A DIFFERENTIABLE DIGITAL MOOG FILTER FOR MACHINE LEARNING APPLICATIONS

Etienne Gerat , Purbaditya Bhattacharya and Udo Zölzer

Department of Signal Processing and Communication
Helmut Schmidt University
Hamburg, Germany
`e.gerat@hsu-hh.de | bhattacp@hsu-hh.de`

ABSTRACT

In this project, a digital ladder filter has been investigated and expanded. This structure is a simplified digital analog model of the well known analog Moog ladder filter. The goal of this paper is to derive the differentiation expressions of this filter with respect to its control parameters in order to integrate it in machine learning systems. The derivation of the backpropagation method is described in this work, it can be generalized to a Moog filter or a similar filter having any number of stages. Subsequently, the example of an adaptive Moog filter is provided. Finally, a machine learning application example is shown where the filter is integrated in a deep learning framework.

1. INTRODUCTION

The Moog ladder filter is a well known analog filter present in numerous synthesizers. It has been introduced in 1965 by Robert Moog [1]. Since then, it has been considered as a central piece of some subtractive synthesizers that gives a very recognizable character to the sound and offers intuitive control parameters. Nowadays, many of the iconic analog synthesizers are digitally modeled to be included in digital hardware or software synthesizers. Several digital models of the Moog filter are already studied using different approaches [2, 3].

In recent years, *machine learning* (ML) has been actively applied to the field of audio signal processing. And it has been stated that classic ML and *deep learning* (DL) structures are not always well adapted to solve audio related problems. Integrating audio systems directly in a DL architecture has been proven to be successful and to achieve good results with smaller architectures [4]. Differentiable function blocks are required to allow backpropagation and thus the integration into DL systems. This paper shows the differentiation process of the chosen Moog filter structure with respect to (w.r.t.) its control parameters, as it can be applied to other *Infinite Impulse Response* (IIR) filters and digital signal processing algorithms [5, 6]. As a proof of concept for the backpropagation capabilities, an adaptive version of the filter has been programmed.

This paper is part of a research project, where a subtractive synthesizer that includes a Moog filter would be differentiated to apply timbre matching using ML. This has already been studied using non gradient-based methods [7, 8] and genetic algorithm [9]. And more recently using Variational Auto-Encoders and Normalizing Flows[10]. Hence, the current work initially investigates the capability of learning the control parameters of a simple Moog filter with the help of ML algorithms.

2. DIGITAL MOOG FILTER

In this paper, a digital Moog filter structure presented by Stilson and Smith [11, 3] is used which aims to simulate the analog Moog ladder filter proposed by Robert Moog in 1965 [1]. In the next sections the Moog filter refers to the Stilson and Smith structure.

2.1. Parameters

The filter has two control parameters, the cutoff frequency f_c and the resonance factor K. The cutoff frequency is present at different places in the filter structure. Stilson and Smith [3] have developed a useful compromise first-order filter that has mostly independent control of the resonance value with the cutoff frequency. The filter coefficients h_0, h_1 and h_2 are parameterized to set the filter behavior close to the expected cutoff frequency. They are defined as

$$h_0 = \frac{\omega_c}{1.3} \qquad h_1 = \frac{0.3\,\omega_c}{1.3} \qquad h_2 = 1 - \omega_c, \tag{1}$$

where ω_c is the angular cutoff frequency related to f_c and the sampling frequency f_s by

$$\omega_c = \frac{2\pi f_c}{L f_s} \qquad f_c = \frac{\omega_c L f_s}{2\pi}. \tag{2}$$

Here, L denotes the oversampling factor and is set to 2. The oversampling helps to achieve stability for high values of ω_c and K.

The resonance parameter K is located in the feedback loop, as visible in Fig. 1. It controls the prominence of the resonance overshoot. It ranges from 0 to 1. Values above 1 lead to self-oscillation and instability.

2.2. Structure

The filter is composed of four cascaded first-order low-pass filters with a feedback loop over the whole cascade. A non-linearity expressed as an hyperbolic tangent is present in the loop to provide a ceiling of the feedback signal. A unit delay is present in the feedback loop to ensure the feasibility of the filter as shown in Fig. 1. Figure 2 illustrates the detail of a stage of the Moog ladder filter.

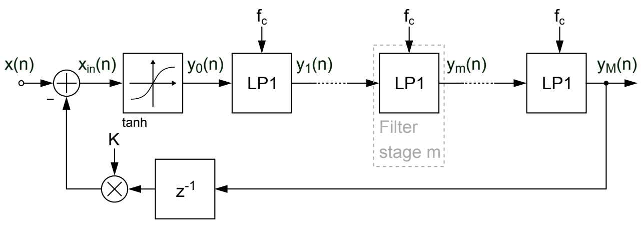


Figure 1: *Block diagram of a Mth order Moog filter structure.*

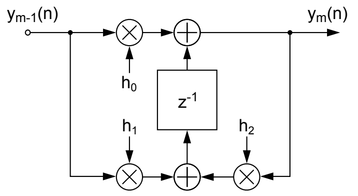


Figure 2: *Block diagram of the mth filter stage.*

The following difference equations describe the signals visible in Fig. 1:

$$x_{\text{in}}(n) = x(n) - K y_4(n-1), \tag{3}$$

$$y_0(n) = \tanh\left(x_{\text{in}}(n)\right), \tag{4}$$

$$y_m(n) = h_0 y_{m-1}(n) + h_1 y_{m-1}(n-1) + h_2 y_m(n-1), \tag{5}$$

where $x(n)$ denotes the input signal and $y_m(n)|_{m=1\cdots 4}$ the outputs of the filter stages from 1 to 4.

3. BACKPROPAGATION

The backpropagation of the error calculated by a loss function through the system allows the adaption of the control parameters to match a target sound. For this purpose, the partial derivatives of the filter output w.r.t. its control parameters must be calculated.

3.1. Partial Derivatives

In this section, the expressions of the partial derivatives w.r.t. the control parameters ω_c and K are derived . It is decided for simplicity to calculate the partial derivative against the angular frequency ω_c defined in Eq. (2).

At first, the derivatives of the filter coefficients h_0, h_1 and h_2 w.r.t. ω_c need to be calculated as follows:

$$\frac{\partial h_0}{\partial \omega_c} = \frac{1}{1.3}, \qquad \frac{\partial h_1}{\partial \omega_c} = \frac{0.3}{1.3}, \qquad \frac{\partial h_2}{\partial \omega_c} = -1. \tag{6}$$

Based on Eq. (5), the partial derivative of an output of a filter stage w.r.t. ω_c

$$\begin{aligned}\frac{\partial y_m(n)}{\partial \omega_c} =& \frac{\partial h_0}{\partial \omega_c} y_{m-1}(n) + \frac{\partial h_1}{\partial \omega_c} y_{m-1}(n-1) + \\ & \frac{\partial h_2}{\partial \omega_c} y_m(n-1) + h_0 \frac{\partial y_{m-1}(n)}{\partial \omega_c} + \\ & h_1 \frac{\partial y_{m-1}(n-1)}{\partial \omega_c} + h_2 \frac{\partial y_m(n-1)}{\partial \omega_c},\end{aligned} \tag{7}$$

where m denotes the index of the filter stages from 1 to 4 and the expression $\frac{\partial y_m(n)}{\partial \omega_c}\big|_{m=0}$ is given by

$$\frac{\partial y_0(n)}{\partial \omega_c} = \left[1 - \tanh(x_{in}(n))^2\right] \frac{\partial x_{in}(n)}{\partial \omega_c}, \tag{8}$$

where

$$\frac{\partial x_{in}(n)}{\partial \omega_c} = -K \frac{\partial y_4(n-1)}{\partial \omega_c}. \tag{9}$$

For the initialization, as $n = 1$, the partial derivatives are initialized as follows:

$$\begin{aligned}\left.\frac{\partial y_m(n)}{\partial \omega_c}\right|_{n=1} =& \frac{\partial h_0}{\partial \omega_c}\, y_{m-1}(1) + \, h_0\, \left.\frac{\partial y_{m-1}(n)}{\partial \omega_c}\right|_{n=1} \\ =& \frac{1}{3} \sum_{k=0}^{m-1} h_0^k y_{m-1-k}(1).\end{aligned} \tag{10}$$

In the next step, the partial derivatives w.r.t. K are calculated and given by

$$\begin{aligned}\frac{\partial y_m(n)}{\partial K} =& h_0 \frac{\partial y_{m-1}(n)}{\partial K} + h_1 \frac{\partial y_{m-1}(n-1)}{\partial K} + \\ & h_2 \frac{\partial y_m(n-1)}{\partial K},\end{aligned} \tag{11}$$

where m denotes the index of the filter stages from 1 to 4 and the expression $\left.\frac{\partial y_m(n)}{\partial K}\right|_{m=0}$ is given by

$$\begin{aligned}\frac{\partial y_0(n)}{\partial K} &= \frac{\partial}{\partial K}\left[\tanh\left(x_{in}(n)\right)\right] \\ &= \left[1 - y_0(n)^2\right]\left[-y_4(n) - K\frac{\partial y_4(n-1)}{\partial K}\right],\end{aligned} \tag{12}$$

where

$$\begin{aligned}\frac{\partial x_{in}(n)}{\partial K} &= \frac{\partial}{\partial K}\left[x(n) - K y_4(n-1)\right] \\ &= -K\frac{\partial y_4(n-1)}{\partial K} - y_4(n-1).\end{aligned} \tag{13}$$

As $n = 1$, the partial derivatives of the individual filter stages m ranging from 1 to 4 are initialized as follow:

$$\left.\frac{\partial y_m(n)}{\partial K}\right|_{n=1} = h_0\, \left.\frac{\partial y_{m-1}(n)}{\partial K}\right|_{n=1}, \tag{14}$$

and the expression $\left.\frac{\partial y_m(n)}{\partial K}\right|_{m=0,n=1}$ is given by

$$\left.\frac{\partial y_0(n)}{\partial K}\right|_{n=1} = -y_M(1)\left[1 - y_0(1)^2\right] \tag{15}$$

for a filter of order M.

It is noteworthy to mention that the expressions of partital derivatives are required to manually construct the backward propagation of the gradients in an environment where automatic derivatives of the forward functions are not calculated, for e.g. MatConvNet [12]. ML environments like Keras[13] and PyTorch [14] provide automatic differentiation of modules constructed by the functions available in their respective packages. However, those environments also offer the possibility to create or alter the backward propagation function, if the automatic backpropagation do not show a stable convergent behavior.

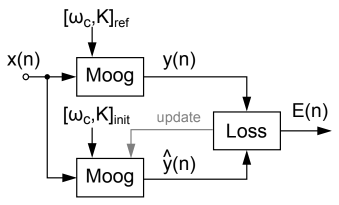


Figure 3: *Simple schematic of a Moog filter adaptation process.*

3.2. Adaptive Moog Filter

In this section, the goal is to verify the functionality of the backpropagation method with the help of an adaptive Moog filter and determine if any gradient tweaking or conditioning is necessary for a simple parameter learning problem . The control parameters of the Moog filter are adapted via backpropagation and a simple parameter update algorithm based on the derivations shown in the previous section is performed.

To adapt a Moog filter, a set of ground truth control parameters $[\omega_c, K]_{\text{ref}}$ are defined initially as shown in Fig. 3. Using a random mixture of various basic signals as the input signal $x(n)$ and the ground truth parameters, the output of Moog filter $y(n)$ is generated and used as the ground truth signal. It is noteworthy to mention that the control signal is oversampled by a factor of $L = 2$ and the corresponding parameters are adjusted. The Moog filter parameters are then initialized with $[\omega_c, K]_{\text{init}}$ and the estimated output signal $\hat{y}(n)$ is compared to the ground truth. The corresponding loss function is given by

$$E = \frac{1}{2}\sum_{n=1}^{N}(|\hat{y}(n)| - |y(n)|)^2, \tag{16}$$

where E denotes the error and $|\cdot|$ denotes the absolute value. The derivative of the error w.r.t. the estimated signal is given by

$$\frac{\partial E}{\partial \hat{y}(n)} = (|\hat{y}(n)| - |y(n)|)\cdot \text{sign}(\hat{y}(n)), \tag{17}$$

where the function $\text{sign}(\hat{y}(n))$ denotes the sign of the estimated signal sample.

The required gradients $\frac{\partial E}{\partial \omega_c}$ and $\frac{\partial E}{\partial K}$ w.r.t. the cutoff frequency f_c and feedback coefficient K are calculated with the help of the chain rule of derivatives and can be given by

$$\frac{\partial E}{\partial \omega_c} = \sum_{n=1}^{N}\frac{\partial E}{\partial \hat{y}(n)}\frac{\partial \hat{y}(n)}{\partial \omega_c}, \tag{18}$$

$$\frac{\partial E}{\partial K} = \sum_{n=1}^{N}\frac{\partial E}{\partial \hat{y}(n)}\frac{\partial \hat{y}(n)}{\partial K}. \tag{19}$$

It is noteworthy to mention that the gradient $\frac{\partial E}{\partial \omega_c}$ is heavily weighted during parameter update such that the initial samples are given more importance. The altered expression can be given by

$$\frac{\partial E}{\partial \omega_c} = \sum_{n=1}^{N}\frac{\partial E}{\partial \hat{y}(n)}\frac{\partial \hat{y}(n)}{\partial \omega_c}\frac{1}{n^k}, \tag{20}$$

where n denotes the sample index and k denotes a positive integer. This change avoids a possible gradient explosion and ensures a stable parameter update and smooth convergence. The expressions in Eq. (20) and Eq. (19) can be derived further with the help of Eq. (17), Eq. (7), and Eq. (11). Finally, gradient clipping is also performed on both the expressions from Eq. (20) and Eq. (19) in order avoid any large gradient jumps and ensure a stable convergence. This can be expressed as

$$\frac{\partial E}{\partial \omega_c} = \min\left(\alpha_{\omega_c}, \left|\frac{\partial E}{\partial \omega_c}\right|\right)\cdot \text{sign}\left(\frac{\partial E}{\partial \omega_c}\right), \tag{21}$$

$$\frac{\partial E}{\partial K} = \min\left(\alpha_{K}, \left|\frac{\partial E}{\partial K}\right|\right)\cdot \text{sign}\left(\frac{\partial E}{\partial K}\right), \tag{22}$$

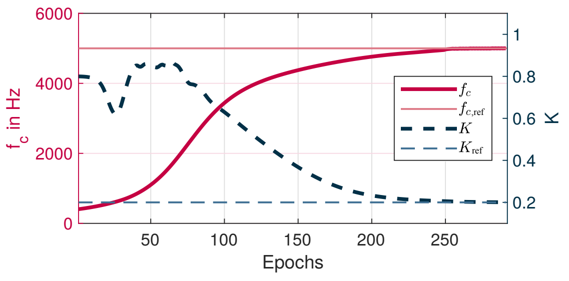
where α_{ω_c} and α_K are two positive small fractional scalars.

The parameters are finally updated using the gradient descent method given by

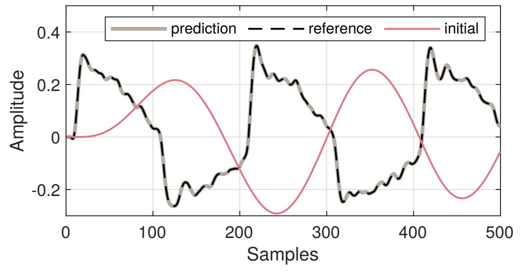
$$\omega_c := \omega_c - \eta_{\omega_c}\cdot\frac{\partial E}{\partial \omega_c}, \tag{23}$$

$$K := K - \eta_K\cdot\frac{\partial E}{\partial K}, \tag{24}$$

where η_{ω_c} and η_K are the corresponding learning rates.



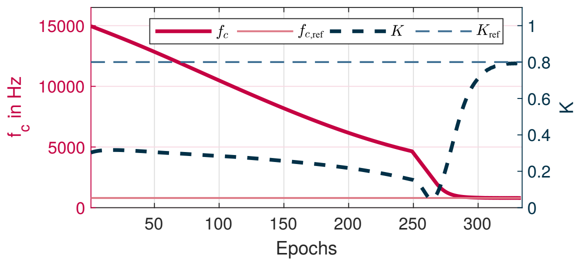
(a) *Parameter adaption curves per epoch.*



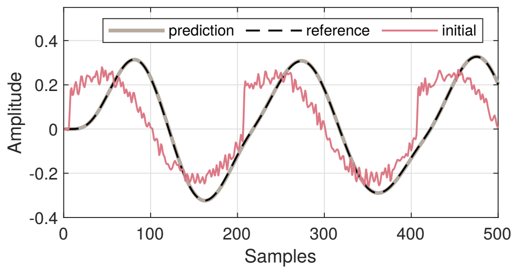
(b) *Predicted, reference and initial signals.*

Figure 4: *Adaption example for:* $f_{c,init} = 400$ *Hz,* $K_{init} = 0.8$, $f_{c,ref} = 5$ *kHz,* $K_{ref} = 0.2$

Figures 4 (a) and 4 (b) show two examples of the evolution of parameters during the adaption process per epoch. In the first example, the target cutoff frequency is set quite above the initial cutoff frequency while the target resonance coefficient is set much below the initial value. In the second example, the target cutoff frequency is set quite below the initial cutoff frequency while the target resonance coefficient is set much above the initial value. In both examples, one can see that the resonance parameter do not converge to a solution until the cutoff frequency gets close enough to its target. This is primarily because of a larger overall error gradient when the cutoff frequencies are far apart. The corresponding initial, reference, and predicted signals are depicted in Fig. 5 (a) and Fig 5 (b).



(a) *Parameter adaption curves per epoch.*



(b) *Predicted, reference and initial signals.*

Figure 5: *Adaption example for: $f_{c,init} = 15\,kHz$, $K_{init} = 0.3$, $f_{c,ref} = 800\,Hz$, $K_{ref} = 0.8$*

4. INTEGRATION IN MACHINE LEARNING

Figure 6 shows a block diagram to illustrate an example of integration of a differentiable subtractive synthesizer with a DL model. During training, the DL model learns to map a reference audio signal y to a set of estimated parameters $\hat{p}$. When applied to the synthesizer, this set of parameter produces an estimated audio output $\hat{y}$ that matches a reference audio input. This process, called tone matching or timbre matching, uses a loss function to evaluate the similarity between reference and estimated outputs. To achieve an update of the DL model weights based on this loss, the error needs to be backpropagated through the synthesizer with regard to its parameters. This should help the timbre matching process to perform better than a system that uses only parameter based loss [15].

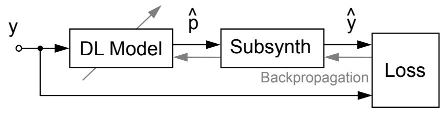


Figure 6: *Simple block diagram of a DL model for subtractive synthesizer parameter estimation.*

The audio signals are not directly fed into the DL model but are subject to pre-processing. A spectro-temporal representation is key to separate the timbre information and the slower temporal changes of the sounds.

Tone Matching or Timbre Matching, has been performed using classical optimization methods but recent approaches tend to use DL methods. While a DL model requires a lot of time for training because of the inclusion of the synthesizer in its end-to-end training process, its performance during testing or evaluation can be quite accurate and fast, particularly if the trained model is small. A DL model can lead to a qualitatively better performance than stochastic optimization based methods [16]. In this work though, instead of the entire synthesizer, only the Moog filter is experimented with and a simple sample based loss function is used to drive an end-to-end learning process.

5. CNN BASED PARAMETER ESTIMATION

In this section, an example application is described where a convolutional neural network (CNN) is used to estimate the control parameters of a Moog filter and Fig. 7 shows the corresponding block diagram. Initially, a dataset is created with a set of predefined input signals x and ground truth control parameters $[\omega_c, K]_{\mathrm{ref}}$ for the Moog filter. The set of output signals y from the filter are collected and their spectral representations Y are used as the input to a CNN. The estimated parameters $[\hat{\omega}_c, \hat{K}]$ from the CNN and the corresponding set of input signals x are used with the Moog filter to generate the estimated output signal $\hat{y}$, The loss computed between y and $\hat{y}$ is used to train the CNN. It is important to mention that a loss between $[\omega_c, K]_{\mathrm{ref}}$ and $[\hat{\omega}_c, \hat{K}]$ can drive the training process, but the goal of this work is to illustrate a successful training via backpropagation through the Moog filter. Additionally, if a CNN has to estimate more control parameters for a complex synthesizer in any later application, the problem might become ill-posed due to a possibility of many parametric solutions. A direct loss between audio signals or their spectral representations should be better for the semantics of sound perception. Both of the loss functions can be used together as well.

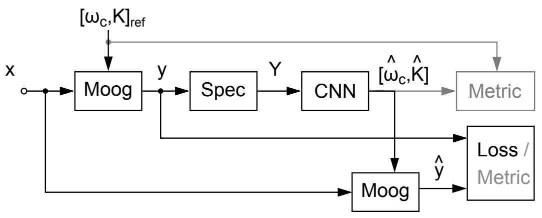


Figure 7: *Block diagram of a CNN integration for Moog filter parameter estimation.*

5.1. Dataset

The dataset used here is composed of 280 audio files at a sample rate of 44.1 Hz. These files are generated using MATLAB. An oscillator produces randomly weighted combinations of five different waveforms (sine, triangle, sawtooth, rectangle and white noise) at 440 Hz that are used as input for the Moog filter. These weights are set to add up to 1. The control parameters of the filter are selected uniformly in the range $[100, 15000]$ Hz for f_c and $[0.1, 0.9]$ for K. They are paired randomly to generate the reference or ground truth audio signals.

As pre-processing, magnitude spectrograms of the output signals from the filter are computed. A window size of 1024 samples, an overlap of 512 samples, and a 256 points DFT are used. The resulting matrix is then resized to match the input of the CNN (128×64). Finally, 256 samples of the input and output audio signals are selected during training to compute the loss. More samples result in longer training times without adding significant improvements.

5.2. CNN Architecture

The architecture of the CNN is illustrated in Fig. 8. The input to the CNN is a resized Spectrogram of the Moog filter output signal. The network has a simple feedforward architecture composed of four blocks. In each of the first three blocks, a convolution layer (Conv) is used followed by a rectified linear unit (ReLU) and a pooling layer (Pool). The Conv layers use filters of size $3 \times 3 \times d$, where d denotes the input feature map depth, and a stride of 2. 96 such filters are used in each Conv layer. The Pool layers perform maximum pooling and use a kernel size of 2×2 with a stride of 2. The ReLU layer uses a leakage factor of 0. The stride in Conv and Pool layers ensures the reduction of spatial dimensions of the feature maps. The final block is composed of a *fully connected layer* (FC) followed by a ReLU layer. The FC layer vectorizes the input feature map and delivers an output vector of the required size. A sigmoid layer can be used instead of a ReLU layer for faster convergence but boundary values of the estimated parameters suffer due to the saturation regions of the function.

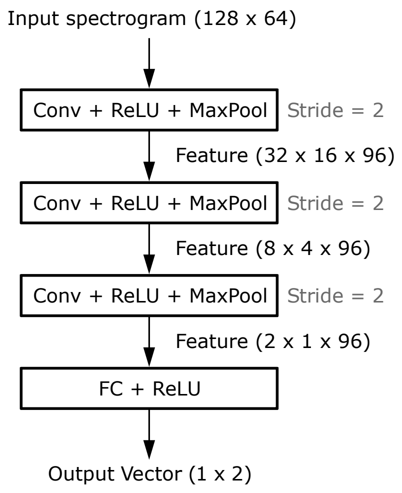
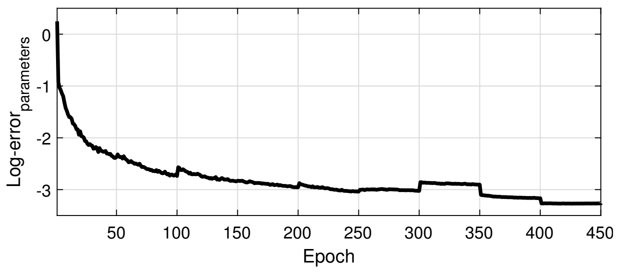


Figure 8: *Block diagram of the CNN architecture.*

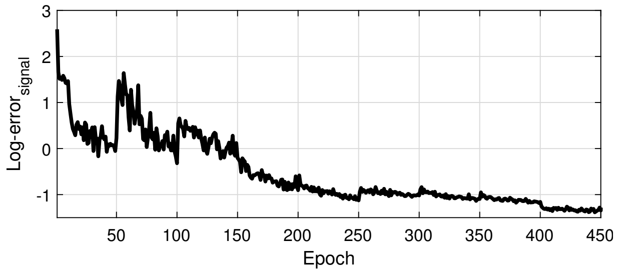
5.3. Model Performance

Initially, CNN models were constructed to be trained with raw audio snippets of multiple sample lengths. For longer audio snippets ($>$ 1024 samples), the models converged initially but stagnated quickly. The models were then trained with the magnitude responses of the raw audio snippets. While the networks were able to reduce the loss and show convergent behavior, particularly for longer audio, the final results were not quite good. Finally, the representation described in Section 5.2 is selected as the CNN input. In order to train the model, the loss function given by Eq. (16) in section 3.2 is used. The gradients w.r.t. the parameters are also constrained by gradient clipping for a stable convergence. Similar to the adaptive Moog filter, the gradient w.r.t. the cut-off frequency is exponentially suppressed in order to assure a more stable convergence. The given model is then trained for about 1000 epochs where the learning rate is reduced linearly from 10^{-4} to 10^{-8}. A batch size of 8 is selected which results in 35 iterations per epoch and the *adam* optimizer [17] is used as the update method. The model is built in the last version of MatConvNet [12] deep learning toolbox for MATLAB. The training is performed on a machine with a Nvidia QUADRO RTX8000 graphical processing unit.

To measure the model performance, sum of squared error (SSE) function between the estimated outputs and the expected ground truths is used. Figure 9 (a) shows the logarithm of the SSE between the estimated $[\hat{\omega}_c, \hat{K}]$ and expected $[\omega_c, K]_{\text{ref}}$ parameters while Fig. 9 (b) shows the logarithm of the SSE between the estimated $\hat{y}(n)$ and expected $y(n)$ signals, measured over the epochs.



(a) *Parameters error.*



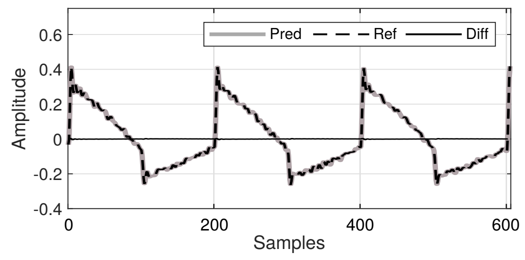
(b) *Signal Error.*

Figure 9: *Training error Curves over epochs.*

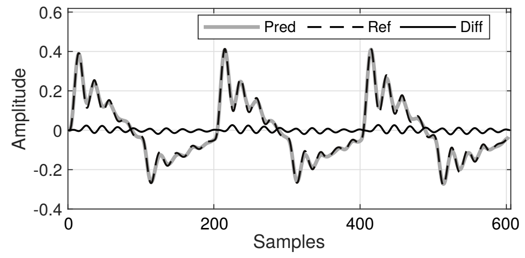
Both of the errors decrease relatively faster within the first 200 epochs and then converges slowly. The training can be performed until 600 epochs beyond which no improvement is achieved. As mentioned previously, the signal error for backpropagation is computed with 256 audio samples. Calculation of the loss with more samples might improve the results but will drastically increase the training time.

Figure 10 shows three audio examples with different types of signals and different sets of parameters. The first 1024 samples of the signals are shown. Figure 10 (a) shows an example ground truth signal generated by a Moog filter for a cutoff frequency of 11549 Hz and resonance coefficient of value 0.309. Based on the corresponding estimated cutoff frequency of 11564 Hz and resonance coefficient of value 0.315, the predicted signal is constructed, denoted by Pred in the plot. The difference between the reference and estimated signal, denoted by Diff, shows a near perfect reconstruction. Figure 10(b) shows another audio example where the predicted cutoff frequency of 3948 Hz is close to the ground truth cutoff frequency of 3867 Hz but the predicted resonance coefficient of value 0.624 is not as close to the ground truth value of 0.712. The difference or error signal is more prominent than the previous examples. The third audio example shown in Fig. 10 (c) has additive noise but the network prediction is still quite good as it closely matches the ground truth values and the difference signal has a low amplitude.

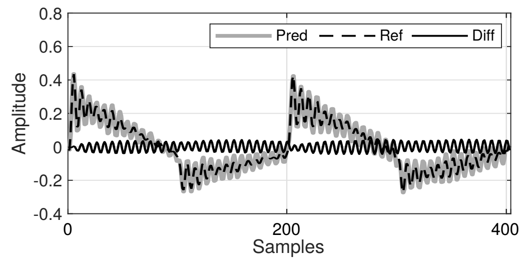
In general, it can be concluded that the network performs ad-



(a) $\hat{f}_c = 11564$, $\hat{K} = 0.315$,
$f_{c,ref} = 11549$, $K_{ref} = 0.309$



(b) $\hat{f}_c = 3948$, $\hat{K} = 0.624$,
$f_{c,ref} = 3867$, $K_{ref} = 0.712$



(c) $\hat{f}_c = 8858$, $\hat{K} = 0.963$,
$f_{c,ref} = 9001$, $K_{ref} = 0.856$

Figure 10: *Direct comparison of signal examples.*

mirably for most examples across all parameter values in experiment. However, some uncertain results are also observed particularly around the lower values of the cutoff frequency. Training with a higher number of samples or including more examples might resolve such uncertainties and can be experimented as part of the future work. Additionally, the network training should also be performed with other forms of loss functions in time or frequency domain to find the best model in terms of error reduction and convergence. Further experiments should be conducted with multiple input representations for the CNN.

6. CONCLUSION

This work is a step toward the modeling of differentiable subtractive synthesizers, which would allow to perform tone matching based on psychoacoustic loss functions. Hence, the presented work should be extended towards the parameter estimation for an entire synthesizer and inclusion of loss functions considering audio semantics. As further work, other blocks of the synthesizer like oscillators, envelope generators and audio effects should be differentiated with respect to their parameters. Experiments should be conducted towards computation of the appropriate input representation for a given CNN model. Since synthesizers can have a large number of time or frequency dependent parameters, multiple time-frequency representations as the model input should be studied. Multiple loss functions in time and frequency domain should also be studied in order to find the most appropriate combination for training a model. Finally the CNN model and its modules should be studied in order to improve the estimation performance. Finally, the synthesizer could be implemented in a PyTorch environment and the performance could be observed and improved.

7. REFERENCES

[1] Robert A. Moog, "A voltage-controlled low-pass high-pass filter for audio signal processing," *Journal of the Audio Engineering Society*, october 1965.

[2] Effrosyni Paschou, Fabian Esqueda, Vesa Välimäki, and John Mourjopoulos, "Modeling and measuring a moog voltage-controlled filter," december 2017, pp. 1641–1647.

[3] Tim Stilson and Julius Smith, "Analyzing the moog VCF with considerations for digital implementation," 1996.

[4] Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts, "DDSP: differentiable digital signal processing," in *International Conference on Learning Representations*, 2020.

[5] Purbaditya Bhattacharya, Patrick Nowak, and Udo Zölzer, "Optimization of cascaded parametric peak and shelving filters with backpropagation algorithm," in *23rd International Conference on Digital Audio Effects (DAFx)*, september 2020.

[6] Boris Kuznetsov, Julian Parker, and Fabian Esqueda, "Differentiable IIR filters for machine learning applications," in *23rd International Conference on Digital Audio Effects (DAFx)*, 2020.

[7] Cheng-Zhi Anna Huang, David Duvenaud, Kenneth C. Arnold, Brenton Partridge, Josiah W. Oberholtzer, and Krzysztof Z. Gajos, "Active learning of intuitive control knobs for synthesizers using gaussian processes," New York, NY, USA, 2014, Association for Computing Machinery.

[8] Matthew D. Hoffman and Perry R. Cook, "Feature-based synthesis: Mapping acoustic and perceptual features onto synthesis parameters," in *International Conference on Mathematics and Computing*, 2006.

[9] Matthew Yee-King and Martin S. Roth, "Synthbot: An unsupervised software synthesizer programmer," 2009.

[10] Philippe Esling, Naotake Masuda, Adrien Bardet, Romeo Despres, and Axel Chemla-Romeu-Santos, "Universal audio synthesizer control with normalizing flows," in *International Conference on Digital Audio Effects (DaFX 2019)*, Birmingham, United Kingdom, september 2019.

[11] Vesa Välimäki, Stefan Bilbao, Julius O. Smith, Jonathan S. Abel, Jyri Pakarinen, and David Berners, *DAFX: Digital Audio Effects*, chapter 12: Virtual analog effects, pp. 279–320, John Wiley & Sons, Ltd, 2011.

[12] Andrea Vedaldi and Karel Lenc, "MatConvNet: Convolutional neural networks for MATLAB," in *Proceedings of the 23rd ACM International Conference on Multimedia*, New York, NY, USA, 2015, MM '15, p. 689–692.

[13] François Chollet et al., "Keras," `https://keras.io`, 2015.

[14] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[15] Matthew John Yee-King, Leon Fedden, and Mark d'Inverno, "Automatic programming of vst sound synthesizers using deep networks and other techniques," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 2, pp. 150–159, 2018.

[16] Matthew Yee-King and Martin Roth, "A comparison of parametric optimisation techniques for musical instrument tone matching," in *Audio Engineering Society Convention 130*, january 2011.

[17] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.