Reverb Deconvolution with Machine Learning

Zachariah Burrows

May 2025

Abstract

In digital audio recording, reverb is an essential effect. It lends authenticity and body to an audio signal. It is useful to extract the "dry", or reverb-less, sound to have a blank slate from which to add further reverb effects. This paper will cover an innovative use of a recurrent neural network (RNN) to analyze reverb'd signals and extract the dry signal. Importantly, this is attempted without the impulse response, which is a difficult problem as deconvolution typically requires both the wet signal and the impulse response. The impulse response is not always available, however, and a deconvolution tool that operates without it would be useful for various reasons, such as extracting the dry signal from an informal audio recording. In addition, this dry signal could then be used to calculate the impulse response with the original wet signal. As an extension, the impulse response, trained on room sizes, could be used to predict the size of the room that the original audio signal was recorded in. This was a stretch goal, and ultimately at the end of this project it was not reached. An RNN was successfully trained on a corpus of anechoic and reverb'd signals, and used to predict on novel inputs. The resulting output had frequent pops and issues with gain levels, but the core signal had been altered with less high frequencies. Future next steps include smoothing the output with overlapping windows and implementing gain normalization, as well as tuning the hyperparameters of the model to maximize its performance.

1 Introduction

In digital signal processing, reverb is an extremely common effect. It can either be recorded naturally into the signal (from the acoustics of the room that the signal was recorded in), or it can be mathematically added to the signal using the process of convolution. Convolution takes an impulse response (or a room's response to a single short "impulse"), and adds it to every signal in the time domain. By modifying the impulse response, reverb can be added to an audio signal to simulate various recording spaces, from a cathedral to a small room.

While adding reverb is often desirable, removing it is also a useful tool. If one wishes to alter the reverb of a signal that is already wet, it must first be

made dry by deconvolution. This might occur if recordings are made in a space and subsequently analyzed.

Traditionally, reverb plugins contain a corpus of impulse responses, which may be convolved with an input to add the desired effects. Similarly, deconvolver plugins might take as an input a wet signal and an impulse response to mathematically perform deconvolution. [8] Typically, these deconvolution plugins use a frequency domain-based method called FFT (Fast Fourier Transform) deconvolution. [8]

2 Related work

Much of the scholarly work that relates to the aim of this project is in the field of image analysis. In the image world, convolution can be considered as the blurring or artifacting of an image. In this instance, the blur pattern is similar to the impulse response in an audio context. [2] Convolutional neural networks (CNNs) use this pattern to analyze images and extract features, searching for "smearing" of the image. They slide filters across the image in a window, comparing image data to find patterns. [2] From a digital signal processing perspective, the "dry" signal would be the image without smearing or artifacts, and the impulse response would be the extracted patterns.

In the field of fluorescence microscopy, where researchers shine light on samples to gain insight of their properties, deconvolution and machine learning can be very useful. In 2022 researchers used deep learning, a derivative of machine learning, to analyze microscopy data with deconvolution. [3] They were able to make predictions four to six times faster than before with a new convolutional network architecture—or method of passing data through the network. [3]

There are some examples of work being done in the machine learning field with audio deconvolution, although this seems to be less common. Students at Stanford published a report describing the use of a neural network to isolate audio sources; [1] this is essentially the same problem, as the goal is to separate several conflicting components of an audio signal. They were able to achieve their goal of successful source separation using a CNN-derivative model. However, their model analyzed signals in the frequency domain, using a short-term Fourier transform (STFT). This project will aim to analyze signals in the time domain.

3 Design

Originally, this project was intended to be a plugin implemented in JUCE. However, it quickly became clear that the amount of data processing involved would not be conducive to an inline plugin. The visual component of the project was changed to a user interface, where users would be able to upload audio files and then analyze them.

3.1 Model architecture

The specific variant of neural network used in this project is called a Long Short-Term Memory (LSTM) Recurrent Neural Network. RNNs analyze sequences of data, rather than a fixed number of inputs as in a traditional neural network. The LSTM variant saves a hidden state during training, and considers changes in the hidden state as part of its predictions, effectively serving as a form of memory for the network. [5] This is especially useful for this project, as the "smearing" of audio is a change over a sequence of time, and examining how the audio changes on a sequential basis could help to reveal the impulse response.

Every machine learning model has a set of hyperparameters, or parameters specific to the model that are tuned to the input and output needs. For this model, there were 256 hidden neurons, 3 hidden layers, and a learning rate of 0.0001. The hidden neurons are weights applied to the input, and hidden implies that they are not given any information. Therefore in this model there are three total layers of 256 neurons, each of which receives information, applies a weight, and passes it to the next layer. The learning rate describes how fast the network "learns", or how extreme it compensates for the loss calculated against the correct answer.

3.2 Training

Because this project deals with raw audio, and thus a lot of data, it was planned to train the model separately, and then to ultimately deconvolve audio it would be sent through the trained network, adjusted according to the weights of the network, and finally predicted on. The model is trained on a set of anechoic recordings made in an anechoic chamber (a room ideally without any reverb). These recordings give the neural network a true example of a "dry" signal. Each recording is made at 44.1 kHz, and that is the final sample rate of the predicted signal.

To pass training data to the model, reverb is artificially added using simple convolution-based methods. Each pair of dry-wet signals are added as key-value pairs to a dictionary, to enable easy access of both pieces of data. For the neural network, the wet signal is the input, and the dry signal is the label (or the "correct" value from which to judge the actual output). Importantly, every signal is converted to mono to avoid broadcasting issues from inputs and outputs of different dimensions. While valuable data is likely lost due to this, the process is already extremely process-heavy and adding another channel could increase computing time significantly.

In training, the model will split each waveform into a segment of predetermined length (here, it was 44100 samples, or at 44.1 kHz a second long). It will select a random start point, and extract the segment from both the wet and dry waveforms. In a loop, for the number of epochs (each time that a segment is passed through the network), the model will select a random wet and dry segment, predict on the wet segment, compare to the dry, and backpropagate the associated loss—meaning the network adjusts all of its parameters to compen-

sate for how close or far it predicted from the correct answer. For this report, the model was trained with 100 epochs. Finally, the model and all of its weights are saved to a local file, from which it can be predicted with.

Because of the computational intensity of this model, and the fact that it deals with raw audio, the model was trained on Nvidia GPUs with CUDA cores. This equipment enables parallel processing on scales impossible with CPUs, and greatly speeds up training times.

3.3 Testing

The model is tested with a slightly different process than training. Because the testing cannot use CUDA cores as most users will not have access to Nvidida hardware, computational efficiency is prioritized. Instead of segment lengths of 44100, testing uses a segment length of 5000. This maximizes the possible amount of parallel processing. Each segment is assigned an available worker, or CPU core, to compute. They are then sent through the network and stitched together in the end to result in an output.

3.4 GUI

The GUI was originally intended to be fully-fledged and sleek. However, given time constraints sleekness was substituted for functionality. The final GUI is extremely basic, providing support for file uploads and local saving. It also features a playback option so that users may listen to their output file before saving. Figure 1 depicts the design. To run the GUI, the `gui.py` file must be ran in the repository.

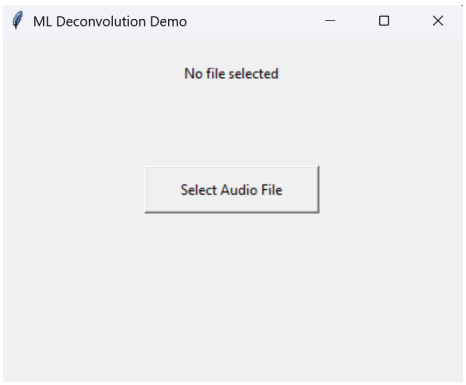

Figure 1: The GUI when the program is first run

3.5 Implementation

The project was implemented using models and model classes from PyTorch [4], an online repository of machine learning resources and models. It additionally

uses `torchaudio` to process wav files and `tkinter` [7] for the basic GUI. All programming was done in Python. [7]

4 Application

This project has been used to predict on several example audio inputs, including the famous acapella recording of Tom's Diner by Suzanne Vega. Figure 2 depicts part of the output waveform overlayed over the input waveform. Visually, the output has slightly fewer peaks and slightly less sharp features. Sonically, the output file sounds less full and rich, which is an indication of the right direction. However, when predicting on files with large amounts of reverb, the model is unable to remove most of it. It is possible that the training set is unrepresentative of natural reverb.

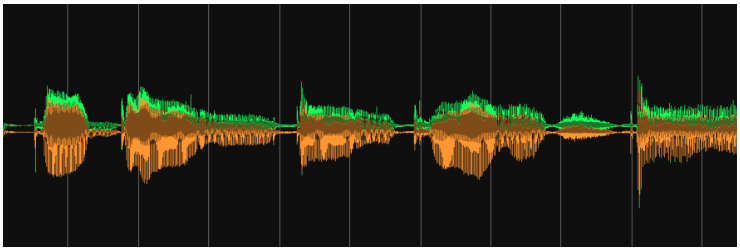

Figure 2: The output signal (orange) overlayed with the input signal (green). Gain is normalized to directly compare the signals.

5 Future work

This project experienced several issues that could be fixed in the future. The primary issue is a constant clicking sound, resulting from the stitching together of the audio signal at the end of the testing sequence. This problem is fixable with an overlapping window approach, where each chunk fades into the next. However, time constraints and recurrent issues hindered this fix.

In addition, as mentioned in the previous section the training set consists of uniformly wet recordings; the same type of reverb was applied to every waveform. This is of course not representative of real world data, and being able to physically record the same sound in both an anechoic chamber and a room with reflections would be ideal.

Further, gain normalization could be implemented in the code. This would help the output of the neural network to focus only on altering necessary qualities of the signal.

Finally, the hyperparameters could be further tuned with a method like grid-search. Due to time constraints, not all hyperparameter options were explored. However, running through all parameters with a brute force method could help to find the ideal combination to yield the best possible output.

6 Conclusion

To summarize, this project set out to create an application that used machine learning to deconvolute a signal with reverb without using an impulse response. In the end, an output was achieved that started to sound like a dry signal. Much more work is needed to be done to fully accomplish the goal that was set. The foundation is laid for this work, and most of the next steps involve modifying work that has already been done.

Ultimately, the prospect of deconvolution with machine learning is not novel; it has been a common practice in the field of image processing for a long time. [6] In the audio field, it is something seen less often. However, there it has the possibility to greatly decrease compute times for deconvolution, effectively allowing anyone to extract a dry signal from an audio recording made anywhere.

References

[1] Ahmed Hamdy, Pratap Kiran Vedula, and Muni Venkata Jasantha Konduru. *Audio Separation and Isolation: A Deep Neural Network Approach.* 2019.

[2] IBM. *What are convolutional neural networks?* URL: `https://www.ibm.com/think/topics/convolutional-neural-networks`. (accessed: 05.18.2025).

[3] Yue Li et al. "Incorporating the image formation process into deep learning improves network performance in deconvolution applications". In: *bioRxiv* (2022). DOI: `10.1101/2022.03.05.483139`. eprint: `https://www.biorxiv.org/content/early/2022/03/06/2022.03.05.483139.full.pdf`. URL: `https://www.biorxiv.org/content/early/2022/03/06/2022.03.05.483139`.

[4] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*. 2019, pp. 8024–8035.

[5] Ralf C. Staudemeyer and Eric Rothstein Morris. *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks.* 2019. arXiv: `1909.09586 [cs.NE]`. URL: `https://arxiv.org/abs/1909.09586`.

[6] Bahareh Tolooshams et al. "Interpretable deep learning for deconvolutional analysis of neural signals". In: *bioRxiv* (2024). DOI: `10.1101/2024.01.05.574379`. eprint: `https://www.biorxiv.org/content/early/2024/01/06/2024.01.05.574379.full.pdf`. URL: `https://www.biorxiv.org/content/early/2024/01/06/2024.01.05.574379`.

[7] Guido Van Rossum. *The Python Library Reference, release 3.8.2.* Python Software Foundation, 2020.

[8] Voxengo. *Voxengo Deconvolver*. URL: `https://www.voxengo.com/product/deconvolver/`. (accessed: 05.18.2025).