DIFFERENTIABLE IIR FILTERS FOR MACHINE LEARNING APPLICATIONS

Boris Kuznetsov, Julian D. Parker and Fabián Esqueda

Native Instruments GmbH
Berlin, Germany
firstname.lastname@native-instruments.de

ABSTRACT

In this paper we present an approach to using traditional digital IIR filter structures inside deep-learning networks trained using back-propagation. We establish the link between such structures and recurrent neural networks. Three different differentiable IIR filter topologies are presented and compared against each other and an established baseline. Additionally, a simple Wiener-Hammerstein model using differentiable IIRs as its filtering component is presented and trained on a guitar signal played through a *Boss DS-1* guitar pedal.

1. INTRODUCTION

The deep learning approach to machine learning has made great strides in many application areas, including signal processing, over the past 10 years. This approach has had success both in generalized audio modelling tasks [1, 2] and in more constrained musical signal processing tasks [3–7].

When applied to 1-dimensional signals, the common building blocks of deep-learning systems can broadly be thought of as non-linear extensions of some familiar signal processing concepts. For example, convolutional neural networks (CNNs) can be thought of as a type of non-linear FIR filter, and recurrent neural networks (RNNs) as a type of non-linear IIR filter. Building up a deep-learning model can be approached much like building up a signal processing chain. Recent work [8] has made this analogy explicit by pointing out that the more general and expressive components of typical deep learning models can be substituted with much more constrained signal processing components in suitable problems, thus encoding structural knowledge in the model and potentially making training easier and models smaller. Engel et al. demonstrate this by constructing a synthesis system using *differentiable* signal processing components including FIR filters, oscillators and convolutions.

In this paper we propose the use of IIR filters in the differentiable DSP (DDSP) context. We discuss how the analogy between IIRs and RNNs present a number of useful insights into how such structures should be trained. We then present some approaches for constructing IIR filters in this context, and compare their performance when applied to several simple example problems. These example problems include fitting linear transfer functions and approximating the behaviour of a non-linear system using a Wiener-Hammerstein model.

2. IIR FILTERS AS RECURRENT NEURAL NETWORKS

The *Elman network* is one of the earliest and simplest forms of a *recurrent neural network*, introduced in 1990 as a way to model human language [9]. It is described by the difference equation:

$$\mathbf{h}[n] = \sigma_h(W_h\mathbf{h}[n-1] + U_h\mathbf{x}[n] + \mathbf{b}_h) \quad (1)$$

$$\mathbf{y}[n] = \sigma_y(W_y\mathbf{h}[n] + \mathbf{b}_y), \quad (2)$$

where the vectors $\mathbf{h}$, $\mathbf{y}$, $\mathbf{x}$ are the hidden states, outputs and inputs of the system, respectively. The matrices W, U are the trainable weights of the RNN, and the vectors $\mathbf{b}$ are the biases. The functions σ are (usually non-linear) activation functions.

Given some familiarity with signal processing, it should be clear that if we choose linear activation functions σ and set the biases $\mathbf{b}$ to zero, we essentially have an all-pole linear IIR filter:

$$\mathbf{h}[n] = W_h\mathbf{h}[n-1] + U_h\mathbf{x}[n] \quad (3)$$

$$\mathbf{y}[n] = W_y\mathbf{h}[n]. \quad (4)$$

This illustrates the strong parallel between RNNs and IIR filters - they are both recursive systems that operate element-by-element on a sequence of numerical data, storing information in their internal states along the way. Hence, if we want to utilize IIR filters in a machine-learning context we can treat them as a specialized type of RNN and re-use the techniques and knowledge already developed around RNNs.

2.1. Training

RNNs are generally trained using backpropagation through time (BPTT), which can be trivially regarded as a regular backpropagation through the computational graph of the network when unrolled over a number of time steps. Instead of applying BPTT on the whole input time-sequence, the dataset is typically split up into short sequences (typically between 512 and 2048 samples for audio problems), over which the BPTT is applied [10] independently. This is known as truncated backpropagation through time (TBPTT). This process can be applied to differentiable IIR filters unmodified.

If multiple filters are to be used, the forward pass of the network can proceed as in a normal signal processing graph. Looping over the input sequences can either be performed for the whole sequence for each filter in turn (analogous to block-based processing in an audio graph), or sample by sample. The computational graphs produced in either approach are equivalent.

2.2. Vanishing or exploding gradients

Typically, RNN structures suffer from two major problems during training - vanishing gradients and exploding gradients. The former is largely mitigated when training a linear IIR filter, whilst the latter still requires some consideration.

Vanishing gradients result from the repeated composition of non-linear activation functions in a deep network, exacerbated by the graph unrolling utilized in BPTT. Since linear IIRs contain no activation functions, they do not suffer from vanishing gradients.

Exploding gradients are typically encountered when the recursion learned by an RNN is unstable, resulting in unbounded growth of the states and output. This is still an issue for IIR filters, but it can largely be mitigated by sensible initialization of coefficients, or by working with higher-level parameters for which stability bounds are easily derived.

2.3. Implementation

To use IIR structures in a machine-learning context, they should be implemented within a framework such as PyTorch or Tensorflow [11,12] that allows automatic differentiation of computational graphs. This is a relatively trivial task as the structures contain only basic arithmetic operations for which the frameworks already have derivatives. Therefore, we only need to implement the forward pass of the system, which is equivalent to its difference equation. Reference implementations for PyTorch are provided at the accompanying website [1].

3. IIR STRUCTURES FOR DDSP

In this section we present a number of IIR filter structures that can be used in a differentiable DSP context.

3.1. First-order section

The general z-domain transfer function for a first-order IIR filter having one pole and one zero is given by:

$$H(z) = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}}. \tag{5}$$

In practice, this transfer function can be implemented using the following difference equation:

$$\begin{aligned} y[n] &= b_0 x[n] + h[n-1] \\ h[n] &= b_1 x[n] - a_1 h[n-1]. \end{aligned} \tag{6}$$

3.1.1. Initialization

It has been shown that initializing the weights of a layer in a neural network with a random distribution is generally desirable [10]. Since the unconditional stability of the system depends solely on the value of a_1, random initialization poses no problems for coefficients b_0 and b_1. For the case of a_1, it can be initialized to any random value that satisfies the stability condition $|a_1| < 1$.

3.2. Direct-Form second-order section

The z-domain transfer function of a general second-order IIR filter having 2 poles and 2 zeros is given by:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}. \tag{7}$$

Since this transfer function is the ratio of two quadratic functions, it is commonly referred to as a *biquad* filter.

[1] https://github.com/boris-kuz/differentiable_iir_filters

One approach would be to train the transfer function directly via the parameters b_0, b_1, b_2, a_1 and a_2. The natural structure for this application would one of the four direct-form IIR topologies, as they contain these parameters directly as coefficients. For this work, the Transposed Direct Form-II (TDF-II) was chosen, given its desirable numerical properties [13]. The difference equation for (7) using TDF-II is given by:

$$\begin{aligned} y[n] &= b_0 x[n] + h_1[n-1] \\ h_1[n] &= b_1 x[n] + h_2[n-1] - a_1 y[n] \\ h_2[n] &= b_2 x[n] - a_2 y[n]. \end{aligned} \tag{8}$$

3.2.1. Initialization

Whilst random initialization can be applied to the coefficients b_0, b_1, b_2 with no restrictions, it poses a problem for coefficients a_1 and a_2. These coefficients define the poles and hence the stability of the system. We need to initialize them in such a way as to ensure that the resulting poles fall inside the unit circle. By solving the quadratic equation in the denominator of the transfer function, we can derive a closed-form expression for the magnitude of the largest pole radius for a given set of coefficients. This expression can be written as:

$$r = \begin{cases} \frac{|a_1|}{2} & \text{if } \; a_1^2 = 4a_2 \\ \frac{|a_1| + \sqrt{a_1^2 - 4a_2}}{2} & \text{if } \; a_1^2 > 4a_2 \\ \sqrt{a_2} & \text{if } \; a_1^2 < 4a_2. \end{cases} \tag{9}$$

The first and last condition dictate $|a_1| \leq 2$ and $|a_2| \leq 1$. Together with the second condition, we get an area of convergence in the shape of a triangle. We can now choose a region inside this triangle for which the stability of the filter is guaranteed. If we place the constraint that a_1, a_2 should be initialised in the same range, the region defined by $|a_1| \leq 0.5$ and $|a_2| < 0.5$ provides the greatest area within the stable limits whilst still being centered on the origin.

3.3. SVF

An alternative approach to producing a second-order filter is to use a filter structure with higher-level parameters. A good fit for this task is the digital state-variable filter (SVF) [14]. This structure has the advantage of decoupled control of both pole frequency and magnitude, along with the possibility to freely mix its three outputs to produce arbitrary zeros. This allows it to produce any second-order transfer function, whilst still having easily interpretable parameters.

The difference equation of the SVF is given by:

$$\begin{aligned} y_{\text{BP}}[n] &= \frac{g(x[n] - h_2[n-1]) + h_1[n-1]}{1 + g(g + 2R)} \\ y_{\text{LP}}[n] &= g\, y_{\text{BP}}[n] + h_2[n-1] \\ y_{\text{HP}}[n] &= x[n] - y_{\text{LP}}[n] - 2R y_{\text{BP}}[n] \\ h_1[n] &= 2y_{\text{BP}} - h_1[n-1] \\ h_2[n] &= 2y_{\text{LP}} - h_2[n-1] \\ y[n] &= c_{\text{HP}}\, y_{\text{HP}} + c_{\text{BP}}\, y_{\text{BP}} + c_{\text{LP}}\, y_{\text{LP}}, \end{aligned} \tag{10}$$

where g is a coefficient controlling the cutoff frequency of the filter (usually derived from an analog frequency using prewarping,

but here used directly), and c_{HP}, c_{BP} and c_{LP}, are the mixing coefficients for the three filter outputs. Parameter R sets the damping of the filter, with self-oscillation occurring at $R = 0$. The trainable parameters for the SVF are then c_{HP}, c_{BP}, c_{LP}, R and g.

3.3.1. Initialization

The parameters $c_{\text{LP}}, c_{\text{BP}}, c_{\text{HP}}$ can be initialized randomly without great concern for the range. On the other hand, parameters g and R must be constrained in their initialization to positive values in order to produce a stable filter. Practically, it makes sense to place an upper bound on both g and R upon initialization. Sensible upper bounds are $R = 1/\sqrt{2}$, which corresponds with a Butterworth filter shape, and $g = 1$, which places the cutoff at approximately 12 kHz for a sample rate of 44.1 kHz.

3.4. State-Space Form

The above described direct-form and SVF structures have the ability to fit any transfer function, but are constrained in their topology. If we want to relax this restriction, we can use the generalized state-space form, given by:

$$\begin{aligned} \mathbf{h}[n+1] &= A\mathbf{h}[n] + B\mathbf{x}[n] \\ \mathbf{y}[n] &= C\mathbf{h}[n] + D\mathbf{x}[n] \end{aligned} \tag{11}$$

where A, B, C, D are matrices.

This form has the additional advantage of scaling to arbitrary order to fit higher-order transfer functions.

3.4.1. Initialization

Random initialization is trivial for the B, C, D matrices, but the A matrix must be handled with care as it defines the stability of the system. We must find a bounding value α of the matrix elements a_{ij} such that any random A matrix containing only values in the interval $[-a, a]$ produces an unconditionally stable system.

It is well known that any linear state-space system is stable if $|\lambda_i| < 1, \forall\ i$ where λ_i are the eigenvalues of the A matrix. It is also well known that the Frobenius matrix norm $||.||_{\text{F}}$ provides an upper-bound for the spectral radius, and hence the magnitude of the largest eigenvalue of a matrix. The Frobenius norm for a square matrix of dimension n is given by:

$$||A||_{\text{F}} = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2} \tag{12}$$

Therefore, we can derive the bounding value α by examining the worst-case scenario where $|a_{ij}| = \alpha, \forall\ i, j$, and $||A||_F = 1$, giving:

$$\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |\alpha|^2} = 1 \tag{13}$$

which implies:

$$\alpha = \frac{1}{n}. \tag{14}$$

Hence, we are guaranteed a stable initial system if A is initialized with uniform random values in the interval $[-1/n, 1/n]$, where n is order of the system.

4. EVALUATION

In this section we evaluate the performance of the proposed structures by examining several linear filtering test cases. A 1D convolutional layer, a common building block in machine learning applications, will be used as a baseline method during the evaluations, as in practice it is equivalent to an FIR filter. Additionally, we present an example application of differentiable IIRs for non-linear system identification. As the choice of amount of epochs trained for each model is arbitrary, constrained only by time and computational resources, the models have been trained until at least two models reached a minimum, i.e. a point at which the loss has not improved for at least 100 epochs.

4.1. Training Data

All models were trained on a signal composed of a 90-second long logarithmic sine sweep ranging from 20 to 20 kHz with a constant amplitude followed by 90 seconds of white noise at 0 dB. For validation and comparison, a 60-second long sine-sweep was used. For the non-linear case, a 90-second electric guitar recording was used for training and a different 60-second recording was used for the validation. All examples were implemented at a fixed sample rate of 48 kHz.

During training, these signals were divided into sequences of 2048 samples, which were then organised into batches to utilize GPU parallelism.

4.2. Loss Function and Loss Surface

The loss function chosen for the evaluation was the commonly-used *mean squared error* (MSE), defined as:

$$\mathcal{E}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2, \tag{15}$$

where Y and $\hat{Y}$ are the target signal and the output of the network, respectively. Before evaluation of the MSE for each sequence of data, both Y and $\hat{Y}$ have the first few samples truncated to allow time for the filters to stabilize. Recent research in audio modelling using deep learning has proposed the use of perceptually-informed loss functions [15]. The use of perceptual loss functions, such as A-weighting, can help improve the perceived accuracy of a network without affecting its performance. Due to the domain-specific nature of the networks proposed in this study, an evaluation of the effects of this advanced training technique falls out of scope and is therefore left for future work.

To evaluate the performance of the proposed structures we examine the magnitude of their frequency response after training as well as their loss history, i.e. the evolution of the model loss as a function of epochs. While the loss history provides important insights into the smoothness of the loss surface of the different networks, it does not give a full picture of the network properties given the chosen structure. This is because the choice of optimizer and its associated hyperparameters, such as learning rate, can easily influence the form this plot takes. In [16], Li et al. propose a method to visualize the evolution of the model loss that is not influenced by the choice of optimizer. This method consists in evaluating the loss of the model at points along a line in parameter space connecting the initialized model and the trained model. Models with a monotonously falling loss are expected to behave

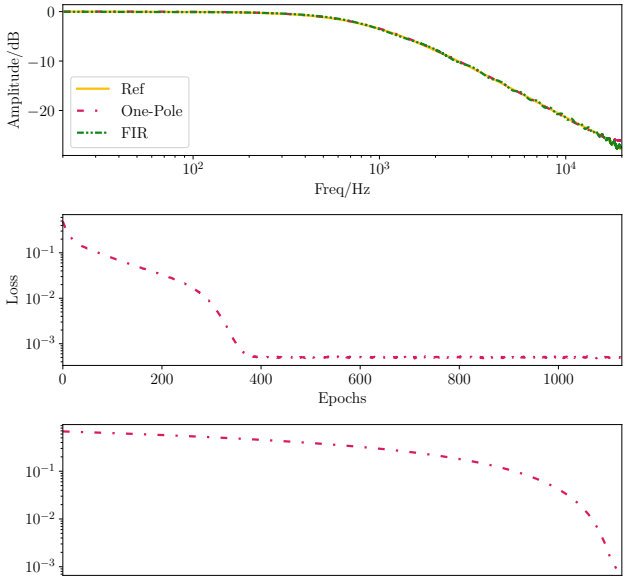

Figure 1: Final frequency response (top) of the reference signal, the FIR-model and the one-pole-model, loss history (middle) and linear interpolation of the loss surface from the initial model to the converged model (bottom) for the model trained on the first-order lowpass.

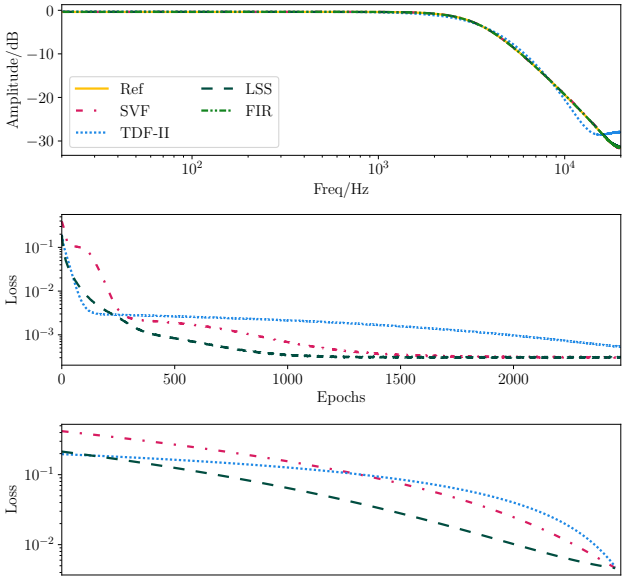

Figure 2: Final frequency response (top), loss history (middle) and linear interpolation of the loss surface from the initial model to the converged model (bottom) for each model trained on the second-order lowpass.

better while training, i.e. converge smoothly to a sensible minimum.

4.3. First-Order Lowpass Filter

In order to validate the differentiable first-order IIR filter, the training signal was processed through a one-pole lowpass with a cutoff frequency at 1 kHz. The reference convolutional layer has a kernel size of 32 and was trained for the same amount of epochs. As can be seen in Figure 1, the differentiable one-pole model achieves a very good fit, only deviating slightly near the Nyquist frequency.

The final MSE values for the trained models are:

Model	Loss
One-Pole	5.061956E-04
FIR	1.832063E-05

4.4. Second-Order Lowpass Filter

The second test case consisted of fitting the response of a single second-order non-resonant lowpass filter with a cutoff frequency at 4 kHz. Four different differentiable structures were trained to match the target response: a TDF-II, an SVF, a linear state-space filter (LSS) of order 2 and a convolutional layer with kernel size of 16. The top part of Fig. 2 shows the magnitude response of the models after training, against that of the reference system. As shown, the SVF and LSS networks achieve a good match of the target response, while DFT-II presents minor deviations in the stopband.

We can gain further insights into these results by observing the evolution of the loss curve during training, depicted in the middle plot of Fig. 2. From this curve we can observe that, while the TDF-II and LSS networks exhibited lower loss values than the SVF during the first epochs, the SVF and LSS converged to a sensible loss much faster than the TDF-II network. These results suggest that either fully restricting the parametrization, as in the SVF, or giving the network complete freedom over both its topology and parametrization, as in the LSS, is more advantageous for training purposes. On the other hand, the TDF-II loss appears to continue decreasing, suggesting that given enough time the model will converge. The loss surface exhibits similar smoothness for all three models as shown in the bottom section of Fig. 2.

The final MSE values for each of the trained models are:

Model	Loss
SVF	3.058363E-04
TDF-II	5.340165E-04
LSS	3.046548E-04
FIR	4.077413E-07

4.5. Three-band Parametric EQ

The second test case consisted of three second-order linear filters in series: a non-resonant highpass filter with a 50-Hz cutoff, a peak filter with a 3-dB gain at 500 Hz and a high-shelf filter with a cutoff at 5 kHz and a gain of –3 dB. The models used to fit the target response were: three TDF-II networks in series, three SVFs in series, a convolutional layer with a kernel size of 1024 samples and an LSS of order 6.

The top part of Fig. 3 shows the magnitude response of the three proposed networks after training and that of the target system. As shown by these results, the TDF-II and LSS networks struggle to fit the target curve particularly below 100 Hz and at the resonant peak. On the other hand, the SVF achieves a good match of the target spectrum, with only minor deviations below

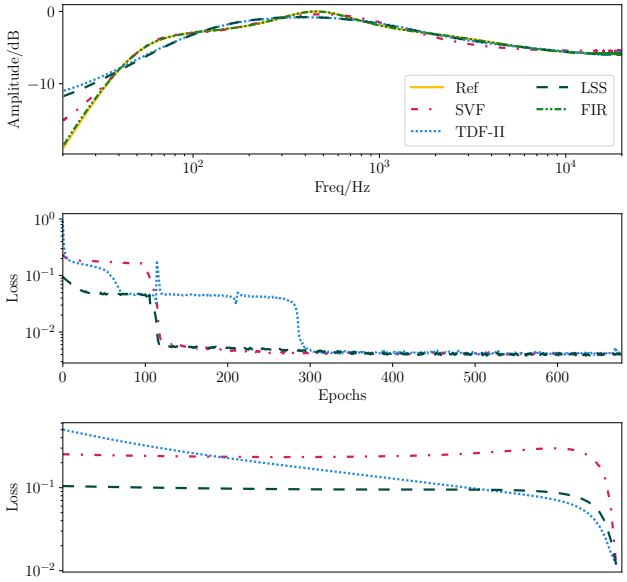

Figure 3: Final frequency response (top), loss history (middle) and linear interpolation of the loss surface from the initial model to the converged model (bottom) for each model trained on the parametric EQ.

Figure 4: Block diagram of a Wiener-Hammerstein model.

approx. 30 Hz. The model loss surfaces, as observed by their linear interpolations the bottom part of Fig. 3, show that both the SVF and the LSS models had a "bump" to overcome, whereas the loss surface of the TDF-II model seems to be smoother. The loss history plot in the middle of Fig. 3 tells a slightly different story, showing the TDF-II model to behave in a more unpredictable manner as compared to the SVF and LSS, which exhibited very similar training performance. In contrast to the previous test case, the loss for the TDF-II model is approximately the same as for the other two, suggesting that for more complex systems the choice of filter structure does not matter as much as for the simple case. While none of the recurrent models fit the target exactly, it is still clear that they approximate the shape nonetheless and are trainable using backpropagation.

The final MSE values for each of the trained models are:

Model	Loss
SVF	3.920308E-03
TDF-II	3.951538E-03
LSS	3.947347E-03
FIR	1.039456E-06

4.6. Wiener-Hammerstein Model

In this section we present an example application of differentiable IIRs within the context of block-oriented nonlinear system modelling using a Wiener-Hammerstein structure. Previous research on virtual analog (VA) modeling has studied the use of block-

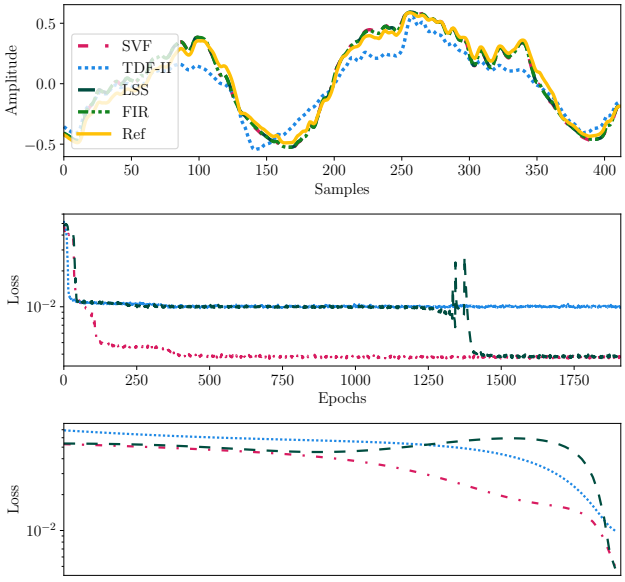

Figure 5: Sample output (top), loss history (middle) and linear interpolation of the loss surface from the initial model to the converged model (bottom) for each Wiener-Hammerstein model trained on the nonlinear signal.

oriented structures to emulate a wide variety of non-linear music systems, such as guitar amplifiers [17] and audio distortion circuits [18–21]. This approach falls under the category of *black-box* modeling, an area of study that deals with the emulation of nonlinear systems based on input–output measurements of the device under study. Figure 4 shows a block diagram representation of a Wiener-Hammerstein model which consists of a static, or memoryless, nonlinearity surrounded by two LTI filters. This same general structure is used, e.g., in [21] to model a tube-based pre-amplifier pedal.

We can design a differentiable Wiener-Hammerstein model by implementing the LTI filters and the static non-linear function in Fig. 4 with differentiable IIR filters and a multi-layer perceptron (MLP) [10], respectively. The proposed architecture has two desirable properties. First, the model is now fully differentiable and all three sections can be trained simultaneously in an end-to-end fashion. Secondly, we do not need to make any assumptions regarding the shape of the non-linear stage. The MLP, being an universal approximator, should be able to learn any arbitrary static non-linear function provided it has enough capacity.

The MLP structure chosen for this specific case consisted of four densely-connected layers, with two being hidden layers of width 20. The first layer maps the input width, which is one in this case, to the width of the hidden layers, i.e. 20. The final layer maps the signal back to the desired output width of one. The activation function was chosen to be $\tanh$ and placed after every hidden layer. Two differentiable second-order IIRs were used to complete the Wiener-Hammerstein model, one before and one after the MLP. This structure can be extended with relative ease, at the cost of increased computational complexity, by increasing the number of filters used and and by modifying the size of the MLP. For the FIR reference model, a kernel size of 128 was chosen for both input and output filters.

The proposed structure was used to model a Boss DS-1 guitar distortion pedal [22]. For this model, all knobs on the pedal were set to 50% of their value. Training was conducted by using a test signal consisting of a 90-second passage of electric guitar playing.

As shown in the top section of Fig. 5, the TDF-II, LSS and FIR models behave quite similarly, while the SVF model diverges slightly. Though none of the models fit the target perfectly, they all perform relatively well. Audio examples for the proposed models and validation signal used can be found in alongside the accompanying code for this paper.

It is notable that the accuracy achieved is comparable to that achieved in [17, 20, 21], but without the use of a specialized measurement signal and the optimization process employed in this previous work. This indicates the suitability of differentiable IIR filters and an end-to-end machine learning approach for this type of black-box modeling of nonlinear systems.

The final MSE for each nonlinear model is:

Model	Loss
SVF	3.690291E-03
TDF-II	1.001991E-02
LSS	3.763787E-03
FIR	4.327687E-04

4.7. Computational Cost

The following table gives an overview of the floating-point operations per sample (adds, multiplies) for each considered structure for a second-order section. The accompanying code has been used as a basis for these operation counts.

Model	Multiplies/Sample	Adds/Sample
SVF	10	8
TDF-II	5	4
LSS	9	6

5. CONCLUSIONS

In this work we made the connection between the RNN, a familiar class of structures in the machine learning domain, and IIR filters, providing several implementations of such filters within a machine learning context. These differentiable IIR filters can be employed as part of established machine learning model structures and training processes, particularly in the audio domain. We evaluated and compared the implementations with a more computationally intensive baseline model and showed that they can, broadly, approach the performance of this baseline. Lastly, we presented an example of how these structures can be integrated within a larger machine learning system by using them along with an MLP to construct a Wiener-Hammerstein model of a *Boss DS-1* pedal.

6. REFERENCES

[1] A. van den Oord et al., "Wavenet: A generative model for raw audio," *CoRR abs/1609.03499*, 2016.

[2] S. Mehri et al., "SampleRNN: An unconditional end-to-end neural audio generation model," in *Proc. Int. Conf. on Learning Representations (ICLR)*, Toulon, France, 2017.

[3] E.-P. Damskägg, L. Juvela, E. Thuillier, and V. Välimäki, "Deep learning for tube amplifier emulation," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Proc. (ICASSP 2019)*, Brighton, UK, 2019, pp. 471–475.

[4] A. Wright, E.-P. Damskägg, and V. Välimäki, "Real-time black-box modelling with recurrent neural networks," in *Proc. 22nd Int. Conf. on Digital Audio Effects (DAFx-19)*, Birmingham, UK, 2019.

[5] V. Välimäki and J. Rämö, "Neurally controlled graphic equalizer," *IEEE/ACM Trans. Audio, Speech, and Lang. Proc,*, vol. 27, no. 12, pp. 2140–2149, 2019.

[6] J. Parker, F. Esqueda, and A. Bergner, "Modelling of nonlinear state-space systems using a deep neural network," in *Proc. 22nd Int. Conf. on Digital Audio Effects (DAFx-19)*, Birmingham, UK, 2019.

[7] M. Martinez Ramirez and J. Reiss, "End-to-end equalization with convolutional neural networks," in *Proc. 21st Int. Conf. on Digital Audio Effects (DAFx-18)*, Aveiro, Portugal, 2018, pp. 296–303.

[8] J. Engel, L. Hantrakul, C. Gu, and A. Roberts, "DDSP: Differentiable digital signal processing," in *Proc. Int. Conf. on Learning Representations (ICLR)*, Addis Ababa, Ethiopia, 2020.

[9] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016, `http://www.deeplearningbook.org`.

[11] A. Paszkeand et al., "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

[12] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, Software available from tensorflow.org.

[13] J. O. Smith III, *Introduction to Digital Filters with Audio Applications*, W3K Publishing, http://www.w3k.org/books/, 2007.

[14] A. Wishnick, "Time-varying filters for musical applications," in *Proc. 17 Int. Conf. Digital Audio Effects (DAFx-14)*, Erlangen, Germany, Sept. 2014, pp. 69–76.

[15] A. Wright and V. Välimäki, "Perceptual loss function for neural modelling of audio systems," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Proc. (ICASSP 2020)*, Barcelona, Spain, 2020, pp. 251–255.

[16] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," *Advances in Neural Information Processing Systems*, pp. 6389–6399, 2018.

[17] F. Eichas, S. Möller, and U. Zölzer, "Block-oriented gray box modeling of guitar amplifiers," in *Proc. 20th Int. Conf. Digital Audio Effects (DAFx-17)*, Edinburgh, UK, Sept. 2017, pp. 184–191.

[18] A. Novak, L. Simon, P. Lotton, and J. Gilbert, "Chebyshev model and synchronized swept sine method in nonlinear audio effect modeling," in *Proc. 13th Int. Conf. Digital Audio Effects (DAFx-10)*, Graz, Austria, Sept. 2010, pp. 423–426.

[19] R. Cauduro Dias de Paiva, J. Pakarinen, and V. Välimäki, "Reduced-complexity modeling of high-order nonlinear audio systems using swept-sine and principal component analysis," in *Proc. 125th AES Conv.*, Helsinki, Finland, March 2012.

[20] F. Eichas, S. Möller, and U. Zölzer, "Block-oriented modeling of distortion audio effects using iterative minimization," in *Proc. 18th Int. Conf. Digital Audio Effects (DAFx-15)*, Trondheim, Norway, Dec. 2015, pp. 243–248.

[21] F. Eichas and U. Zölzer, "Black-box modeling of distortion circuits with block-oriented models," in *Proc. 19th Int. Conf. Digital Audio Effects (DAFx-16)*, Brno, Czech Republic, Sept. 2016, pp. 39–45.

[22] Roland Corp., "*Boss DS-1 Service Notes*," Dec. 1994.