Joel Shapiro
Dr. Shibberu
Oct. 28, 2016
MA 490

# Deep Signal Processing

## overview

The objective of this project is to evaluate the effectiveness of doing audio effect emulation using deep learning. For audio, there are two main classifications for tools: generators and effects. A generator is something which takes non-audio input, either physical or midi, and creates audio out of it. This would include instruments, synthesizers, drums - basically anything that really stands out as being traditionally musical. The other category, effects, are elements which take audio as input and transform it into some other audio to output. This can range from a simple filter to more complex effects such as distortion or reverb. Even the echo of a room is an effect. The idea behind this project is to see if we can't train a network to emulate one of these effects using deep learning. Audio is an interesting medium to work in for machine learning, as like image data the output can be judged both quantitatively and qualitatively.

Audio itself is a complex structure; the additive property of waves can cause some unforeseen outcomes. Effects can be composed of a complex network of variables. On top of that, digital audio data is inherently convoluted: it is stored as a time series of points which are sampled from the audio signal itself. These points are fast fourier transformed back into the signal whenever the audio is ready to be output. Because of this, a lot of the information which is altered by effects is hidden behind this signal processing problem. In the past, doing signal processing in machine learning involved doing some manual decomposition of the input in order to abstract away the signal processing. Often audio would be rendered into images of the spectrogram, which show the frequency distribution of the audio. While this would probably work as means of identifying different effects put on the audio, it doesn't allow us a way to generate the audio back as output. For that, we need to do actual signal processing in order to detect the features that matter.

The current progress on this project is available at github.com/jshap70/DeepSP

## sample types

The audio itself is a wide spectrum of sources which have a wide range of complexity. On the easier side, we have simple sine and triangle waves that move through a frequency range. More difficult samples include piano recordings and voice data. There is a selection of simple and complex filters being put on the audio, as it will be interesting to see the impact filter complexity has on training difficulty.

## batching and data sampling

Looking at the data itself, the wav files are stereo 16bit PCM (integer)) files. To begin with, I converted the data to a 32bit float wav file and normalized the audio to fit within that standard. I converted the data to numpy arrays, keeping the audio in stereo. I could have split apart each file into mono tracks, effectively doubling the amount of data I would be training on, but I chose not to because some effects might have stereo-dependent elements to them.

Before we begin batching, validation and testing data is extracted prior to the batching to ensure that it is not trained on at all. Because it is time series data the batching is a bit trickier. Although the data needs to be kept in contiguous chunks, we can still extract smaller sections of it to train on so we don't have to train on the whole dataset at one time. I implemented a batching system that does a scrolling window selection of the audio for discrete time periods. If the offset that each window is from each other is smaller than the length of each sample, then there will be some overlap of the batches. Initially, I have set up up the batches in a way that means every datapoint will be used 10 times. After generating the batches I shuffle them out of order so that we can get a more even distribution.

## networks & how that makes sense for our problem

Starting off I used a standard, fully connected regression neural network with a depth of _(still working)_ layers. The goal of this network was to try to overfit the training data to show that it can at least be brute forced.

Because this problem is attempting to directly emulate a filtering effect, it seems somewhat intuitive that this problem would be decently well represented by a convolutional network. If we could get the neural net to understand the audio input, we could train a convolutional layer to understand the change in the data from the input to the output. This might also allow us to combine convolutional layers which are trained from different filters, but more on that later.

// RNN's and how they might help with time series data