Audio De-clipping Via Deep Learning

Gabriel Lasry & Alon Feldinger

Contents

Abstract

This project aims at exploring techniques for audio de-clipping using deep learning algorithms. Throw out this work, we have reviewed some of the existing techniques, implemented an RNN architecture based on LSTM unit cells, and a state-of-the-art architecture named DEMUCS that was originally implemented for the task of music source separation, and was adapted for the task of de-clipping. We show that this architecture achieves results that are comparable to A-SPADE, which is considered one of the SOTA algorithms in this context.

Introduction and Background

Audio signals may contain speech, music, or individual instruments. Since audio signals are stored in digital media files, and recorded, played, and streamed via digital formats, the audio is limited in dynamic range. This limitation depends on the number of bits used for the storage of each audio sample, or on the word-length of the analog-to-digital and digital-to-analog conversions. This phenomenon is called *audio clipping*.

It's called clipping because that's what the clipping waveform ends up looking like a smooth, rounded sine wave has its peaks and troughs 'clipped off,' resulting in the flat plateau of a square wave (see Fig. 1). When this plateau is played back in your loudspeaker, it's an unnatural and jarring event that sounds horrible and can even lead to blown speakers.

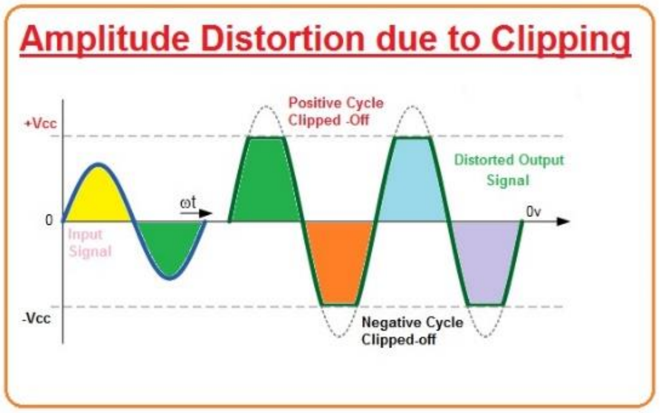

Amplitude Distortion due to Clipping
Positive Cycle Clipped -Off
+Vcc
Distorted Output Signal
ωt
0
0v
Input Signal
-Vcc
Negative Cycle Clipped-off

Figure 1: Clipped waveform

When speech or music is subject to clipping, information is lost, and the content will sound distorted to the listener, especially if the amount of clipping is more than a few decibels. The longer the clipping is, the more information is lost, and the more there is a need to understand the context of the signal and the music content being lost.

This makes *machine learning*, and specifically *deep-learning*, a natural candidate for interpolating long clipped areas, or areas with many audio samples getting clipped.

Over the years, many algorithms were proposed to deal with the de-clipping task. In Section 2, we will review the methods that we implemented in this project and will be used later, as benchmarks to compare our data-driven solution.

Review of De-clipping Algorithms

In this Section, we review some of the interpolation methods for de-clipping audio signals. First, we describe two classical methods, i.e., polynomial interpolation and a sparse-recovery-based scheme that uses prior knowledge of audio features to efficiently solve an optimization problem. Later, we review some data-driven algorithms that can be used for de-clipping purposes.

Polynomial interpolation

In *polynomial Interpolation*, the aim is to fit a L degree polynomial that best fits the data $(\{\mathbf{x}\}, \{y\})_1^N$, N being the number of data points. Mathematically,

$$L(\mathbf{x}) = \sum_{l=0}^{L} b_l \mathbf{x}^l,$$

where b_n are the polynomial coefficients. If $N < L$, we can fit the data points in infinite ways (*under-determined*), if *N*=*L* there is a single set of coefficients that fits the data exactly, and if $N > L$ the polynomial fits the data by minimizing the distance (in some sense) between the data samples and the polynomial at the same points (*over-determined*). In matrix form, the expression above can be written as

$$L(\mathbf{x}) = \mathbf{A}(\mathbf{x})\mathbf{b},$$

where the matrix $\mathbf{A}^{N\times L}$ is composed of the basis functions, in this case the polynomials, sampled at the data points, such that

$$\mathbf{A}(\mathbf{x}) = \begin{bmatrix} 1 & \mathbf{x}_1 & \dots & \mathbf{x}_1^L \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \mathbf{x}_N & \dots & \mathbf{x}_N^L \end{bmatrix}.$$

For the de-clipping task, the data set is composed of a given number of samples before and after the clipped area. Based on the data points and a polynomial degree, we can find the coefficient vector $\mathbf{b}$ with which we interpolate the missing samples.

When performing polynomial interpolation, arises the question of what degree of polynomial to fit. Fitting a low-order polynomial can result in a big difference between the *de-clipped* samples and the original ones. Fitting a high-order polynomial is also tricky as we obtain way too many options on how to reconstruct the signal. In machine learning terms, this trade-off is known as the *bias-variance* trade-off.

For this project, we decide on performing 3rd degree interpolations, or *cubic*, as we assume that the clipped areas are relatively narrow, and the waveform behaves smoothly.

In optimization techniques, a well-known strategy for dealing with ill-posed problems (under-determined systems) is to impose *sparsity* on the solution. In the case of a polynomial representation, this would mean imposing the solution to be a linear combination of only a few polynomial functions, even though the degree is high. Such an optimization scheme would be of the form

$$\min ||\mathbf{A}(\mathbf{x})\mathbf{b} - \mathbf{y}||_2 + \lambda ||\mathbf{b}||_0,$$

$|| \, ||_0$ being the zero-norm that is known to embrace sparsity.

One of the SOTA (state of the art) algorithms for de-clipping purposes is the *SPArse DEclipper,* or *SPADE* [1]*,* and is based on imposing sparsity in the reconstruction procedure. The algorithm is explained in the following subsection.

SPADE

The SPADE algorithm aims at reconstructing the distorted signal by means of mathematical optimization. First, let us define the problem of de-clipping. Let us denote the original samples vector as **x,** and the clipped version of it **y.** We can divide the samples of **y** to three categories. The set of samples that were not affected by the clipping are denominated Ω_r, and the clipped samples with positive and negative magnitudes are denominated Ω^+, Ω^-, respectively. The de-clipped signal $\hat{\mathbf{x}}$ must satisfy the following conditions:

$$\hat{\mathbf{x}}_r = \mathbf{y}_r$$

$$\hat{\mathbf{x}}^+ > \mathbf{y}^+$$

$$\hat{\mathbf{x}}^- < \mathbf{y}^-.$$

Let us denote these three constraints on $\hat{\mathbf{x}}$ as $\Gamma(y)$. It is well-known that the energy of audio signals is often concentrated in a small number of frequency components at every given time-frame [2]. Therefore, if we take a small segment of the signal, or window, and represent it in the frequency domain, the spectral coefficients will tend to be sparse. Mathematically, we can denote the Fourier analysis operator as $\mathbf{A}$**,** and the spectral representation of $\mathbf{x}$ as $\mathbf{Ax} = \mathbf{z}$, where $\mathbf{z}$ are the frequency coefficients. Assuming that the energy is concentrated in a limited number of spectral bins, yields the sparsity of $\mathbf{z}$.

We are now ready to formulate the optimization problem to be:

$$\min_{\mathbf{x},\mathbf{z}} \|\boldsymbol{z}\|_0 + \|\mathbf{Ax} - \mathbf{z}\|_2 + \mathbf{1}_{\Gamma(\boldsymbol{y})}$$

where $\mathbf{1}_{\Gamma(\boldsymbol{y})}$ is the indicator function of the constraint set $\Gamma(\boldsymbol{y})$**.**

The optimization problem stated above is a *non-convex* one, as the zero-norm is not a convex function. The authors propose to solve this problem iteratively via the Alternation Direction Method of Multipliers (ADMM), reducing every step to a convex optimization problem.

DNN-based De-clipping

Several works propose recovering the original audio from a distorted signal [3] [4] [5]. These techniques can be divided into two types, depending on the representation of the input signal: (i) Frequency domain via the STFT [3] [4]. This way, the audio signal is represented as a spectrogram and processed as an image. U-Net, which was originally developed for image segmentation [6] is an example of a convolutional neural network that was used for audio de-clipping. (ii) Time-domain [5]. The main advantage of treating the waveform directly is the consideration of the phase information, that for most frequency-based techniques is ignored (only the magnitude spectrogram is treated). Wave-U-Net is an adaptation of U-Net that works directly on the waveform and was originally implemented for audio separation tasks [7] and can be easily adapted to perform de-clipping tasks.

Wave-U-Net follows an encoder-decoder architecture. In every encoder layer, a feature map is extracted via a convolutional layer and then down-sampled. In this way, the NN extracts features in different time scales. In every decoder layer, the signal is up-sampled, combined with the corresponding feature map that was extracted in the encoder, and then a convolutional layer is applied. While the main motivation to add the feature maps in the decoder layers comes from empirical performances, an advantage of the skip connections is to give direct access to the original signal, and it allows for direct transfer of the phase of the input signal to the output. A diagram of the Wave-U-Net architecture is shown in Fig. 2.

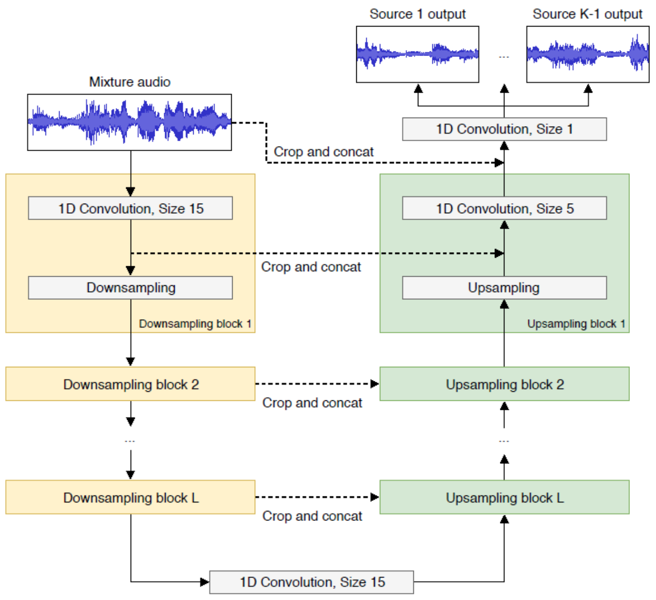
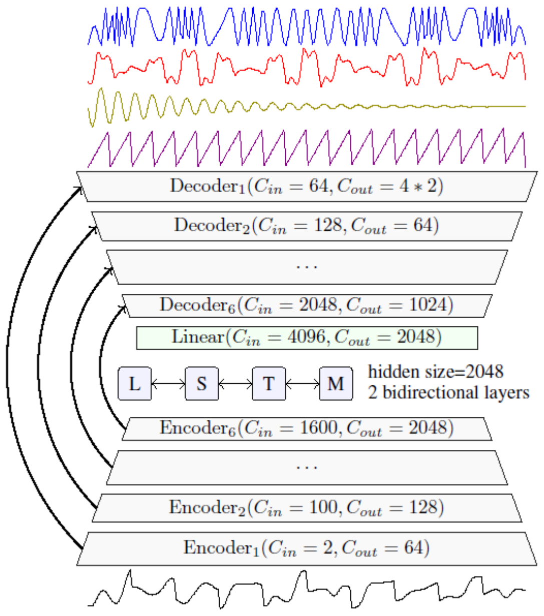


Figure 2: Wave-U-Net architecture

In 2022 an improvement to Wave-U-Net, named DEMUCS, was proposed, surpassing all other time-, and frequency-domain techniques for audio separation in terms of sound-to-distortion ratio (SDR). In this project, we adapted the DEMUCS architecture for the de-clipping task. In the following paragraph, we describe the architecture.

The encoder is composed of 6 layers. Every block is composed of a convolutional layer with C_{in} input channels and $2C_{\text{in}}$ output channels (except from the first layer that outputs 64 channels) followed by a ReLU activation function. The down-sampling of the signal is achieved using a stride of 4 in the convolutional layer. Then, a 1x1 convolution is applied followed by a Gated Linear Unit (GLU) that acts as a self-attention module, weighing the channels based on the input. The output of every block is sent to the next encoding layer and saved as skip connections at the decoder. Between the encoder and the decoder, they use a two-layer bidirectional LSTM block that keeps the dimensionality of the input.

The decoder is mostly the inverse of the encoder. In every decoder layer, there is a convolutional layer followed by a GLU activation function. The output is up-sampled using a transposed convolutional layer that divides the number of channels by a factor of two (except from the last layer that reduces the number of channels to one) and then a ReLU activation function is applied. A diagram of the entire architecture is depicted in Fig. 3.



$\text{Decoder}_1(C_{in} = 64, C_{out} = 4 * 2)$
$\text{Decoder}_2(C_{in} = 128, C_{out} = 64)$
...
$\text{Decoder}_6(C_{in} = 2048, C_{out} = 1024)$
$\text{Linear}(C_{in} = 4096, C_{out} = 2048)$
L
S
T
M
hidden size=2048
2 bidirectional layers
$\text{Encoder}_6(C_{in} = 1600, C_{out} = 2048)$
...
$\text{Encoder}_2(C_{in} = 100, C_{out} = 128)$
$\text{Encoder}_1(C_{in} = 2, C_{out} = 64)$

Figure 3: DEMUCS architecture

Experimental Setup

We use the LibriSpeech [8] data set for training and testing. We train the models with 2400 files of a few seconds duration, and we test the model with 270 files of the same lengths. All tracks are sampled at a rate of 16 kHz with a bit depth of 16 bits per sample.

We simulate clipped tracks $y(t)$ by hard-thresholding all clean files $x(t)$ to 0.2 of the maximum value, such that,

$$y(t) = \begin{cases} x(t), & |x(t)| < 0.2 \cdot \max|x(t)| \\ 0.2 \cdot \max|x(t)| \cdot \operatorname{sign}\big(x(t)\big), & |x(t)| \geq 0.2 \cdot \max|x(t)| \end{cases}.$$

See Fig. 4.

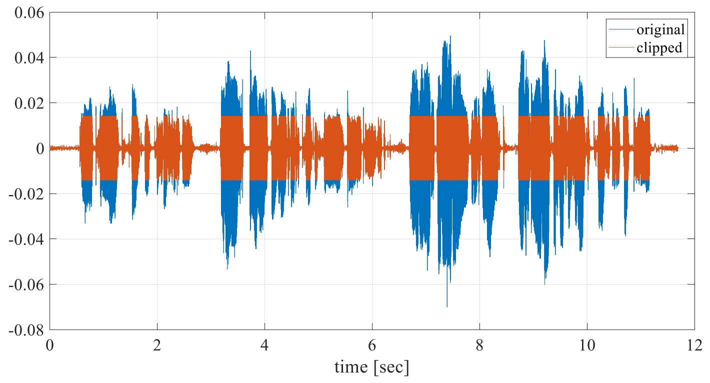


Figure 4: Original (blue line) and clipped (red line) waveforms.

The clipped signal is then split into windows of 1 [sec] with 50% overlap. Then, we send to the inference scheme all windows that have been clipped, and last, we reconstruct the signal by averaging the reconstructed values of the overlapping windows.

We train the model over 40 epochs with a batch size of 5 windows, use the MSE as the loss function, and use the Adam optimizer with a learning rate of 3e-4. The learning curve is depicted in Fig. 5.

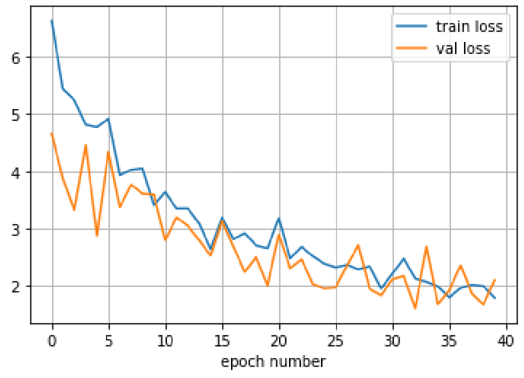


Figure 5: DEMUCS learning curve.

The evaluation of the models is carried out via two methods. The *first evaluation* metric is the mean squared error (MSE) between the original and de-clipped signals. The second evaluation metric is the *Perceptual Evaluation of Audio Quality* (PEAQ) [9] that measures the amount of degradation between two audio signals using perceptual properties of the human ear.

Results

Table 1 shows the results of the DEMUCS algorithm compared to three other techniques: cubic interpolation, A-SPADE, and a trained RNN built of four BLSTM layers with hidden states of dimensionality of 64.

	MSE	**PEAQ**
Clipped	6.9e-4	2.62
Cubic interpolation	6e-4	2.43
A-SPADE	1.5e-4	**3.67**
RNN	2.4e-4	3.12
DEMUCS	**8.7e-5**	**3.67**

Table 1: De-clipping results

In terms of MSE, the DEMUCS architecture surpasses all other models, and in terms of the PEAQ, both A-SPADE and DEMUCS achieve similar results. In terms of inference time, DEMUCS brings faster inference than A-SPADE, the last being an iterative method.

The evaluation of the A-SPADE algorithm was achieved using the code provided by the authors and can be found in [10].

Conclusions and Future Work

In this project, we have not explored architectures that treat the signals in the frequency domain. A direct follow-up of this project would be to implement the newer versions of DEMUCS that treat the signals both in the time and frequency domains and make an inference based on both paths [11]. Another architecture that is worth exploring is APPLADE, a model-based deep learning scheme for de-clipping audio signals, that is based on A-SPADE, and integrates a learned operator in the thresholding step of every iteration. In the proposed paper they report slightly better and faster inference than A-SPADE [12].

Bibliography

[1] P. R. O. M. Z. P. Pavel Záviška, "A PROPER VERSION OF SYNTHESIS-BASED SPARSE AUDIO DECLIPPER," 2020.

[2] N. B. R. G. Srđan Kitić, "Sparsity and cosparsity for audio declipping: a flexible non-convex approach," 2015.

[3] A. J. M. M. G. S. G. F. Hamidreza Baradaran Kashani, "Image to Image Translation based on Convolutional Neural Network Approach for Speech Declipping," 2019.

[4] W. M. a. E. A. P. Habets, "Declipping Speech Using Deep Filtering".

[5] K. F. a. A. Scherlis, "WaveMedic: Convolutional Neural Networks for Speech Audio Enhancement".

[6] P. F. a. T. B. O. Ronneberger, "U-Net: Convolutional Networks for Biomedical Image Segmentation".

[7] S. E. S. D. Daniel Stoller, "Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation".

[8] G. C. D. P. S. K. Vassil Panayotov, "LIBRISPEECH: AN ASR CORPUS BASED ON PUBLIC DOMAIN AUDIO BOOKS".

[9] T. Thiede, W. C. Treurniet, R. Bitto, C. Schmidmer, T. Sporer, J. G. Beerends and C. Colomes, "PEAQ - The ITU Standard for Objective Measurement of Perceived Audio Quality".

[10] N. B. R. G. Srđan Kitić, "GitHub," [Online]. Available: https://github.com/andryr/spade-declipping.

[11] A. Défossez, "Hybrid Spectrogram and Waveform Source Separation," *Electrical Engineering and Systems Science, Audio and Speech Processing,* 2022.

[12] K. Y. M. Y. Y. O. Tomoro Tanaka, "APPLADE: Adjustable Plug-and-Play Audio Declipper Combining DNN with Sparse Optimization," *Electrical Engineering and Systems Science, Audio and Speech Processing,* 2022.