GREY-BOX MODELLING OF DYNAMIC RANGE COMPRESSION

*Alec Wright and Vesa Välimäki**

Acoustics Lab, Department of Signal Processing and Acoustics
Aalto University
Espoo, Finland
alec.wright@aalto.fi

ABSTRACT

This paper explores the digital emulation of analog dynamic range compressors, proposing a grey-box model that uses a combination of traditional signal processing techniques and machine learning. The main idea is to use the structure of a traditional digital compressor in a machine learning framework, so it can be trained end-to-end to create a virtual analog model of a compressor from data. The complexity of the model can be adjusted, allowing a trade-off between the model accuracy and computational cost. The proposed model has interpretable components, so its behaviour can be controlled more readily after training in comparison to a black-box model. The result is a model that achieves similar accuracy to a black-box baseline, whilst requiring less than 10% of the number of operations per sample at runtime.

1. INTRODUCTION

Dynamic Range Compression (DRC) is a nonlinear audio effect which adjusts the dynamic range of the input signal [1]. The use of DRC is widespread in music production, mastering, and broadcasting. Typically DRC is applied to reduce the level of the loud parts of a signal, whilst leaving the quieter parts unaltered. Generally, the input signal is used to calculate a time-varying gain envelope, which is then applied to the signal.

Digital compressor algorithms are a popular research topic, exploring themes ranging from DRC algorithm design [1, 2, 3, 4], to methods for automation of DRC parameters [5, 6, 7, 8], or inversion of DRC [9, 10]. Studies have also approached the simulation of analog compressors by creating virtual analog models of specific compressor devices [11], or of different components used in the devices, such as optocouplers [12] or amplifiers [13].

Deep learning approaches have also been applied to DRC modelling [14, 15, 16]. An advantage to this approach is that a model can be trained to emulate a specific compressor using data recorded from the device, removing the requirement to design a new model for each compressor. The disadvantages are that the models can be impractical for real-time applications, due to high-computational cost at inference time, non-causality, or both. Furthermore, whilst it is possible to incorporate the user controls of the effect via conditioning, they are essentially black-box models and as such it is hard to interpret or control their behaviour.

* This research belongs to the activities of the Nordic Sound and Music Computing Network—NordicSMC (NordForsk project number 86892).

Recent research into deep learning for musical instrument synthesis has proposed the combination of digital signal processing methods, such as oscillators and filters, with deep learning, allowing the resulting model to be trained from data whilst retaining components whose behaviour can be more readily understood and controlled at inference time [17]. This has also been applied to modelling non-linear audio effects such as distortion pedals [18, 19], by using memoryless nonlinearities and Infinite Impulse Response (IIR) filters within a machine learning framework.

In this paper, we apply this approach to create a grey-box DRC model based on an existing topology. This has the advantage that the behaviour of the resulting model is easier to understand and manipulate after training, whilst still allowing a specific analog device to be modelled from data.

The paper is structured as follows. Section 2 summarises the basics of DRC. Section 3 introduces the novel machine learning approach for DRC modelling. Section 4 presents results of model validation using an artificial dataset. Section 5 describes the analog compressor and architectures used to model it. Section 6 reports the results of the training, and Section 7 concludes the paper.

2. DIGITAL DYNAMIC RANGE COMPRESSION

The core idea of this paper is to utilise the components and structure of a traditional digital DRC, along with some neural network based components, to create a grey-box DRC model that can be fit to data recorded from a target device. The defining behaviour of a DRC is that it applies a time-varying gain to the input signal, with the time-varying gain envelope being calculated by a side chain. Typically the input to the side chain is the signal being compressed, although the compressor can also be controlled by another audio signal if desired. How the side chain calculates the time-varying gain envelope defines the behaviour of the compressor. A digital compressor generally consists of three blocks, a static gain computer, a level detector, and a make-up gain. These are all described in more detail in the following sections.

2.1. Static Gain Computers

A static gain computer is a memoryless nonlinear function, that maps input level to output level. Traditionally it is implemented as a hard-knee or soft-knee compression curve. For the hard-knee, the parameters are the threshold and the ratio. When it exceeds the threshold, the signal level is reduced by a factor determined by the signal level and the ratio, according to the equation [1]:

$$y_{gc} = \begin{cases} x_{gc}, & \text{when } x_{gc} \leq T \\ T + (x_{gc} - T)/R, & \text{when } x_{gc} > T, \end{cases} \quad (1)$$

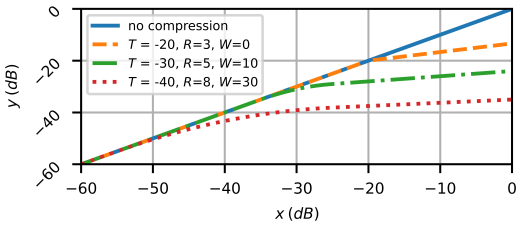


Figure 1: *Hard and soft knee static compression curves, where T is threshold, R is ratio, and W is the knee width.*

where x_{gc} is the input to the gain computer, y_{gc} is the corresponding output and T and R are the threshold and ratio, respectively.

For the soft knee, a knee width parameter is added, which introduces a transition region. The soft knee is defined by [1]:

$$y_{gc} = \begin{cases} x_{gc}, & 2(x_{gc} - T) < -W \\ x_{gc} + \frac{(\frac{1}{R}-1)(x_{gc}-T+\frac{W}{2})^2}{2W}, & 2|x_{gc} - T| \leq W \\ T + \frac{(x_{gc}-T)}{R}, & 2(x_{gc} - T) > W, \end{cases} \quad (2)$$

where W is the knee width. An example of the above static compression curves for various parameters are shown in Fig. 1. Note that when $W = 0$, the soft-knee is identical to the hard-knee.

2.2. Level Detectors

A level detector prevents sudden gain changes being applied to the signal, which can result in unpleasant sounding compression artifacts, by producing a smooth gain envelope. It can be applied directly to the side chain input, or, to the gain envelope produced by the gain computer. A common choice for a level detector filter is the digital one-pole filter, with the difference equation [1]:

$$y_d[n] = \alpha y_d[n-1] + (1-\alpha)x_d[n], \quad (3)$$

where α is the filter coefficient and $x_d[n]$ and $y_d[n]$ are, respectively, the filter input and output at time n. Stability of this filter is ensured if $|\alpha| < 1$. The step response of the filter is given by [1]:

$$y_d[n] = 1 - \alpha^n \quad \text{for} \quad x_d[n] = 1, n \geq 1. \quad (4)$$

The time constant, τ, determines the amount of time it takes for the filter output to reach $1 - 1/e$ of its final value. The value of α that produces a desired time constant can be found by:

$$\alpha = e^{-1/\tau f_s}, \quad (5)$$

where τ is the desired time constant in seconds and f_s is the sample rate in Hz.

When describing compressor behaviour, the *attack* and *release* times refer to how quickly the compressor acts. The attack time determines how quickly the compressor responds to an increasing input signal level, and thus how quickly the subsequent gain decrease is applied. Likewise, the release time determines how long the applied gain takes to return back to the normal level, once the input signal level begins to fall. Independent attack and release times can be achieved by defining time constants τ_a and τ_a for the attack and release phases, respectively, then finding the corresponding filter coefficients α_a and α_r. The level detector filter can then be implemented as follows [1]:

$$y_d[n] = \begin{cases} \alpha_a y_d[n-1] + (1-\alpha_a)x_d[n], & x_d[n] > y_d[n-1] \\ \alpha_r y_d[n-1] + (1-\alpha_r)x_d[n], & x_d[n] \leq y_d[n-1]. \end{cases} \quad (6)$$

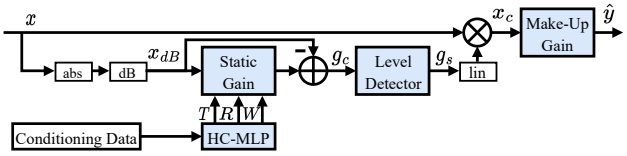


Figure 2: *Diagram of the compressor structure used in this paper. Blue boxes are components with parameters learned from data.*

2.3. Make-Up Gain

Finally, the reduction in signal level introduced by the previous stages can be compensated for by the make-up gain module. The make-up gain is applied uniformly to the output signal, and in a digital DRC this is simply a constant gain factor.

3. PROPOSED MODEL

The compressor model used in this paper, depicted in Fig. 2, takes on the 'log-domain detector' structure described in [1]. It has the same three stages that were described in Sec. 2 , a static gain computer, a level-detector, and a make-up gain.

3.1. Static Gain Computer

The static gain computer is a memoryless nonlinear function that outputs a target gain reduction for each sample in the model input:

$$g_c[n] = f_{gc}(x_{\text{dB}}[n], \theta_{gc}, c), \quad (7)$$

where $g_c[n]$ is the calculated gain reduction to be applied at discrete time n, $x_{\text{dB}}[n]$ is the input signal level in dB at time n, θ_{gc} are the learned parameters of the gain computer, and c is the conditioning information describing the user settings of the device being modelled. For this paper, the static gain computer, f_{gc}, was implemented as a hard or soft knee compression curve, see Sec. 2.1.

The user controls of the compressor can be incorporated into the model, by providing information representing the device control settings. Following recent work modelling distortion circuits [18], we use a hyperconditioning Multilayer Perceptron (HC-MLP) network to predict the parameters of the static compression curve based on the conditioning information. θ_{gc} is thus defined as the parameters of the HC-MLP.

The conditioning information is first normalised to the range $[-1, 1]$, and then provided as an input to the HC-MLP, which predicts the corresponding threshold, ratio and knee width. The HC-MLP has an input size equal to the number of conditioning values, and an output size corresponding to the number of parameters of the static gain function. The HC-MLP has two hidden layers of size 20, each followed by a ReLU activation function. The output layer is followed by a sigmoid activation function and appropriate scaling and offset factors to ensure the threshold T, ratio R and knee width W meet the conditions $-80 < T < 0$, $1 < R < 30$ and $0 < W < 30$.

The target output gain, y_{dB}, is calculated according to Eq. (1) or Eq. (2), and the target gain reduction is given by:

$$g_c[n] = y_{dB}[n] - x_{dB}[n]. \quad (8)$$

3.2. Level Detector

Smoothing is applied to the gain envelope to prevent discontinuities and compression artifacts. The gain g_c is first normalised to the range $[0, 1]$, assuming a minimum possible gain reduction of

0 dB and a maximum possible gain reduction of 80 dB. The level detector has memory and takes the output of the gain computer, g_c, to produce the smoothed gain envelope:

$$g_s[n] = f_{gs}(g_c[n], s_{gs}[n-1], \theta_{gs}), \tag{9}$$

where $g_s[n]$ is the smoothed gain to be applied at discrete time n, $s_{gs}[n-1]$ is the state of the level detector from the previous time step, and θ_{gs} are the learned parameters of the level detector. The exact form of $s_{gs}[n-1]$ depends on the chosen strucutre of the level detector, and is described in the following sections. The smoothed gain signal is then scaled back to the range $[-80, 0]$ and converted to the linear domain, before being applied to the input.

3.2.1. One-Pole Filter

Smoothing can be carried out by the one-pole filter defined in Eq. (3). In this case the level detector learns the time-constant, τ, which is parameterised by a sigmoid function, limiting it to the range $0\,\text{s} < \tau < 1.0\,\text{s}$. The one-pole filter coefficient α is calculated according to Eq. (5). This ensures the stability of the one-pole filter. In this case, θ_{gs} is the learned time constant, τ, and the state s_{gs} is simply the previous output of the one-pole filter.

For training purposes, implementing this filter recursively slows the training time considerably, as has been noted in previous work [18]. To reduce training time, the one-pole filter was applied in the frequency domain. To achieve this, we take the Discrete Fourier Transform (DFT) of the input to the filter, and multiply it with the filter frequency response. The Inverse Discrete Fourier Transform (IDFT) is applied to the result to produce the output of the filter in the time-domain. This follows the approach described in [18].

Training can be accelerated by increasing the frequency at which the compressor model parameters are updated. This can be achieved by updating the parameters many times whilst processing a longer segment of audio, a process known as Truncated Back-propagation Through Time (TBPTT) [20]. This presents an issue when implementing the one-pole filter in the frequency domain, as the filter parameters will change after each sub-segment of the input is processed. Simply applying frequency domain filtering to each sub-segment will result in an inaccurate approximation of the one-pole filter unless inputs from previous sub-segments are also included.

To solve this problem, we use an input buffer. The buffer is L samples in length, and is initialised with zeros. Each sub-segment is N_{in} samples in length, and as each sub-segment is processed it is added to the buffer in a First-In First-Out fashion. The filter is applied to the whole input buffer, and the last N_{in} samples of the output are taken as the output of the filter. This process is depicted in Fig. 3. The one-pole filter has a very long impulse response as α approaches 1, so the input buffer length was selected to be 80 000 samples long. Although this is unnecessarily long to sufficiently approximate the one-pole filter for most values of α, this method was still found to be approximately five times faster during training in comparison to implementing the IIR filter as a straightforward recursion.

3.2.2. Switching One-Pole Filter

In practice, it is desirable for the compressor to have independent attack and release times, as this allows users greater control over the compressor behaviour. This can be implemented trivially by learning two time-constants, τ_A and τ_R, representing the attack and release times respectively. The one-pole filter can then be implemented according to Eq. (6). It should be noted that in this case, the filter is time-varying as the filter coefficients depend on the previous output and the current input. This means that it must be implemented recursively, which slows training considerably. In this case, θ_{gs} is the learned attack and decay time constants, τ_a and τ_r, and the state s_{gs} is still simply the previous output of the one-pole filter.

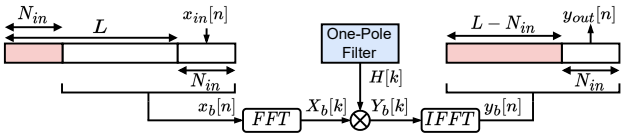


Figure 3: *Frequency-domain convolution with the frequency response of the one-pole filter. The beginning of the output signal $y_{out}(n)$ is discarded because it contains the starting transient.*

3.2.3. RNN-Modulated One-Pole Filter

The switching one-pole filter is a simple example of a time-varying filter, where the filter's time constant depends on the current input and previous output of the filter. This is however somewhat limited as the one-pole filter only has two possible values to switch between. The time-constant of the one-pole filter can also be varied more freely, for example, by training a Recurrent Neural Network (RNN) to predict appropriate time-constants. We propose using a standard Elman-RNN [21] followed by a linear layer.

The RNN has a hidden size of 4, and its input is the unsmoothed gain reduction, g_c. An affine transformation is then applied to the RNN hidden state via a linear layer with an output size of 1. A sigmoid activation function is applied to the output of the linear layer, limiting it to be between the values of 0 and 1, and the resulting sequence is taken as the predicted time-constant for the one-pole filter, with the filter coefficient α being changed at each time-step.

Modulation of the one-pole filter should allow the model to have greater control over the gain smoothing process. The introduction of the RNN does decrease the interpretability of the level detector somewhat, as the time-constants in traditional a digital DRC are generally stable over time. However, as the level detector is still a one-pole filter, it's behaviour is still relatively easy to interpret, and this configuration ensures that the gain smoothing doesn't scale or shift the input gains, as would be possible if the gain smoothing was carried out directly by the recurrent model. In this case, θ_{gs} is the weights and biases of the RNN and linear layer, and the state, s_{gs}, is the previous output of the one-pole filter as well as the hidden state of the RNN.

3.3. Make-Up Gain

In a digital compressor, the make-up gain can be trivially implemented as a constant gain factor applied to the output. However in an analog compressor the make-up gain might be implemented via a VCA or a vacuum-tube amplifier. The make-up gain in the proposed model can thus be implemented either as a constant gain factor, hereafter referred to as a Static Gain, or as a small Gated Recurrent Unit (GRU) [22]. In both cases the make-up gain is applied directly to the time-domain signal, after the time-varying gain envelope g_s has been applied.

The GRU model is identical to a previously proposed model applied to guitar amplifiers and distortion effects [23]. It con-

sists of a single-layer GRU, followed by an affine transformation. The GRU has a hidden size of 4. The input to the make-up gain GRU is the input signal after the time-varying gain envelope has been applied, x_c. The affine transformation is applied via a fully-connected layer, reducing the hidden size of the GRU to a single value, representing the compressor output, $\hat{y}$, at that time-step. To reduce the ability of the GRU make-up gain to simply learn to emulate the behaviour of the whole compressor, it is not provided the conditioning information describing the compressor settings. Whilst it is still possible that the GRU could infer the conditioning parameters indirectly from the signal envelope provided to it, the small hidden size should significantly limit its ability to do so.

3.4. Loss Functions

When evaluating the 'SignalTrain' dataset of the LA-2A compressor (see Sec. 5 for a description of this dataset) it was noted that the compressor output contains low frequency oscillating noise which is not present in the compressor input. It is common practice to apply pre-emphasis filtering to the model output and the target before calculating the loss, to emphasise or attenuate certain frequencies [24, 25]. As the introduction of low frequency noise is not a desired characteristic of the compressor, a DC-blocking pre-filter was applied to the model output and target, prior to the loss calculation, with the following transfer function:

$$H(z) = \frac{1 - z^{-1}}{1 - 0.995z^{-1}}. \tag{10}$$

To avoid unnecessary recursions in the training process, the filter was applied via an FIR approximation. The FIR filter coefficients were obtained by truncating the impulse response of the filter to 2000 samples.

3.4.1. Error-to-Signal Ratio

The Error-To-Signal (ESR) loss has been used in a number of other papers focussing on modelling non-linear audio circuits [23, 24, 26, 27]. The ESR is the squared error in the time-domain, divided by the energy of the target signal. For a segment of training signal of length N, the ESR is given by:

$$\mathcal{E}_{\text{ESR}} = \frac{\sum_{n=0}^{N-1} |y_p[n] - \hat{y_p}[n]|^2}{\sum_{n=0}^{N-1} |y_p[n]|^2}, \tag{11}$$

where y_p is the DC-blocker pre-filtered target signal, and $\hat{y_p}$ is the compressor model DC-blocker pre-filtered output.

3.4.2. Short-Time Energy-Loss

One potential issue with the ESR loss is that it requires strict alignment in the time domain. For the models proposed in this paper where a static make-up is applied (as opposed to the GRU make-up gain), the model is only able to apply a time-varying gain envelope to the input signal. As such it is incapable of applying any phase shifts or sign inversions to the input. This means that, when applying the ESR loss, slight time misalignment or phase differences between the model output and target will result in a large loss, even if the two signals are perceptually very similar.

In light of this, we propose using a loss function based on the difference between the short-time energy of the output and target signals, similar to one proposed in [28]. The DC-blocker pre-filtered target and model output signals, y_p and $\hat{y_p}$, are first divided

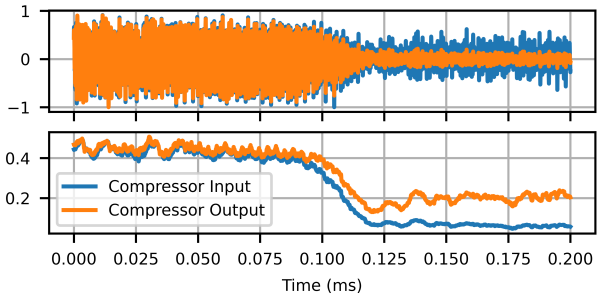


Figure 4: *Example of audio processed by the LA-2A compressor with peak reduction set to 90, (top) input and (middle) output signals from the data set and (bottom) the corresponding RMS envelopes. Signals have been normalised to 0 dBFS and RMS envelopes were calculated using a sliding window of 250 samples.*

into M overlapping frames of length N_f, referred to as Y_p and $\hat{Y}_p$. The short-time energy loss is given by:

$$\mathcal{E}_{\text{STE}} = \frac{1}{MN_f} \sum_{m=0}^{M-1} | \sum_{n=0}^{N_f-1} Y_p[n,m]^2 - \hat{Y}_p[n,m]^2 |. \tag{12}$$

The loss function can be applied at multiple resolutions. The overall loss is then calculated as the mean of the short-time energy losses:

$$\mathcal{E}_{\text{MSTE}} = \frac{1}{I} \sum_{i=1}^{I} \mathcal{E}_{\text{STE,i}}, \tag{13}$$

where I is the number of different frame sizes the loss is calculated for. This Multi-Resolution Short-Time Energy (MSTE) loss was applied at four different resolutions in this study, using frame sizes of 8, 16, 32, and 64 and hop sizes of 2, 4, 8, and 16, respectively.

3.5. Training Hyperparameters

To train the model, the training dataset was split into 2.5 s segments. For each segment the first 4096 samples were processed without updating the network's parameters, to allow any components of the model with a state to initialise. The remaining input was processed with TBPTT being applied every 8128 samples and The training dataset was processed using a batch size of 25. The validation loss was calculated four times per epoch. Training was run for a maximum of 50 epochs. Early stopping was applied, with a validation patience of 5. The models were trained using the Adam optimizer [29], with an initial learning rate of 5×10^{-3}. Models were trained on a GPU, and training took around 30 minutes when using the single parameter one-pole filter level detector, and up to 16 hours otherwise. A reference PyTorch implementation is provided at[1].

4. TOY PROBLEM – MODELLING A DIGITAL COMPRESSOR

To verify the modelling approach and training procedure, the model was trained to recreate the behaviour of a digital compressor. As input material for the digital compressor, data from the Signal-Train dataset [15], which is described in Sec. 5, was used. Ten minutes of audio was selected as the training dataset, and 90 seconds of audio for the validation dataset. In this case, a test dataset is unnecessary, as the success of the modelling procedure

[1] https://github.com/Alec-Wright/GreyBoxDRC

can be observed directly based on the learned parameters of the compressor model.

The target digital compressor was implemented as described in Sec. 2, with hard-knee static gain curve (Eq. (1)), one-pole filter level detector with independent attack and release times (Eq. (6)) and no make-up gain. The training dataset was processed a number of times, with the hard-knee threshold and ratio parameters being varied. For each dataset, the threshold was set to either -15 dB, -20 dB, -25 dB or -30 dB and the ratio was set to 2, 4, 6 or 8. All possible permutations of these setting were used resulting in 16 combinations of threshold and ratio. The attack and release times and were set to 20 ms and 50 ms respectively. To better emulate the data recorded from an analog device, and to test the robustness of the training process, pink noise at a level of -40 dB was added to the training data targets.

Models were trained using the soft-knee compression curve. For the level detector either the switching one-pole or RNN modulated one-pole was used, and for the make-up gain both the static gain and GRU were trialled. Models were trained using either the ESR or the MSTE loss function. Conditioning information representing the threshold and ratio used to generate the data was provided to the model via a HC-MLP. It should be noted that as the HC-MLP is a fully connected network, it must learn that the two conditioning values it receives as input are independent of each other. That is, the input conditioning value representing the threshold, should have no influence over the ratio predicted by the HC-MLP, and vice versa.

The success of the modelling can be evaluated by how closely the conditioning values provided to the model match the predicted threshold and ratio. Results for two of the trained models, over a range of input threshold and ratio values, are shown in Fig. 5. The models were selected to demonstrate that the correct compression curve parameters can be inferred using the MSTE loss, and that the GRU make-up gain does not simply bypass the compression curve altogether and learn to apply the compression itself. It can be seen that for both models the threshold parameter is learned very accurately, with only minor fluctuations from the correct values. For the ratio parameter, the models learned the mapping well for low ratios, but a small error can be observed for higher values. For both of the models shown in Fig. 5, the learned attack and release times were within 0.02 ms of the target attack and release times. Results for the other models are omitted here for brevity, but generally similar behaviour was observed.

5. TELETRONIX LA-2A COMPRESSOR MODELLING

The proposed compressor model was tested on emulating the behaviour of the Teletronix LA-2A compressor. This is an optical compressor, which uses the control voltage to drive a light source. The light source controls the resistance of a light-dependent resistor (LDR) which is connected to a voltage divider in the main signal path. The effect is that as the brightness of the light increases, the resistance of the LDR decreases, resulting in a gain reduction. Fig. 4 shows an example comparing the input and resulting output of a signal processed by the device.

The data used to train the model was taken from the SignalTrain dataset [15], which consists of input-output audio processed by the LA-2A compressor, with settings of the *peak reduction* knob (from 0 to -100), and the compressor either set to *limit* or *compress*. It includes content such as lone instrument and full band recordings, as well as electronic music. It also includes synthetic

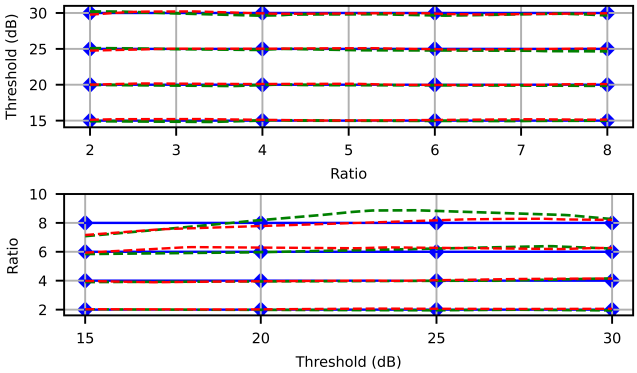


Figure 5: *Example outputs from HC-MLP trained on the toy problem, markers indicate threshold and ratio values seen during training, solid lines indicate threshold and ratio values input to the HC-MLP, dashed lines indicate the threshold and ratio predicted by the HC-MLP, for (green) model trained with $\mathcal{E}_{MSTE}$, soft-knee, switching one-pole level detector and no make-up gain, and (red) model trained with $\mathcal{E}_{ESR}$, soft-knee, switching one-pole level detector and GRU make-up gain.*

data, consisting of noise bursts mixed with tones.

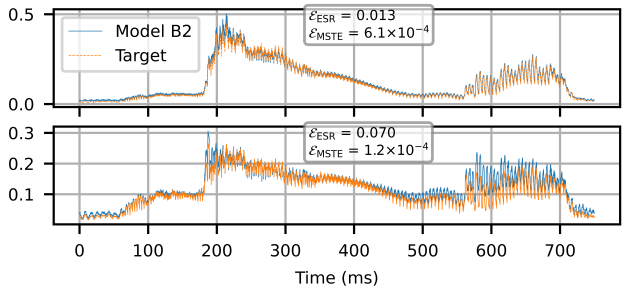
To train the model a subset of the dataset was used, which only included examples where the compressor was set to compress mode, with the peak reduction setting, ranging from 0 to 100, in increments of 10. This results in a total of 11 parameter settings with a total of 20 min of audio for each, with the input audio being the same for each setting. An identical 80:10:10 split of the dataset was applied for each parameter setting, to create the training, validation and test datasets. A further test dataset was created, consisting of the same input audio from the original test dataset, but using the remaining 'peak reduction' settings that were not included in the training dataset. These range from 5 to 95, in increments of 10, resulting in a total of 10 parameter settings, all of which were unseen during training. This dataset is intended to test how well the trained model generalises to unseen conditioning values, and will now be referred to as the unseen test dataset.

5.1. Model Architectures

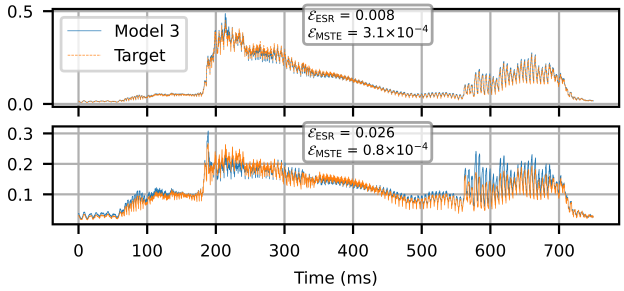
Models were trained either using the hard-knee or soft-knee compression curve. The three level-detectors introduced in Sec. 3.2 were all tested, these are, the one-pole filter, the switching one-pole filter, and the RNN-modulated one-pole filter. The make-up gain was either the Static Gain, in which case the model was trained using the $\mathcal{E}_{\text{MSTE}}$ loss, or a GRU, in which case it was trained using the $\mathcal{E}_{\text{ESR}}$ loss.

6. RESULTS

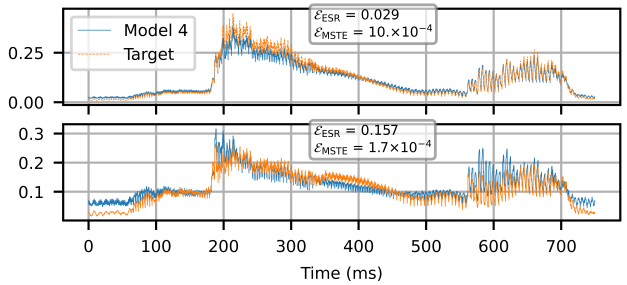
A black-box model consisting of a GRU followed by an affine transformation was used as a baseline. This model has previously been applied to guitar amplifiers [23, 27]. Recent work used this model to profile the LA-2A compressor [16], conducting a listening test which showed slight but perceptible differences between the LA-2A and the baseline. Two baseline models were created, with hidden sizes of 8 and 32. The conditioning data was first normalised to the range $[-1, 1]$ and then provided to the network by concatenating it to the input.



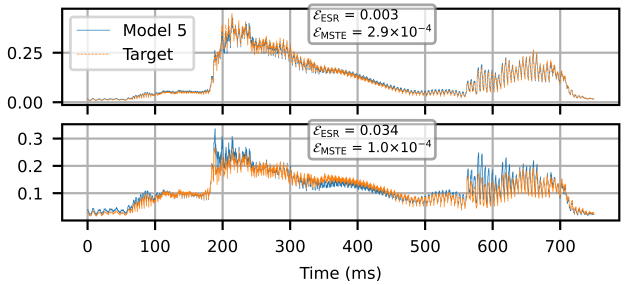
(a) *RMS envelopes for Model B2 output and target, Model B2 is baseline GRU model with hidden size 32, for LA-2A with peak reduction set to 60 (top) and 90 (bottom).*



(b) *RMS envelopes of Model 3 output and target, with soft-knee compression curve, RNN modulated one-pole level detector and static make-up gain, for LA-2A with peak reduction set to 60 (top) and 90 (bottom).*



(c) *RMS envelopes of Model 4 output and target, with hard-knee compression curve, one-pole level detector and GRU make-up gain, for LA-2A with peak reduction set to 60 (top) and 90 (bottom).*



(d) *RMS envelopes of Model 5 output and target, with hard-knee compression curve, switching one-pole level detector and GRU make-up gain, for LA-2A with peak reduction set to 60 (top) and 90 (bottom).*

Figure 6: *RMS envelopes produced by various compressor models, with RMS window size of 100 samples*

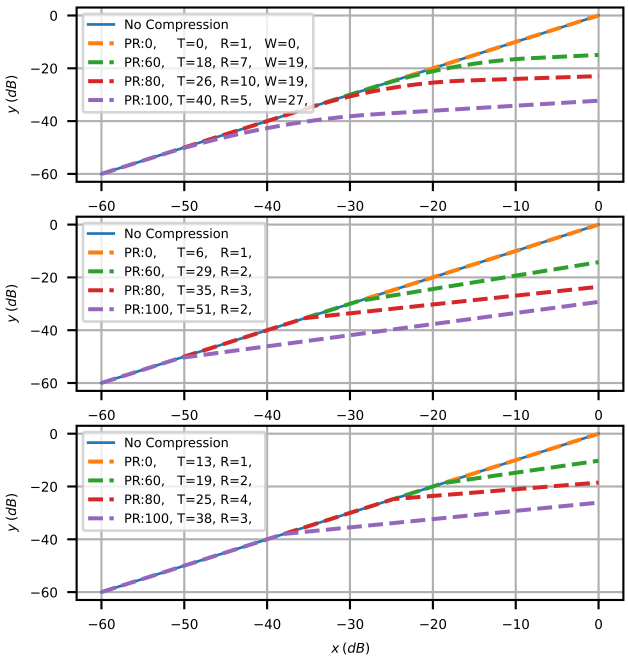


Figure 7: *Input-Output characteristic learned for various LA-2A peak reduction settings, for (top) Model 3 (soft-knee, RNN modulated one-pole level detector, and static make-up gain), (middle) Model 4, (hard-knee, one-pole level detector and GRU make-up gain) and (bottom) Model 5 (hard-knee, switching one-pole level detector and GRU make-up gain).*

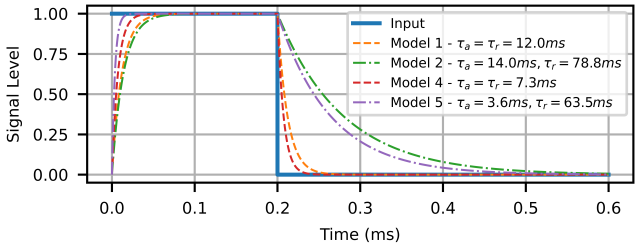


Figure 8: *Level detector attack and release behaviour learned by the compressor models.*

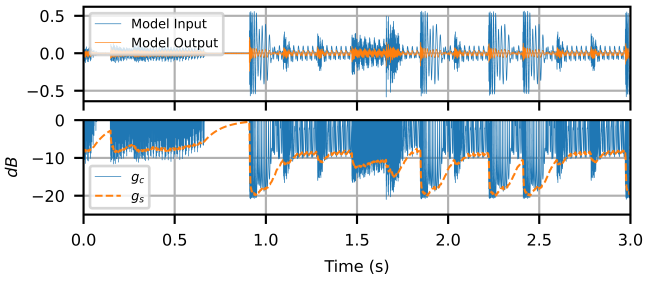


Figure 9: *Plots showing (top) input, (middle) static and smoothed gain envelopes and (bottom) output of the model, for Model 5 (hard-knee, switching one-pole level detector and GRU make-up gain) of the LA-2A compressor with peak reduction set to 90.*

Table 1: *Results of the proposed model and the baseline model [16] when emulating the LA-2A compressor.*

Model #	Model			ops/sample	Test Loss $\mathcal{E}_{\text{ESR}}$	Test Loss $\mathcal{E}_{\text{MSTE}}$	Test Loss - Unseen $\mathcal{E}_{\text{ESR}}$	Test Loss - Unseen $\mathcal{E}_{\text{MSTE}}$
B1	Baseline - Small GRU (hidden size 8)			1273	0.041	8.3×10^{-4}	0.037	7.6×10^{-4}
B2	Baseline - Big GRU (hidden size 32)			9697	**0.023**	$\mathbf{5.5 \times 10^{-4}}$	0.020	4.8×10^{-4}
	Static Gain	Level Detector	Make-Up Gain					
1	Hard Knee	One-Pole	Static Gain	11	0.095	8.4×10^{-4}	0.090	7.6×10^{-4}
2	Hard Knee	Switching One-Pole	Static Gain	12	**0.089**	8.0×10^{-4}	0.085	7.4×10^{-4}
3	Soft Knee	RNN Mod. One-Pole	Static Gain	184	0.092	$\mathbf{7.4 \times 10^{-4}}$	0.089	6.7×10^{-4}
4	Hard Knee	One-Pole	GRU	523	0.045	8.2×10^{-4}	0.043	7.9×10^{-4}
5	Hard Knee	Switching One-Pole	GRU	528	**0.031**	$\mathbf{7.1 \times 10^{-4}}$	0.028	6.8×10^{-4}
6	Hard Knee	RNN Mod. One-Pole	GRU	700	0.032	7.5×10^{-4}	0.029	7.0×10^{-4}

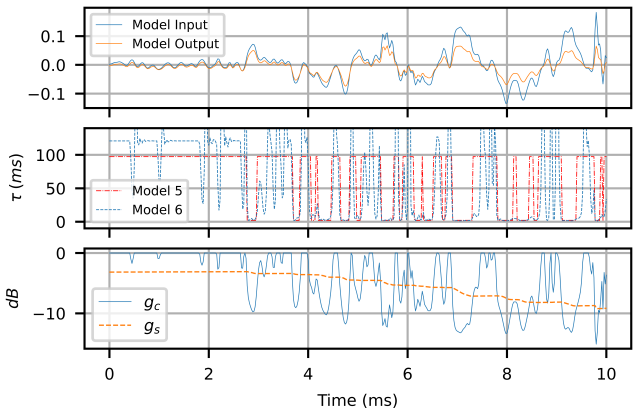


Figure 10: *Plots showing (top) input and model output, (middle) RNN modulated one-pole time constant with switching one-pole filter time constant for comparison, and (bottom) static and smoothed gain envelope, for Model 6 (hard-knee, RNN modulated one-pole filter and GRU make-up gain) emulating the LA-2A compressor with peak reduction set to 90.*

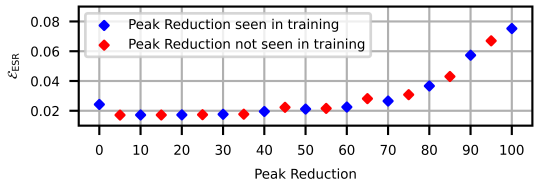


Figure 11: *Test losses for Model 6, against different settings of peak reduction, modelling the LA-2A compressor.*

A selection of the results of the trained models are presented in Table 1. For each possible combination of make-up gain and level detector, the model with either the hard or soft knee is presented, depending on which achieved the lowest test loss. An estimate of the computational cost of the model at runtime is also provided, by calculating the number of floating-point operations required per sample of output, as suggested in [30]. The tanh and sigmoid nonlinearities are assumed to cost 30 floating point operations.

The results show that Model B2, the baseline model with a hidden size of 32, achieves the best loss values, however the computational cost is considerably higher in comparison to the other models. Models 5 and 6 are less accurate than the large GRU baseline, although the difference fairly small, and the proposed models require less than 10% of the number of floating point operations per sample. They also outperform the small baseline model, Model B1, whilst being less expensive to run.

In Table 1 the performance of the models on the unseen test dataset, which consists of peak reduction values that were excluded from the training dataset, can also be observed. In all cases the test loss decreases for the unseen test dataset, indicating that the models generalise well. This can also be seen in Fig. 11, which shows the test loss for Model 6 evaluated at different settings of peak reduction.

Fig. 6 shows the RMS envelope of some of the models outputs, compared to the target signal, indicating fairly close agreement. It can be observed that for the higher peak reduction setting, the model performance is noticeably worse. This can also be seen in the ESR losses shown in Fig. 11. A possible contributing factor to this is that the Signal-to-Noise Ratio (SNR) in the compressor output is decreasing as the amount of compression applied increases. For extreme peak reduction settings the low SNR might make it challenging for the model to learn, as the loss function becomes increasingly dominated by noise that is not present in the compressor input. Sound examples are also provided at the accompanying webpage [2].

Fig. 7 shows the learned compression curves for some of the models, for varying settings of the peak reduction conditioning value. As expected, the static gain curves learn to apply more compression as the peak reduction increases. There is generally good agreement between the models.

The results also indicate that the single parameter one-pole filter performs worse than either the switching or RNN modulated one-pole filter. The learned one-pole filter step responses are shown in the Fig. 8, which show that the learned filters all have an attack time constant of around 15 ms or less. However, for the switching one-pole filter the release time is much longer, indicating that a fast attack and slow release is required to successfully model the target device. Example output from Model 5, which used the switching one-pole filter, is shown in Fig. 9

The results don't indicate much difference in performance between the RNN modulated one-pole and the switching one-pole level detector. Example output from Model 6 can be seen in Fig. 10. It can be observed that the learned behaviour is somewhat similar to that of the switching one-pole, with the time-constant becoming very small when the input signal is large, and increasing once the input decreases again.

[2] http://research.spa.aalto.fi/publications/papers/dafx22-DDRC

7. CONCLUSION

In this work, we proposed a new method for virtual analog modelling of dynamic range compressors, which created a grey-box model using a combination of existing techniques and neural network components. The resulting models can be trained using data to emulate a specific device in an end-to-end fashion. As the model structure follows that of a traditional digital DRC, parameters such as threshold and ratio can be easily adjusted after training. We trained our proposed model, as well as a black-box baseline model, to emulate an analog compressor, and found that our model was able to achieve comparable performance at a much lower computational cost. Future work should involve subjective evaluation so that the perceptual quality of the models can be better understood.

8. REFERENCES

[1] D. Giannoulis, M. Massberg, and J. D. Reiss, "Digital dynamic range compressor design—A tutorial and analysis," *J. Audio Eng. Soc.*, vol. 60, no. 6, pp. 399–408, 2012.

[2] E. F. Stikvoort, "Digital dynamic range compressor for audio," *J. Audio Eng. Soc.*, vol. 34, no. 1/2, pp. 3–9, 1986.

[3] G. Ramos, "Block processing strategies for computationally efficient dynamic range controllers," in *Proc. Int. Conf. Digital Audio Effects*, Paris, France, 2011, pp. 253–256.

[4] L. McCormack and V. Välimäki, "FFT-based dynamic range compression," in *Proc. 14th Sound and Music Comput. Conf.*, Espoo, Finland, 2017, pp. 42–49.

[5] J. Maddams, S. Finn, and J. D. Reiss, "An autonomous method for multi-track dynamic range compression," in *Proc. Int. Conf. Digital Audio Effects*, York, UK, 2012, pp. 1–8.

[6] D. Giannoulis, M. Massberg, and J. D. Reiss, "Parameter automation in a dynamic range compressor," *J. Audio Eng. Soc.*, vol. 61, no. 10, pp. 716–726, 2013.

[7] D. Sheng and G. Fazekas, "Automatic control of the dynamic range compressor using a regression model and a reference sound," in *Proc. Int. Conf. Digital Audio Effects*, Edinburgh, UK, 2017, pp. 160–167.

[8] D. Sheng and G. Fazekas, "A feature learning siamese model for intelligent control of the dynamic range compressor," in *Proc. Int. Joint Conf. on Neural Networks (IJCNN)*, Budapest, Hungary, 2019, pp. 1–8.

[9] B. Lachaise and L. Daudet, "Inverting dynamics compression with minimal side information," in *Proc. Int. Conf. Digital Audio Effects*, Helsinki, Finland, 2008, pp. 1–6.

[10] S. Gorlow and J. D. Reiss, "Model-based inversion of dynamic range compression," *IEEE Trans. Audio, Speech, Language Process.*, vol. 21, no. 7, pp. 1434–1444, 2013.

[11] O Kröning, K Dempwolf, and U Zölzer, "Analysis and simulation of an analog guitar compressor," in *Proc. Int. Conf. Digital Audio Effects*, Paris, France, 2011, pp. 205–208.

[12] F. Eichas and U. Zölzer, "Modeling of an optocoupler-based audio dynamic range control circuit," in *Novel Opt. Syst. Design and Optim. XIX*, 2016, vol. 9948, p. 99480W.

[13] J. Schimmel, "Using nonlinear amplifier simulation in dynamic range controllers," in *Proc. Int. Conf. Digital Audio Effects*, London, UK, 2003.

[14] M. Martínez Ramírez, E. Benetos, and J. D. Reiss, "A general-purpose deep learning approach to model time-varying audio effects," in *Proc. Int. Conf. Digital Audio Effects*. Birmingham, UK, 2019.

[15] S. Hawley, B. Colburn, and S. I. Mimilakis, "Profiling audio compressors with deep neural networks," in *Audio Eng. Soc. Conv. 147*, New York, USA, 2019.

[16] C. J. Steinmetz and J. D. Reiss, "Efficient neural networks for real-time modeling of analog dynamic range compression," in *Audio Eng. Soc. Conv. 151*, 2022.

[17] J. Engel, L. Hantrakul, C. Gu, and A. Roberts, "DDSP: differentiable digital signal processing," in *Proc. Int. Conf. Learning Representations*, Addis Ababa, Ethiopia, 2020.

[18] S. Nercessian, A. Sarroff, and K. J. Werner, "Lightweight and interpretable neural modeling of an audio distortion effect using hyperconditioned differentiable biquads," in *Proc. IEEE ICASSP*, 2021, pp. 890–894.

[19] B. Kuznetsov, J. Parker, and F. Esqueda, "Differentiable IIR filters for machine learning applications," in *Proc. Int. Conf. Digital Audio Effects*, Online, 2020, pp. 297–303.

[20] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

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

[22] K. Cho, B. Van Merrienboer., D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *Proc. SSST-8, 8th Workshop on Syntax, Semantics and Struct. in Statistical Transl.*, Doha, Qatar, 2014, pp. 103–111.

[23] A. Wright, E.-P. Damskägg, and V. Välimäki, "Real-time black-box modelling with recurrent neural networks," in *Proc. Int. Conf. Digital Audio Effects*, Birmingham, UK, 2019, pp. 173–180.

[24] E.-P. Damskägg, L. Juvela, E. Thuillier, and V. Välimäki, "Deep learning for tube amplifier emulation," in *Proc. IEEE ICASSP*, Brighton, UK, 2019, pp. 471–475.

[25] A. Wright and V. Välimäki, "Perceptual loss function for neural modeling of audio systems," in *Proc. IEEE ICASSP*, Barcelona, Spain, 2020, pp. 251–255.

[26] E.-P. Damskägg, L. Juvela, and V. Välimäki, "Real-time modeling of audio distortion circuits with deep learning," in *Proc. Int. Sound and Music Computing Conf. (SMC-19)*, Malaga, Spain, 2019, pp. 332–339.

[27] A. Wright, E.-P. Damskägg, L. Juvela, and V. Välimäki, "Real-time guitar amplifier emulation with deep learning," *Appl. Sci.*, vol. 10, no. 3, 2020.

[28] Q. Tian, Y. Chen, Z. Zhang, H. Lu, L. Chen, L. Xie, and S. Liu, "TFGAN: Time and frequency domain based generative adversarial network for high-fidelity speech synthesis," *arXiv preprint arXiv:2011.12206*, 2020.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learning Representations*, San Diego, CA, 2015.

[30] J. Parker, F. Esqueda, and A. Bergner, "Modelling of nonlinear state-space systems using a deep neural network," in *Proc. Int. Conf. Digital Audio Effects*, Birmingham, UK, 2019, pp. 2–6.