MATRICES ANALYSIS OF NEURAL NETWORK ARCHITECTURES FOR AUDIO SIGNAL CLASSIFICATION

V. S. Paul Institute of Sound and Vibration Research, University of Southampton, UK
P. A. Nelson Institute of Sound and Vibration Research, University of Southampton, UK

1 INTRODUCTION

The development of computational power in recent years has enabled the use of large training datasets in machine learning. Training neural networks can still be very time consuming and this has led to the development of computationally efficient neural network implementations such as TensorFlow\(^1\), PyTorch\(^2\), Keras and Scikit-Learn\(^3\) or the MATLAB Deep Learning Toolbox\(^4\). With the availability of such powerful deep learning frameworks, experiments research in machine learning are often conducted using such software as “black boxes”, with little focus on the properties of the networks and their evolution during training. Understanding the mathematics of neural network models is a first step to better understanding how information propagates through a network architecture. Besides the original papers describing the different neural network architectures such as Multi-Layer Perceptrons\(^5\) (MLPs), Recurrent Neural Networks\(^6\)\(^7\) (RNNs), Long Short-Term Memory Networks\(^8\) (LSTMs) or Convolutional Neural Networks\(^9\) (CNNs), there are few accessible introductions that focus on the basic neural network models. There are also several papers and books that discuss the implementation of the underlying mathematics, although most of these publications either discuss only one network architecture in detail\(^10\)–\(^13\) or analyse the most popular network models, but without going into the mathematical details\(^14\)\(^15\). The main contribution of this paper is a derivation in matrix form of the forward and backward propagation equations for an MLP with any number of hidden layers. The analysis described can also be applied to other architectures. The equations derived will be implemented in MATLAB, and as an initial example of an application to audio signal classification, the network will be trained to distinguish between two closely related spectra. The singular value decomposition (SVD) is applied to the weight matrices during the weight update process to better understand the behaviour of the network. The initial results are discussed, and areas for future work proposed.

2 MATRIX ANALYSIS OF THE MULTI-LAYER PERCEPTRON

2.1 Notation and mathematical rules

To define some general notations, the input into a network is a vector \(\mathbf{x} \in \mathbb{R}^{D \times 1}\), where \(D\) corresponds to the number of neurons. The general notation describing the value of neurons in the hidden layer is \(\mathbf{z} \in \mathbb{R}^{M \times 1}\) and a superscript can be added (ex. \(\mathbf{z}^{(1)}\)) to differentiate between hidden layers. The values of the neurons in the output layer is denoted as \(\hat{\mathbf{y}} \in \mathbb{R}^{K \times 1}\), where \(\hat{\mathbf{y}}\) is the estimated output of the network. Both \(M\) and \(K\) correspond to the dimension of the particular layer. The weight matrices are defined as \(\mathbf{W}\) and the dimensions of the weight matrices depend on the dimensions of the layers that they link. A superscript can be added (ex. \(\mathbf{W}^{(l)}\)) to denote the layer number. The activation functions will be denoted as a general function \(h(\cdot)\). As a cost function, the cross-entropy will be used for the simulations in this paper.
Some mathematical definitions need to be described prior to the derivation of the backpropagation equations and, in particular, the distinction should be made between the “numerator” and the “denominator” convention\textsuperscript{16}. In this work the numerator convention is used and the gradient equations can be described in terms of dimensions as shown by Magnus\textsuperscript{17}. Thus the gradient of a scalar with respect to a $D$x$1$ vector results in a $1$x$D$ vector, the gradient of a scalar with respect to a $D$x$P$ matrix results in a $D$x$P$ matrix and the gradient of a $D$x$1$ vector with respect to a $P$x$1$ vector results in a $D$x$P$ Jacobian matrix. Finally, the gradient of a $D$x$1$ vector with respect to a $D$x$P$ matrix is technically a tensor, but the use of tensors is avoided here by transforming the matrix into a vector using the $vec(\cdot)$ operator\textsuperscript{17}. The transformation is computed by stacking the columns of the matrix under each other, thus forming a column vector. This trick can be used in combination with the theorem proposed by Roth\textsuperscript{18}, which describes the relation between three matrices $A \in \mathbb{R}^{Dx1}$, $B \in \mathbb{R}^{Px1}$ and $C \in \mathbb{R}^{Mx1}$ which states that

$$
vec(ABC) = (C^T \otimes A)vec(B),
$$

where $\otimes$ is the Kronecker product.

### 2.2 Multi-Layer Perceptrons

This section will describe the derivation of both forward and backward propagation for the MLP network, with emphasis on the gradient computation for the backpropagation. The equations will be presented while following the rules of matrix calculus and properties of the chain rule. The MLPs consist of an input layer $x \in \mathbb{R}^{Dx1}$, an output layer $\hat{y} \in \mathbb{R}^{Kx1}$ and one or several hidden layers $z$, each with individual number of layers having different dimensions. The MLP architecture with only one hidden layer is illustrated in Figure 1.

![Figure 1 – MLP model with one hidden layer](image)

The forward pass from the input to the output layer can be written in matrix form as

$$a^{(2)} = W^{(2)}x + b^{(2)}$$
$$z^{(2)} = h(a^{(2)})$$
$$a^{(1)} = W^{(1)}z^{(2)} + b^{(1)}$$
$$z^{(1)} = h(a^{(1)}) = \hat{y},$$

where $b^{(2)} \in \mathbb{R}^{Mx1}$ and $b^{(1)} \in \mathbb{R}^{Kx1}$ are the bias vectors for the particular layers and $h(\cdot)$ is a non-linear activation function, which is usually the sigmoid, the tanh, the ReLU, or the softmax function\textsuperscript{19}. It should be noted that the superscripts of the variables are used in an inverse order so that the backpropagation equations can be more easily generalized. Since the cost function used here is the cross-entropy, the error $E_C$ at the output of the MLP at the $k$-th neuron and the corresponding vector of errors $e_C$ are given by

Vol. 42. Pt. 3. 2020
\[ E_k = -y_k \log \hat{y}_k \]  
\[ e^T = [E_1 \ E_2 \ ... \ E_K] \]  

The backpropagation technique will now be explained for the network architecture shown in Figure 1. Consider the change of the error vector e with respect to the weight matrix \( W^{(1)} \) and bias vector \( b^{(1)} \), the partial derivative can be written using the chain rule as

\[
\frac{\partial e}{\partial w^{(1)}} = \frac{\partial e}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial w^{(1)}}
\]
\[
\frac{\partial e}{\partial b^{(1)}} = \frac{\partial e}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial b^{(1)}}
\]

where \( W^{(1)} \) has been transformed into a vector \( w^{(1)} \) by using the vec operator (i.e. \( w = vec(W) \)). The new vector is \( w^{(1)} \in \mathbb{R}^{K \times 1} \) and the definitions of the partial derivative of a vector with respect to another vector can be applied. The gradient with respect to the bias vector has a similar expression. Only the gradient with respect to the weights will be derived in what follows and only the final equation for the bias vector will be given here. The partial derivative of a \( K \times 1 \) vector with respect to a \( K \times 1 \) vector, using the numerator notation, results in a \( K \times K \times M \) matrix, also known as the Jacobian matrix.

The result of the first term in the chain rule in equation (8) will be a Jacobian matrix since it is the partial derivative of a vector with respect to another vector

\[
\frac{\partial e}{\partial z^{(1)}} = \begin{bmatrix}
\frac{\partial e_1}{\partial z_1^{(1)}} & \cdots & \frac{\partial e_1}{\partial z_K^{(1)}} \\
\frac{\partial e_2}{\partial z_1^{(1)}} & \cdots & \frac{\partial e_2}{\partial z_K^{(1)}} \\
\vdots & \ddots & \vdots \\
\frac{\partial e_K}{\partial z_1^{(1)}} & \cdots & \frac{\partial e_K}{\partial z_K^{(1)}}
\end{bmatrix}
\]

Considering only a single term, \( \frac{\partial e_k}{\partial z_k^{(1)}} \) for example, each term can be computed using the expressions from the cost function:

\[
\frac{\partial e_k}{\partial z_k^{(1)}} = \frac{\partial}{\partial z_k^{(1)}} (-y_k \log (z_k^{(1)})) = -\frac{y_k}{z_k^{(1)}} = \delta_k.
\]

Note that each value of e depends only on the corresponding value of \( z^{(1)} \) and therefore all elements of one row in the matrix are zero except those for which the subscript of e equals the subscript of \( z^{(1)} \). The matrix above can be simplified to give the diagonal matrix \( D \), where

\[
\frac{\partial e}{\partial z^{(1)}} = \begin{bmatrix}
\frac{\partial e_1}{\partial z_1^{(1)}} & \cdots & \frac{\partial e_1}{\partial z_K^{(1)}} \\
\frac{\partial e_2}{\partial z_1^{(1)}} & \cdots & \frac{\partial e_2}{\partial z_K^{(1)}} \\
\vdots & \ddots & \vdots \\
\frac{\partial e_K}{\partial z_1^{(1)}} & \cdots & \frac{\partial e_K}{\partial z_K^{(1)}}
\end{bmatrix} = D.
\]

The second term \( \frac{\partial z^{(1)}}{\partial a^{(1)}} \) can be calculated from the relationship \( z^{(1)} = h(a^{(1)}) \). There are two cases that need to be discussed here, depending on whether either the softmax function or the other non-linear activation functions are used. The softmax function is usually used for classification tasks in conjunction with the cross-entropy loss and is mathematically defined as
\[
\hat{y} = z^{(1)} = \text{softmax}(a^{(1)}) = \frac{\exp(a_i^{(1)})}{\sum_{k=1}^{K} \exp(a_k^{(1)})},
\]

where \( K \) is the total number of entries in the output layer \( a^{(1)} \). The function rescales a vector of values between 0 and 1, where all elements of that vector sum up to 1. It assigns a value closer to one to the largest component in the vector and pushes the lower values towards zero. The softmax function is often applied to classification tasks. The derivative of the output \( z^{(1)} \) with respect to the input vector into the softmax function \( a^{(1)} \) can be written for each entry of \( z_i^{(1)} \) and \( a_k^{(1)} \) as

\[
\frac{\partial z_i^{(1)}}{\partial a_k^{(1)}} = \frac{\partial}{\partial a_k^{(1)}} \left( \frac{e^{a_i^{(1)}}}{\sum_{k=1}^{K} e^{a_k^{(1)}}} \right),
\]

where \( i, k \) are the indices of the values in the vectors. Performing the differentiation of each term results in

\[
\frac{\partial z_i^{(1)}}{\partial a_k^{(1)}} = \begin{cases} 
z_k^{(1)}(1 - z_i^{(1)}) & \text{if } i = k \\
-z_k^{(1)}z_i^{(1)} & \text{otherwise.}
\end{cases}
\]

Using this, the final gradient term \( \frac{\partial z^{(1)}}{\partial a^{(1)}} \) is given by the Jacobian matrix

\[
\frac{\partial z^{(1)}}{\partial a^{(1)}} = \frac{\partial}{\partial a^{(1)}} \left( \frac{e^{a^{(1)}}}{\sum_{k=1}^{K} e^{a_k^{(1)}}} \right) = \begin{bmatrix}
z_1^{(1)}(1 - z_1^{(1)}) & -z_1^{(1)}z_2^{(1)} & \cdots & -z_1^{(1)}z_k^{(1)} \\
-z_2^{(1)}z_1^{(1)} & z_2^{(1)}(1 - z_2^{(1)}) & \cdots & -z_2^{(1)}z_k^{(1)} \\
\vdots & \vdots & \ddots & \vdots \\
-z_k^{(1)}z_1^{(1)} & z_k^{(1)}z_2^{(1)} & \cdots & z_k^{(1)}(1 - z_k^{(1)})
\end{bmatrix} = H_s^{(1)}
\]

If the sigmoid, tanh or ReLU functions are used instead of the softmax in the output layer, the general notation \( h(\cdot) \) can be used and the gradient results in a diagonal matrix \( H^{(1)} \), where

\[
\frac{\partial z^{(1)}}{\partial a^{(1)}} = \begin{bmatrix}
\frac{\partial z_1^{(1)}}{\partial a_1^{(1)}} & \cdots & \frac{\partial z_1^{(1)}}{\partial a_k^{(1)}} \\
\frac{\partial z_2^{(1)}}{\partial a_1^{(1)}} & \vdots & \ddots \\
\vdots & \ddots & \ddots \\
\frac{\partial z_k^{(1)}}{\partial a_1^{(1)}} & \cdots & \frac{\partial z_k^{(1)}}{\partial a_k^{(1)}}
\end{bmatrix} = \begin{bmatrix}
h'(a_1^{(1)}) & 0 & \cdots & 0 \\
0 & h'(a_2^{(1)}) & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & h'(a_k^{(1)})
\end{bmatrix} = H^{(1)},
\]

since any \( k \)-th element \( \frac{\partial z_k^{(1)}}{\partial a_k^{(1)}} \) can be written as

\[
\frac{\partial z_k^{(1)}}{\partial a_k^{(1)}} = \frac{\partial}{\partial a_k^{(1)}} h(a_k^{(1)}) = h'(a_k^{(1)}).
\]

It is important to note the differences between \( H_s^{(1)} \) and \( H^{(1)} \) when implementing the backpropagation. Since the softmax function considers all output neurons for every \( k \)-th output, \( H_s^{(1)} \) is not a diagonal matrix as in the case of \( H^{(1)} \). For the sake of a general gradient derivation, the general notation \( h(\cdot) \) will be used in what follows. To calculate the last term in the chain rule of equation (8), Roth’s theorem is used. Using the theorem to rewrite the forward propagation for \( a^{(1)} = W^{(1)}z^{(2)} \),
$$a^{(1)} = W^{(1)}z^{(2)} + b^{(1)} = I^{(1)}W^{(1)}z^{(2)} + b^{(1)}$$  \hspace{1cm} (19)$$

where $I^{(1)} \in R^{K \times K}$ is the identity matrix. Applying Roth's theorem,

$$\text{vec}(a^{(1)}) = (z^{(2)^T} \otimes I^{(1)}) \text{vec}(W^{(1)}) + b^{(1)}$$  \hspace{1cm} (20)$$

The vec operator transforms the matrix into a vector and therefore, the final expression is given by

$$a^{(1)} = (z^{(2)^T} \otimes I^{(1)})w^{(1)} + b^{(1)}.$$  \hspace{1cm} (21)$$

Using these expressions, the last term $\partial a^{(1)}/ \partial w^{(1)}$ in the chain rule can be simplified to

$$\frac{\partial a^{(1)}}{\partial w^{(1)}} = \frac{\partial}{\partial w^{(1)}}[(z^{(2)^T} \otimes I^{(1)})w^{(1)} + b^{(1)}] = z^{(2)^T} \otimes I^{(1)}.$$  \hspace{1cm} (22)$$

Finally, the gradient of the error with respect to the first set of weights is given by

$$\frac{\partial e}{\partial w^{(1)}} = DH^{(1)}(z^{(2)^T} \otimes I^{(1)}).$$  \hspace{1cm} (23)$$

By following a very similar procedure, the gradient of the error with respect to the bias vector $b^{(1)}$ is given by

$$\frac{\partial e}{\partial b^{(1)}} = DH^{(1)}I^{(1)}.$$  \hspace{1cm} (24)$$

In these expressions, $D$ is the diagonal matrix containing the errors $\delta_k$, $H^{(1)} \in R^{K \times K}$ is the diagonal matrix containing the derivatives of the non-linear activation functions $h'(\cdot)$ and $I^{(1)} \in R^{K \times K}$ is the identity matrix corresponding to the same dimensions as $H^{(1)}$. If one checks the dimensions, it can be seen that the matrix multiplications result in a matrix of expected dimensions.

Next, the influence of the weight matrix $W^{(2)}$ and bias vector $b^{(2)}$ can be analysed. The gradient of the error vector $e$ can be computed by extending the chain rule from above such that

$$\frac{\partial e}{\partial w^{(2)}} = \frac{\partial e}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial w^{(2)}}$$  \hspace{1cm} (25)$$

$$\frac{\partial e}{\partial b^{(2)}} = \frac{\partial e}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial b^{(2)}}.$$  \hspace{1cm} (26)$$

where $w^{(2)} = \text{vec}(W^{(2)})$. The first term of the chain rule has been calculated above and is the diagonal error matrix $D$. The second term is given by

$$\frac{\partial z^{(1)}}{\partial z^{(2)}} = \frac{\partial z^{(1)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial z^{(2)}}.$$  \hspace{1cm} (27)$$

The first term from above has already been computed in equation (17) and is denoted as $H^{(1)}$. The second fraction can be easily computed as

$$\frac{\partial a^{(1)}}{\partial z^{(2)}} = \frac{\partial}{\partial z^{(2)}}(W^{(1)}z^{(2)} + b^{(1)}) = W^{(1)}.$$  \hspace{1cm} (28)$$

Using the expression for both fractions, the second term of the gradient is given by

$$\frac{\partial z^{(1)}}{\partial z^{(2)}} = H^{(1)}W^{(1)}.$$  \hspace{1cm} (29)$$
The third term $\partial z^{(2)} / \partial w^{(2)}$ can be calculated from equation $z^{(2)} = h(W^{(2)}x + b^{(2)})$ and equation (17), where the output is a matrix of the derivatives of the non-linear activation functions. Therefore,

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

(30)

The first term is the $H^{(2)}$ matrix, based on the calculations from equation (17), and the second fraction is given by

$$\frac{\partial z^{(2)}}{\partial w^{(2)}} = H^{(2)} \frac{\partial}{\partial w^{(2)}} (I^{(2)}W^{(2)}x + b^{(2)}).$$

(31)

Again, applying Roth’s theorem, the fraction can be written as

$$\frac{\partial z^{(2)}}{\partial w^{(2)}} = H^{(2)} \frac{\partial}{\partial w^{(2)}} [(x^T \otimes I^{(2)}) \text{vec}(W^{(2)}) + b^{(2)}] = H^{(2)} (x^T \otimes I^{(2)}),$$

(32)

since the derivative of the weight vector with respect to the same vector can be simplified. Note that the same mathematical process as in equation (22) is used, but here the identity matrix $I^{(2)} \in R^{MxM}$.

Combining the three expressions of the chain rule terms, $\partial e / \partial w^{(2)}$ can be written in matrix form using only the information from the forward propagation as

$$\frac{\partial e}{\partial w^{(2)}} = DH^{(1)}W^{(1)}H^{(2)}(x^T \otimes I^{(2)})$$

(33)

and equation (31) is checked, the result has the expected dimensions.

After computing the gradients with respect to each set of weights, the weight matrices can be updated using the gradient descent algorithm or one of its variants. First, note that the Jacobian matrix of the gradient $\partial e / \partial w$ using the numerator notation has the form

$$\frac{\partial e}{\partial w} = \begin{bmatrix} \frac{\partial e_1}{\partial w_{11}} & 0 & \ldots & 0 \\
0 & \frac{\partial e_2}{\partial w_{21}} & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & \frac{\partial e_K}{\partial w_{K1}} \end{bmatrix},$$

(35)

noting that each $k$-th error term depends only on the $k$-th row of the weight matrix, If the $KxKM$ matrix above is pre-multiplied by a row vector of ones $r^T$ of dimension $K$, the result is a $1xM$ vector given by

$$r^T \frac{\partial e}{\partial w} = \begin{bmatrix} \frac{\partial e_1}{\partial w_{11}} & \frac{\partial e_2}{\partial w_{21}} & \ldots & \frac{\partial e_K}{\partial w_{K1}} \\
\frac{\partial e_1}{\partial w_{12}} & \frac{\partial e_2}{\partial w_{22}} & \ldots & \frac{\partial e_K}{\partial w_{K2}} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial e_1}{\partial w_{1M}} & \frac{\partial e_2}{\partial w_{2M}} & \ldots & \frac{\partial e_K}{\partial w_{KM}} \end{bmatrix}. $$

(36)

Since the result is a row vector, the update rules for the weight vectors using the method of steepest descent are given by

$$w^{(1)} = w^{(1)} - \eta \left( \frac{\partial e}{\partial w^{(1)}} \right)^T r$$

(37)
\[ b^{(1)} = b^{(1)} - \eta \left( \frac{\partial e}{\partial b^{(1)}} \right)^T r \]  
\[ w^{(2)} = w^{(2)} - \eta \left( \frac{\partial e}{\partial w^{(2)}} \right)^T r \]  
\[ b^{(2)} = b^{(2)} - \eta \left( \frac{\partial e}{\partial b^{(2)}} \right)^T r, \]

where \( \eta \) is the learning rate. Each weight vector can then be transformed back to a matrix with an inverse function of the \( \text{vec}(\cdot) \) operation, so that the weight matrices \( W \) can be examined during the process of backpropagation.

### 2.2.1 Extensions of backpropagation for any number of layers

Let a second hidden layer be introduced between the input \( x \) and the hidden layer \( z^{(2)} \). This will be denoted as \( z^{(3)} \) assuming it has \( P \) neurons. A new matrix of weights needs to be introduced, \( W^{(3)} \in \mathbb{R}^{D \times P} \). The gradient \( \partial e / \partial w^{(3)} \) is given by the chain rule as

\[
\frac{\partial e}{\partial w^{(3)}} = \frac{\partial e}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial w^{(3)}}. 
\]

The first two terms have already been computed above, so that \( \partial e / \partial z^{(1)} = D \) and \( \partial z^{(1)} / \partial z^{(2)} = H^{(1)} W^{(1)} \). The same processes from above can be used to compute the next two terms. First,

\[
\frac{\partial z^{(2)}}{\partial z^{(3)}} = \frac{\partial z^{(2)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial z^{(3)}} = H^{(2)} \frac{\partial}{\partial z^{(3)}} \left( W^{(2)} z^{(3)} \right) = H^{(2)} W^{(2)}. 
\]

Note that \( H^{(2)} \) results from a similar calculation as equation (17). Next, the last term for both gradients can be computed as in equation (32) and the result is

\[
\frac{\partial z^{(3)}}{\partial w^{(3)}} = H^{(3)} (x^T \otimes I^{(3)}) 
\]

\[
\frac{\partial z^{(3)}}{\partial b^{(3)}} = H^{(3)}. 
\]

The final equation for the gradient with respect to the weights and biases is given in matrix form as

\[
\frac{\partial e}{\partial w^{(3)}} = DH^{(1)} W^{(1)} H^{(2)} W^{(2)} H^{(3)} (x^T \otimes I^{(3)}) 
\]

\[
\frac{\partial e}{\partial b^{(3)}} = DH^{(1)} W^{(1)} H^{(2)} W^{(2)} H^{(3)} I^{(3)}. 
\]

By extending the network even further, a pattern emerges, so that a more general gradient equation can be defined for an arbitrary number of layers. Assuming an MLP architecture with an arbitrary number of layers \( L \), where each layer has an arbitrary number of neurons, the general equation for the gradient of the \( L \)-th set of weights and biases can be written as
\[
\frac{\partial e}{\partial W^{(l)}} = D \left[ \prod_{i=1}^{L} W^{(l-1)}H^{(l)} \right] \left( z^{(L+1)T} \otimes I^{(l)} \right)
\]
\[
\frac{\partial e}{\partial b^{(l)}} = D \left[ \prod_{i=1}^{L} W^{(l-1)}H^{(l)}I^{(l)} \right],
\]

where \( W^{(1)} \) is assumed to be an identity matrix, so that for \( l = 1 \), the weight matrix does not influence \( H^{(1)} \) and if \( L \) is the first set of weights from the input to the first hidden layer, then \( z^{(L+1)} \) is the input \( x \). Also, in the case of the biases, note that the identity matrix has the same dimensions as \( H \). The symbol \( \prod \) is used here to define the repeated multiplication of successive terms.

3 APPLICATIONS OF THE MLP FOR CLASSIFICATION TASKS

3.1 Description of the task

This section will describe the implementation of the MLP equations derived above for a classification task, where the goal is to distinguish between two very closely related audio spectra. An interesting fundamental question that arises is the extent to which two closely related spectra can be distinguished from one another by an MLP trained to recognize the two signals. The results will be discussed and the performance of different MLP architectures will be compared. In addition, an analysis of the network behaviour during the training will be presented and initial results discussed.

An MLP network is trained to classify two spectra, that from visualizing the FFT spectrum cannot be easily differentiated. The training data is synthesised by generating white noise signals passed through bandpass filters of different centre frequencies. The database was built by generating 1000 white noise signals bandpass filtered by a lower centre frequency \( f_{\text{low}} \) and another 1000 signals bandpassed by a filter with a higher centre frequency \( f_{\text{high}} \). Using a sample frequency of 16000 Hz and an FFT size of 512, Figure 2 shows on the left side two spectra of white noise signals bandpass filtered by \( f_{\text{low}} = 705 \) Hz and \( f_{\text{high}} = 710 \) Hz, both filters with a bandwidth of 10 Hz. On the right side, are shown the spectra of the two types of signals averaged over all 1000 signals. Note that the FFT resolution is the sample frequency divided by the FFT size and is 31.25 Hz. Each signal is 300 ms long. The input size of the MLP network equals the FFT size and in order to learn the features, different network architectures were tested. Since this is a classification task and there are two output classes, the output layer has two neurons and the output is expected to be 1 for one class or 0 for the other class. In the output layer, the softmax function is used to estimate the output. For the following simulations, the learning rate \( \eta \) was set to a constant value of \( \eta = 0.005 \) and the Adam optimizer\(^{20}\) was used to update the weights. A batch size of 64 was used, which means that the weights were updated after every 64 analysed signals. More about different batch size implementations is available\(^{21}\).

To better understand the MLP model, the network was trained for two classification tasks of different difficulties. For the easy task, where \( f_{\text{low}} = 650 \) Hz and \( f_{\text{high}} = 710 \) Hz, the MLP model easily achieved at least 98% on average, and for the difficult task, where \( f_{\text{low}} = 705 \) Hz and \( f_{\text{high}} = 710 \) Hz, the MLP model achieved, depending on the architecture, on average an accuracy of around 63%. To investigate the behaviour of the network, the simulations were run using our implemented framework, although most of the results were double checked using the MATLAB Deep Learning framework.
3.2 Results

The first simulations were implemented to compare the performances of the MLP models while using the Adam optimizer instead of the usual Stochastic Gradient Descent (SGD) algorithm. The work undertaken so far suggests that on average the performance of the MLP model slightly increases if the Adam optimizer is used. In addition to this finding, the model achieves its highest accuracy faster using Adam. Therefore, for the rest of the simulations, the Adam optimizer was used to train the network.

Next, the benchmark for further investigations was a simple architecture with no hidden layers, where the input was weighted by a weight matrix and was sent to the output layer through a softmax function. Simulations so far show that for both the easy and the hard tasks, this simple algorithm can be trained to classify the two signals, but with a slightly lower validation accuracy than if one hidden layer is placed between the input and the output. For example, for the easy task the very simple architecture with no hidden layer achieves on average a validation accuracy of 98-99% and if one hidden layer of 10 neurons is inserted, the accuracy rises to 100% very quickly. The same pattern can be observed for the difficult task, where one hidden layer helps the MLP to achieve a better validation accuracy (62.6%) faster compared to no hidden layer (61.25%).

Further simulations were implemented to compare the performance of MLP models with different numbers of hidden layers, each with the same number of 10 neurons. Preliminary results suggest that on average there is no significant difference between the MLP with one or several hidden layers. Table 1 shows the validation accuracy for both tasks using from 1 to 5 hidden layers.

<table>
<thead>
<tr>
<th>Nr. hidden layers</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy %</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy task</td>
<td>99</td>
<td>99</td>
<td>99.2</td>
<td>99</td>
<td>99.3</td>
</tr>
<tr>
<td>Hard task</td>
<td>62.2</td>
<td>60.3</td>
<td>60.4</td>
<td>60.6</td>
<td>59.1</td>
</tr>
</tbody>
</table>

For the hard task, it can be seen that the validation accuracy decreases a little with an increased number of hidden layers and this is mainly because of overfitting. These findings are in accordance with well-established research\(^{22,23}\), where the authors showed that an MLP with one hidden layer has the potential to be an universal approximator and is able to learn any nonlinear function. In contrast to these findings it has been shown\(^{24}\) that a deep MLP can learn more easily and that the depth can play an important role. For the initial application in this paper, more hidden layers do not appear to increase the learning capabilities of the MLP and therefore the next results were computed using only one hidden layer.
To analyse the effect of the size of the hidden layer, MLP models with one hidden layer and different number of neurons were trained for the same two classification tasks. Table 2 shows the averaged validation accuracy for each MLP model.

<table>
<thead>
<tr>
<th>Nr. neurons</th>
<th>Easy task</th>
<th>Hard task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy %</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>94.1</td>
<td>58.05</td>
</tr>
<tr>
<td>4</td>
<td>99</td>
<td>61.3</td>
</tr>
<tr>
<td>8</td>
<td>98.95</td>
<td>61.25</td>
</tr>
<tr>
<td>16</td>
<td>98.8</td>
<td>61.35</td>
</tr>
</tbody>
</table>

When the hidden layer has 2 neurons, simulations show that the network is not sufficiently robust, since in some trials, the validation performance was 50%, meaning that the network did not learn anything at all. From 10 trials for the easy task, 9 of them achieved 99% and one was 50%. For the hard task, 3 trials out of 10 achieved an accuracy of 50%. The main outcome from these results is that the validation accuracy is not changing significantly with an increased number of neurons above four. However, it was found that the training accuracy increases with an increased number of neurons. In other words, the network can learn better if the hidden layer contains more neurons. The main reason why the validation accuracy is not increasing as well is likely to be overfitting, which means that the network learns the training features too well but cannot generalize well. Fortunately, there are several methods that can reduce this problem and help the network generalize better. In other words, an increased number of neurons in the hidden layer is not directly improving the validation accuracy but helps the network to learn better and with the help of some optimization techniques can improve the performance.

A final experiment was run to check how the accuracy increases with a decreasing level of difficulty, while having as reference the spectrum resolution of 31.25 Hz. The difference between the centre frequencies $\Delta f$ was increased from 5 Hz to 60 Hz and for each task 10 trials were run. Table 3 shows the validation accuracy on average for 5 different values of $\Delta f$, with an MLP model with one hidden layer and 10 neurons.

<table>
<thead>
<tr>
<th>$\Delta f$ (Hz)</th>
<th>5</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy %</td>
<td>61.4</td>
<td>85.15</td>
<td>94.35</td>
<td>97.8</td>
<td>100</td>
</tr>
</tbody>
</table>

It is interesting to observe that if $\Delta f$ is close to the FFT resolution or higher, the validation accuracy is around 95% or higher. If $\Delta f$ is 20 or less, the accuracy decreases drastically by at least 10%.

Motivated by these results, the network behaviour during training was investigated. The weight matrices after every update were analysed by looking at the singular values for the two classification tasks. The singular value decomposition (SVD) of the weight matrix between the input layer and the hidden layer of 16 neurons was observed during and after the training. Figure 3 shows the different behaviour of the singular values for an easy classification task, where $f_{\text{low}} = 650$ Hz and $f_{\text{high}} = 710$ Hz and where the accuracy achieves 100% and a more difficult task, where $f_{\text{low}} = 705$ Hz and $f_{\text{high}} = 710$ Hz with an accuracy of 63.5%. It is interesting to observe that most singular values of the weight matrix during the training of the easy task remain constant and very close to zero, while only the first 2 singular values are dominant. In comparison, the difficult task involves far more singular values in the weight matrix. It is also interesting to observe how the singular values change during the weight updates. It can be seen that in the left figure, the performance is unlikely to improve with more iterations, since there is little sign that the model will develop new relations between the input and the hidden layer. Since the accuracy achieved 100%, the two dominant singular values remain almost constant during the last weight iterations.
DISCUSSION AND CONCLUSION

This paper gave an insight into the implementation of the MLP, while deriving the mathematics from first principles in order to understand how information is propagated through the network architecture. The gradient equations were written in a general form, which can be easily implemented in any programming language, and a model problem was used to validate the mathematics. An audio signal classification task was used as an initial exploration of the network training behavior. It was shown that the MLP can distinguish between two spectra, although the results suggest that the spectral resolution of the input FFT still limits the ability of the network to classify. The extraction of other spectral features will be a further priority for such an approach to audio signal classification.

The equations derived above are expressed in matrix form, so that each variable can be visualized during training. This enables the SVD to be easily applied to the weight matrices. Note that other papers have used the SVD in combination with neural networks\textsuperscript{28-31} for the purposes of dimensionality reduction. The behavior of the singular values of the weight matrices examined here showed a clear distinction between behaviors depending on the difficulty of the initial classification task examined. This will also be investigated further.

REFERENCES


