Component Compressed Music

Laryn Qi
UC Berkeley CNMAT
larynqi@berkeley.edu
Code

I. Introduction

The dawn of the digital age has brought about numerous new challenges for all who enjoy music. As dependence on live music has diminished and in its stead, digitally distributed music has risen, people have quickly noticed the need for audio compression to make time and space efficient transfer and storage. In fact, audio files in their raw, uncompressed form (wav) average around 10mb per minute. However, since the early-twentieth century, advances in computation have also opened the door for new algorithms and techniques to tackle these problems.

The field of audio compression can be split into two subdomains: lossless and lossy compression methods. Lossless compression is designed to minimize audio file sizes while maintaining the ability to reconstruct one-hundred percent of the original signal; this type of compression is suited for those who value maintaining the exact original data/signal when compressing. On the other hand, lossy compression exploits certain attributes of the sound alongside human factors to produce an even greater compression (in terms of file size reduction) that, when played back, is perceptually indistinguishable from the original.

II. State of the Art

For many, the problem of audio compression is considered solved. With the onset of audio coding schemes such as MPEG Layer-3 ("mp3") in the late-twentieth century, researchers in Germany were able to bring hours of music into the pockets of music fans. Unlike lossless compression codecs such as Free Lossless Audio Codec ("FLAC"), which uses run-length encoding and linear predictions to ensure complete reconstruction, mp3 takes advantage of human aural limitations to filter out sounds that cannot be picked up by the human ear anyway. mp3 boasts file size reductions by a factor over ten wit the algorithm that follows:

1) Filter bank: Perform multiple FFT's on the input signal to get its frequency response split between subbands.
1) Psychoacoustic model: Perform a sequence of FFT's on the input signal to map amplitude against frequency and ensure that noise produced by approximations can be masked by strong tonal signals.
2) Bit allocation: Allocate more bits to frequency subbands with higher mask-to-noise ratios. Use Huffman codes to encode quantized values.
3) Bit stream formatting: Process results of prior steps and return an encoded bitstream.

III. Methodology

In this paper, I propose a different approach to audio compression: Component Compressed Music ("ccm"). Like mp3, ccm is also a lossy compression algorithm. Instead of using somewhat subjective measures such as human noise-detection thresholds, ccm uses Principal Component Analysis ("PCA"), a purely mathematical technique traditionally used within the unsupervised learning subsect of machine learning, in an effort to achieve either faster encoding, improved space efficiency, or both. In machine learning, PCA is used in dimensionality reduction, or, in other words, to reduce the complexity of data while keeping its core features intact. Specifically, PCA finds orthogonal components and ranks them in order of percentage of variance of the input explained. In ccm, PCA is used to extract what linear algebra deems are the key features of the signal. This result is used to discover which dimensions of the signal are not contributing much to the overall sound and can therefore be thrown out without detriment. In essence, that is all ccm does: pre-process the signal, run PCA on the reshaped data, remove principal components past some threshold, and transform the signal back to its original shape. The algorithm also relies on a few hyperparameters:

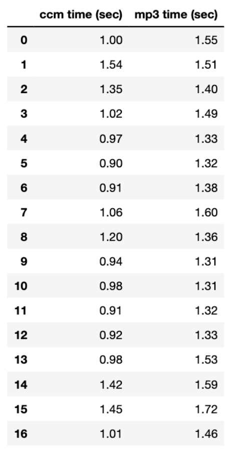
1) 'n_dim': The desired number of dimensions the input signal should be modeled with.
2) 'n_com': The desired number of principal components to include in the compressed, dimensionality-reduced representation of the signal

An in-depth, step-by-step walkthrough and explanation of the algorithm follows:

1) Pre-processing: Pad the input signal until the number of samples is a multiple of 'n_dim'. Reshape the signal to be of 'n_dim' dimensions/columns.
2) PCA: Use Singular Value Decomposition (SVD) to compute U, Σ, and V.T matrices. Σ importantly provides us with a measure of how much variance each principal component explains (in descending order). We use this information to select the most important principal components.
3) Compression: Slice off unwanted columns of the signal, keeping only 'n_com' principal components. Reconstruct the transformed signal using PCA reconstruction ($U\Sigma V.T$).
4) Transform: Reshape the reconstructed signal back to its original shape (so it can be encoded as an audio file).

IV. Experiments & Evaluation

Experiments are performed primarily on two audio files: ‘simple.wav‘ and ‘complex.wav‘. The former is a converted MIDI file and features only piano as such; the latter is a more musically complicated song created using a variety of Magenta’s, a team within Google Brain, computer music tools (sourced with ‘simple.wav‘). Interested in the efficiency of the algorithm, I measure empirical time and space data between ccm and mp3 (on both ‘simple.wav‘ and ‘complex.wav‘). With ccm, I use ‘n_dim‘=1024 and ‘n_com‘=32. With mp3, I use a bitrate of 192k.



	ccm time (sec)	mp3 time (sec)
0	1.00	1.55
1	1.54	1.51
2	1.35	1.40
3	1.02	1.49
4	0.97	1.33
5	0.90	1.32
6	0.91	1.38
7	1.06	1.60
8	1.20	1.36
9	0.94	1.31
10	0.98	1.31
11	0.91	1.32
12	0.92	1.33
13	0.98	1.53
14	1.42	1.59
15	1.45	1.72
16	1.01	1.46

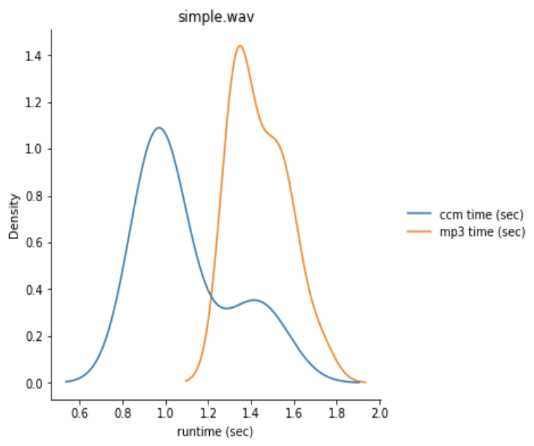
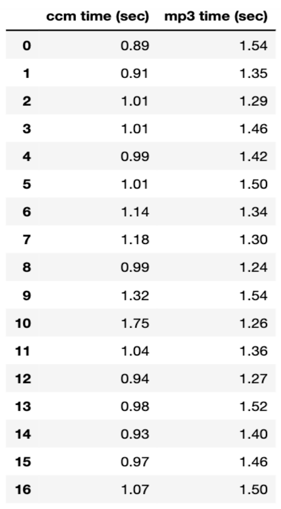


Fig. 1. ‘simple.wav‘ time comparison

```
ccm compressed simple.wav to 50.01% of its original size
mp3 compressed simple.wav to 13.62% of its original size
```

Fig. 2. ‘simple.wav‘ space comparison

In both the simple and complex case, ccm runs around 50% faster than ffmpeg’s mp3 compression on average (Fig 1., Fig 3.). However, on the space side of things, ccm consistently compresses the file to half its original size while mp3 compresses the file to ranging from around 1/10 to around 1/3 of its original size (Fig 2., Fig 4.). Perhaps most importantly, the mp3 compression sounds much, much more like the original audio than ccm result (see Github repo).



	ccm time (sec)	mp3 time (sec)
0	0.89	1.54
1	0.91	1.35
2	1.01	1.29
3	1.01	1.46
4	0.99	1.42
5	1.01	1.50
6	1.14	1.34
7	1.18	1.30
8	0.99	1.24
9	1.32	1.54
10	1.75	1.26
11	1.04	1.36
12	0.94	1.27
13	0.98	1.52
14	0.93	1.40
15	0.97	1.46
16	1.07	1.50

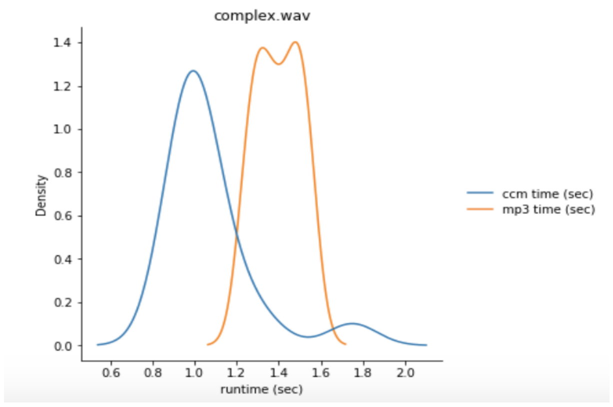


Fig. 3. ‘complex.wav‘ time comparison

```
ccm compressed complex.wav to 50.0% of its original size
mp3 compressed complex.wav to 31.27% of its original size
```

Fig. 4. ‘complex.wav‘ space comparison

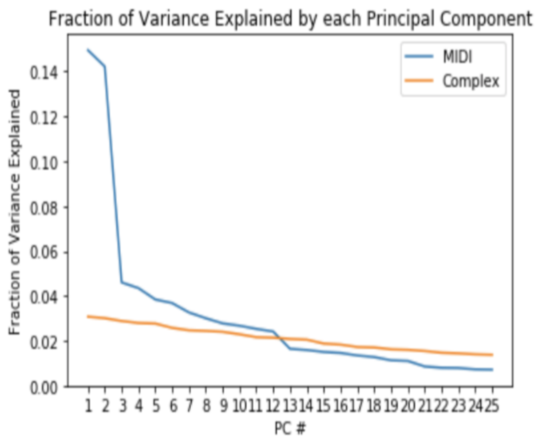


Fig. 5. ‘simple.wav‘ - ‘complex.wav‘ PCA comparison

Beyond comparing ccm and mp3, there is value in exploring PCA's results on on simple versus complex signals (Fig. 5). In a machine learning context, having many principal components contribute to the variance suggests a dataset with high rank and vice versa. Here, we see that with simple audio such as a MIDI piece, a much higher percentage of the variance is captured by the first (and few following) principal components, suggesting the music is rather simplistic. On the other hand, a complex song results in a less steep slope with a more consistent decline in variance explained, suggesting that there are many components essential to reconstructing the signal. Consequently, we hear that the ccm compression of 'simple.wav' sounds much closer to the original compared to the ccm compression of 'complex.wav'.

V. Conclusion & Future Work

To improve ccm, many changes can be made in the way of dynamic, intelligent hyperparameter selection. The number of dimensions to represent signal with and number of principal components to keep is highly dependent on the signal being compressed, so using this correlation would be beneficial to the scheme. I imagine basing hyperparameters off the complexity of the audio would benefit the algorithm's performance greatly, for example thresholding the number of components based on percentage of variance captured.

Furthermore, it'd be interesting to hypothesize what a principal component really is from an artistic standpoint. In the case of datasets, its much easier to visualize exactly along what axes the principal components lie along. With music, it's a much more interesting question to be answered.

In data science contexts, PCA is often used for Exploratory Data Analysis ("EDA") to find the absolutely essential, defining characteristics of the data. Even if the reconstruction is imperfect, the trends in the data still hold. However, when the data is not inherently low rank, it's dangerous to draw conclusions from PCA. From a musical perspective, it's very difficult to even hypothesize what a principal component is in a piece of music. In that sense, this approach to audio compression may be fundamentally flawed in that it seeks to discard a piece of music's defining vectors which, once performed, will always result in a perceptually significant loss. This makes less a difference in general machine learning classification, regression, or modeling tasks, for example, but when compressing music, the goal is not to create a model of the original sound, but instead to create a representation that allows people to enjoy it in its fullness without losing any perceptible pieces of it. mp3 cleverly incorporates human factors via psychoacoustic masking to keep everything humans can perceive, discarding only the sounds which humans cannot physically hear.

Although these days, hardware has been improving, networks have been speeding up, and disk storage capacities have been growing, the supposedly solved problem of audio compression is still a fascinating one to explore. As the field of machine learning deepens, the possibilities for musical applications can only grow in tandem.

References

[1] B. Speice, "Audio Compression using PCA" https://bspeice.github.io/audio-compression-using-pca.html

[2] A. Joseph, J. Hug, J. DeNero, S. Lau, S. Rampure, A. Bray, and J. Gonzalez, DATA100 SP21 Lecture 23, "Principal Component Analysis" https://ds100.org/sp21/lecture/lec23/

[3] C. Cella, MUSIC159 SP21 Lecture 8, "Sound Types"

[4] "The mp3 History" https://www.mp3-history.com/en/technology.html

[5] E. Roberts, "Data Compression" https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossy/mp3/algorithm.htm

[6] Computerphile, "Digital Audio Compression" https://www.youtube.com/watch?v=KGZ0een8vSE

[7] Computerphile, "How Digital Audio Works" https://www.youtube.com/watch?v=KGZ0een8vSE

[8] "FLAC" https://xiph.org/flac/

[9] Google Brain, Magenta https://magenta.tensorflow.org/demos/colab/