction in the test (generalization) accuracy. The algorithm provides a method for tailoring the dimensions of the weight

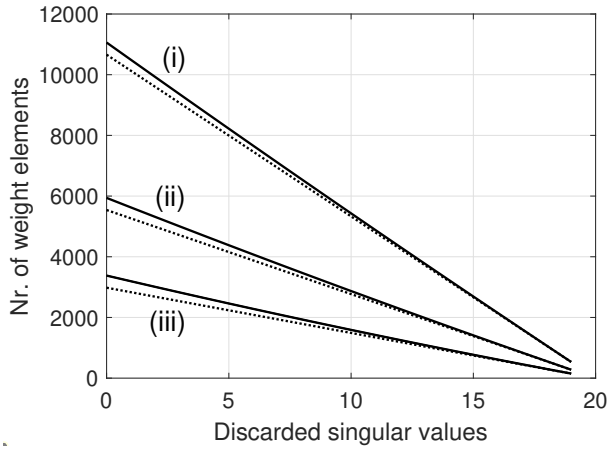


FIG. 3. The reduction in the number of weight elements to be trained as singular values are discarded for a weight matrix $\mathbf{W}^{(2)}$ with a total number of weight elements: (i) 512x20 = 10240, (ii) 256x20 = 5120 and (iii) 128x20 = 2560. The solid lines shows the number of weight elements when all three matrices in the SVD are trained and the dotted lines when these are reduced to two.

matrices to most efficiently solve the problem presented to the network. The implementation and its use is based on previous observations of the behaviour of singular values of weight matrices during MLP training (Paul and Nelson, 2020). This showed that, during network training, the singular values of the weight matrix evolved over time and that a number of singular values would become dominant in magnitude, depending on the difficulty of the classification task. An example of the behaviour of singular values of the weight matrix $\mathbf{W}^{(2)}$ during training of a multiclass classification problem is shown in Figure 4. The general MLP was trained to classify five different classes. Each of the five classes corresponded to a different time averaged acoustic spectrum generated by passing white noise through bandpass filters of different centre frequencies, as described in more detail below. It can

be observed that in this case, around 5 singular values became dominant during the weight updates. The number of dominant singular values is different depending on the difficulty of the task (see Paul and Nelson, 2020). The algorithm presented here is based on the

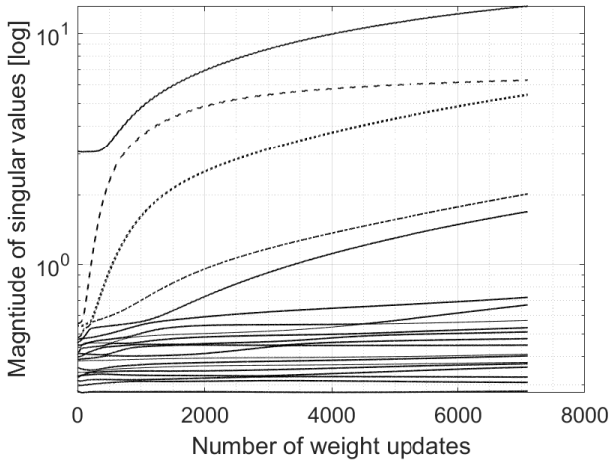


FIG. 4. Behaviour of singular values of the weight matrix between input and hidden layer during training for a classification task with five output classes, each class comprising a different acoustic spectrum. The number of neurons in the hidden layer was set to 20 and thus $\mathbf{W}^{(2)}$ had 20 singular values. The input layer had 256 neurons and the output had 5 neurons, corresponding to the 5 classes. The network was trained using the Adam optimizer (see Kingma and Ba, 2015).

progressive discarding of the smallest singular values during training. This is achieved using two parameters; a discarding threshold and specific times during training at which singular values are discarded. The threshold is defined to be a fraction of the highest singular value at each discarding point. The discarding points in time are defined by a vector that specifies the number of iterations of the backpropagation algorithm at which singular values are discarded. It should also be made clear that training continues as usual (by backpropagation) between

the points in time at which singular values are discarded. The three matrices after each discarding point are defined as $\mathbf{U}_\mathrm{D}, \mathbf{\Sigma}_\mathrm{D}, \mathbf{V}_\mathrm{D}^\mathrm{T}$, which correspond to the truncated versions of $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^\mathrm{T}$. The network trains with a new architecture after every discarding point, such that the component matrices $\mathbf{U}_\mathrm{D}, \mathbf{\Sigma}_\mathrm{D}, \mathbf{V}_\mathrm{D}^\mathrm{T}$ are updated at each iteration using the gradient expressions defined in the equations above. The updated matrices are defined as $\hat{\mathbf{U}}_\mathrm{D}, \hat{\mathbf{\Sigma}}_\mathrm{D}, \hat{\mathbf{V}}_\mathrm{D}^\mathrm{T}$. It should also be noted that during the update process these matrices then cease to be an exact SVD of the weight matrix. At the discarding points, the weight matrix is reassembled from $\mathbf{W}^{(2)} = \hat{\mathbf{U}}_\mathrm{D}\hat{\mathbf{\Sigma}}_\mathrm{D}\hat{\mathbf{V}}_\mathrm{D}^\mathrm{T}$ and a new SVD is undertaken before the singular values below the threshold are discarded.

Both key parameters (singular value threshold and discarding points) are defined based on initial simulations and previous observations (Paul and Nelson, 2020). During MLP training, the cost function usually has the steepest decrease during the initial iterations. Based on the observations in Cai *et al.* (2014) and some initial simulations with different lower bounds, results indicate that the MLP needs an initial number of updates to train the weight matrix, before the SVD matrices can contain useful information. The largest changes in the weight matrices happen during the first iterations and the singular values also tend to change rapidly. It is helpful to start discarding singular values as soon as possible to reduce the dimensions of the weight matrices. The sequence used to define the discarding points has been defined logarithmically, with more frequent discarding points towards the start of training and reducing frequency as training progresses. The number of iterations that determine the discarding points can be computed from

$$d(n) = d(n-1) \cdot 10^{(b-a)/(N-1)}, \tag{65}$$

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. In the simulations presented here, the lower bound of the sequence is defined to be 5 iterations and the higher bound is 67 iterations, or roughly 2/3 of the total number of iterations. Thus the values of a and b are given by $a = \log_{10}(5)$ and $b = \log_{10}(67)$. Being a logarithmic sequence, the vector of discarding points contains more values close to the lower bound and the spacing of discarding points is increasing while getting closer to the higher bound. At every discarding point, the algorithm checks if any singular values are lower than the threshold and if this is not the case, the architecture is not changed.

B. Description of the classification task

This section describes the classification task used both as an example of network training and to evaluate the performance of the MLP-SVD algorithm. The goal is to investigate how effectively the MLP might be able to differentiate between acoustic spectra that might not be easily distinguishable from one another by using conventional power spectral analysis. The training data used in these model problems is synthesised by generating white noise signals in the time domain that are passed through bandpass filters having different centre frequencies and bandwidths. The difference equation of the second-order bandpass filter is presented in Lane (1990) and is expressed as

$$y(n) = 2\alpha[x(n) - x(n-2)] + \gamma y(n-1) - \beta y(n-2). \tag{66}$$

The parameters α, γ, β can be computed using the centre frequencies, the Q-factor and the sampling frequency f_s. Figures 5(a) and 5(b) show the frequency response function of the bandpass filters used to generate the training data. Figure 5(a) shows a series of ten "narrow band" filters having bandwidth $B = 10$ Hz at centre frequencies of between 710 Hz and 440 Hz with a spacing between centre frequencies of $\Delta f = 30$ Hz. The bandwidth is defined as the difference between the upper and the lower frequency points at a 3 dB reduction in the gain of the filter. Similarly, Figure 5(b) shows ten frequency response functions of "broad band" filters having a bandwidth $B = 100$ Hz with a separation $\Delta f = 30$ Hz. Depending on

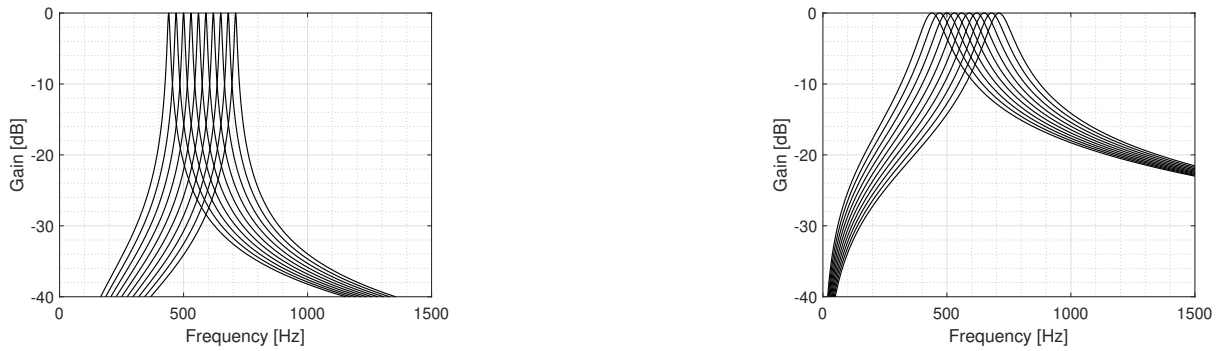


FIG. 5. Frequency response of 10 bandpass filters with centre frequencies between 710 Hz and 440 Hz, a spacing between centre frequencies of 30 Hz and a bandwidth of: (a) 10 Hz and (b) 100 Hz.

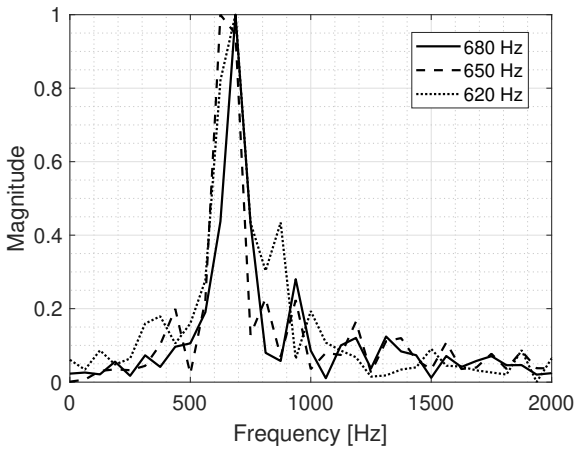
the classification task attempted, a database of time histories can be built from the outputs of the filters. These time histories, of duration 300 ms at a sampling frequency of 16 kHz are then transformed into the frequency domain using an FFT size of 256. The network is trained by presenting the magnitude of the FFT to the input nodes of the MLP and

comparing the estimated output to a target output where each class is mapped to a binary vector. For example, if there are 5 labels to classify, the target output for the first class would have the form $\mathbf{y}^{\mathrm{T}} = [1, 0, 0, 0, 0]$. The target output for the second class would be $\mathbf{y}^{\mathrm{T}} = [0, 1, 0, 0, 0]$ and so forth. The network is then trained to estimate an output that is as close as possible to the defined targets. As an example of the raw spectra presented to the input nodes of the MLP, Figure 6 shows a comparison between a single signal and 1000 averaged signals of the database from 3 filters separated by $\Delta f = 30$ Hz having centre frequencies between 620 Hz and 680 Hz. Not surprisingly, since the spectral resolution of the FFT in this case is 62.5 Hz, the differences in the spectra are difficult to detect by inspection, especially in the case of the FFT based on a single time history sample.

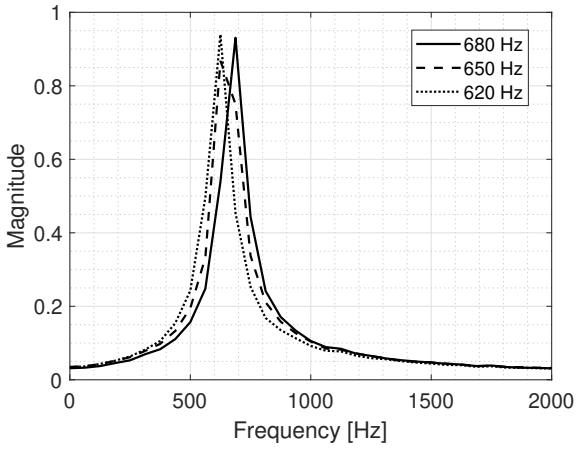
IV. RESULTS AND DISCUSSION

A. Network parameters used in evaluating performance

In order to enable a fair comparison between the standard MLP and the MLP-SVD network architectures, all initial parameters were chosen to be the same. The training database was generated with 1000 signals from each class of bandpass filtered white noise and both training and testing datasets were shuffled before starting the training. Each sample of time history generated from the bandpass filters was then converted into the frequency domain using an FFT before presenting the magnitude spectra to the network. Note that no phase information was used in these studies and the possibility of either using both real and imaginary parts of the FFT and/or a complex valued neural network (CVNN) remains



(a)



(b)

FIG. 6. Spectrum of: (a) a single 256 point sample of filter output time history for three filter centre frequencies: 680 Hz, 650 Hz and 620 Hz and (b) 1000 averaged 256 samples of filter output time history for the same filter centre frequencies.

to be investigated. A good review of the use of CVNNs and related issues is presented by Bassey *et al.* (2021). The FFT size was chosen to be 256 and the input into the network architecture was half of the FFT size plus one, because of the mirrored frequency spectrum. Both MLP and MLP-SVD networks have one hidden layer. The number of output nodes in the network corresponds to the number of classes of bandpass filtered white noise. The

number of neurons in the hidden layer is set to 20 for all simulations, unless otherwise specified. The learning rate was chosen to be $\eta = 0.0005$, since this value was shown in initial simulations to be suitable for these applications.

Both MLP and MLP-SVD network are trained using the Adam optimizer (Kingma and Ba, 2015), which is a variant of the well established stochastic gradient descent algorithm. The Adam optimizer uses "momentum" information to adapt the learning rate based on the behaviour of the training. In our initial simulations, this was shown to help the network converge faster. For all simulations, a batch size of 64 was used, which means that the network is presented with 64 input signals from the database, the gradients for each weight in the network are computed for each input signal, and then the average gradient for each weight is computed before updating the weights. After the weights are updated using the averaged gradients, the next batch of 64 signals is sent into the network. A training iteration (also known as a single epoch) ends after all 1000 signals from the database, divided into batches of 64, are presented to the network. The use of batches influences the training time and also the rate of learning. Since the error is averaged over the number of signals in the batch, if the batch is very small and the weights are updated more times during one iteration, the cost function will be "noisier", and this helps regularise the problem and enhances the generalization accuracy. Too small a batch size results in a longer training time, since the weights are updated too frequently. Similarly, a large batch size will result in faster training time, but with less noisy gradients and therefore a poorer generalization performance. More about the batch size implementation and its influence on the training of neural networks can

be found in Masters and Luschi (2018). The training of the MLP and MLP-SVD networks was stopped after 100 iterations, since initial simulations showed that a longer training for these applications will almost always result in overfitting (Hawkins, 2004). In simple terms, the occurrence of overfitting means that the network learns the training features too well, but can no longer generalize effectively, resulting in a decrease in network performance.

The MLP-SVD network has two additional initial parameters that need to be defined. The first one is the threshold for discarding the singular values, which based on initial simulations was defined to be 0.1 times the value of the largest singular value at that particular iteration. A higher threshold usually results in a discarding of too many singular values at one time, which in some cases can cause training difficulties. A lower threshold value results in a longer training time, since not enough singular values are discarded at each discarding point. The logarithmic sequence of discarding points is the second parameter and as described above, the best lower bound in terms of computational time and performance was found to be after 5 iterations whilst the higher bound was set to be 2/3 of the maximum number of iterations. This was found to be sufficient to allow the MLP-SVD network to converge.

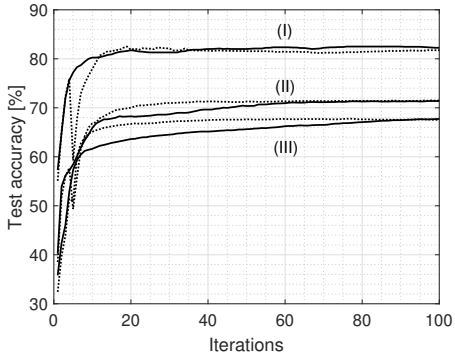
B. Comparison between MLP and MLP-SVD algorithms

For the comparison of the two algorithms, six different simulations will be used. The number of output classes will be changed, so that the neural networks will be trained for tasks of six different difficulties. The classes used are for "narrow band" spectra and "broad

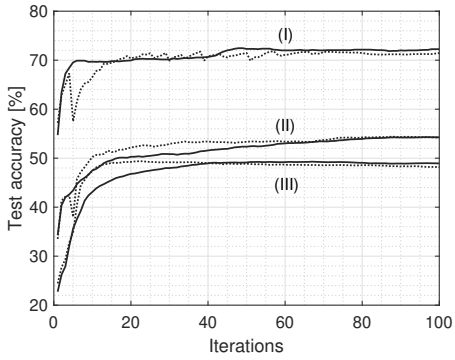
band" spectra where in each case either 2, 5 or 10 spectra are used. The results presented are averaged over 10 trials, meaning that for each set of parameters, both networks were trained 10 times. This was undertaken since the network initializes the weights with random numbers and therefore the training performance differs slightly between trials. It was found that 10 trials were sufficient to be establish a reliable average, since further trials were not found to enhance the accuracy or the training time to any significant degree.

Figures 7(a) and 7(b) illustrate the comparison between the MLP and the MLP-SVD algorithms in terms of test accuracy as a function of time (iterations) for 3 output classes (2, 5, 10) and two different bandwidths $B = 10$ Hz, $B = 100$ Hz.

The test accuracy is defined as the accuracy computed by presenting to the network a test dataset that is different from the training dataset. The accuracy can be calculated from the confusion matrix, which corresponds to the error matrix in classification problems (Stehman, 1997). Each of the columns in the confusion matrix contains the total number of signals that were estimated to be in one class, while each row corresponds to the number of signals presented from one class. This allows evaluation of the number of target signals from one class that are correctly estimated by the network. An example of the confusion matrix for 3 different bandpass filtered white noises having centre frequencies $f_{c_1}, f_{c_2}, f_{c_3}$ is illustrated in Table I, where the bold underlined values represent the number of correct predictions for each class and the rest of the cells correspond to errors. The accuracy can be computed by summing up the correct estimations and dividing by the total number of signals. For the example above, this would be $(85 + 60 + 80)/(270) = 0.83$ or 83 percent.



(a)



(b)

FIG. 7. Comparison between the MLP (solid line) and the MLP-SVD (dotted line) performance for (i) 2 output classes; (ii) 5 output classes; (iii) 10 output classes with a bandpass filter bandwidth of: (a) $B = 10$ Hz and (b) $B = 100$ Hz.

Note that even though the spectral resolution is 62.5 Hz, both network architectures are able to make a distinction between the different filter outputs, where the difference between the centre frequency of the signals is $\Delta f = 30$ Hz. In the case where the bandwidth of the bandpass filter is increased to 100 Hz, it can be observed that the performance of both networks is lower, since the signals are harder to distinguish. As expected, the generalization accuracy decreases in both cases if there are more output classes. Jumps in the test accuracy of the MLP-SVD algorithm during training sometimes appear at the singular

TABLE I. Example of a confusion matrix for 3 classes of different centre frequencies

S = 270	Estimated f_{c_1}	Estimated f_{c_2}	Estimated f_{c_3}
Target f_{c_1}	**85**	5	0
Target f_{c_2}	15	**60**	15
Target f_{c_3}	2	8	**80**

value discarding points. For all three applications, the biggest jump appears at iteration 5, since that is the point where the algorithm first discards singular values and the network needs some time to train with the new architecture. Whilst the curves illustrate an average over 10 test trials, there are no other large deviations during training, even though singular values are being discarded until 2/3 of the total number of iterations.

Table II compares, for both choices of bandwidth, the final test accuracy, the number of singular values remaining at the end of training, and the total computational time. Since for every trial the weights are initialised randomly, the final number of singular values remaining can differ slightly. This can be seen in Table II, where for the $B = 10$ Hz case, the number of singular values remaining for the easy task (2 output classes) is only 2 or 3, where most trials end with 2 singular values. If the difficulty is increased (5 output classes), the 10 trials end with 2 or 3 singular values, however most trials end with 3 singular values. For the hard task (10 output classes) all 10 trials ended with 4 singular values. However, for the case where the bandwidth of the bandpass filter is increased, there is no obvious increase in the number of remaining singular values. The regular MLP has a very slightly higher

accuracy than the MLP-SVD algorithm for almost all simulations, although this difference is not significant. The main conclusion from these results is that the MLP-SVD algorithm needs around 2/3 of the training time of the MLP algorithm to achieve almost the same accuracy.

TABLE II. Comparison between the MLP and the MLP-SVD for three different applications

Bandwidth		10 Hz		100 Hz	
		MLP	MLP-SVD	MLP	MLP-SVD
Test Accuracy [%]	2 classes	82.5	82	72.25	72
	5 classes	71.4	71.5	54.3	54.2
	10 classes	67.7	67.6	49	48.5
Singular values	2 classes	n/a	2-3	n/a	2-3
	5 classes	n/a	2-3	n/a	2-3
	10 classes	n/a	4	n/a	2-4
Training time [s]	2 classes	6	4	5.4	4.8
	5 classes	17	11	11.2	7
	10 classes	36.3	24.7	33.7	23.7

The results demonstrate that the number of discarded singular values plays a significant role in the training time and performance of the MLP-SVD algorithm. From some initial simulations it was observed that if the threshold is set to be too high, only few singular values are discarded during training and therefore the training time is not significantly reduced. On the other hand, if the threshold is set too low, too many singular values are discarded

at each stage and useful information is lost. This results in a reduction in the ability of the network to learn the patterns in the data presented.

C. Influence of filter bandwidth and hidden layer dimensions

Further simulations were undertaken to investigate the performance of the MLP-SVD for different values of Δf and for different hidden layer dimensions. In the first case, the difference between centre frequencies is changed, so that $\Delta f = 5, 30, 60$ Hz. The number of output classes is set to be 5 and the hidden layer is defined to have 20 neurons. Figure 8 illustrates the test accuracy of the MLP-SVD for the three tasks, averaged again over 10 trials. Even though the spectrum resolution is 62.5 Hz, the MLP-SVD can differentiate

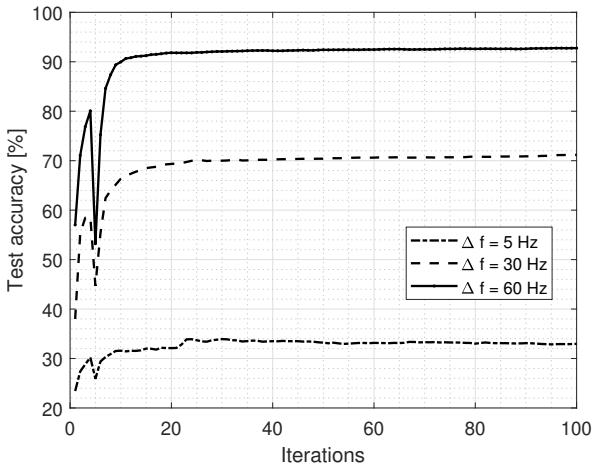
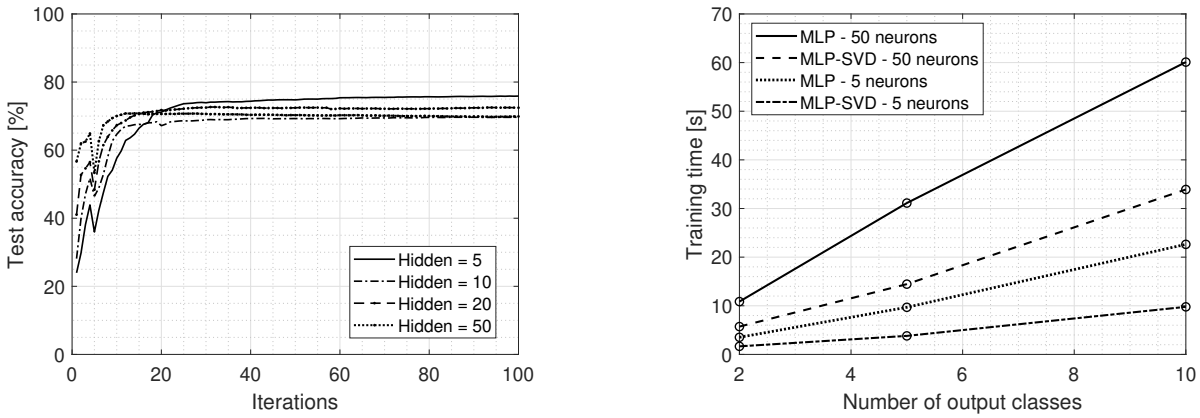


FIG. 8. Test accuracy of the MLP-SVD for five output classes with three different values of filter centre frequency separation Δf given by 5 Hz (dotted line) 30 Hz (dashed line) and 60 Hz (solid line).

between the 5 classes for both $\Delta f = 60$ Hz and $\Delta f = 30$ Hz with an accuracy of above

50%. As might be expected, the most difficult task where $\Delta f = 5$ Hz, cannot be successfully accomplished by the network with a test accuracy of only 30%.

The influence of the dimension of the hidden layer was investigated by using 5, 10, 20 and 50 neurons. The number of output classes was set to 5 and the centre frequencies of the bandpass filters are spaced by $\Delta f = 30$ Hz. Figure 9 illustrates the averaged results of the MLP-SVD for the four cases in the left plot, while comparing the reduction in training time for two of these cases in the right figure.



(a) (b)

FIG. 9. (a) Test accuracy of the MLP-SVD for different dimensions of the hidden layer and (b) training time saved depending on the dimensions of the hidden layer

The results show that an increased dimension in the hidden layer does not necessarily increase the network performance, but helps it to achieve a high test accuracy faster. A more detailed discussion about the influence of the hidden layer on the performance is presented in Paul and Nelson (2020). The right figure shows that a larger hidden layer dimension (50 neurons) increases the training time regardless of the applications, since more parameters

need to be trained. It is important to note that even with an increased number of neurons in the hidden layer, the same number of singular values remain after the discarding process. Thus the greater the number of neurons in the hidden layer, the greater the number of singular values discarded during training, and greater is the saving in training time. This is shown in the right figure, where for a hidden layer of 50 neurons, the time saved for any of the three cases is higher than for a network with 5 neurons in the hidden layer.

V. CONCLUSIONS

This paper has presented a detailed analysis in matrix form of the multilayer perceptron (MLP) and the associated backpropagation algorithm. The approach taken enables the application of the singular value decomposition (SVD) of the weight matrices in the MLP and shows how the component matrices of the SVD can also be updated by backpropagation. A training algorithm, the MLP-SVD algorithm, has been introduced that is based on the sequential discarding of singular values during training. The algorithm has been successfully applied to some model audio signal classification tasks and compared in performance with the standard MLP algorithm. It was shown that for the classification problems considered here, the technique can reduce training time typically by 1/2 and still achieve a similar classification accuracy. The performance of the MLP-SVD approach will be problem-dependent and requires the parameters of the algorithm to be chosen accordingly. However, the method seems well-suited to acoustical problems where the training data, such as acoustic spectra, has high dimensions. It is also likely that this technique can be successfully applied to other neural network architectures.

603 **ACKNOWLEDGMENTS**

604 V.S. Paul is grateful for the support of the UK Engineering and Physical Sciences Research
605 Council through the Doctoral Training Partnership.

606 **APPENDIX:**

607 The gradient of the loss function with respect to the weights in the output layer is given
608 by the row vector

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}^{(1)}} = \mathbf{d}^{\mathrm{T}}\mathbf{H}_{\mathrm{E}}^{(1)}(\mathbf{z}^{(2)\mathrm{T}} \otimes \mathbf{I}^{(1)}), \tag{A.1}$$

609 This expression can be transposed to give the column vector

$$\left[\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}^{(1)}}\right]^{\mathrm{T}} = (\mathbf{z}^{(2)} \otimes \mathbf{I}^{(1)})\mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}}\mathbf{d}, \tag{A.2}$$

610 where the identities $(\mathbf{ABC})^{\mathrm{T}} = \mathbf{C}^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}$ and $(\mathbf{A} \otimes \mathbf{B})^{\mathrm{T}} = \mathbf{A}^{\mathrm{T}} \otimes \mathbf{B}^{\mathrm{T}}$ have been used.
611 Note that the term involving the Kronecker product can be written as

$$\mathbf{z}^{(2)} \otimes \mathbf{I}^{(1)} = \begin{bmatrix} z_1^{(2)}\mathbf{I}^{(2)} \\ z_2^{(2)}\mathbf{I}^{(2)} \\ \vdots \\ z_J^{(2)}\mathbf{I}^{(2)} \end{bmatrix} \tag{A.3}$$

612 and the weight matrix $\mathbf{W}^{(1)}$ can be written in terms of its J component column vectors

$$\mathbf{W}^{(1)} = \begin{bmatrix} \mathbf{w}_1^{(1)} & \mathbf{w}_2^{(1)} & \dots & \mathbf{w}_J^{(1)} \end{bmatrix}, \tag{A.4}$$

where each column vector has K values. The transposed gradient expression can thus be written as

$$\begin{bmatrix} \\ \frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}_1^{(1)}} \\ \\ \frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}_2^{(1)}} \\ \\ \vdots \\ \\ \frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}_J^{(1)}} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} z_1^{(2)}\mathbf{I}^{(1)}\mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}}\mathbf{d} \\ z_2^{(2)}\mathbf{I}^{(1)}\mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}}\mathbf{d} \\ \vdots \\ z_J^{(2)}\mathbf{I}^{(1)}\mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}}\mathbf{d} \end{bmatrix}, \tag{A.5}$$

where each element $\partial L_{\mathrm{E}}/\partial \mathbf{w}_j^{(1)}$ is a Kx1 row vector. Since $z_1\mathbf{I}^{(1)}\mathbf{H}^{(1)} = z_1\mathbf{H}^{(1)}$ the above expression can be rewritten as the matrix given by

$$\left[\left(\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}_1^{(1)}}\right)^{\mathrm{T}} \left(\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}_2^{(1)}}\right)^{\mathrm{T}} \ \ldots \ \left(\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}_J^{(1)}}\right)^{\mathrm{T}} \right] = \left[z_1^{(2)}\mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}}\mathbf{d} \ \ z_2^{(2)}\mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}}\mathbf{d} \ \ldots \ z_J^{(2)}\mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}}\mathbf{d} \right]. \tag{A.6}$$

The matrix on the left hand side is now in the form of the required matrix of derivatives of the loss function with respect to the matrix $\mathbf{W}^{(1)}$. The right hand side can be written in terms of the transpose of the vector $\mathbf{z}^{(2)}$ and therefore the above expression can be written as

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{W}^{(1)}} = \mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}}\mathbf{d}\mathbf{z}^{(2)\mathrm{T}}. \tag{A.7}$$

In order to compare the results of the current analysis with classical approaches to back-propagation (see Bishop, 1995 p. 146; Rumelhart *et al.*, 1985, 1986), it is helpful to consider the case where the loss function is the sum of squared errors and the matrix $\mathbf{H}_{\mathrm{E}}^{(1)}$ is replaced by the diagonal matrix $\mathbf{H}^{(1)}$. The diagonal terms of this matrix are given by $h'(a_k^{(1)})$ and can be conveniently written as $h_k'^{(1)}$. Using this abbreviation, the element-wise multiplication of

terms in the above matrices results in

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{W}^{(1)}} = \begin{bmatrix} h_1'^{(1)} d_1 z_1^{(2)} & h_1'^{(1)} d_1 z_2^{(2)} & \dots & h_1'^{(1)} d_1 z_J^{(2)} \\ h_2'^{(1)} d_2 z_1^{(2)} & h_2'^{(1)} d_2 z_2^{(2)} & \dots & h_2'^{(1)} d_2 z_J^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ h_K'^{(1)} d_K z_1^{(2)} & h_K'^{(1)} d_K z_2^{(2)} & \dots & h_K'^{(1)} d_K z_J^{(2)} \end{bmatrix}. \tag{A.8}$$

The individual terms in this matrix can be clearly identified as corresponding to those resulting from classical analyses (see Bishop, 1995 p. 146; Rumelhart *et al.*, 1985, 1986). Now turning to the weights in the hidden layer, the transpose of the gradient equation in this case can be written as

$$\left[\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{w}^{(2)}}\right]^{\mathrm{T}} = (\mathbf{x} \otimes \mathbf{I}^{(2)}) \mathbf{H}^{(2)} \mathbf{W}^{(1)\mathrm{T}} \mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}} \mathbf{d}. \tag{A.9}$$

Using an identical approach to that used for the output layer above, the matrix of the derivatives of the loss function with respect to the weights of the matrix $\mathbf{W}^{(2)}$ can be written as

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{W}^{(2)}} = \mathbf{H}^{(2)} \mathbf{W}^{(1)\mathrm{T}} \mathbf{H}_{\mathrm{E}}^{(1)\mathrm{T}} \mathbf{d}\mathbf{x}^{\mathrm{T}}. \tag{A.10}$$

Again, if the loss function is considered to be the sum of squared errors and the matrix $\mathbf{H}_{\mathrm{E}}^{(1)}$ is replaced by the diagonal matrix $\mathbf{H}^{(1)}$, and the matrix $\mathbf{H}^{(2)}$ has diagonal elements written as $h_k'^{(2)}$, then the element-wise multiplication of this matrix product results in

$$\frac{\partial L_{\mathrm{E}}}{\partial \mathbf{W}^{(2)}} = \begin{bmatrix} h_1'^{(2)} \sum_{k=1}^{K} w_{k1}^{(1)} h_k'^{(1)} d_k x_1 & h_1'^{(2)} \sum_{k=1}^{K} w_{k1}^{(1)} h_k'^{(1)} d_k x_2 & \dots & h_1'^{(2)} \sum_{k=1}^{K} w_{k1}^{(1)} h_k'^{(1)} d_k x_I \\ h_2'^{(2)} \sum_{k=1}^{K} w_{k2}^{(1)} h_k'^{(1)} d_k x_1 & h_2'^{(2)} \sum_{k=1}^{K} w_{k2}^{(1)} h_k'^{(1)} d_k x_2 & \dots & h_2'^{(2)} \sum_{k=1}^{K} w_{k2}^{(1)} h_k'^{(1)} d_k x_I \\ & \vdots & & \\ h_J'^{(2)} \sum_{k=1}^{K} w_{kJ}^{(1)} h_k'^{(1)} d_k x_1 & h_J'^{(2)} \sum_{k=1}^{K} w_{kJ}^{(1)} h_k'^{(1)} d_k x_2 & \dots & h_J'^{(2)} \sum_{k=1}^{K} w_{kJ}^{(1)} h_k'^{(1)} d_k x_I \end{bmatrix}. \tag{A.11}$$

637 This matrix of gradients again corresponds to the element-wise equations derived from clas-
638 sical analyses (see Bishop, 1995 p. 146; Rumelhart *et al.*, 1985, 1986).

639

640 Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S.,
641 Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard,
642 M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga,
643 R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I.,
644 Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden,
645 P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (**2015**). "TensorFlow: Large-scale
646 machine learning on heterogeneous systems" https://www.tensorflow.org/, software
647 available from tensorflow.org.

648 Aggarwal, C. C. (**2018**). *Neural Networks and Deep Learning: A Textbook*, 1st ed. (Springer
649 Publishing Company, Incorporated, Gewerbestrasse 11, 6330 Cham, Switzerland), pp. 1–
650 497.

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

653 Bermeitinger, B., Hrycej, T., and Handschuh, S. (**2019**). "Singular value decomposition
654 and neural networks," in *Artificial Neural Networks and Machine Learning - ICANN*
655 *2019: Deep Learning - 28th International Conference on Artificial Neural Networks,*
656 *Munich, Germany, September 17-19, 2019, Proceedings, Part II*, Springer, Berlin, Vol.
657 11728, pp. 153–164, https://doi.org/10.1007/978-3-030-30484-3_13, doi: 10.1007/

978-3-030-30484-3_13.

Bianco, M. J., Gerstoft, P., Traer, J., Ozanich, E., Roch, M. A., Gannot, S., and Deledalle, C.-A. (**2019**). "Machine learning in acoustics: Theory and applications," The Journal of the Acoustical Society of America **146**(5), 3590–3628.

Bishop, C. M. (**1995**). *Neural Networks for Pattern Recognition* (CLARENDON PRESS, OXFORD), pp. 1–482.

Bishop, C. M. (**2006**). *Pattern recognition and machine learning* (Springer, 233 Spring Street, New York, NY 10013, USA), pp. 1–738.

Bridle, J. S. (**1990**). "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing* (Springer), pp. 227–236.

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, Gold Coast, QLD, Australia, December 1-5, 2014. Proceedings*, Springer, Berlin, Vol. 8862, pp. 820–826, https://doi.org/10.1007/978-3-319-13560-1_65, doi: 10.1007/978-3-319-13560-1_65.

Caswell, H. (**2019**). "Matrix calculus and notation," in *Sensitivity Analysis: Matrix Methods in Demography and Ecology* (Springer Nature, London, United Kingdom), pp. 13–28.

Cauchy, A. (**1847**). "Méthode générale pour la résolution des systemes d'équations simultanées [general method for solving systems of simultaneous equations]," Comp. Rend. Sci. Paris **25**(1847), 536–538.

Choudhary, T., Mishra, V., Goswami, A., and Sarangapani, J. (**2020**). "A comprehensive survey on model compression and acceleration," Artificial Intelligence Review 1–43.

Eckart, C., and Young, G. (**1936**). "The approximation of one matrix by another of lower rank," Psychometrika **1**(3), 211–218.

Géron, A. (**2019**). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems* (O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472), pp. 1–541.

Golub, G. H., and van Loan, C. F. (**2013**). *Matrix Computations*, fourth ed. (JHU Press, Baltimore, Maryland, United States), pp. 1–756.

Goodfellow, I., Bengio, Y., and Courville, A. (**2016**). *Deep Learning* (The MIT Press, Cambridge, United States), pp. 1–773.

Graves, A. (**2012**). *Supervised Sequence Labelling with Recurrent Neural Networks* (Springer, Berlin), pp. 1–131.

Hawkins, D. M. (**2004**). "The problem of overfitting," Journal of chemical information and computer sciences **44**(1), 1–12.

Hochreiter, S., and Schmidhuber, J. (**1997**). "Long short-term memory," Neural computation **9**(8), 1735–1780.

Kingma, D. P., and Ba, J. (**2015**). "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, http://arxiv.org/abs/1412.6980.

Klema, V., and Laub, A. (**1980**). "The singular value decomposition: Its computation and some applications," IEEE Transactions on automatic control **25**(2), 164–176.

Lane, J. E. (**1990**). "Pitch detection using a tunable iir filter," Computer Music Journal **14**(3), 46–59.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (**1989**). "Backpropagation applied to handwritten zip code recognition," Neural computation **1**(4), 541–551.

Magnus, J. R. (**2010**). "On the concept of matrix derivative," Journal of Multivariate Analysis **101**(9), 2200–2206.

Magnus, J. R., and Neudecker, H. (**2019**). *Matrix Differential Calculus with Applications in Statistics and Econometrics* (WILEY; Third Edition, Chichester, United Kingdom), pp. 1–479.

Masters, D., and Luschi, C. (**2018**). "Revisiting small batch training for deep neural networks," arXiv preprint arXiv:1804.07612 .

Nielsen, M. A. (**2015**). *Neural networks and deep learning*, **25** (Determination press), pp. 1–216.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (**2017**). "Automatic differentiation in pytorch," .

Paul, V., and Nelson, P. A. (**2020**). "Matrix analysis of neural network architectures for audio signal classification," in *Proceedings of the Institute of Acoustics Vol. 42. Pt. 3.*, Milton Keynes, United Kingdom.

Roth, W. E. (**1934**). "On direct product matrices," Bulletin of the American Mathematical Society **40**(6), 461–468.

722 Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (**1985**). "Learning internal represen-
723 tations by error propagation," Technical Report.

724 Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (**1986**). "Learning representations by
725 back-propagating errors," Nature **323**(6088), 533–536.

726 Stehman, S. V. (**1997**). "Selecting and interpreting measures of thematic classification ac-
727 curacy," Remote sensing of Environment **62**(1), 77–89.

728 The MathWorks, I. (**2020**). *Deep Learning Toolbox*, Natick, Massachusetts, United State,
729 https://www.mathworks.com/solutions/deep-learning.html.

730 Xu, P. (**1998**). "Truncated svd methods for discrete linear ill-posed problems," Geophysical
731 Journal International **135**(2), 505–514.

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