Musical MCMC

Harley Patton, Navneedh Maudgalya, Rahul Gupta

April 18th, 2018

1 Introduction

We sought to use the Markov Chain Monte Carlo method to sample from the waveform distributions of arbitrary pieces of music to construct our own version of the music. We discretized music samples and used the Metropolis Hastings algorithm to synthesize novel music by comparing our music's waveform to a prior distribution and the transition probabilities of moving from one sound value to the next. By sampling the MCMC's resulting distribution, we can produce a new unique piece of music that somewhat matches the general characteristics of the original piece.

2 Methods

2.1 Preprocessing the Music

Using pydub[1], we loaded fifteen, three second music samples from random parts of a song. Hoping to generate similar yet novel music from pieces of these music samples, we originally stored the sample's waveform as an array of amplitude values over time. However, this approach made it difficult to avoid the noise in the waveform and generate large pieces of music. Hence, we chose two strategies to split the samples into notes with actual musical content:

- **Method 1**: Split the three second music samples into three, variable length notes based on sudden changes in amplitude
- **Method 2**: Split the three second music samples into three, one second notes regardless of the waveform.

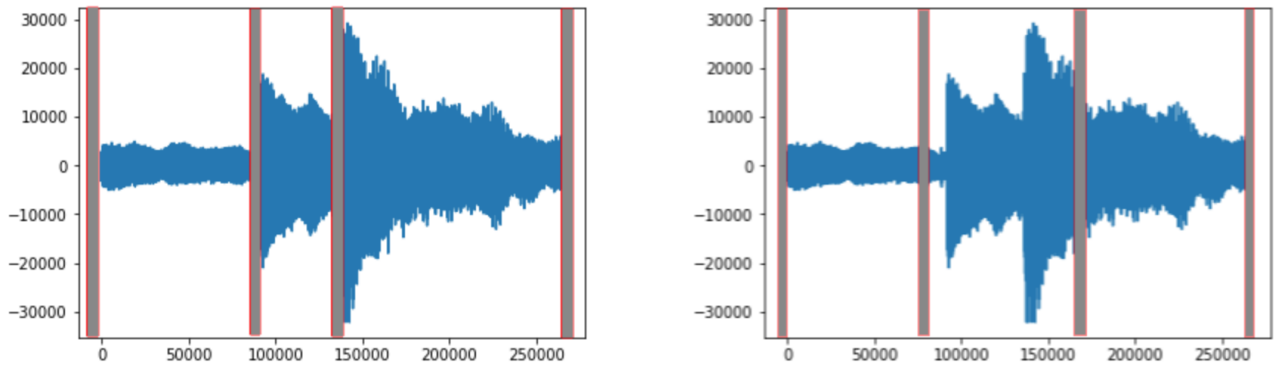

Figure 1: Method 1 (left) and method 2 (right) split the music sample into different sized notes for the MCMC algorithm. The vertical grey bars denote partitions between notes.

These new notes were uniquely indexed and mapped to their corresponding waveform array, leaving us with more manageable content to use with our MCMC algorithm.

[1] https://github.com/jiaaro/pydub

2.2 The MCMC Algorithm

The song was divided into n samples of which $m << n$ were selected. Since each sample had 3 one-second notes, there were a total of $3m$ states. The initial state was chosen from one of these.

The proposal function was generated by setting k of n states from the sample to a random state. The value k is a hyperparameter and it's limiting behavior, zero or n, will be expanded upon in the Experiments section. The lab's implementation of the candidate sampling did not allow for repetition. However, allowing repetition is reasonable in this use case since loops are often present in songs.

Given that candidate generation has been discussed, we will now talk about the acceptance procedure. The transition matrix is generated as follows. Initially, the song was divided into three second samples. Of the n samples, $m << n$ were selected by random sampling without replacement. Each sample was further divided into three smaller segments (we will refer to these as *notes* going forward). This enabled us to establish ground truth transition probability, namely $p = 1$, between note x and $x + 1$ within the same sample.

The transition probability from note y to x was a modified similarity measure between y and $x - 1$ (the note that immediately preceded x). The similarity measure between notes a and b is defined as $\frac{a^T b}{\|a\|\|b\|}$. The transition probability from last note of sample c and first note of sample $c + 1$ was 0 since the aforementioned definition breaks down. It was also 0 from note a to a, explaining the dark-colored, zero probability diagonal. This baseline similarity measure was divided by the maximum similarity over all previously calculated similarity measures. These values, which ranged from [0,1] were pushed closer to 1 using $f(x) = \frac{2}{1+e^{-lx}} - 1$ with $l = 13$ as seen in Figure 2. The value l is a hyperparameter.

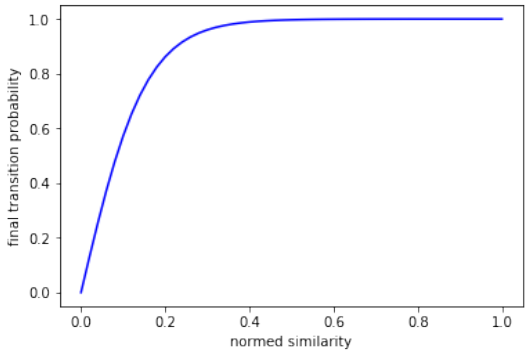

Figure 2: Squashing the pre-probabilities to improve transitions

By plotting histograms of various samples, we observed that the amplitude of the sounds in a music piece is Gaussian with near-zero mean. For simplicity, we assumed that the mean was zero since the standard deviation was significantly greater (in the 1000's). We used MLE to estimate the variance of each sample and averaged the samples to get a final estimate. We used this variance estimate in our acceptance function to ensure the music we produce comes from a similar normal distribution.

The scoring function $P_c(i, i + 1)/P_s(i, i + 1)$ (where c is the candidate and s is the sample) as used previously in lab was modified to include the Gaussian prior calculated using MLE as discussed in above. By narrowing the exploration space, this had the effect of removing outliers which would be heard as sharp changes and large amplitude values in the generated song piece. Our final scoring function used both the Gaussian prior and transition matrix.

In our experiment, we used the Metropolis-Hastings algorithm which is part of a family of MCMC algorithms for generating a sequence of random samples from a distribution from which direct sampling is computationally infeasible. Metropolis-Hastings is run for q iterations, where q is yet another hyperparameter. The proposal function generates a candidate state with the previous as input. The scorer function generates a probability score which is used to decide whether to accept or reject the candidate state. A procedure for generating the starting state is also provided.

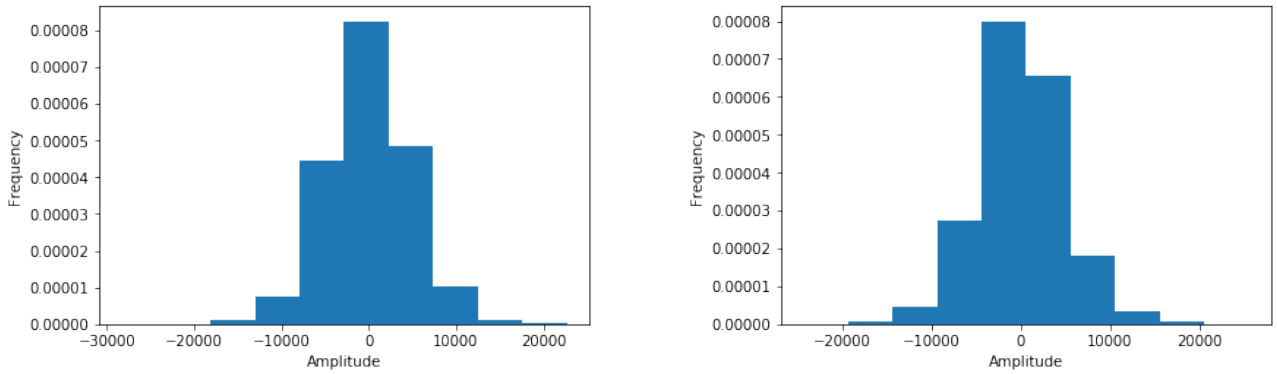

Figure 3: These are two histograms of amplitudes of two samples of the music. They look like zero-mean Gaussians.

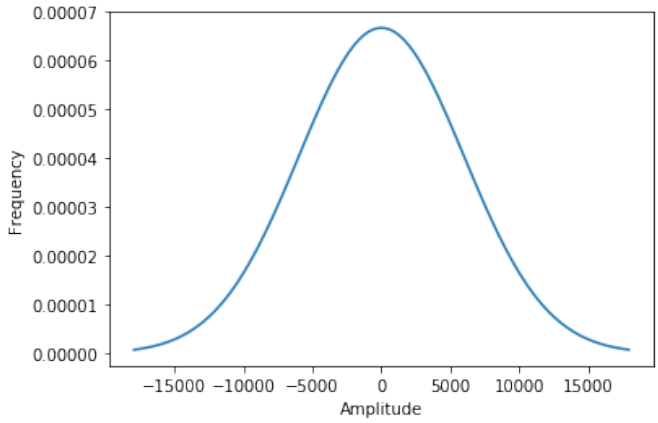

Figure 4: This is the prior distribution that was generated with the zero-mean simplification and variance being the average of the sample variances.

3 Experiments

We began our experimentation with the simplest possible case: a song whose notes are chosen from a very small range (an octave, for example) and for which the notes are of relatively uniform frequency and duration. With such a song it is not difficult to extract where in the waveform each note occurs, and the limited range of notes results in a transition matrix that can be efficiently calculated. For this part of our experiment, we used the Ode to Joy from Beethoven's Symphony No. 9. The song is short enough that the transition probabilities can be calculated from the sheet music[2] directly, as shown in Figure 5. The concentration of probability near the diagonal of the matrix corresponds to the fact that almost all notes in the Ode to Joy are followed by either themselves or one of their immediate neighbors on the major scale. The sparsity of the matrix along the bottom and far right are due to the fact that the Ode to Joy primarily consists of the notes C, D, E, F, and G; and rarely uses the notes A and B. A random walk along the Markov chain produced by this distribution produces a song[3] that sounds very similar in theme to the Ode to Joy.

Sampling from the waveform distribution of an arbitrary piece of music is much more complicated, since there is no constraint on how wide of a range of notes are included, how the spacing between notes varies, or how the duration of a given note varies. An arbitrarily large note range causes the alphabet size of notes to grow, and can make calculating the transition matrix intractable. The possible variance in note spacing and duration make systematically picking out each note from the waveform difficult as well. As a result, we

[2]https://www.8notes.com/scores/435.asp
[3]Available in our submission at otj_walk.mp3

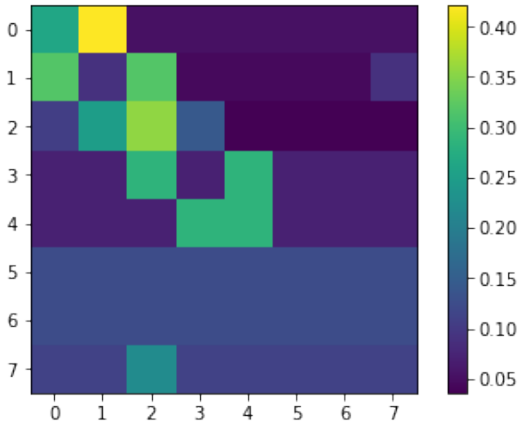

Figure 5: The transition matrix for Beethoven's Ode to Joy. The numbers 0-6 correspond to the notes C, D, E, F, G, A, B; and the number 7 corresponds to a beat of rest.

moved away from using individual notes as characters for our MCMC and instead used continuous chunks of music, as outlined in the previous section. We tested this method on the rap song M.A.A.D. City by Kendrick Lamar and the piano song Awestruck by Cellophane Sam. The transition matrices between music chunks for both songs is shown in Figure 6. The transition matrix for M.A.A.D. City is significantly sparser than that of Awestruck. This is a consequence of some of the structural differences between rap music and classical piano; mainly, the piano music is more melodic and features more frequent transitions between neighboring notes as well as repetitions of musical blocks. For this reason, we used Awestruck as our song for the rest of our experiments. A random walk along the resulting Markov chain produces music[4] similar in its characteristics to the original song. The waveform of the first 24 seconds of both the original and generated versions of this song can be seen in Figure 7.

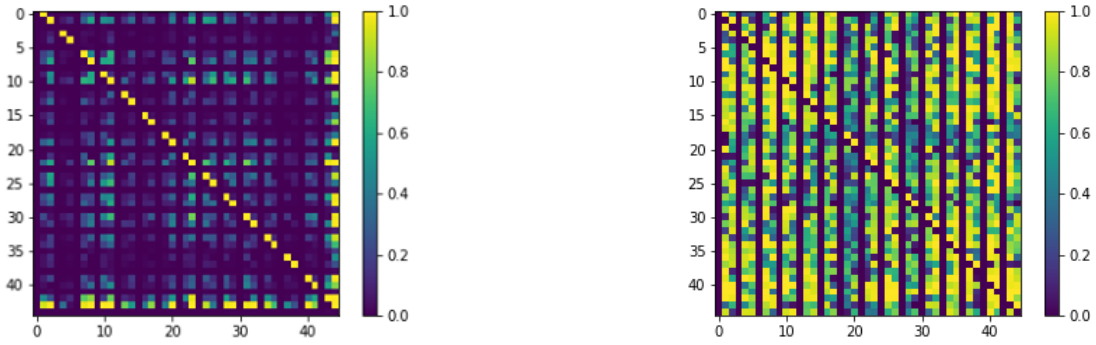

Figure 6: The transition matrix for Kendrick Lamar's M.A.A.D. City (left) and Cellophane Sam's Awestruck (right). The entries correspond to transition probabilities that have been re-scaled and monotonically raised for visibility. The indices along the axis correspond to the 45 music chunks derived from the songs' waveform.

The hyperparameters discussed below affected the nature of our outcomes:

- **Strategy to Split Music Sample**: We produced better music samples by splitting the original samples into three equal parts. This method worked well because all the notes' vectors were the same length, providing more accurate similarity measurements for the transition matrix. Also, every three second sample did not have three distinct notes, making it difficult to distinguish the notes using method 1.

[4]Available in our submission at awestruck_walk.mp3

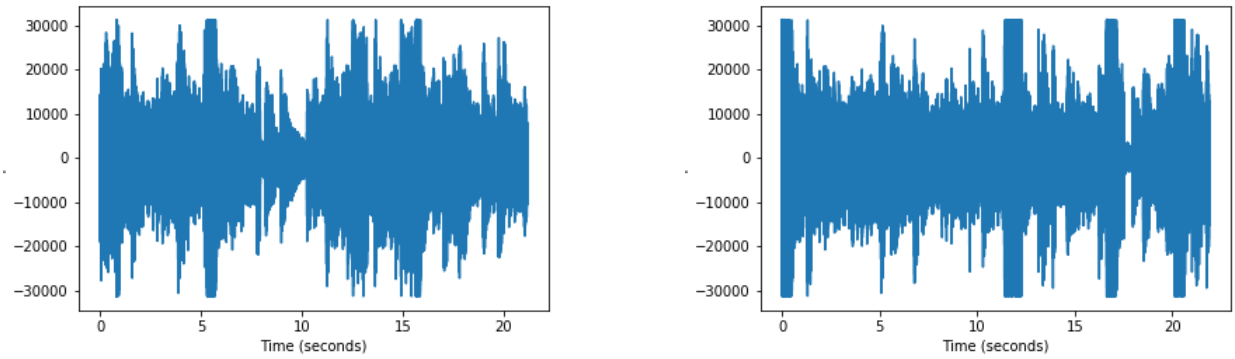

Figure 7: Both the original (left) and artificially generated (right) waveforms of the first 24 seconds of Cellophane Sam's Awestruck.

- **Num. of Original Music Samples**: We used 15 samples, randomly chosen from a given song. Increasing this hyperparameter, increased the size of our transition matrix quadratically and the number of usable music notes linearly. A larger number of music samples allows us to better represent complex music with different notes, but struggles to represent simpler music.
- **Num. of Amplitude Values used to Score with Prior**: We used 1000 amplitude values from the proposed and candidate arrays to score with our prior normal distribution for the acceptance scorer function. Using more values would be computationally more expensive but would allow the MCMC algorithm to propose and accept a new waveform more likely sampled from the distribution of the original song.
- **Num. of Iterations of MCMC**: We chose to run the MCMC algorithm 100 times to produce novel music samples in a timely manner.
- **Num. of Value Changes in Proposal Function**: When proposing a new array representing our novel music sample, we chose to alter four notes. If the proposal function altered more notes during each proposal, our MCMC algorithm would sample from larger portions of the sample space but would take longer to converge on a final music sample. However, with fewer note changes, we sample from a smaller sample space but the probability that we converge on a music sample sooner is higher.

4 Conclusion

Using an MCMC to generate music is most effective when the music sample used to generate the transitions is as simple as possible. The ideal song for this would be one that contains very few notes, almost all of which come at the same frequency and last for the same duration. Since arbitrary music does not guarantee any of these constraints, we found it necessary to introduce a new way to build a transition matrix for a given sample of music. By splitting the music into fixed length chunks and then extracting individual notes, we have come up with a method to calculate transition probabilities between similar-sounding chuncks of music. From this, we have been able to use an MCMC to create novel music that sounds similar to its source. Although our MCMC was able to recreate this music, there are many other models that may be more qualified for this task. In particular, both Generative Adverserial Networks (GANs)[5] and Variational Autoencoders (VAEs)[6] have shown promise when it comes to music generation as well.

[5]https://arxiv.org/abs/1703.10847
[6]https://arxiv.org/abs/1711.07050