Audio Engineering Society

Conference Paper 2

Presented at the AES International Conference on Artificial Intelligence and Machine Learning for Audio 2025
September 8–10, London, UK

This paper was peer-reviewed as a complete manuscript for presentation at this conference. This paper is available in the AES E-Library (http://www.aes.org/e-lib), all rights reserved. Reproduction of this paper, or any portion thereof, is not permitted without direct permission from the Journal of the Audio Engineering Society.

Sound Matching an Analogue Levelling Amplifier Using the Newton-Raphson Method

Chin-Yun Yu[1] and György Fazekas[1]

[1] *Centre for Digital Music, Queen Mary University of London, London, UK*

Correspondence should be addressed to Chin-Yun Yu (chin-yun.yu@qmul.ac.uk)

ABSTRACT

Automatic differentiation through digital signal processing algorithms for virtual analogue modelling has recently gained popularity. These algorithms are typically more computationally efficient than black-box neural networks that rely on dense matrix multiplications. Due to their differentiable nature, they can be integrated with neural networks and jointly trained using gradient descent algorithms, resulting in more efficient systems. Furthermore, signal processing algorithms have significantly fewer parameters than neural networks, allowing the application of the Newton-Raphson method. This method offers faster and more robust convergence than gradient descent at the cost of quadratic storage. This paper presents a method to emulate analogue levelling amplifiers using a feed-forward digital compressor with parameters optimised via the Newton-Raphson method. We demonstrate that a digital compressor can successfully approximate the behaviour of our target unit, the Teletronix LA-2A. Different strategies for computing the Hessian matrix are benchmarked. We leverage parallel algorithms for recursive filters to achieve efficient training on modern GPUs. The resulting model is made into a VST plugin and is open-sourced.

1 Introduction

Controlling the dynamic range of an audio signal is a fundamental task in audio engineering. Audio effects such as compressors, limiters, and expanders are designed to achieve this goal and are often used in music production, broadcasting, and live sound reinforcement. Analogue compressors developed in the early days have iconic sounds, such as the popular Teletronix LA-2A, which is known for its smooth and musical compression. Considerable interest has been devoted to replicating their characteristics in the digital world, known as virtual analogue (VA) modelling [1].

1.1 Problem Definition

Let us denote an audio stimulus as $x(t)$ and its discrete version as $\mathbf{x} = [x(0), x(T), \ldots, x((N-1)T)]^\top \in \mathbb{R}^N$ with $T \in \mathbb{R}_+$ being the sampling period. We want to model a target analogue compressor $\delta : \mathbb{R}^\infty \to \mathbb{R}$, specifically, its behaviour to the stimulus $x(t)$ as $y(t) = \delta(x(s) \mid s \leq t)$. We denote $\mathbf{y} = [y(0), y(T), \ldots, y((N-1)T)]^\top \in \mathbb{R}^N$ as the sampled compressed signal. Given a controllable base model $f : \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}^N$ parametrised by M parameters, we aim to find the optimal parameters $\boldsymbol{\theta}^* \in \mathbb{R}^M$ that minimises the distance between the output $\hat{\mathbf{y}} = f(\mathbf{x}, \boldsymbol{\theta})$ and $\mathbf{y}$ according to a distance function $\mathscr{D} : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}_+$.

1.2 Related Work

In recent years, neural networks (NNs) with various architectures have become popular base models for f. Hawley et al. [2] are pioneers in this approach, using an U-Net-like autoencoder with skip connections to model the LA-2A end-to-end. Steinmetz and Reiss [3] proposed temporal convolutional networks (TCNs) to enhance efficiency while maintaining large receptive fields of the networks. Different conditioning strategies have been explored to improve the modelling ability [4, 5]. More recent works explored the use of state-space models for VA modelling [6, 7]. The choice of architecture is crucial since compressors are used in real-time applications, and the computational cost is a concern. Despite impressive results, the black-box nature of NNs makes the learnt parameters difficult to interpret and limits creative control.

Among previous works, Wright and Välimäki [8] proposed a grey-box approach that combines NNs and signal processing components. A feed-forward compressor is used as f with a lightweight Gated Recurrent Unit (GRU) at the end to model time-varying non-linear make-up gain. This approach achieves competitive results while requiring only 5% of arithmetic operations compared to a single GRU. The learnt control parameters also give us a better understanding of the target unit's behaviour. Yu et al. [9] further improve their training speed by implementing a specialised kernel for backpropagating the gradients through the ballistics module of the compressor.

1.3 Motivation

Both [8] and [9] use gradient descent optimisation to find the optimal feed-forward compressor parameters for VA modelling. Gradient descent is widely used in training neural networks instead of more robust methods like Newton-Raphson (NR) due to its simplicity and lower storage cost, favouring methods with many parameters ($M \gg 10^6$). However, since the feed-forward compressor has only a handful of parameters ($M < 10$), the quadratic storage cost of NR is feasible on modern computers. NR converges faster and more robustly than gradient descent when the objective function is convex around the solution and twice differentiable. This motivates us to explore the NR method for VA modelling of analogue compressors. Although NR has been used to solve differential equations in white-box models, such as wave digital filters [10] and the physical modelling of musical instruments [11], to the authors' knowledge, it has not been used for grey-box modelling of analogue compressors.

1.4 Contributions

In this article, we formulate VA modelling as a sound matching problem, aiming to find the optimal parameters of a digital compressor f with the closest match to the target sound. We verify the feasibility of using the NR method for modelling the LA-2A compressor. Moreover, we further accelerate the backpropagation method from [9] on modern GPUs using parallel algorithms for recursive filters. The learnt mapping from the circuit's peak reduction to the parameters is presented, providing an interpretable and intuitive way to control the compressor. We made the resulting model into a VST plugin, open-sourced under the MPL-2.0 license[1], for validation and as a viable music production tool.

2 Methodology

Continuing the work from [8, 9], we use a feed-forward compressor as the base model f for its efficiency and high modelling capability. We will discuss the choice of distance function in Section 2.2. Since the input signal $\mathbf{x}$ is fixed during optimisation, we use $f_{\mathbf{x}} : \mathbb{R}^M \to \mathbb{R}^N$ to denote the curried f given $\mathbf{x}$, to simplify the later derivative notations.

2.1 Feed-Forward Compressor (f)

We adopt `torchcomp` [9], a differentiable implementation of the feed-forward compressor from [12]. A digital compressor is defined as:

$$\hat{y}[n] = x[n]g[n] = x[n]g\left(s[m], \boldsymbol{\theta} \mid m \leq n\right), \qquad (1)$$

where $g : \mathbb{R}^{n+1} \times \mathbb{R} \to \mathbb{R}_+$ is the gain reduction function with parameters $\boldsymbol{\theta}$ and $s[m]$ is the side-chain signal. The design is feed-forward when $s[m] = x[m]$ [13]. We remove the root mean square (RMS) level detector from the design for two reasons. Firstly, Table 7 in [9] shows the converged RMS smoothing coefficient is close to one, implying the smoothing effect is negligible. Secondly, we found that when running the NR method, the converged $\mathscr{D}(\hat{\mathbf{y}}, \mathbf{y})$ is slightly lower without it. A possible explanation is that the extra degree of freedom

[1] github.com/aim-qmul/4a2a

introduced by the RMS filter creates a local minimum that traps the NR optimisation, a phenomenon we will discuss in Section 2.2.

The resulting compressor consists of five parameters ($M = 5$): threshold (CT), ratio (R), attack time (t_{at}), release time (t_{rt}), and make-up gain (γ). Threshold and make-up gain are expressed in decibels, and the attack/release times are in milliseconds. Threshold controls the level at which the compressor starts to work, and the ratio determines the amount of gain reduction. The make-up gain adjusts the output level to compensate for the gain reduction. The attack and release time control the ballistics of the gain reduction, which is defined as:

$$g[n] = \begin{cases} \alpha_{at}\hat{g}[n] + (1-\alpha_{at})g[n-1] & \text{if } \hat{g}[n] < g[n-1], \\ \alpha_{rt}\hat{g}[n] + (1-\alpha_{rt})g[n-1] & \text{otherwise,} \end{cases} \tag{2}$$

$$\alpha_{at/rt} = 1 - \exp\left(-\frac{2200T}{t_{at/rt}}\right). \tag{3}$$

$\hat{g}[n]$ is the gain reduction before applying the ballistics. Equation (2) is a recursive smoothing filter, and the attack and release times determine how fast the filter responds to the loudness change. The whole signal flow is shown in Fig. 1.

Both threshold and make-up gain can be any real numbers, but not for the other parameters. We also want to bound the parameters to some reasonable range that is common in practice. To enforce these constraints, we define the parameter vector as $\theta = \{CT, \gamma, \hat{R}, \hat{\alpha}_{at}, \hat{\alpha}_{rt}\} \in \mathbb{R}^5$ and get

$$\{R, \alpha_{at}, \alpha_{rt}\} = \Big\{ \lambda(u) \mid u \in \{\hat{R}, \hat{\alpha}_{at}, \hat{\alpha}_{rt}\}, \\ \lambda(u) = \theta_{min}(u) + \big(\theta_{max}(u) - \theta_{min}(u)\big)\sigma(u) \Big\}, \tag{4}$$

where $\sigma : \mathbb{R} \to [0,1]$ is the monotonic sigmoid function, and $\theta_{min} : \mathbb{R} \to \mathbb{R}$ and $\theta_{max} : \mathbb{R} \to \mathbb{R}$ return the lower and upper bounds of the parameters, respectively.

2.2 Newton-Raphson Optimisation

Newton-Raphson is a fundamental optimisation method that finds the minimum of a smooth and convex function $\mathscr{L} : \mathbb{R}^M \to \mathbb{R}$. It updates the parameter θ by the following equation:

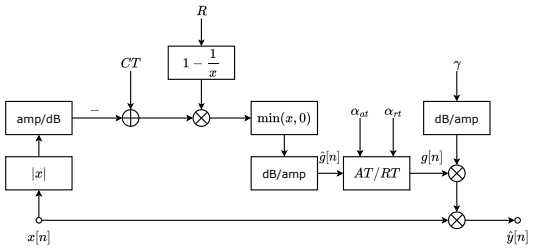


Fig. 1: Block diagram of the feed-forward compressor. "AT/RT" refer to the ballistics control in Eq. (2).

$$\theta \mapsto \theta - [\nabla^2 \mathscr{L}(\theta)]^{-1} \nabla \mathscr{L}(\theta). \tag{5}$$

Here $\mathscr{L}(\theta) = \mathscr{D}(f_{\mathbf{x}}(\theta), \mathbf{y})$, and $\nabla\mathscr{L}(\theta) \in \mathbb{R}^M$ and $\nabla^2\mathscr{L}(\theta) \in \mathbb{R}^{M\times M}$ are the gradient and Hessian matrix of $\mathscr{L}$ at θ, respectively. NR leverages second-order information from the Hessian; thus, it guarantees convergence to a local minimum and converges faster than gradient descent, especially when the initial point is close to the minimum.

NR has two requirements that need to be satisfied:

1. $\mathscr{L}$ must be convex around the optimal θ^*.
2. $\mathscr{L}$ must be twice differentiable.

We validate 1) empirically by observing the convergence behaviour, which we will discuss more in Section 4.4. For 2), we set the distance function to be the square distance $\mathscr{D}(f_{\mathbf{x}}(\theta), \mathbf{y}) = (f_{\mathbf{x}}(\theta) - \mathbf{y})^\top (f_{\mathbf{x}}(\theta) - \mathbf{y})$, which is twice differentiable. Regarding the differentiability of the feed-forward compressor f, although `torchcomp` computes the sub-gradients for operations that are not differentiable everywhere (the absolute and minimum functions in Fig. 1 and the if-else statement in Eq. (2)), it is shown that they are accurate enough for first-order optimisation. Since its gradient backpropagation function is also differentiable, we can quickly evaluate its second-order derivatives within automatic differentiation frameworks like PyTorch [14].

3 Hessian Computation

`torchcomp` implements the Vector-Jacobian product (VJP) [15, 16] $\mathbf{v} \mapsto \mathbf{v}^\top \nabla f_{\mathbf{x}}(\theta)$ of the compressor, which lets us compute the VJP of the loss function $\mathbf{v} \mapsto \mathbf{v}^\top \nabla \mathscr{L}(\theta)$ using backpropagation where

$\nabla\mathscr{L}(\theta) = \nabla_{f_\mathbf{x}(\theta)}\mathscr{D}(f_\mathbf{x}(\theta),\mathbf{y})\nabla_\theta f_\mathbf{x}(\theta)$. We get the gradient by setting $\mathbf{v}=1$.

To get the Hessian matrix $\nabla^2\mathscr{L}(\theta)$, note that the Hessian matrix is the Jacobian of $\nabla\mathscr{L}(\theta)$. Given an identity matrix $\mathbf{I}\in\mathbb{R}^{M\times M}$, if we pass the i^{th} column of $\mathbf{I}$ into the VJP of $\nabla\mathscr{L}:\mathbb{R}^M\mapsto\mathbb{R}^M$, which is $\mathbf{v}\mapsto\mathbf{v}^\top\nabla^2\mathscr{L}(\theta)$, the result will be the i^{th} row of the Hessian $\nabla^2\mathscr{L}(\theta)$; computing the entire Hessian equals passing every column of $\mathbf{I}$ to the VJP function in parallel ($\mathbf{I}^\top\nabla^2\mathscr{L}(\theta)$).

Although we can get the Hessian using VJP twice (which is known as *reverse-over-reverse* mode) as we show above, the second (sometimes also called outer) Jacobian can also be computed with Jacobian-Vector product (JVP) $\mathbf{v}\mapsto\nabla h(\theta)\mathbf{v}$ [17]. This is called *forward-over-reverse* mode where $h=\nabla\mathscr{L}$. Since the Jacobian of $\nabla h(\theta)$ is a square matrix, computing it by passing the basis vectors, either from the right or the left, has to be repeated the same number of times. The efficiency thus depends on the computational costs of VJP and JVP of $h:\mathbb{R}^M\to\mathbb{R}^M$, although in practice, the *forward-over-reverse* strategy is usually preferred. For a fair comparison, we derive the JVP of the function $\mathbf{v}\mapsto\mathbf{v}^\top\nabla f_\mathbf{x}(\theta)$, specifically, the backpropagation of the ballistics module in Eq. (2), as it involves recursive filters and is non-trivial to implement in PyTorch.

3.1 Forward-Mode Gradients for Time-Varying One-Pole Filters

To get the gradients of Eq. (2), let us rewrite it as a time-varying one-pole filter:

$$\beta[n] = 1-\alpha_{at}^{\zeta[n]}\alpha_{rt}^{1-\zeta[n]}, \tag{6}$$

$$\tilde{g}[n] = (1-\beta[n])\hat{g}[n], \tag{7}$$

$$\underbrace{g[n]}_{\text{output}} = \underbrace{\tilde{g}[n]}_{\text{input}} + \underbrace{\beta[n]}_{\text{multiplier}}\ \underbrace{g[n-1]}_{\text{previous output}}, \tag{8}$$

where $\zeta[n]\in\{0,1\}$ is the attack and release phase indicator function. We colour-coded Eq. (8) to emphasise the recursions as it will show up multiple times in the derivations, and we have special treatment for it in Section 4.2.

According to [9], backpropagating $\nabla_{g[n]}\mathscr{L}(\theta)$ through Eq. (8) can be expressed as passing through a time-varying one-pole filter in reversed time:

$$\nabla_{\tilde{g}[n]}\mathscr{L}(\theta) = \nabla_{g[n]}\mathscr{L}(\theta) + \beta[n+1]\nabla_{\tilde{g}[n+1]}\mathscr{L}(\theta). \tag{9}$$

The gradients $\nabla_{\beta[n]}\mathscr{L}(\theta)$ can be computed using chain rule:

$$\nabla_{\beta[n]}\mathscr{L}(\theta) = \nabla_{\tilde{g}[n]}\mathscr{L}(\theta)(g[n-1]-\hat{g}[n]). \tag{10}$$

The gradients $\nabla_{\alpha_{at}}\mathscr{L}(\theta)$, $\nabla_{\alpha_{rt}}\mathscr{L}(\theta)$, and $\nabla_{\hat{g}[n]}\mathscr{L}(\theta)$ are trivial to compute given Eq. (6) and Eq. (7).

Backpropagation of $\nabla_{\tilde{g}[n]}\nabla\mathscr{L}(\theta)$ (we simplify $\nabla_{\nabla_*\mathscr{L}(\theta)}$ as ∇_* for brevity) through Eq. (9) can be done in the same way as in Eq. (8):

$$\nabla_{g[n]}\nabla\mathscr{L}(\theta) = \nabla_{\tilde{g}[n]}\nabla\mathscr{L}(\theta) + \beta[n]\nabla_{g[n-1]}\nabla\mathscr{L}(\theta), \tag{11}$$

$$\nabla_{\beta[n+1]}\nabla\mathscr{L}(\theta) = \nabla_{g[n]}\nabla\mathscr{L}(\theta)\nabla_{\tilde{g}[n+1]}\mathscr{L}(\theta). \tag{12}$$

For forward propagation (JVP) on Eq. (9), let us use Eq. (8) as an example. Since in the forward mode the Jacobians are multiplied from right to left (opposite to the backpropagation), we need to apply Eq. (9) and Eq. (10) in reverse order and traverse the computational graph from the outputs back to the inputs. This also means applying recursive filters in the opposite direction. Given forwarded gradients $\nabla_\theta\tilde{g}[n]$ and $\nabla_\theta\beta[n]$, the forward version of Eq. (9) is:

$$\nabla_\theta g[n] = [\nabla_\theta\tilde{g}[n] + \nabla_\theta\beta[n]\,(g[n-1]-\hat{g}[n])] + \beta[n]\nabla_\theta g[n-1], \tag{13}$$

which is just changing the input of Eq. (8) to $\nabla_\theta\tilde{g}[n] + \nabla_\theta\beta[n](g[n-1]-\hat{g}[n])$. Applying the same process to Eq. (9), we can get the forward version of Eq. (11) similar to Eq. (13) as:

$$\nabla_\theta\nabla_{\tilde{g}[n]}\mathscr{L}(\theta) = [\nabla_\theta\nabla_{g[n]}\mathscr{L}(\theta) + \nabla_\theta\beta[n]\nabla_{\tilde{g}[n+1]}\mathscr{L}(\theta)] + \beta[n+1]\nabla_\theta\nabla_{\tilde{g}[n+1]}\mathscr{L}(\theta). \tag{14}$$

Comparing the VJP of Eq. (9) (Eq. (11) and Eq. (12) combined) to its JVP (Eq. (14)), we see that the JVP has one more addition than the VJP. Nevertheless, the extra addition does not occur within the recursion, and modern computers are very good at vectorised atomic addition, so the computation time is often negligible.

In summary, in this section, we show that propagating the gradients of a time-varying one-pole filter in forward mode equals filtering the gradients with the same filter[2]. The resulting Eq. (13) is required for the JVP of the inner Jacobian $\mathbf{v}\mapsto\nabla\mathscr{L}(\theta)\mathbf{v}$, while Eq. (14) is required for the JVP of the outer Jacobian $\mathbf{v}\mapsto\nabla h(\theta)\mathbf{v}$.

[2]This result generalises to higher order all-pole filters as well.

4 Experiment

4.1 Dataset

We use the recordings from SignalTrain [18] as it is the largest curated analogue compressor dataset known to the authors and has been utilised in many previous works [3, 8, 9, 5]. It contains paired recordings sampled at 44.1 kHz from LA-2A. Technically, the LA-2A is a levelling amplifier (LA) and does not have the parameters mentioned in Section 2.1. It has only one knob, called *peak reduction*, that controls the compression. The peak reduction was sampled from 0 to 100 with a spacing of 5, and the gain value was fixed for the compressor and limiter modes. We select recordings labelled with `3c` in the file name, which use an identical 20 min audio stimulus consisting of both musical and synthetic sounds for all configurations.

4.2 Acceleration Techniques

We implement everything in PyTorch. To accelerate the training process, we split the audio into 12 s chunks with 1 s overlap, so $f_{\mathbf{x}}$, which is inherently sequential in contrast to $\nabla f_{\mathbf{x}}$, can be parallelised at the batch level. The overlap parts are used as a warm-up, and the loss is computed only on each chunk's last 11 s. Following [8], we apply a pre-emphasis filter $\frac{1-z^{-1}}{1-0.995z^{-1}}$ to $\mathbf{y}$ and $\hat{\mathbf{y}}$ prior to computing the loss to reduce low-frequency noise.

Furthermore, we identify that a lot of the operations are one-pole filters (Eq. (8)), and a parallel algorithm exists for these filters called *parallel associative scan* [19]. The idea is to convert the recursion into a form like $s[0] \oplus s[1] \oplus \ldots \oplus s[n]$ where $\oplus$ is associative, then the scan algorithm can be applied. We use the CUDA implementation from [20] to parallelise the coloured equations (except Eq. (8)) and the pre-emphasis filter on the GPU.

4.3 Analysis on Computational Cost

PyTorch has two implementations for computing the Hessian. One lies under the `autograd` module, which is the default implementation and has been around since PyTorch 1.5. It only has full coverage for *reverse-over-reverse* (hereafter referred to as *rev-rev*) mode. The other implementation is the `func` module, introduced in PyTorch 2.0, which is relatively new but allows for the flexible composition of forward and reverse mode Jacobians. We evaluate all the four combination strategies for the Hessian $\nabla^2 \mathscr{L}(\boldsymbol{\theta})$ using `func` and compare them to the default `autograd` implementation.

Table 1: The memory cost (in MB) and runtime (in milliseconds) of different hessian computation strategies in PyTorch.

	`autograd`	`func`			
	rev-rev	rev-rev	fwd-rev	rev-fwd	fwd-fwd
Memory	**1066**	2358	3534	6306	5364
Time	**26.5**	28.1	28.1	64.7	38.9

We report the computational cost in Table 1, measured on an RTX 3060 with a batch size of 16. From Table 1, we see that the *rev-rev* and *fwd-rev* modes are equally fast, which implies that the extra addition we mentioned in Section 3 does not impact the performance. The *rev-fwd* mode is the slowest and most memory-consuming. We see that *fwd-rev* requires more memory than *rev-rev*, probably due to non-optimal memory utilisation for the forward gradients in our implementation[3]. The table also shows inherent overhead in memory and time for the `func` module, as the numbers are all bigger than those of `autograd`. This is likely because this feature is still in the Beta stage and has not been optimised enough.

Based on the above findings, we choose the `autograd` implementation. All the computations for each configuration can be fitted on a single RTX 3060 with 12GB of VRAM, and no mini-batching is required. The resulting optimisation takes roughly just 3.5 s for one NR update. Moreover, training on the entire dataset takes less than 20 min, instead of hours as reported in [8].

4.4 Optimisation Strategy

We perform the damped NR method with backtracking line search [21]. In each step, the parameters are updated as $\boldsymbol{\theta} \mapsto \boldsymbol{\theta} - \tau \boldsymbol{\nu}$, where $\tau \in [0,1]$ is the step size and $\boldsymbol{\nu}$ is the solution of $\nabla^2 \mathscr{L}(\boldsymbol{\theta})\boldsymbol{\nu} = \nabla \mathscr{L}(\boldsymbol{\theta})$. We use an equation solver to avoid performing matrix inversion for better numerical stability. Starting from $\tau = 1$, we divide τ by 2 until the Armijo–Goldstein

[3] In our filter implementation, only one dimension of the tensor is treated as batch dimension. The straightforward way to call the JVP M times in parallel using the same function is to duplicate the parameters M times along the batch dimension.

condition [22] $\mathscr{L}(\theta - \tau v) \leq \mathscr{L}(\theta) - \alpha\tau\nabla\mathscr{L}(\theta)^\top v$ is satisfied with $\alpha = 0.0001$. If the Hessian matrix is not positive semi-definite (which implies negative curvature), we randomly sample a new direction orthogonal to v and repeat the process.

We find different optimal parameters for each sampled peak reduction. Since the NR method is sensitive to the initialisation, we start from 100 peak reduction which has the heaviest compression so it is easier to fit, and then use the previous result to initialise the subsequent peak reduction, e.g., $100 \rightarrow 95 \rightarrow 90$, etc. This ensures the starting point for each configuration is as close to the solution as possible. The initial values are determined empirically based on early experiments and shared beliefs of the target circuit, which are $CT = -36$ dB, $\gamma = 0$ dB, $R = 4$, $t_{at} = 1$ ms, and $t_{rt} = 200$ ms. We restrict the range of the parameters to $R \in [1, 20]$, $t_{at} \in [0.1, 100]$ ms, and $t_{rt} \in [10, 1000]$ ms.

The optimisation converges in less than 10 iterations in most cases. We stopped the process at 40 peak reduction, as the NR method could not find a consistent solution for lower peak reductions. The optimisation encounters negative curvature structure frequently [4]. We investigated the data and found that the compression is less noticeable in this range and is surpassed by other non-linearities that a feed-forward compressor cannot capture. For other peak reductions, the NR method converges to the same solution in multiple different initialisations, verifying that $\mathscr{L}(\theta)$ is mostly convex around the optimal solution.

4.5 Results

> *...The average compression ratio is always set at roughly 3:1, while the average Attack time is 10 milliseconds, and the Release time is about 60 milliseconds for 50% of the release, and anywhere from 1 to 15 seconds for the rest.* [5]

The learnt configurations are shown in Fig. 2. The mappings to the parameters are nearly identical between the compressor and limiter modes, except for the ratio, which is slightly higher in the limiter mode but exhibits a similar trend. The ratio is typically around 4:1 in compressor mode, slightly deviating from the manufacturer's stated ratio of 3:1. The attack and release times vary exponentially with the peak reduction, rather than fixed values. Since there is only one release time parameter, it does not capture the documented two-stage release behaviour of the LA-2A, but an average of them.

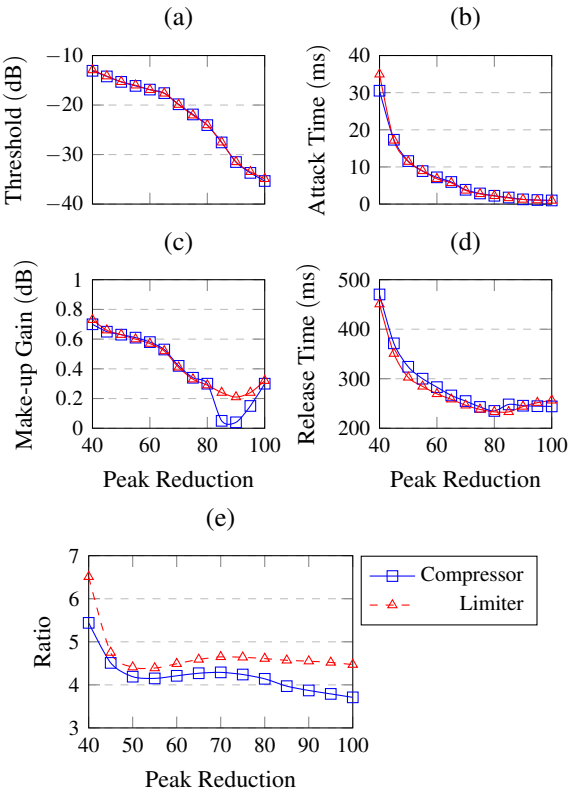


Fig. 2: The mapping from LA-2A peak reduction to the compressor's parameters.

5 Evaluation

We compare our compressor (denoted as 4A-2A and "4" stands for feed-forward) with the following baselines: the official UAD LA-2A plugin [6], the emulation plugin CA-2A by Cakewalk [7], the emulation plugin CLA-2A by Waves [8], and the grey-box compressor from [8]. We evaluate 4A-2A using its training data, as the five-parameter model cannot fully represent the LA-2A and no overfitting is observed on this dataset

[4] For peak reductions over 40, this rarely happens (it happens a few times when random initialisation is used).

[5] www.uaudio.com/blog/la-2a-collection-tips-tricks

[6] www.uaudio.com/uad-plugins/compressors-limiters/teletronix-la-2a-tube-compressor

[7] legacy.cakewalk.com/Products/CA-2A

[8] www.waves.com/plugins/cla-2a-compressor-limiter

with more complex models [8]. We evaluate performance using the Error-to-Signal Ratio (ESR) and the Loudness Dynamic Range difference (ΔLDR). ESR is defined as $\frac{(\mathbf{y}-\hat{\mathbf{y}})^\top(\mathbf{y}-\hat{\mathbf{y}})}{\mathbf{y}^\top\mathbf{y}}$. ΔLDR is the difference between the LDR [23] of the target signal and the predicted signal. The same pre-emphasis filter is applied to all audio before evaluations.

To calculate LDR given $\mathbf{y}$, we first compute the log ratio between the short-term and long-term RMS envelopes as

$$L_{\mathbf{y}}[n] = 10\log_{10}\frac{\mathrm{RMS}_{\mathrm{short}}(y[n])}{\mathrm{RMS}_{\mathrm{long}}(y[n])}. \tag{15}$$

The integration time for the short-term RMS is set to 50 ms and the long-term RMS is set to 3 s where the details can be found in [23]. We define $\mathrm{LDR}(\mathbf{y})$ as

$$\mathrm{LDR}(\mathbf{y}) = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} L_{\mathbf{y}}[n]^2}, \tag{16}$$

which tells how much the loudness varies locally (microdynamics) in $\mathbf{y}$. ΔLDR is simply $\mathrm{LDR}(\hat{\mathbf{y}}) - \mathrm{LDR}(\mathbf{y})$ so if $\Delta\mathrm{LDR} > 0$, it means $\hat{\mathbf{y}}$ is less compressed than $\mathbf{y}$ and vice versa.

Since we do not know what setup and gain staging were used to model the commercial plugins, the plugins' input and output gains must be tuned. We manually adjust the input gain so that the sum of its ΔLDR across peak reductions is close to zero, ensuring its compression curve matches the SignalTrain dataset. We then adjust the output gain so that its overall ESR is minimised. We render the audio through plugins using `pedalboard` [24].

We choose the Model 5 configuration for the grey-box compressor, their best-performing model. It is architecture-wise very similar to our 4A-2A, but the make-up gain γ is replaced with a GRU. Thus, we follow their setup to train a GRU on the same training set but pre-processed by 4A-2A with $\gamma = 0$ dB. We set the batch size to 26 (13 configurations × two modes) and the sequence length to 12288. The training converges after around eight epochs. We stop the training at the 15$^{\text{th}}$ epoch and pick the checkpoint with the lowest loss. The process takes less than 30 minutes on an RTX 3060 Laptop GPU. We denote this model as 4A-2A-G.

The evaluation results are shown in Fig. 3. 4A-2A-G has the lowest ESR and is close to those reported in [8]. 4A-2A shows comparable performance to the official UAD plugin. At around 75 peak reduction, the LA-2A behaves most like the feed-forward compressor, thus having the lowest ESR. Beyond this point, the 4A-2A's ESR increases, and the errors are more for the limiter mode. For the commercial plugins, their curves are less smooth and have a few apparent spikes. Among them, the CA-2A has the highest ESR in general and even reaches over 40 % at 100 peak reduction.

Regarding their ΔLDR profiles, we see that CA-2A varies the most with a maximum mismatch of 1 dB at 100 peak reduction. UAD and CLA-2A exhibit similar patterns of difference, which can be divided into three stages: from 40 to 60 (peak reduction), their compressions are lighter than the ground truth; then, the compressions become heavier from 60 to 80 and then return to lighter levels beyond 80. 4A-2A and 4A-2A-G have their ΔLDR very close to zero thus out-performing others, where 4A-2A-G being the closest.

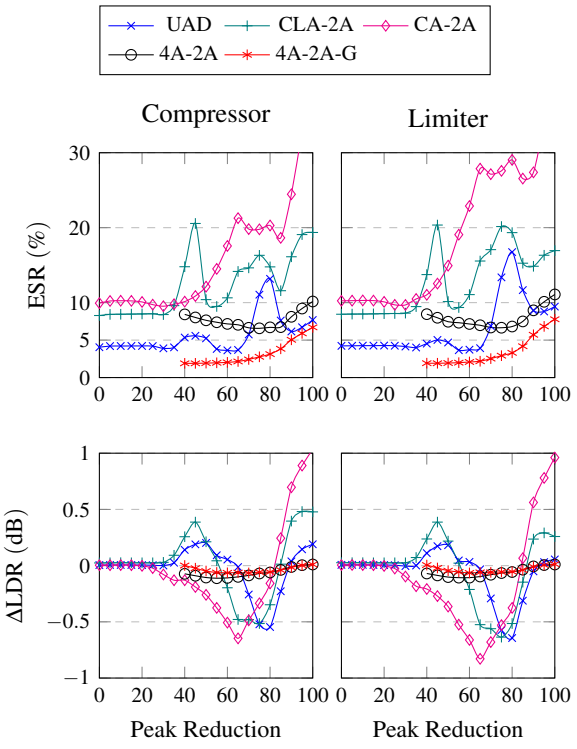


Fig. 3: ESR and ΔLDR of all the evaluated models under different peak reduction and mode.

6 Application: the 4A-2A Plugin

We implement our 4A-2A compressor as a real-time audio plugin using JUCE [25], providing it as a cre-

ative tool for audio engineers. In the plugin interface (Fig. 4), we have five sliders that control the parameters of the underlying compressor. At the bottom, there is one slider for peak reduction and a switch for the mode. The peak reduction slider is linked to the above compressor's parameters, mapping the peak reduction to them using the learnt mappings from Fig. 2. The mode switch toggles between the compressor and limiter profiles. The user can adjust the peak reduction and observe the compressor's behaviour in real-time. If they want to fine-tune the compressor, they can adjust the parameters manually.

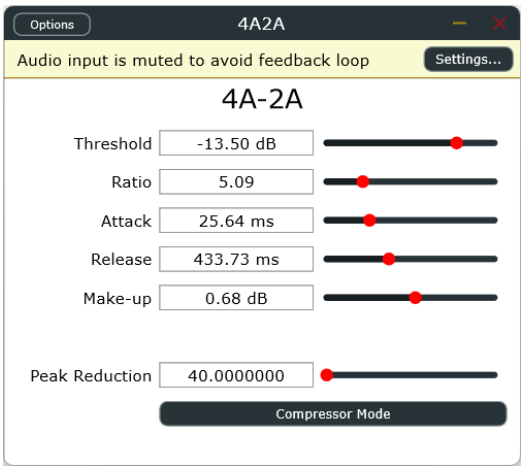


Fig. 4: The GUI of the 4A-2A plugin.

The peak reductions in SignalTrain are sampled on a grid. Interpolation is required for continuous mapping $[40,100] \rightarrow \mathbb{R}^5$. To evaluate the interpolation error, we exclude peak reductions $\{45,55,\ldots,95\}$ and use the remaining 7 points to interpolate them using either linear or cubic spline interpolation. Their ESR is shown in Fig. 5. Linear interpolation and spline interpolation perform the worst at 65 and 95 peak reduction, respectively. On average, the spline interpolation has a slightly higher ESR than the linear interpolation. We suppose this finding generalises to denser grids and choose the linear interpolation for the plugin.

We made the GRU model from 4A-2A-G into a standalone plugin. The GRU model is very lightweight, consisting of only one layer with eight hidden units. We wrapped the GRU model as a PyTorch script module using Neutone [26], which can be loaded into their Neutone FX plugin [9]. Cascading the 4A-2A and the

[9] github.com/Neutone/neutone_sdk

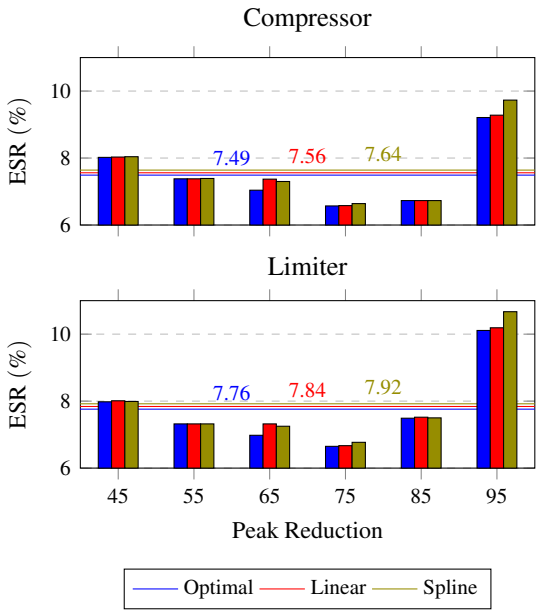


Fig. 5: Comparison of different interpolation methods. The horizontal lines are the averages.

GRU model resembles the 4A-2A-G baseline, a real-time implementation of the Model 5 emulator from [8], with the feed-forward part being trained using the NR method, outperforming all commercial plugins in our evaluation.

Comparing the residuals with or without the GRU make-up gain shown in Fig. 6, the most noticeable difference is in the high-frequency range (> 4 kHz). The GRU successfully captures the non-linearities of these analogue circuits, which cannot be modelled by a simple digital compressor, and spreads out the errors across frequencies.

7 Discussion

The NR method effectively finds a unique solution to match the LA-2A's response to the square distance criterion. There is room for improvement, as LA-2A is technically a feedback compressor; therefore, the signal flows of 4A-2A and LA-2A are not precisely matched. By using a digital feedback compressor ($s[n] = \hat{y}[n-1]$) and modelling the two-stage release behaviour explicitly, we can potentially improve the performance. However, efficiently backpropagating gradients through the feedback path is challenging and requires custom implementations. In contrast, our grey-box sound-matching

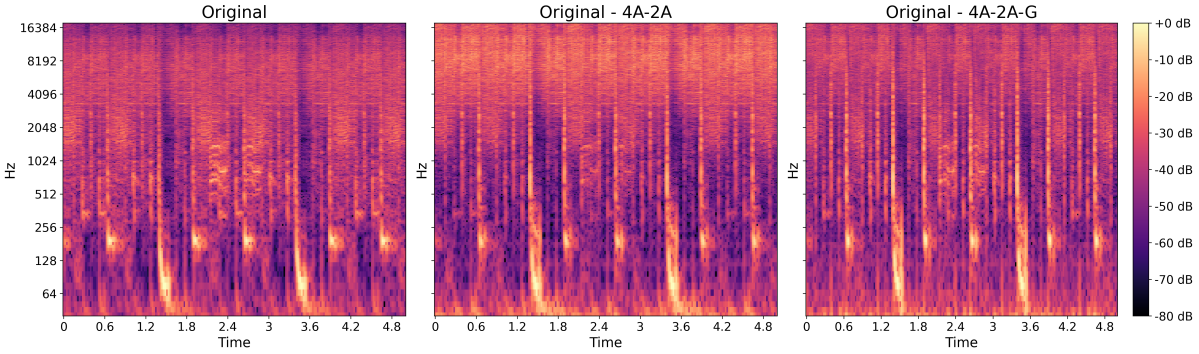


Fig. 6: The spectrograms of a drum loop from the SignalTrain and the residuals ($\mathbf{y} - \hat{\mathbf{y}}$) of the model's predictions with 75 peak reductions operated in compressor mode.

approach can be generalised to any dynamic range controller circuit, and our trial on the LA-2A already yields satisfactory results.

Due to the variations between analogue circuits, the inferior plugin performance we see in Section 5 is expected. Their modelled LA-2A might behave slightly differently from the SignalTrain one. It is interesting to note that the UAD and CLA-2A have similar ESR/∇LDR patterns, which could be due to similar target circuits or modelling approaches.

We evaluate the entire Hessian matrix, as the number of parameters is smaller. However, for more complex effects and longer effect chains, the Hessian can become too large to fit in memory. Methods like conjugate gradients using Hessian-vector product $\mathbf{v} \mapsto \nabla^2 \mathscr{L}(\theta)\mathbf{v}$ can be used to solve the equation $\nabla^2 \mathscr{L}(\theta)\nu = \nabla \mathscr{L}(\theta)$ without explicitly computing the Hessian [17, 27]. Nevertheless, increasing the number of parameters also raises the possibility of encountering more local minima, making it harder to find the optimal solution.

8 Conclusion

This paper presented a method to emulate the LA-2A optical compressor using a feed-forward digital compressor with parameters optimised via the Newton-Raphson method. Our approach successfully turned the LA-2A's response into interpretable parameters that can be used for creative control. The efficient implementation of ballistics and the parallel associative scan algorithm for gradient computation significantly accelerate optimisation. Forward-mode gradient propagation through the compressor is derived to benchmark different strategies for computing the Hessian.

Our findings suggest that the Newton-Raphson method is a promising approach for the grey-box modelling of audio compressors, providing a foundation for further research and development in this area. Further improvements could be made by using a feedback design and explicitly modelling the LA-2A's two-stage release behaviour. Future work could extend Newton-Raphson optimisation to other analogue compressors or non-linear audio effects. Another direction could be reducing the computational costs of the Hessian matrix for more complex signal chains.

Acknowledgements

The first author is a research student supported jointly by UKRI (grant number EP/S022694/1) and Queen Mary University of London.

References

[1] Eichas, F., Gerat, E., and Zölzer, U., "Virtual analog modeling of dynamic range compression systems," in *AES 142nd Convention*, 2017.

[2] Hawley, S. H., Colburn, B., and Mimilakis, S. I., "Profiling Audio Compressors with Deep Neural Networks," in *AES 147th Convention*, 2019.

[3] Steinmetz, C. J. and Reiss, J. D., "Efficient neural networks for real-time modeling of analog dynamic range compression," in *AES 152nd Convention*, 2022.

[4] Comunità, M., Steinmetz, C. J., Phan, H., and Reiss, J. D., "Modelling black-box audio effects with time-varying feature modulation," in *Proc. IEEE ICASSP*, pp. 1–5, 2023.

[5] Yeh, Y.-T., Hsiao, W.-Y., and Yang, Y.-H., "Hyper Recurrent Neural Network: Condition Mechanisms for Black-box Audio Effect Modeling," in *Proc. DAFx*, pp. 97–104, 2024.

[6] Simionato, R. and Fasciani, S., "Comparative Study of Recurrent Neural Networks for Virtual Analog Audio Effects Modeling," *arXiv preprint arXiv:2405.04124*, 2024.

[7] Simionato, R. and Fasciani, S., "Modeling Time-Variant Responses of Optical Compressors with Selective State Space Models," *Journal of the Audio Engineering Society*, 73, pp. 144–165, 2025.

[8] Wright, A. and Välimäki, V., "Grey-box modelling of dynamic range compression," in *Proc. DAFx*, pp. 304–311, 2022.

[9] Yu, C.-Y., Mitcheltree, C., Carson, A., Bilbao, S., Reiss, J. D., and Fazekas, G., "Differentiable All-pole Filters for Time-varying Audio Systems," in *Proc. DAFx*, pp. 345–352, 2024.

[10] Bernardini, A., Bozzo, E., Fontana, F., and Sarti, A., "A Wave Digital Newton-Raphson Method for Virtual Analog Modeling of Audio Circuits with Multiple One-Port Nonlinearities," *IEEE/ACM TASLP*, 29, pp. 2162–2173, 2021.

[11] Bilbao, S., Torin, A., and Chatziioannou, V., "Numerical Modeling of Collisions in Musical Instruments," *Acta Acustica united with Acustica*, 101(1), pp. 155–173, 2015.

[12] Zölzer, U., *DAFX: Digital Audio Effects*, chapter Nonlinear Processing, pp. 110–112, John Wiley & Sons, 2011, ISBN 0470665998.

[13] Giannoulis, D., Massberg, M., and Reiss, J. D., "Digital dynamic range compressor design—A tutorial and analysis," *Journal of the Audio Engineering Society*, 60(6), pp. 399–408, 2012.

[14] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. NeurIPS*, pp. 8026 – 8037, 2019.

[15] Rirsch, M. W. and Smale, S., *Differential equations, dynamical systems, and linear algebra*, ACADEMIC PRESS. INC., 1974.

[16] Robeyns, M., "Of VJPs and JVPs," `https://maximerobeyns.com/of_vjps_and_jvps`, 2023, [Accessed: 2025-02-17].

[17] Dagréou, M., Ablin, P., Vaiter, S., and Moreau, T., "How to compute Hessian-vector products?" in *ICLR Blogposts*, 2024, `https://iclr-blogposts.github.io/2024/blog/bench-hvp/`.

[18] Colburn, B. and Hawley, S., "SignalTrain LA2A Dataset," 2020, doi:10.5281/zenodo.3824876.

[19] Blelloch, G. E., "Prefix Sums and Their Applications," Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, 1990.

[20] Martin, E. and Cundy, C., "Parallelizing Linear Recurrent Neural Nets Over Sequence Length," in *ICLR*, 2018.

[21] Moré, J. J. and Sorensen, D. C., "Newton's method," Technical report, Argonne National Lab.(ANL), Argonne, IL (United States), 1982.

[22] Armijo, L., "Minimization of functions having Lipschitz continuous first partial derivatives," *Pacific Journal of Mathematics*, 16(1), pp. 1 – 3, 1966.

[23] Nercessian, S., McClellan, R., and Lukin, A., "A direct microdynamics adjusting processor with matching paradigm and differentiable implementation," in *Proc. DAFx*, 2022.

[24] Sobot, P., "Pedalboard," 2021, doi:10.5281/zenodo.7817838.

[25] Raw Material Software Limited, "JUCE: Jules' Utility Class Extensions," `https://juce.com/`, 2025, [Accessed: 2025-02-16].

[26] Fyfe, A. and Mitcheltree, C., "Neutone-Real-time AI audio plugin for DAWs," in *Audio Developer Conference*, 2022.

[27] Hestenes, M. R. and Stiefel, E., "Methods of conjugate gradients for solving linear systems," *Journal of research of the National Bureau of Standards*, 49, pp. 409–435, 1952.