# NEURAL THIRD-OCTAVE GRAPHIC EQUALIZER

*Jussi Rämö and Vesa Välimäki**

Acoustics Lab, Dept. of Signal Processing and Acoustics
Aalto University
Espoo, Finland
jussi.ramo@aalto.fi

## ABSTRACT

This paper proposes to speed up the design of a third-order graphic equalizer by training a neural network to imitate its gain optimization. Instead of using the neural network to learn to design the graphic equalizer by optimizing its magnitude response, we present the network only with example command gains and the corresponding optimized gains, which are obtained with a previously proposed least-squares-based method. We presented this idea recently for the octave graphic equalizer with 10 band filters and extend it here to the third-octave case. Instead of a network with a single hidden layer, which we previously used, this task appears to require two hidden layers. This paper shows that good results can be reached with a neural network having 62 and 31 units in the first and the second hidden layer, respectively. After the training, the resulting network can quickly and accurately design a third-order graphic equalizer with a maximum error of 1.2 dB. The computing of the filter gains is over 350 times faster with the neural network than with the original optimization method. The method is easy to apply, and may thus lead to widespread use of accurate digital graphic equalizers.

## 1. INTRODUCTION

The design of a graphic equalizer (GEQ) has advanced considerably in the past few years [1, 2]. Much research has been conducted to improve the design of both the cascade [3–8] and the parallel GEQs [9–13]. Currently it is possible to design either a cascade [2, 7] or a parallel GEQ [11–13] to have a maximum error of 1 dB, which is often considered sufficient for hi-fi audio. However, the design still requires optimization, which includes matrix operations, when the command gains are changed. This means that the accurate design of a GEQ needs large computational resources, if the parameters need to be updated quickly, such as in low-latency real-time applications.

We have recently proposed the idea of simplifying the calculation of filter gain optimization in a cascade graphic equalizer using a neural network [14], instead of the previous heavier method, which requires the calculation of DFT and matrix inversions. The training of the neural network becomes easy, when the network is presented with the pairs of command gains and the corresponding optimized gains obtained with an accurate design method. Then the task of the neural network is to imitate the nonlinear mapping, which the optimization method uses. This is simpler than using the neural network to learn to design the graphic equalizer by optimizing its magnitude response. It is also a different approach than the teaching of an equalizer using a neural network directly from an audio signal [15]. The training using the gain pairs was applied first to the cascade octave GEQ using a conventional perceptron with a single hidden layer [14].

The neural network introduces an error, when it approximates the nonlinear mapping. In [14] it was shown that a perceptron having twice as many hidden layer cells as input parameters was large enough for good approximation. The number of input parameters was 10 in the case of an octave GEQ, so 20 hidden layer cells were needed [14]. The approximation error can be kept smaller than 0.085 dB, which is sufficient for a maximum error of 0.7 dB for the GEQ itself [14].

In this paper, we apply the same idea to the design of a very common large GEQ, which has third-octave-octave bands. The third-octave GEQ has 31 bands to control the signal gain on narrow bands over the whole audio frequency range from 20 Hz to 20,000 Hz. This paper shows that the complexity of the problem is much larger than in the case of the octave GEQ, which has only 10 bands, and, consequently, a neural network with a single large hidden layer may not learn the mapping sufficiently accurately. We thus test a larger network structure having two hidden layers. It seems necessary that one of the hidden layers should contain twice as many nodes as the input layer.

The rest of this paper is organized as follows. Section 2 briefly recapitulates the design of a cascade third-octave GEQ, which will be approximated with the neural net. Section 3 explains the structure and training of the neural network. Section 4 presents validation and results of this work. Section 5 concludes this paper.

## 2. THIRD-OCTAVE GRAPHIC EQ DESIGN

An accurate design for a third-octave cascade graphic EQ (ACGE3) was proposed at the DAFx-17 conference [2]. The method is an extension of the corresponding accurate GEQ design for the octave case with ten bands [7]. Both designs take the user-set command gain values as inputs and then optimize the filter gains by evaluating the interaction between different band filters, which are second-order IIR filters. Each band filter is designed as a specific parametric equalizer, which is controllable at its own center frequency and at the center frequencies of its neighboring bands by defining the bandwidth in an unusual manner. This parametric equalizer is a modification of the design proposed by Orfanidis in his textbook [16].

The transfer function of the second-order band filter with user-

* This research is related to the "Nordic Sound and Music Computing Network—NordicSMC", NordForsk project number 86892.

Table 1: *Center frequencies $f_c$ and bandwiths $B$ for third-octave bands $m$.*

| $m$ | $f_c$ (Hz) | $B$ (Hz) | $m$ | $f_c$ (Hz) | $B$ (Hz) | $m$ | $f_c$ (Hz) | $B$ (Hz) | $m$ | $f_c$ (Hz) | $B$ (Hz) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 19.69 | 9.178 | **9** | 125.0 | 58.28 | **17** | 793.7 | 370.0 | **25** | 5040 | 2350 |
| **2** | 24.80 | 11.56 | **10** | 157.5 | 73.43 | **18** | 1000 | 466.2 | **26** | 6350 | 2846* |
| **3** | 31.25 | 14.57 | **11** | 198.4 | 92.51 | **19** | 1260 | 587.4 | **27** | 8000 | 3502* |
| **4** | 39.37 | 18.36 | **12** | 250.0 | 116.6 | **20** | 1587 | 740.1 | **28** | 10080 | 4253* |
| **5** | 49.61 | 23.13 | **13** | 315.0 | 146.9 | **21** | 2000 | 932.4 | **29** | 12700 | 5038* |
| **6** | 62.50 | 29.14 | **14** | 396.9 | 185.0 | **22** | 2520 | 1175 | **30** | 16000 | 5689* |
| **7** | 78.75 | 36.71 | **15** | 500.0 | 233.1 | **23** | 3175 | 1480 | **31** | 20160 | 5570* |
| **8** | 99.21 | 46.25 | **16** | 630.0 | 293.7 | **24** | 4000 | 1865 | | | |

* Manually adjusted bandwidths due to warping close to the Nyquist frequency.

set linear gain $G_m$ is [2]

$$H_m(z) = b_{0,m}\frac{1 + b_{1,m}z^{-1} + b_{2,m}z^{-2}}{1 + a_{1,m}z^{-1} + a_{2,m}z^{-2}}, \tag{1}$$

where

$$\begin{aligned} b_{0,m} &= \frac{1+\beta_m}{1+G_m\beta_m}, \\ b_{1,m} &= -2\frac{\cos(\omega_{c,m})}{1+G_m\beta_m}, \qquad a_{1,m} = -2\frac{\cos(\omega_{c,m})}{1+\beta_m}, \\ b_{2,m} &= \frac{1-G_m\beta_m}{1+G_m\beta_m}, \qquad a_{2,m} = \frac{1-\beta_m}{1+\beta_m}, \end{aligned} \tag{2}$$

where

$$\beta_m = \begin{cases} \sqrt{\dfrac{|G_{\mathrm{B},m}^2 - 1|}{|G_m^2 - G_{\mathrm{B},m}^2|}}\tan\left(\dfrac{B_m}{2}\right), & \text{when } G_m \neq 1, \\ \tan\left(\dfrac{B_m}{2}\right), & \text{when } G_m = 1, \end{cases} \tag{3}$$

$$g_{B,m} = cg_m, \quad \text{where } c = 0.4, \tag{4}$$

$$\omega_{c,m} = 2\pi f_{c,m}/f_s, \tag{5}$$

with $g_{\mathrm{B},m} = 20\log(G_{\mathrm{B},m})$ and $g_m = 20\log(G_m)$. The sampling rate $f_s$ used throughout this work is 44.1 kHz. Table 1 shows the center frequencies $f_{c,m}$ and bandwidths $B_m$ of the third-octave bands used in this work.

One such second-order IIR filter is used per band, see Fig. 1(a), and all the 31 filters are cascaded to form the overall transfer function of the GEQ:

$$H(z) = \prod_{m=1}^{31} H_m(z), \tag{6}$$

as illustrated in Fig. 1(b). The gain factor $G_0$ in front of the graphic equalizer in Fig. 1(b) is the product of the scaling coefficients $b_{0,m}$ of the band filters:

$$G_0 = \prod_{m=1}^{31} b_{0,m}. \tag{7}$$

This way the multiplier related to the scaling factor $b_{0,m}$ can be removed from each band filter section, as can be seen in Fig. 1(a), which saves $M - 1$ multiplications in total [13].

### 2.1. Least Squares Optimization of Filter Gains

The optimal filter gains for the cascade graphic equalizer are solved using the least-squares method with the help of an interaction matrix [7]. The magnitude response of each equalizer filter with an example gain (17 dB is used in this work) is evaluated at the third-octave center frequencies and at their geometric means. These data are used to form the interaction matrix $\mathbf{B_0}$, which represents the leakage caused by each band filter to the other frequency points. Each row of the interaction matrix contains the normalized magnitude response of the $m^{\text{th}}$ band filter sampled at the 61 prescribed frequencies. Because of the normalization, the value of the interaction matrix at the center frequency of the filter itself is always 1.0, since the magnitude response is divided by the filter gain. Furthermore, an additional iteration is used, which calculates another interaction matrix based on the filter gains obtained as the first LS solution. The second interaction matrix is used for further optimization [7]. This iteration round helps to restrict the approximation error in the magnitude response to be less than $\pm 1$ dB, which was the design goal during the development of ACGE3 [2]. The

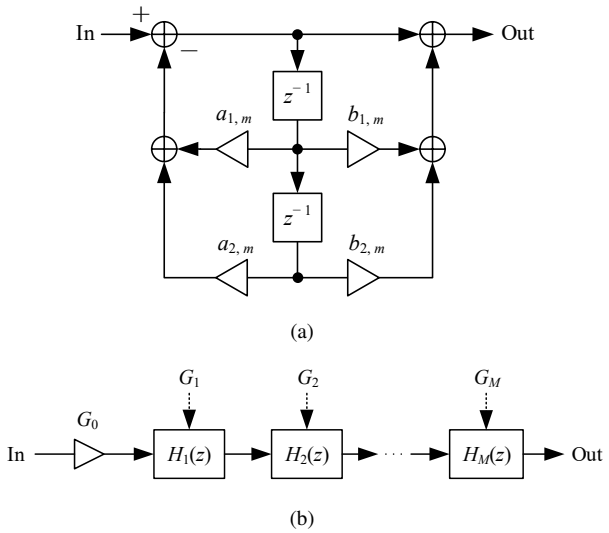


Figure 1: *(a) The second-order IIR filter structure of each band filter $H_m(z)$, and (b) the graphic equalizer structure containing a series of such filters and showing the filter gain controls, $G_m$. In the third-octave design, the number of filter sections is $M = 31$.*

matrix inversions cause this optimization method to be computationally costly.

## 3. NEURAL NETWORK

### 3.1. Training Data

The training data for the feedforward neural net is created using the ACGE3 design [2], which was reviewed in Sec. 2. With that design it is possible to create a huge number of input-output gain pairs, where the input values are the user-set command gains between −12 dB and 12 dB and the outputs are the optimized filter gains used in the underlying filter design, see Sec. 2.1.

For this work we created 1500 input-output pairs with random input gains using the ACGE3 algorithm. Six special gain configurations, known to be hard for GEQs, were included in the training data. They were two constants cases with all gains set to +12 dB and all gains at −12 dB, and two zigzag cases [2], as well as two hard configurations which are special zigzag settings[1].

### 3.2. Network Structure and Training

By definition, the third-octave EQ has 31 frequency bands, meaning it has 31 user-adjustable command gains. Thus the neural network has 31 nodes in its input layer, one for each band's gain setting. The ACGE3 design is implemented using one second-order IIR filter ber band, resulting in 31 optimized gain values for the EQ filters. Thus, the size of the output layer is also set to 31.

After initial training tests of the neural network it was decided that the network structure should be in the form of 31-J-K-31, i.e., it should have two hidden layers of size $J$ and $K$. After training several different prototype neural networks we settled on the layer sizes of $J = 62$ and $K = 31$. Based on our previous experiments with the octave GEQ it is beneficial to have the size of the first hidden layer twice the size of the input layer [14]. Figure 2 shows the structure of the neural network, where $g_1$, $g_2$, ..., $g_{31}$ are the user-set command gains in dB and $g_{\text{opt},1}$, $g_{\text{opt},2}$, ..., $g_{\text{opt},31}$ are the optimized filter gains in dB.

The neural network was trained using Matlab's `fitnet` function, which is a function-fitting neural network that is able to form a generalization of the input-output relation of the training data. Thus, after the network is trained, it is possible to use it to generate outputs for inputs that were not in the training dataset. The training algorithm was selected to be `trainbr`, a Bayesian regularization backpropagation algorithm [17]. It updates the weight and bias values according to the Levenberg-Marquardt (LM) optimization [18, Ch. 12]. The LM algorithm provides a desirable compromise between speed and guaranteed convergence of steepest descent [18], while the Bayesian regularization also ensures that the resulting network generalizes well by minimizing a combination of the squared errors and the network weights [17].

The training dataset was split into two sets, a training set (70% of the whole dataset) and test data (the remaining 30%). The test data is not used in the training per se, it is only used to monitor the performance of the model to unseen data during the training. The stopping conditions were set so that the training would continue until it is converged. With Bayesian regularization, a good indication of convergence is when the LM $\mu$ parameter reaches a high value (Matlab's default is $10^{10}$). However, after 15,000 epochs the training was stopped, before reaching the maximum $\mu$. One epoch takes approximately a minute calculate, when using 12 parallel CPUs, so it is quite time consuming to train the neural net. Thus, increasing the training time could still improve the accuracy of the proposed neural net.

[1] $\mathbf{g}_{\text{special}} = [12\ -12\ -12\ 12\ -12\ -12\ -12\ 12\ -12\ -12\ 12\ -12\ -12\ 12\ -12\ -12\ 12\ -12\ -12\ 12\ -12\ -12\ -12\ 12\ -12\ -12\ 12]^T$ and its opposite $-\mathbf{g}_{\text{special}}$.

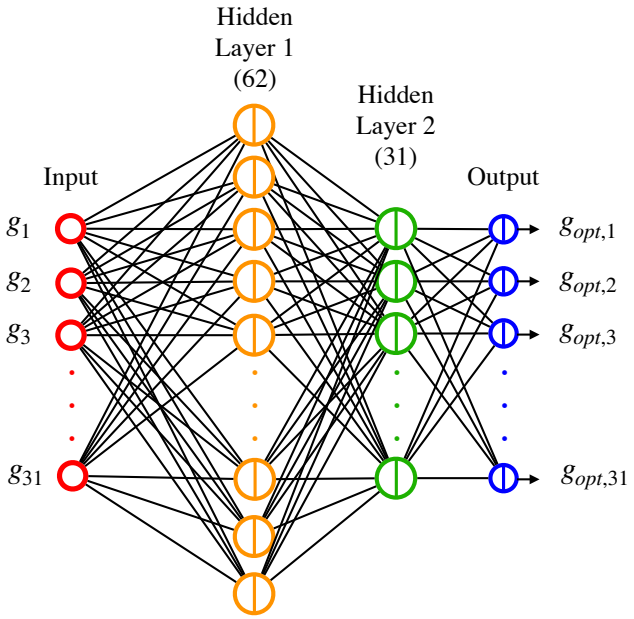


Figure 2: *Neural network presented in this work has 31 inputs and outputs, and two hidden layers.*

### 3.3. Final Neural Network

Figure 2 shows the resulting neural network while Fig. 3 depicts individual neurons in the hidden layers and the output layer. In Fig. 3, the leftmost neuron is the $j^{\text{th}}$ neuron of hidden layer 1. Its inputs are the scaled user-set command gains $g'_1$, $g'_2$, ..., $g'_{31}$, since the neural network assumes that the input data has values between −1 and 1. Matlab does the scaling automatically during training using `mapminmax` function. The $j^{\text{th}}$ neuron uses the weights $w^1_{j,1}$, $w^1_{j,2}$, ..., $w^1_{j,31}$ to scale the inputs, sums them and adds the bias value $\theta^1_j$ to the sum, and then uses the nonlinear sigmoid function $\sigma$ to calculate the output $o^1_j$ of for the neuron:

$$o^1_j = \sigma\left(\sum_{m=1}^{M=31} w^1_{j,m} g'_m + \theta^1_j\right), \tag{8}$$

where $\sigma$ is equivalent to $\tanh(x) = 2/(1 + e^{-2x}) - 1$.

The output of a neuron in the second hidden layer is calculated in similar manner as in Eq. (8), but now the inputs are the outputs from every neuron in hidden layer 1. The output of the $k^{\text{th}}$ neuron of hidden layer 2 is calculated as

$$o^2_k = \sigma\left(\sum_{j=1}^{J=62} w^2_{k,j} o^1_j + \theta^1_k\right), \tag{9}$$

and finally the $m^{\text{th}}$ neuron in the output layer outputs the optimized

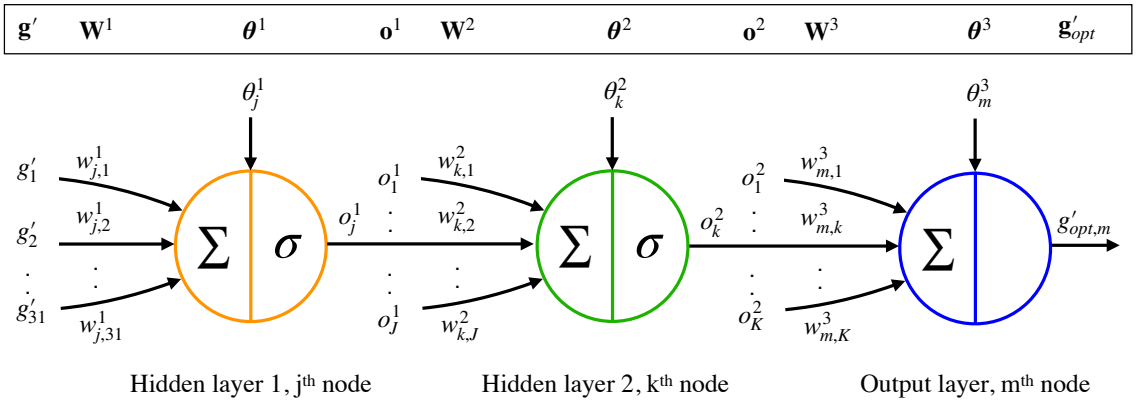


Figure 3: *Structure of individual neurons in the neural net. Cf. Fig. 2.*

gain for the $m^{\text{th}}$ filter by calculating

$$g'_{\text{opt},m} = \sum_{k=1}^{K=31} w^3_{m,k} o^2_k + \theta^3_m. \tag{10}$$

Equations (8)–(10), which are used for running the neural network, can be written in matrix form as

$$\mathbf{g}' = 2 \cdot \frac{\mathbf{g} - \mathbf{x}_{\text{min}}}{\mathbf{x}_{\text{max}} - \mathbf{x}_{\text{min}}} - 1, \tag{11}$$

$$\mathbf{o^1} = \tanh(\mathbf{W}^1 \mathbf{g}' + \boldsymbol{\theta}^1), \tag{12}$$

$$\mathbf{o^2} = \tanh(\mathbf{W}^2 \mathbf{o}^1 + \boldsymbol{\theta}^2), \tag{13}$$

$$\mathbf{g'_{opt}} = \mathbf{W}^3 \mathbf{o}^2 + \boldsymbol{\theta}^3, \tag{14}$$

$$\mathbf{g_{opt}} = (\mathbf{t}_{\text{max}} - \mathbf{t}_{\text{min}}) \frac{\mathbf{g'_{opt}} + 1}{2} + \mathbf{t}_{\text{min}}, \tag{15}$$

where all the vectors and matrices correspond to those shown in the top part of Fig. 3. That is, Eq. (11) maps the user-set dB-gain values $\mathbf{g} \in [-12\ 12]$ to $\mathbf{g}' \in [-1\ 1]$, where all $x_{\text{min},m} = -12$ and $x_{\text{max},m} = 12$. Eq. (12) calculates the outputs $\mathbf{o}^1$ of hidden layer 1 based on $\mathbf{g}'$ by using weights $\mathbf{W}^1$, bias values $\boldsymbol{\theta}^1$, and the nonlinear transfer function $\tanh()$. Similarly, Eq. (13) uses all of the outputs $\mathbf{o}^1$ of hidden layer 1 to calculate the outputs of hidden layer 2 using a different set of weights $\mathbf{W}^2$ and bias values $\boldsymbol{\theta}^2$, including the nonlinear sigmoid function. The output layer takes the outputs $\mathbf{o}^2$ of hidden layer 2 as its inputs and weights them with $\mathbf{W}^3$ and adds the bias values defined in $\boldsymbol{\theta}^3$. Note that the output layer has no nonlinearity in it. Finally, the output layer of the neural network outputs the optimized gain vector $\mathbf{g}'_{\text{opt}}$ that have values between $[-1\ 1]$, which are then mapped to dB values based on the maximum and minimum values found in the training data targets, $\mathbf{t}_{\text{max}}$ and $\mathbf{t}_{\text{min}}$, respectively.

With these three weight matrixes, three bias vectors, and four output/input extreme values, it is possible to run the neural network for any arbitrary user command gain configurations (between $-12$ and 12 dB). We will provide all of the needed parameters to run the model.

## 4. RESULTS AND VALIDATION

In order to validate the actual performance and accuracy of the proposed third-octave neural GEQ (NGEQ3), we need to compare it against ACGE3, which was used to train the network. In order to do this, a validation dataset of 10,000 random command gain settings was created.

### 4.1. Computational Performance

The main purpose of substituting the ACGE3 filter optimization with a neural network is to computationally simplify the procedure so that Fourier transforms and matrix inversions are not needed. Although the designing and training of neural networks may take some time, running a trained neural network is often computationally quite straightforward. The neural network proposed in this work has 4929 parameters, consisting of the weights and biases, however, the main computation consists of only three matrix multiplications and additions, and two `tanh` calculations for vectors of sizes 62 and 31, see Eqs. (12)–(14).

To evaluate the computational time of the filter optimization, the validation dataset of 10,000 input command gains were optimized and the averages of the optimization times were recorded. The results are shown in Table 2. As can be seen, the proposed NGEQ3 optimization (13 $\mu$s) is much faster than that of the original ACGE3 (4661 $\mu$s). The ACGE3 optimization is heavier than the proposed NGEQ3 optimization, since it requires the calculation and inversion of the interaction matrix, during the iteration round, and several matrix multiplications. The interaction matrix is constructed by using the discrete-time Fourier transform which is used to evaluate the magnitude response of the band filters at 61 frequency points, consisting of the 31 third-octave center frequencies and their midpoints. The matrix inversion requires the computing of the Penrose-Moore pseudoinverse for the resulting 61-by-31 interaction matrix, which involves a matrix inversion and three matrix multiplications [7].

Table 2: *Comparison of computing times of the third-octave ACGE3 and proposed NGEQ3 methods, average of 10,000 trials. The fastest case in each column is highlighted.*

| | Gain optimization | Coefficient update | Total |
|---|---|---|---|
| ACGE3 (DAFx-17) | 4661 $\mu$s | 57 $\mu$s | 4718 $\mu$s |
| NGEQ3 (proposed) | **13 $\mu$s** | 57 $\mu$s | **70 $\mu$s** |

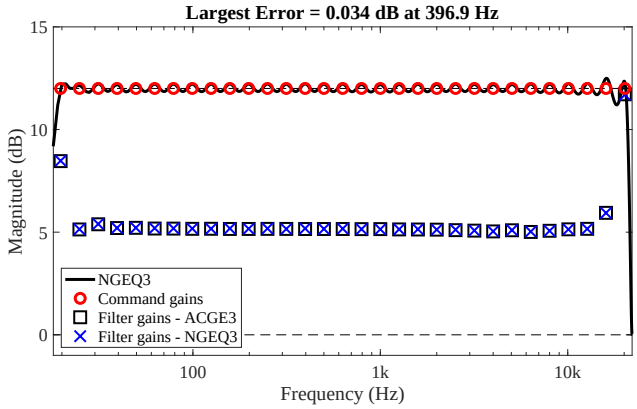


Figure 4: *Comparison of ACGE3 and NGEQ3 filter optimization, when all command gains are set to 12 dB.*

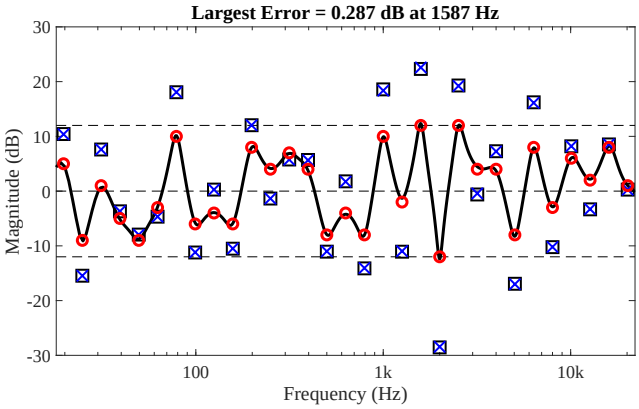


Figure 6: *Worst case scenario based on the validation dataset of 10,000 gain configurations. See the legend in Fig. 4.*

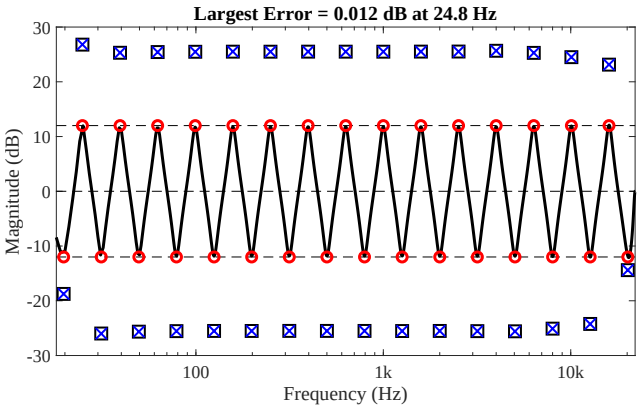


Figure 5: *Alternating ±12 zigzag command gain settings. See the legend in Fig. 4.*

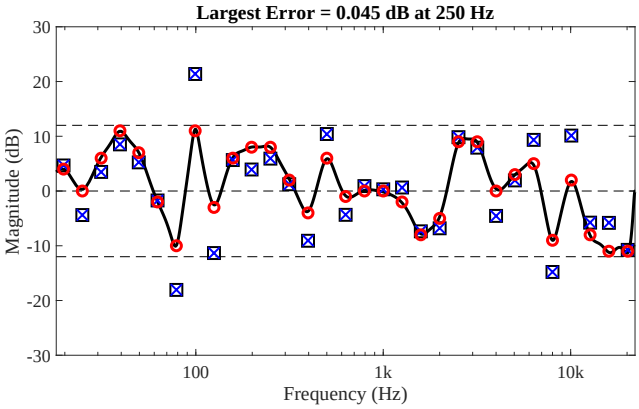


Figure 7: *Random command gain settings illustrating the mean accuracy of NGEQ3. See the legend in Fig. 4.*

Furthermore, the calculation of the filter coefficients takes approximately 57 $\mu$s, which is the same for both methods, meaning that the NGEQ3 gain optimization is even faster than the actual filter design.

### 4.2. Accuracy

While getting the implementation of the filter gain optimization faster can be essential to certain applications, the proposed method needs to be accurate in order to be useful. Figures 4 and 5 show magnitude responses of two example runs of the proposed neural network. Both cases are known to be challenging for a GEQ, and thus, both of these example cases were also included in the training dataset. Figure 4 shows a gain setting where all command gains are set to +12 dB, while Fig. 5 shows a gain setting with alternating commands at ±12 dB. In both figures, red circles (○) are the user-set command gains, black squares (□) are the ACGE3 optimized filter gains, blue crosses (×) are the optimized filter gains by the proposed NGEQ3, and the black line plots the magnitude response of the whole NGEQ3. Thus, in ideal case the crosses should lie inside the squares (⊠). Furthermore, the horizontal dashed lines plot the zero line, as well as the used maximum and minimum values ±12 of the command gains.

These two examples clearly illustrate the importance of filter gain optimization, since it is evident that the optimized filter gains (□ and ×) can be totally different than the actual user-set command gains (○). In Fig. 4, where all the gains are set to +12 dB, the optimized gains are considably smaller that the command gains, so that the final response settles at 12 dB. On the other hand, in the zigzag case in Fig. 5, the optimized gains are more than twice the value of the user-set command gains.

The accuracy of the proposed NGEQ3 was evaluated using the same validation dataset as above. The proper error evaluation is to compare NGEQ3 to ACGE3, since that is how the neural network was trained. That is, a perfect neural net with zero error would produce identical responses (and errors) with ACGE3. However,

Table 3: *Magnitude-response errors in dB at command point frequencies for 10,000 random gain settings.*

| | **ACGE** | | **Commands** | |
|---|---|---|---|---|
| | Max | Mean Max | Max | Mean Max |
| ACGE3 (DAFx-17) | – | – | 1.1 | 0.53 |
| NGEQ3 (proposed) | 0.28 | 0.07 | 1.2 | 0.53 |

the absolute errors in respect to the user-set command gains are what eventually matters to the end user.

Table 3 shows the accuracy validation results. Each row in the table compares the absolute error, calculated at the defined command gain frequencies where the target can be specified, to ACGE3 and to the actual user-set command gains values. The largest error in NGEQ3 with respect to ACGE3 was 0.28 dB[2]. This case is plotted in Fig. 6, where the largest error occurs at 1587 Hz. However, the magnitude response still goes through the command gain setting (○), so there is no visible error for the end user.

Finally, Fig. 7 shows a random gain setting (not included in the validation or training datasets) to illustrate how small the error typically is. As can be seen in Table 3, the mean value of all the maximum errors (mean max) over the 10,000 sample validation dataset, when compared to ACGE3, was 0.07 dB, which is incredibly small. Furthermore, the last two columns of the table show the maximum and average of all of the maximum errors calculated against the user-set command gains. As can be seen, the overall maximum errors of ACGE3 and the proposed NGEQ3 are almost the same and close to 1 dB, whereas the mean max of the validation dataset is the same for both methods, approximately 0.5 dB.

## 5. CONCLUSIONS

This paper proposed to simplify the calculation of the gain optimization of a third-octave graphic equalizers using a neural network. This became possible after our team recently proposed an accurate graphic equalizer design method, which optimizes filter gains based on user-defined command gains. The filter gains are determined using a least-squares technique with one iteration and then, as all parameters are known, the IIR filter coefficients are computed using closed-form formulas. Thus, the main complication in the design has been the filter gain optimization.

In this work, the command gain-filter gain vector pairs obtained with the accurate design method are used as training data for a multilayer neural network. After the training, the LS optimization can be replaced with the neural network. The computing of the filter gains is over 350 times faster with the neural network than with the original LS method. The filter coefficients are finally computed using traditional closed-form formulas, which now takes more time than the gain optimization. The proposed method turns accurate graphic equalizer design easy and fast. The associated Matlab code is available online at `http://research.spa.aalto.fi/publications/papers/dafx19-ngeq/`.

While in this work the neural network was trained by using the input-output gain pairs from a previously known optimization algorithm, in the future, it could be interesting to explore the possibilities to train a neural network with a novel cost function based on the actual gains of a GEQ.

## 6. REFERENCES

[1] V. Välimäki and J. D. Reiss, "All about audio equalization: Solutions and frontiers," *Appl. Sci.*, vol. 6, no. 129/5, pp. 1–46, May 2016.

[2] J. Liski and V. Välimäki, "The quest for the best graphic equalizer," in *Proc. Int. Conf. Digital Audio Effects (DAFx-17)*, Edinburgh, UK, Sep. 2017, pp. 95–102.

[3] J. S. Abel and D. P. Berners, "Filter design using second-order peaking and shelving sections," in *Proc. Int. Computer Music Conf.*, Miami, FL, USA, Nov. 2004.

[4] M. Holters and U. Zölzer, "Graphic equalizer design using higher-order recursive filters," in *Proc. Int. Conf. Digital Audio Effects (DAFx-06)*, Montreal, Canada, Sep. 2006, pp. 37–40.

[5] J. Rämö and V. Välimäki, "Optimizing a high-order graphic equalizer for audio processing," *IEEE Signal Process. Lett.*, vol. 21, no. 3, pp. 301–305, Mar. 2014.

[6] R. J. Oliver and J.-M. Jot, "Efficient multi-band digital audio graphic equalizer with accurate frequency response control," in *Proc. Audio Eng. Soc. 139th Conv.*, New York, NY, USA, Oct. 2015.

[7] V. Välimäki and J. Liski, "Accurate cascade graphic equalizer," *IEEE Signal Process. Lett.*, vol. 24, no. 2, pp. 176–180, Feb. 2017.

[8] S. Prince and K. R. S. Kumar, "A novel Nth-order IIR filter-based graphic equalizer optimized through genetic algorithm for computing filter order," *Soft Comput.*, vol. 23, no. 8, pp. 2683–2691, Apr. 2019.

[9] S. Tassart, "Graphical equalization using interpolated filter banks," *J. Audio Eng. Soc.*, vol. 61, no. 5, pp. 263–279, May 2013.

[10] Z. Chen, G. S. Geng, F. L. Yin, and J. Hao, "A pre-distortion based design method for digital audio graphic equalizer," *Digital Signal Process.*, vol. 25, pp. 296–302, Feb. 2014.

[11] J. Rämö, V. Välimäki, and B. Bank, "High-precision parallel graphic equalizer," *IEEE/ACM Trans. Audio Speech Lang. Process.*, vol. 22, no. 12, pp. 1894–1904, Dec. 2014.

[12] B. Bank, J. A. Belloch, and V. Välimäki, "Efficient design of a parallel graphic equalizer," *J. Audio Eng. Soc.*, vol. 65, no. 10, pp. 817–825, Oct. 2017.

[13] J. Liski, B. Bank, J. O. Smith, and V. Välimäki, "Converting series biquad filters into delayed parallel form: Application to graphic equalizers," *IEEE Trans. Signal Process.*, vol. 67, no. 14, pp. 3785–3795, Jul. 2019.

[14] V. Välimäki and J. Rämö, "Neural graphic equalizer," *IEEE/ACM Trans. Audio Speech Lang. Process.*, submitted for publication in Jan. 2019.

[15] M. A. Martínez Ramírez and J. D. Reiss, "End-to-end equalization with convolutional neural networks," in *Proc. 21st Int. Conf. Digital Audio Effects (DAFx-18)*, Aveiro, Portugal, Sep. 2018, pp. 296–303.

[16] S. J. Orfanidis, *Introduction to Signal Processing*, Prentice-Hall, Upper Saddle River, NJ, 1996.

[17] F. D. Foresee and M. T. Hagan, "Gauss-Newton approximation to Bayesian learning," in *Proc. IEEE Int. Conf. Neural Networks (ICNN'97)*, Houston, TX, USA, Jun. 1997, pp. 1930–1935.

[18] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesús, *Neural Network Design*, Second edition, 2014. [E-Book] Available: http://hagan.okstate.edu/nnd.html.

[2] When the neural network was trained using a single layer with 62 nodes, the maximum error was 0.52 dB, which was considered to be too large for our purposes.