Blind estimation of audio effects using an auto-encoder approach and differentiable digital signal processing

Côme Peladeau, Geoffroy Peeters

▶ **To cite this version:**

Côme Peladeau, Geoffroy Peeters. Blind estimation of audio effects using an auto-encoder approach and differentiable digital signal processing. ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, Apr 2024, Seoul, South Korea. pp.856-860, 10.1109/ICASSP48485.2024.10448301 . hal-04539329

HAL Id: hal-04539329
https://hal.science/hal-04539329v1

Submitted on 9 Apr 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

BLIND ESTIMATION OF AUDIO EFFECTS USING AN AUTO-ENCODER APPROACH AND DIFFERENTIABLE DIGITAL SIGNAL PROCESSING

Côme Peladeau and Geoffroy Peeters

LTCI - Télécom-Paris, Institut Polytechnique de Paris, France

ABSTRACT

Blind Estimation of Audio Effects (BE-AFX) aims at estimating the audio effects (AFXs) applied to an original, unprocessed audio sample solely based on the processed audio sample. To train such a system traditional approaches optimize a loss between ground truth and estimated AFX parameters. This involves knowing the exact implementation of the AFXs used for the process. In this work, we propose an alternative solution that eliminates the requirement for knowing this implementation. Instead, we introduce an auto-encoder approach, which optimizes an audio quality metric. We explore, suggest, and compare various implementations of commonly used mastering AFXs, using differential signal processing or neural approximations. Our findings demonstrate that our auto-encoder approach yields superior estimates of the audio quality produced by a chain of AFXs, compared to the traditional parameter-based approach, even if the latter provides a more accurate parameter estimation.

Index Terms— audio effects, differentiable digital signal processing, neural proxy, deep learning

1. INTRODUCTION

Audio effects (AFXs) play an essential role in music production. They are used during mixing to sculpt sounds for artistic purposes or context requirements (such as when a sound needs to be mixed with others). They are used during mastering, the final stage of production, to improve the clarity of a given mix, adapt it for a given media (such as vinyl or streaming), or harmonize it with other tracks in the album. For these reasons, its automatization has been the subject of several softwares[1] which allow learning the mastering EQ of a given track to apply it to another. In this work, we study the generalization to other common mastering AFXs.

Blind Estimation of Audio Effects (BE-AFX) aims at estimating the audio effects (AFXs) applied to an original, unprocessed (dry) audio sample $\mathbf{x}$ solely based on the observation of the processed (wet) audio sample $\mathbf{y}$. This estimation takes the form of the AFXs and their parameters $\mathbf{p}$.

1.1. Related works

For long BE-AFX techniques have been based on explicit rules and assumptions. For example, Ávila et al. [1] proposed to estimate the curve of a memoryless non-linear distortion by assuming that the unprocessed signal has the statistics of a Gaussian white noise. However, nowadays, most BE-AFX approaches rely on training neural networks. Indeed, following SincNet [2] and DDSP [3], modeling audio processes as differential processes has allowed developing differentiable AFXs as specialized neural networks layers with interpretable parameters [4, 5, 6]. Because they are differentiable, they can be integrated transparently in a neural network. Since then, differentiable AFXs have been used for many tasks: automatic mixing and mastering [7, 8], production style transfer [6] or estimation of audio effects [9].

In the case of BE-AFX, neural networks are usually trained to minimize a loss function that aims at reconstructing the AFX chain and its parameters as did Hinrichs et al. [10] or Lee et al. [11]. However, while their approach is flexible, its training requires the knowledge of the used AFXs and their parameters. Our approach does not. Also, as we will highlight in this work, a parameter distance does not translate well to a perceptual distance between audio effects. This is why we will propose here the use of an audio distance.

Estimation of audio effects with an audio loss function and differentiable audio effects has already been investigated. For example, Colonel et al. [12] used it for non-linear distortion using a differentiable Wiener-Hammerstein model and also in [9] for a complete mixing setting. However, in both cases, their approaches require paired $\mathbf{x}$ and $\mathbf{y}$ data for the estimation, i.e. they did not perform the blind estimation. In this work, we perform blind estimation, i.e. we aim at estimating the AFXs applied to $\mathbf{x}$ using only the knowledge of $\mathbf{y}$.

1.2. Proposal

To solve the BE-AFX problem, we propose an auto-encoder approach which is illustrated in Figure 1. In the left part, we construct synthetic processed mixes by applying a set of (synthesis) audio effects $\{e^s\}$ with known parameters $\mathbf{p}$ to an unprocessed mix $\mathbf{x}$. The results are our ground-truth processed mixes $\mathbf{y}$. Using only $\mathbf{y}$, an analysis network f^a then estimates the set of parameters $\hat{\mathbf{p}}$ to be used to process $\mathbf{x}$ with (analysis) audio effects $\{e^a\}$ to produce an estimated audio sample $\hat{\mathbf{y}}$. The analysis network f^a is trained to minimize an audio loss function between $\hat{\mathbf{y}}$ and $\mathbf{y}$ so that $\hat{\mathbf{y}} \approx \mathbf{y}$. It therefore closely matches the formulation of an auto-encoder.

Doing so, when $\{e^a\}$=$\{e^s\}$, f^a implicitly learns to replicate the parameters $\mathbf{p}$ given only $\mathbf{y}$. When $\{e^a\} \neq \{e^s\}$, f^a learns parameters to be used for $\{e^a\}$ such that the effect of the analysis chain sounds similar to the synthesis chain.

1.3. Paper organization

To be able to estimate $\mathbf{p}$ using this auto-encoder, we should be able to "differentiate" the audio effects $\{e^a\}$ (i.e. compute the derivative of their outputs $\hat{\mathbf{y}}$ w.r.t. their inputs $\hat{\mathbf{p}}$). We therefore discuss various implementations of the audio effects in part 2.1. To be able to estimate $\mathbf{p}$ we should define an architecture for the analysis network f^a. We therefore discuss various possible architectures in part 2.2. During evaluation (part 3), we first decide for each type of effect what

This work was performed under fundings by the 'Hi! PARIS Call for Collaborative and scientific Projects 2022' and the 'AQUA-RIUS project founded by the ANR-22-CE23-0022 program'.

[1] such as Izotope Ozone 11 or Sonible smart:EQ 4.

©2024 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

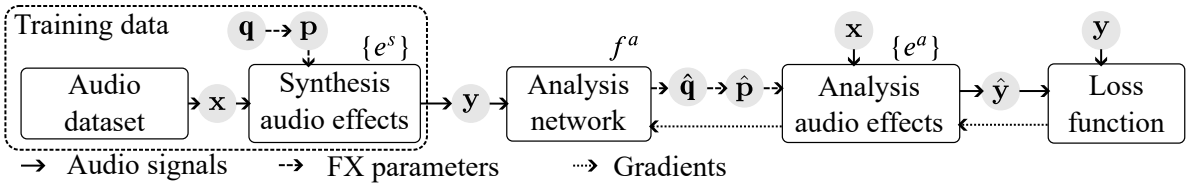

Fig. 1. Proposed auto-encoder approach for Blind Estimation of Audio Effects (BE-AFX).

is the best implementation e^a (among those of 2.1) and architecture of f^a (among those of 2.2) to reconstruct $\mathbf{y}$. We then compare our proposed approach (audio reconstruction $\hat{\mathbf{y}} \approx \mathbf{y}$) to the previously proposed approach (parameter reconstruction $\hat{\mathbf{p}} \approx \mathbf{p}$). Finally, we evaluate the joint estimation of the whole AFX chain defined as the succession of an equalizer, a dynamic range compressor (DRC), and a clipper. We conclude in part 4 and propose future directions.

We provide the code of this study to ensure replicability as well as audio examples.[2]

2. PROPOSAL

2.1. Audio effects

In this work, we only consider the 3 following AFXs commonly used for mastering [13]: - an **equalizer** (a cascade of linear filters which modifies the level of different frequency bands), - a **dynamic range compressor** (which reduces the signal level when it is too loud, leaving it untouched otherwise), and - a **soft clipper** (which clips the signal peaks producing harmonic distortion). We distinguish their implementation for synthesis $\{e^s\}$ and for analysis $\{e^a\}$.

$\{e^s\}$ is the implementation of the effects that have been used to create the observed master $\mathbf{y}$. In a real scenario, it is unknown.

$\{e^a\}$ is the implementation we use in our model to (1) predict the parameters $\hat{\mathbf{p}}$ of the effects or (2) replicate the resulting process of the mastering. To perform (1) (comparing the estimated $\hat{\mathbf{p}}$ to the ground-truth $\mathbf{p}$) the implementation of the effect in $\{e^s\}$ and $\{e^a\}$ should be similar. To perform (1) and (2), the implementation in $\{e^a\}$ should be differentiable (to estimate the parameters) or, if not, we will have to use neural networks to approximate the effects.

We now detail the implementation of the three AFXs for synthesis and analysis and list them in Table 1. The audio is normalized to 0 dBFS before each effect

Table 1. Considered audio effects and their implementations.

Effect	Implementation	Synthesis	Analysis
Equalizer	Parametric	√	√
	Graphic		√
Compressor	DSP	√	
	Simplified DSP		√
	NP		√
	Hybdrid NP		√
Clipper	Parametric	√	√
	Taylor		√
	Chebyshev		√

[2]https://peladeaucome.github.io/ICASSP-2024-BEAFX-using-DDSP/

2.1.1. *Equalizer*

For **synthesis**, we use a 5-band parametric equalizer: 1 low-shelf, 3 peak, 1 high-shelf.

For **analysis**, we use either this parametric equalizer or a 10-band graphic equalizer. Each band of the graphic equalizer has a bandwidth of 2 octaves [14].

Each parametric band has 3 parameters: center frequency, gain, and quality factor. Each band of the graphic equalizer has only 1 parameter: its gain.

Frequencies of parametric bands are logarithmic parameters, gains (in dB) and quality factors are linear. Differentiable filters are implemented in the frequency domain [4] as we find the time-aliasing error small enough for training a neural network.

2.1.2. *Dynamic range compressor (DRC)*

For synthesis, we use the DSP compressor proposed in [15]. It has 5 parameters: threshold, ratio, attack time, release time, and knee width.

For analysis, we either use a simplified DSP compressor or a Neural Proxy (NP). The simplified DSP compressor is the DSP compressor of [15] but with the attack and release time linked, as proposed by [6] to reduce the computation time. The NP compressor [16] is trained to approximate a DSP compressor[3]. It uses a Temporal Convolution Network (TCN) conditioned with FiLM [17] layers. In [16] the NP directly outputs $\hat{\mathbf{y}}$. In our case, we use the same TCN architecture but propose to replace its output activation with a sigmoid such that it provides the compressor gain factor $\mathbf{g}$ to be applied over time n: $\hat{y}[n] = g[n] \cdot x[n]$. [4]

Once trained, the NP compressor, being differentiable, can be inserted in the analysis chain $\{e^a\}$ to train the analysis network f^a and obtain its compressor parameters $\hat{\mathbf{p}}$. We can of course use $\hat{\mathbf{p}}$ to process $\mathbf{x}$ with the NP compressor but also use $\hat{\mathbf{p}}$ to process $\mathbf{x}$ with the DSP compressor. We name the latter "hybrid NP compressor". Since it is not differentiable, we only use it during validation and testing (not during training). It has already been used in [6].

The compressors' ratio and time parameters are logarithmic while their threshold and knee (both in dB) are linear.

[3]To train it, we first process a set of $\mathbf{x}$ with the DSP compressor using known parameters $\mathbf{p}$. The output provides ground-truths $\mathbf{y}$. We then train the parameters θ of the NP compressor conditioned with the same parameters $\mathbf{p}$ such that its output $\hat{\mathbf{y}} = f_\theta(\mathbf{x}; \mathbf{p}) \approx \mathbf{y}$.

[4]We found by experiment that this modification allows to largely reduce the number of parameters (number of TCN channels) with equivalent performances. With 8 channels, our causal model with a receptive field of 3 s duration has a test mean average error (MAE) of 0.0060 while the TCN from [16] has a test MAE of 0.050.

2.1.3. Clipper

We propose 3 implementations of the clipper: parametric (both for synthesis and analysis), Taylor, and Chebyshev (only for analysis).

The **parametric** clipper is defined by the function f defined in eq. (1) with hardness parameter h blending between tanh, cubic, and hard clipping:

$$f(x,h) = \begin{cases} (1-h)\tanh(x) + h f_{\text{cubic}}(x), & h \in [0;1], \\ (2-h) f_{\text{cubic}}(x) + (h-1) f_{\text{hard}}(x), & h \in (1;2]. \end{cases} \tag{1}$$

with :

$$f_{\text{hard}}(x) = \max(-1, \min(1, x))$$

$$f_{\text{cubic}}(x) = \begin{cases} x + 4x^3/27, & x \in [-\frac{2}{3}; \frac{2}{3}], \\ \text{sign}(x), & |x| > \frac{2}{3}. \end{cases} \tag{2}$$

The effect used for synthesis is constructed with the following parameters: gain g (in dB), offset o, and hardness h.

$$y[n] = \left(f(g \cdot x[n] + o, h) - f(o, h)\right)/g. \tag{3}$$

We also use two other memory-less models. Both have been proposed for memory-less distortion identification. The **Taylor** clipper is inspired by Taylor series expansions [1]:

$$y[n] = \sum_{h=0}^{H-1} g_h x[n]^h. \tag{4}$$

The **Chebyshev** clipper is inspired by Chebyshev's polynomials. It has been used for non-linear audio effect identification [18]:

$$y[n] = \sum_{h=0}^{H-1} g_h T_h(y[n]). \tag{5}$$

with $g_h \in [-1;1]$ being the effect's parameters and T_n:

$$\begin{aligned} &T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x), \qquad \forall n \geq 2, \\ &T_0(x) = 1, \qquad T_1(x) = x. \end{aligned} \tag{6}$$

In both cases, the parameters to be estimated are the $\{g_h\}$ and we set $H = 24$. All clipper parameters are linear.

2.1.4. Parameter ranges

All AFX parameters, $p_c/\hat{p}_c \in [p_{c,\min}; p_{c,\max}]$, are derived from "normalized" AFX parameters, $q_c/\hat{q}_c \in [0;1]$. Linear parameters (see above) are derived from $q_c/\hat{q}_c$ using an affine transformation:

$$p_c = (p_{c,\max} - p_{c,\min}) q_c + p_{c,\min}. \tag{7}$$

Logarithmic parameters (see above) are derived from $q_c/\hat{q}_c$ using an exponential transformation:

$$p_c = e^{\left(\log(p_{c,\max}) - \log(p_{c,\min})\right) q_c} p_{c,\min}. \tag{8}$$

2.2. Analysis network f^a

The analysis network is divided into two parts:

1. An **encoder**; which outputs a time invariant embedding. We describe and compare below 3 architectures for this encoder.
2. A MLP with 4 layers of size 2048, 1024, 512, and C where C is the total number of parameters $\hat{\mathbf{p}}$ of the effect chain $\{e^a\}$. Each hidden layer is followed by a Batchnorm-1D and a PReLU. The output layer, which estimates normalized parameters $\hat{\mathbf{q}}$, is followed by a sigmoid.

For the encoder, we compare three popular architectures commonly used in the Music Information Retrieval field:

MEE the Music Effects Encoder proposed by [19]. It consists of a cascade of residual 1D convolutional layers,

TE a Timbre Encoder inspired by [20]. It consists of a single 2D convolution layer with multiple sizes of filters. The conv-2D is applied on the CQT [21] of the signal as implemented in [22],

TFE a Time+Frequency Encoder inspired by [23]. It consists of two 2D convolutional nets, one focusing on highlighting temporal motifs, and the second on frequency motifs. The network's input is the CQT of the signal.

MEE has 88M parameters, TE 2.8M, and TFE 3.4M.

3. EVALUATION

3.1. Dataset

For evaluation, we use the *mix files* of the MUSDB18 [24] dataset. From those, we extract randomly clips of 10 s duration. Those are then converted to mono and peak-normalized to 0 dBFS. For each, we randomly pick the normalized parameters $\mathbf{q} \sim \mathcal{U}(0,1)$, convert them to $\mathbf{p}$ and apply them to the clip. We use the training, validation, and testing splits proposed by MUSDB18 [24].

3.2. Training

In the following, we compare 2 approaches for training f^a:

Audio reconstruction $\hat{\mathbf{y}} \approx \mathbf{y}$ (our proposal): we minimize the ℓ^1 norm between the log-magnitude Mel-spectrograms of $\mathbf{y}$ and $\hat{\mathbf{y}}$ as implemented in [25] [5] :

$$\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}} = \left\| \log\left\{\text{Mel}(|\hat{\mathbf{y}}|)\right\} - \log\left\{\text{Mel}(|\mathbf{y}|)\right\} \right\|_1. \tag{9}$$

Parameter reconstruction $\hat{\mathbf{q}} \approx \mathbf{q}$ (previously proposed approach): we minimize the ℓ^2 norm between $\hat{\mathbf{q}}$ and $\mathbf{q}$:

$$\text{MSE}_{\hat{\mathbf{q}},\mathbf{q}} = \frac{1}{C} \sum_{c=0}^{C-1} (\hat{q}_c - q_c)^2. \tag{10}$$

Training details: Each model f^a is trained using the ADAM algorithm with a learning rate of 10^{-4} and a batch size of 16 during a maximum of 400 epochs, where a single epoch is defined as 430 training examples (5 from each song of the training subset). The learning rate is scheduled to decrease by a factor of 10 when the best validation score has not been improved for 30 epochs. Training stops after 150 epochs without improvement. To ensure the reliability of computed scores the validation is run 5 times and the test 10 times.

3.3. Performance metrics.

For evaluation, we compute the following metrics:

- $\text{MSE}_{\hat{\mathbf{y}},\mathbf{y}}$: the MSE between $\hat{\mathbf{y}}$ and $\mathbf{y}$
- $\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$ as defined above
- $\text{MSE}_{\hat{\mathbf{q}},\mathbf{q}}$: the MSE between $\hat{\mathbf{q}}$ and $\mathbf{q}$

To make audio loss and metrics invariant to sound level, we normalize $\mathbf{y}$ and $\hat{\mathbf{y}}$ by their respective RMS values[6].

[5] We consider this metric as a surrogate for "audio quality" and are aware that it does not cover the whole extent of audio quality. Note that this loss has been previously used, for example in [26].

[6] Note that this normalization is applied as a static gain on the whole signal and therefore does not interfere with the dynamic gain of the DRC.

3.4. Results

3.4.1. Single effect estimation.

For each type of effect we first decide what is the best implementation (among those of 2.1) and architecture (among those of 2.2) to reconstruct $\mathbf{y}$. For each, we also indicate: - **Random $\hat{\mathbf{q}}$**: the results obtained with a random choice of $\hat{\mathbf{q}}$ (rather than the estimated one), - $\mathcal{L}(\mathbf{x}, \mathbf{y})$: the value of the loss when comparing the input $\mathbf{x}$ to the output $\mathbf{y}$.

For the **equalizer** (Table 2), Parametric and Graphic provide similar results. Since $\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$ indicates the difference between spectras, it is more suited to measure the performances of an EQ than $\text{MSE}_{\hat{\mathbf{y}},\mathbf{y}}$. We therefore focus on $\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$ and conclude that the best (0.32) configuration is to use the Parametric implementation for $\{e^a\}$ and TFE for f^a.

For the **compressor** (Table 3), the best (0.011, 0.076) configuration is to use the Hybrid NP for $\{e^a\}$ and use MEE for f^a. As a reminder, the Hybrid NP compressor uses the NP compressor to estimate $\hat{\mathbf{p}}$ but the DSP compressor (the same used for synthesis) to get $\hat{\mathbf{y}}$. This works better than using the NP compressor (0.014, 0.098) or the simplified DSP compressor (0.041, 0.16) This is because the latter links the attack and release time parameters which might be too restrictive. The fact that the Hybrid NP works better indicates that our proxy performs well enough for the task of estimating parameters usable with the DSP Compressor.

For the **clipping** (Table 4), the best configuration is to use the Parametric clipper (the same used for synthesis) for $\{e^a\}$ and use MEE for f^a.

It should be noted that in all three cases, the best results are obtained when $\{e^a\} = \{e^s\}$.

3.4.2. Training method comparison.

We now compare our proposed training method (based on audio reconstruction $\hat{\mathbf{y}} \approx \mathbf{y}$) to the previously proposed one (based on parameter reconstruction $\hat{\mathbf{q}} \approx \mathbf{q}$). Results are indicated in Table 5. For each single effect, we use the best configuration found above: f^a=TFE for equalizer, MEE for compression and clipping.

For **equalization**, in terms of audio quality ($\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$), the network trained to minimize $\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$ outperforms (0.32) the one that minimizes $\text{MSE}_{\hat{\mathbf{q}},\mathbf{q}}$ (0.40). But in terms of parameter estimation ($\text{MSE}_{\hat{\mathbf{q}},\mathbf{q}}$) minimizing directly $\text{MSE}_{\hat{\mathbf{q}},\mathbf{q}}$ leads to better results (0.072).

For **compression** and **clipping**, training by minimizing the parameter distance ($\text{MSE}_{\hat{\mathbf{q}},\mathbf{q}}$) leads to better results both in terms of audio quality ($\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$=0.069, 0.064) and parameter estimation ($\text{MSE}_{\hat{\mathbf{q}},\mathbf{q}}$=0.069, 0.028).

3.4.3. Effects chain estimation

We finally evaluate the estimation of the whole **chain** of effect (equalizer→compressor→clipper). In this case, we use f^a=TFE for all. As for audio quality, we see (row "Chain" in Table 5) that our approach (minimizing $\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$) leads to the best results: $\text{MSE}_{\hat{\mathbf{y}},\mathbf{y}}$=0.31 and $\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$=0.40. However, as can be predicted, minimizing directly $\text{MSE}_{\hat{\mathbf{q}},\mathbf{q}}$ leads to better estimation of the parameters: $\text{MSE}_{\hat{\mathbf{q}},\mathbf{q}}$=0.072. These contrasting outcomes underscore that achieving accurate parameter estimation ($\hat{\mathbf{p}} \approx \mathbf{p}$) does not guarantee high audio quality ($\hat{\mathbf{y}} \approx \mathbf{y}$). While our approach may not yield the best parameter estimation, it does yield the best audio transform estimation.

Table 2. Results by implementation of $\{e^a\}$ and encoder f^a for equalisation. The best results are indicated in bold.

Equalizer	Parametric		Graphic	
Metrics	$\text{MSE}_{\hat{\mathbf{y}},\mathbf{y}}$	$\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$	$\text{MSE}_{\hat{\mathbf{y}},\mathbf{y}}$	$\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$
MEE	0.35	0.41	0.38	0.41
TE	0.35	0.46	0.39	0.48
TFE	0.28	**0.32**	**0.25**	0.34
Random $\hat{\mathbf{q}}$	0.86	0.89	1.1	1.1
$\mathcal{L}(\mathbf{x}, \mathbf{y})$	0.48	0.64	0.48	0.63

Table 3. Same for dynamic range compression.

Comp.	NP		Hybrid NP		Simpl. DSP	
Metrics	$\text{MSE}_{\hat{\mathbf{y}},\mathbf{y}}$	$\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$	$\text{MSE}_{\hat{\mathbf{y}},\mathbf{y}}$	$\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$	$\text{MSE}_{\hat{\mathbf{y}},\mathbf{y}}$	$\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$
MEE	0.014	0.098	**0.011**	**0.076**	0.041	0.16
TE	0.020	0.12	0.019	0.11	0.042	0.17
TFE	0.015	0.11	0.012	0.086	0.042	0.17
Random $\hat{\mathbf{q}}$	0.038	0.16	0.036	0.15	0.13	0.66
$\mathcal{L}(\mathbf{x}, \mathbf{y})$	0.041	0.16	0.041	0.16	0.040	0.16

Table 4. Same for clipping

Clipper	Parametric		Taylor		Chebyshev	
Metrics	$\text{MSE}_{\hat{\mathbf{y}},\mathbf{y}}$	$\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$	$\text{MSE}_{\hat{\mathbf{y}},\mathbf{y}}$	$\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$	$\text{MSE}_{\hat{\mathbf{y}},\mathbf{y}}$	$\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$
MEE	**0.0045**	**0.072**	3.4	0.15	0.65	0.17
TE	0.011	0.090	2.9	0.21	3.2	0.15
TFE	0.0067	0.075	2.1	0.16	1.34	0.12
Random $\hat{\mathbf{q}}$	0.078	0.29	1.9	0.39	2.0	1.1
$\mathcal{L}(\mathbf{x}, \mathbf{y})$	0.078	0.34	0.079	0.34	0.079	0.34

Table 5. Comparison between training using $\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$ and $\text{MSE}_{\hat{\mathbf{q}},\mathbf{q}}$ for single effects and whole FX chain.

Loss	$\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$			$\text{MSE}_{\hat{\mathbf{q}},\mathbf{q}}$		
Metrics	$\text{MSE}_{\hat{\mathbf{y}},\mathbf{y}}$	$\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$	$\text{MSE}_{\hat{\mathbf{q}},\mathbf{q}}$	$\text{MSE}_{\hat{\mathbf{y}},\mathbf{y}}$	$\mathcal{L}^{\text{Mel}}_{\hat{\mathbf{y}},\mathbf{y}}$	$\text{MSE}_{\hat{\mathbf{q}},\mathbf{q}}$
Eq.	0.28	**0.32**	0.089	**0.27**	0.40	**0.072**
Comp.	0.011	0.076	0.11	**0.0081**	**0.069**	**0.069**
Clip.	0.0045	0.072	0.044	**0.0041**	**0.064**	**0.028**
Chain	**0.31**	**0.40**	0.10	0.33	0.49	**0.072**

4. CONCLUSION

In this work, we proposed an auto-encoder approach for Blind Estimation of Audio Effects. Given only processed (wet) audio signals, we train a neural network to estimate AFX parameters such that when used for effects applied to an unprocessed (dry) signal it approximates the processed (wet) signal. This allows training a network using real dry/wet data pairs without knowing the exact effect implementation. We show that our audio-based method better replicates the audio quality of the mastering process than the previous parameter-based method.

Further work will focus on performing subjective perceptual experiments, including other important mastering effects in the chain and testing their estimation on real mastered music productions.

5. REFERENCES

[1] Flávio R. Ávila and Luiz W. P. Biscainho, "ML estimation of memoryless nonlinear distortions in audio signals," in *Proc. of IEEE ICASSP (International Conference on Acoustics, Speech, and Signal Processing)*. May 2014, pp. 4493–4497, IEEE.

[2] Mirco Ravanelli and Yoshua Bengio, "Speaker recognition from raw waveform with SincNet," in *IEEE SLT (Spoken Language Technology Workshop)*, December 2018, pp. 1021–1028.

[3] Jesse Engel, Lamtharn (Hanoi) Hantrakul, Chenjie Gu, and Adam Roberts, "DDSP: Differentiable digital signal processing," in *Proc. of ICLR (International Conference on Learning Representations)*. April 2020, ICLR.

[4] Shahan Nercessian, "Neural parametric equalizer matching using differentiable biquads," in *Proc. of DAFx (International Conference on Digital Audio Effects)*, September 2020, pp. 265–272.

[5] Sungho Lee, Hyeong-Seok Choi, and Kyogu Lee, "Differentiable artificial reverberation," *Audio, Speech and Language Processing, IEEE Transactions on*, vol. 30, pp. 2541–2556, 2022.

[6] Christian J. Steinmetz, Nicholas J. Bryan, and Joshua D. Reiss, "Style transfer of audio effects with differentiable signal processing," *JAES (Journal of the Audio Engineering Society)*, vol. 70, no. 9, pp. 708–721, September 2022.

[7] Christian J. Steinmetz, Jordi Pons, Santiago Pascual, and Joan Serrà, "Automatic multitrack mixing with a differentiable mixing console of neural audio effects," in *Proc. of IEEE ICASSP (International Conference on Acoustics, Speech, and Signal Processing)*. June 2021, pp. 71–75, IEEE.

[8] Marco A. Martínez Ramírez, Oliver Wang, Paris Smaragdis, and Nicholas J. Bryan, "Differentiable signal processing with black-box audio effects," in *Proc. of IEEE ICASSP (International Conference on Acoustics, Speech, and Signal Processing)*. June 2021, pp. 66–70, IEEE.

[9] Joseph T. Colonel and Joshua D. Reiss, "Reverse engineering of a recording mix with differentiable digital signal processing," *JASA (Journal of the Acoustical Society of America)*, vol. 150, no. 1, pp. 608–619, July 2021.

[10] Reemt Hinrichs, Kevin Gerkens, and Jörn Ostermann, "Classification of Guitar Effects and Extraction of Their Parameter Settings from Instrument Mixes Using Convolutional Neural Networks," in *Artificial Intelligence in Music, Sound, Art and Design*, Tiago Martins, Nereida Rodríguez-Fernández, and Sérgio M. Rebelo, Eds., vol. 13221, pp. 101–116. Springer International Publishing, 2022.

[11] Sungho Lee, Jaehyun Park, Seungryeol Paik, and Kyogu Lee, "Blind estimation of audio processing graph," in *Proc. of IEEE ICASSP (International Conference on Acoustics, Speech, and Signal Processing)*. June 2023, pp. 1–5, IEEE.

[12] Joseph T. Colonel, Marco Comunità, and Joshua D. Reiss, "Reverse Engineering Memoryless Distortion Effects with Differentiable Waveshapers," in *Audio Engineering Society Convention 153*. 2022, p. 10, AES.

[13] Udo Zölzer, Ed., *DAFX: Digital Audio Effects*, Wiley, 2nd edition, 2011.

[14] Robert Bristow-Johnson, "RBJ audio-EQ-cookbook," 2005, https://www.musicdsp.org/en/latest/Filters/197-rbj-audio-eq-cookbook.html.

[15] Dimitrios Giannoulis, Michael Massberg, and Joshua D. Reiss, "Digital dynamic range compressor design— a tutorial and analysis," *JAES (Journal of the Audio Engineering Society)*, vol. 60, no. 6, 2012.

[16] Christian J. Steinmetz and Joshua D. Reiss, "Efficient neural networks for real-time modeling of analog dynamic range compression," *JAES (Journal of the Audio Engineering Society)*, May 2022.

[17] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville, "FiLM: Visual Reasoning with a General Conditioning Layer," in *Proc. of the AAAI Conference on Artificial Intelligence*. April 2018, vol. 32, AAAI.

[18] Antonin Novak, Laurent Simon, Pierrick Lotton, and Joël Gilbert, "Chebyshev Model and Synchronized Swept Sine Method in Nonlinear Audio Effect Modeling," in *Proc. of DAFx (International Conference on Digital Audio Effects)*, September 2010.

[19] Junghyun Koo, Seungryeol Paik, and Kyogu Lee, "End-to-end music remastering system using self-supervised and adversarial training," in *Proc. of IEEE ICASSP (International Conference on Acoustics, Speech, and Signal Processing)*. May 2022, pp. 4608–4612, IEEE.

[20] Jordi Pons, Olga Slizovskaia, Rong Gong, Emilia Gómez, and Xavier Serra, "Timbre analysis of music audio signals with convolutional neural networks," in *Proc. of EUSIPCO (European Signal Processing Conference)*, August 2017, pp. 2744–2748.

[21] Judith C. Brown, "Calculation of a constant *Q* spectral transform," *JASA (Journal of the Acoustical Society of America)*, vol. 89, no. 1, pp. 425–434, January 1991.

[22] Kin Wai Cheuk, Hans Anderson, Kat Agres, and Dorien Herremans, "nnAudio: An on-the-fly GPU audio to spectrogram conversion toolbox using 1D convolutional neural networks," *IEEE Access*, vol. 8, pp. 161981–162003, 2020.

[23] Jordi Pons, *Deep Neural Networks for Music and Audio Tagging*, Ph.D. thesis, Universitat Pompeu Fabra, 2019.

[24] Zafar Rafii, Antoine Liutkus, Fabian-Robert Stöter, Stylianos Ioannis Mimilakis, and Rachel Bittner, "MUSDB18 - a corpus for music separation," 10.5281/zenodo.1117372.

[25] Christian J. Steinmetz and Joshua D. Reiss, "Auraloss: Audio-focused loss functions in PyTorch," in *Digital Music Research Network One-Day Workshop 2020*, 2020.

[26] Jiaqi Su, Yunyun Wang, Adam Finkelstein, and Zeyu Jin, "Bandwidth extension is all you need," in *Proc. of IEEE ICASSP (International Conference on Acoustics, Speech, and Signal Processing)*, June 2021, pp. 696–700.