Robot music: generating beats using factor graphs

Saturday 16th December, 2017

Han Lin Aung, Justin Xu, Sicheng Zeng

Keywords—Music, K-Means, Maximum Likelihood Expectation, Bayesian Network

INTRODUCTION

Music is a popular project topic in the history of CS221, whether it is jazz generation, running adversarial networks to produce similar songs to those in a database, or imitating the style of Bach's baroque work. However, getting good results historically proves to be difficult, no matter what approach or niche of music is attempted, so we wanted to approach the topic in our own way, using a preexisting knowledge of music theory to gain a step up on our music generation, and skip ahead to the good part, where it starts to actually sound like music that was made 'by hand'.

Typical music structure, no matter the genre, is subdivided into different parts - usually one or two melody lines, bass or beat lines, and then supporting accompaniment lines. We hoped that music generation would be better if we took advantage of this structure, and generated the parts separately. So, we focused on generating beats. Training on existing beats, we would generate new beats similar to the beats in the training set.

RELATED WORK

There have been several previous 221 projects with music generation from existing music as their goal. For instance, Kai-Chieh Huang, Quinlan Jung, and Jennifer Lu's "Algorithmic Music Composition using Recurrent Neural Networking" looks at generation of music having learned on a specific composer's music in one genre, and attempts to create music indistinguishable from music that the original composer created [1].

Chris Pesto's 221 final project was simply titled "Music Generation", and had a more general objective, to generate new music using a convolutional neural network trained on a giant database of music. Pesto's paper states that he used music21 to parse his data and uses Google's trained Project Magenta AI to judge whether or not the resulting music was successful [2].

Yet another 221 final project by Allen Huang and Raymond Wu is titled "Deep Learning for Music", and is focused on creating melodies and harmonies from using only deep neural nets trained on classical music. Our approach is similar to theirs as both of us focused on a narrow aspect of music, them melodies and harmonies (which are heavily codified and algorithmic in traditional music theory), and us beat lines[3].

Other, non-221 examples of generating music include the previously mentioned Project Magenta, which uses TensorFlow to create art and music, and deepjazz, which is trained on an eight minute long jazz piece to generate similar sounding endless jazz. These non-CS221 projects have had more time to train than we would've had, and although they generate decent sounding music, we chose to not use a similar, neural network-based approach, to avoid training times and have flexibility and instant musical feedback.

FINAL PROBLEM DEFINITION AND STRUCTURE

To approach the problem, we utilized an online beat generator website called Splice: https://splice.com/sounds/beatmaker. Splice defines a beat as a repeating two measures of music. Each measure is further subdivided into 16 moments, during each of which a sound could choose to either start playing or not. If a sound's length is longer than a moment, then the sound persists for the full duration, rather than cutting off when the moment is over. We had eight sounds, or instruments, playing simultaneously in our beats: kick, snare, open hat,

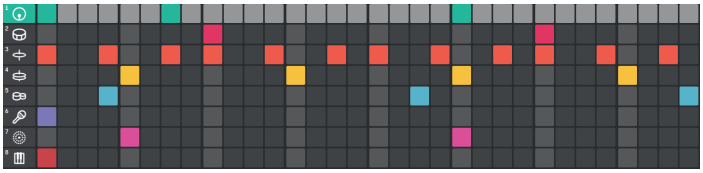

Fig. 1. An example of someone's beat.

closed hat, mid tom, percussion, fx, and stabs.

The search space is quite large, as there are 256 boxes that can be turned on or off, making there 2^{256} possible unique beats.

Our objective was to create an AI that would look at other people's beats and learn from them to make its own beats. Using this model, we would automatically generate a new beat after training on training samples that we would scrape from other people's creations on the website.

Oracle

We considered the most popular of the human made music to be our oracle, our best possible beats. In order to determine how close we were to the oracle, we mass surveyed random people and had them listen to a set of music, of which half is music we generated, and half is original music files. Without knowing whether the music was original or machine generated, they rated the music from 1 to 10 overall, giving us a true rating on our music.

Data

We initially manually got links to splice beats from the popular/sample beats that were front and center on the website, which amounted to around 20 unique beats. Later, we web-scraped twitter to search for splice beatmaker links, which allowed us to get a data set with a size on the order of around 700.

Bayesian Network

We approached the problem by representing it as a Bayesian network, where each variable represented one individual square on the grid. A variable would be assigned to either 0 or 1, depending on if the space on the grid is turned on or off. This way, the Bayesian network would model the conditional probabilities of certain nodes given the assignment of others. The music theory reasoning here is that certain sounds are more likely to play if other sounds had been played before or were also playing at that moment.

After developing a Bayesian network with all the necessary probabilities decided, we would dynamically build up a new assignment to the factor graph. We started from the top left most node and traversed from left to right, then down a row and left to right again, until reaching the bottom right most node. In order for this to work, we ensured that in any of our Bayesian network variations, the parents of a node are either above or to the left of it, so that we can always calculate the conditional probabilities exactly before we assign the next node. We set a corresponding node to 'on' with probability based on the conditional model that we received from our model.

By utilizing a stochastic model to generate our beats, we allow for there to be more variety in the beats that we create, rather than one best beat. It was important, however, to balance variance and determinism to ensure that we were not overfitting to our dataset, and generating a beat pattern too similar any of our inputted beat patterns. As a result, we also include a similarity metric to our generated beats that measures the maximal similarity to any assignment in our training set.

Baseline Model: Naive Bayes

The baseline algorithm that we used is Naive Bayes, which found the probabilities by assuming all the blocks are independent from each other. In terms of a factor graph, this corresponded to a Bayesian Network where each node was independent from each other, or a graph with no connections between nodes.

The optimal probability assignments were calculated by taking the ratio of the number of times that a node is turned on over the total number of training examples. This way, we captured the average probability that a node was turned on.

Naive Bayes Results

The results of the baseline produced reasonable sounding beats when we initially tested on our small 20 sample size training set. Irregularities in rhythm

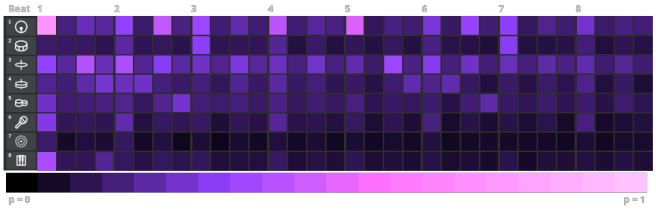

Fig. 2. MLE probabilities using all of the beat data

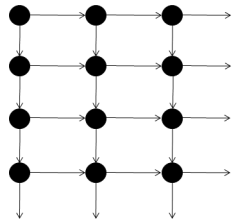

Fig. 3. Bayesian network including connections to only the instrument right above

sometimes occurred, where mismatched rhythms from different samples seem to have gotten mixed with each other.

This problem was exacerbated when we moved on to a larger ∼700 sized data set. Because of the wide variety of beat patterns and density of notes, the algorithm tended to create strange, unnatural sounding rhythm patterns almost every time the algorithm was run.

Complex Bayesian Network Model

Figures 3 and 4 show the two Bayesian networks we implemented.

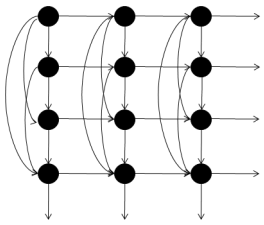

Fig. 4. Bayesian Network with connections to all instruments above

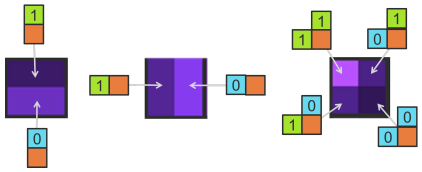

Fig. 5. A key to interpret conditional probabilities

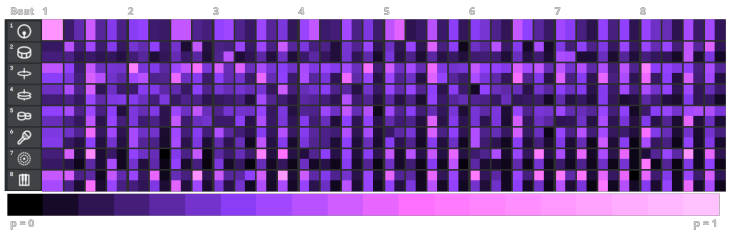

Fig. 6. Conditional Probabilities using all of the beat data and the Bayesian network in figure 3

Figure 3 includes two factors for each node - the node to the left and the node above.

Figure 4 has, for each node, a condition on the node to the left and and every node in the same column in the rows above.

These conditions were chosen because a beat pattern is a time dependent series, and the on and offs naturally progress as we go from left to right. Because the rows represent the different instruments, it also made sense to create relationships between instruments at the same beat.

Because of our condition that we could only make parents for a node if it was either above or to the left, Figure 2 could only include connections to the instruments above it.

We performed maximum likelihood expectation (MLE) over our training set to get the ideal set of conditional probabilities for each setup.

In theory, having more connections would probably make for a more predictive and complete model. However, given that our training set was on the order of ∼700, and that for instruments on the bottoms row, the number of factors was 8, there were a total of $2^8 = 256$ possible ways to condition on some of these variables. As a result, many of these conditional probabilities became 1 or 0 due to the low number of samples with the respective assignments.

This led to some nodes being deterministically assigned, which was acceptable, as it could represent certain rules in beat patterns. However, it meant we had to be careful about overfitting or reducing variance.

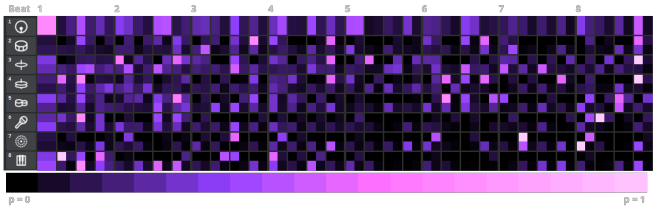

Fig. 7. Minimum Cluster Conditional Probabilities

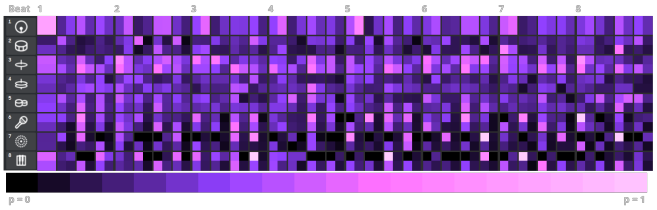

Fig. 8. Middle Cluster Conditional Probabilities

K Means

Preliminary experimentation resulted in clean music that mostly made sense. Especially when we were using the smaller 20 sample size training set, the music came out quite nice.

However, some of the problems that were a part of the Naive Bayes were also prevalent when we transitioned into a larger training set.

Reviewing some of the training samples, we noticed that there was quite a large variety in the densities of music. Some included beats with very high density, nearing having half of the possible blocks turned on, while other samples were quite sparse, with around 10% filled.

High and low density beats typically had quite different methods to making a solid beat. High density beats typically wanted more consecutive and regular structure, while low density wanted more spaced out structures, with off beats thrown in there every now and again.

When combining the two densities, as our algorithm did, it led to a bad combination of intensity and sparsity because of the way that values averaged out, which was not something that our probabilistic model could easily solve by itself.

As a result, we implemented K-means to separate

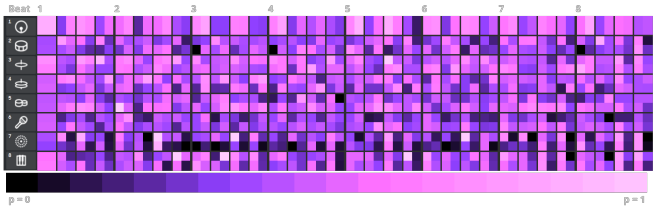

Fig. 9. Max Cluster Conditional Probabilities

out our training data into different clusters based on the density.

We experimented with 3 clusters and 5 clusters, getting similar results for both, and ultimately ran our 3 cluster algorithm to generate our desired outputs.

The centroid values for our 3 clusters ended up being around $20, 50,$ and 100 nodes turned on out of a total of 256 nodes. This resulted in much cleaner music, with the high and low intensity clusters producing vastly different kinds of beats.

TABLE I. K-means splitting

Metric	Low	Medium	High
Centroid Values	20	48	101
Number Files	455	258	52

Similarity Metric

We defined **node similarity** between two beats as the total number of nodes both on or both off in both beats and divided by the total number of nodes per beat, which is always 256.

This seemed like an accurate way to measure similarity, but for the higher density or lower density assignments, since there was always a lot of 1's or a lot of 0's, there would be a lot of similarity sheerly from that, so we also wanted a secondary similarity metric.

We defined **assigned node similarity** between two beats, a generated beat and a training example, as the number of times that a node was assigned to 1 on both beats and divided by the total number of 1's in the training example.

We then found the largest similarity metric over all training examples and outputted it.

TABLE II. AVERAGE SIMILARITY METRICS

Similarity	Low	Medium	High
Node Similarity	0.94	0.789	0.64
Assigned Node Similarity	0.5	0.41	0.45

Our metrics showed that Node similarity was quite high for the low density, as expected because of a lot of the 0 space. However, when factoring in the assigned node similarity, we found that none of the algorithms overfit the data too heavily, showing that we successfully created new, unique beats.

ALGORITHM FOR ASSIGNMENT GENERATION

The outline for our overall algorithm is below.

Algorithm 1: Assignment Generation Algorithm

```
Read in Data into a dictionary
Run k-means on note density to get
  assignments and centroid values
Choose a cluster and get the corresponding
  files in the cluster
ZeroCounts = {}
OneCounts = {} for all training examples in
  cluster do
    for node in array of beats do
        Create tuple based on conditions of
          parents of node
        if node is 0 then
            ZeroCounts[tuple] += 1
        else
            OneCounts[tuple] += 1
        end
    end
end
ConditionalProbs[tuple] =
            OneCounts[tuple]
  OneCounts[tuple]+ZeroCounts[tuple]
for node in array of beats do
    generate tuple for node
    assignment[node] = 1 with probability
      ConditionalProbs[tuple]
end
```

RESULTS AND ERROR ANALYSIS

We surveyed people to give ratings on a scale of 1-10 on 6 clips of music, 3 of which were made by humans and 3 of which were made by AI. The order of the music clips are randomized so that people did not know which songs were produced by AI and which songs were produced by casual human beatmakers. After we collected our survey responses, of which we got over 70, we found that the mean and standard deviation of the ratings produced by our algorithm were 4.1289

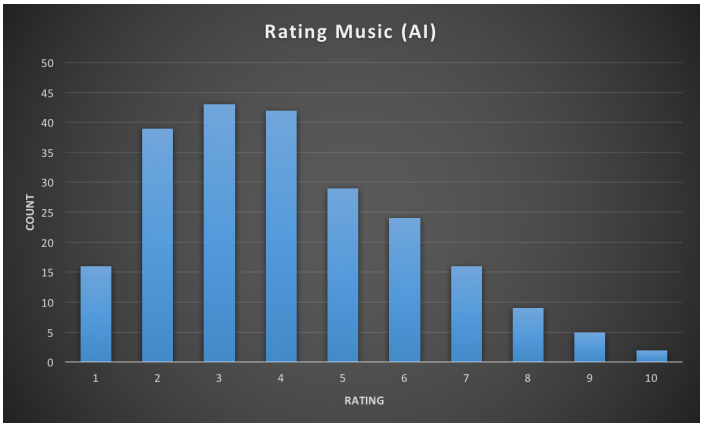

Fig. 10. Ratings of music clips produced by AI

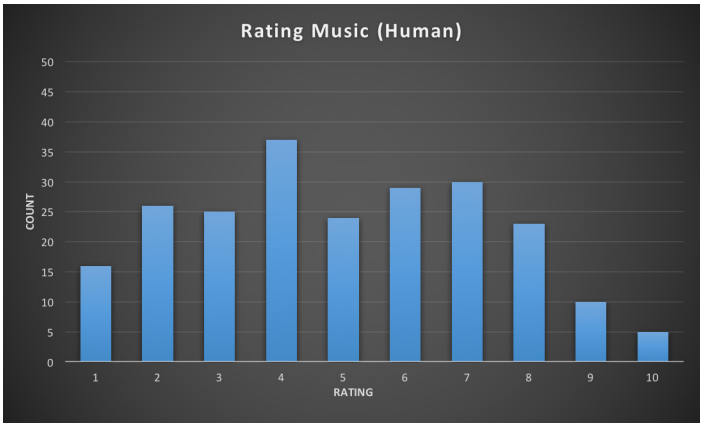

Fig. 11. Ratings of music clips produced by humans

TABLE III. RATINGS WITH AI VS HUMAN

Mode of production	Mean	Standard deviation
Human	4.9733	2.3791
AI	4.1289	2.0586

TABLE IV. AI RATINGS SEPARATED INTO THE DIFFERENT DENSITIES

Density	Mean	Standard deviation
Low	4.48	2.18
Medium	4.267	2.00
High	3.64	1.93

and 2.0586 respectively. As for the beats produced by our humans, the mean and standard deviation of the ratings were 4.9733 and 2.3791 respectively. Hence, the ratings produced by our algorithm scored slightly less (0.84 less) than that of humans. However, the standard deviation (spread) of the ratings produced by our algorithm is slightly less (approximately 0.2 less) than that of the humans. Additionally, the standard deviation was much greater than the difference between the two means. As a result, there does not seem to be a significant difference between the ratings of beats produced by AI or by humans.

Our survey results also indicated that low-density beats scored a higher mean rating than other density beats. In fact, according to our survey results, the lower the density the beats are trained on, the higher the rating. While we cannot conclusively claim this finding due to lack of sufficient amount of survey data, this is a future area of work we can look into.

While our beats are comparable to that of a casual beatmaker, some of the beats that are produced are not as high quality as others, and some of them are definitely of lower quality compared to the oracle. When we inspect the beats, we realize that there are certain beats that still sound cluttered (especially for beats trained on high and medium density datasets) and irregularities in the rhythm that do not necessarily sound nice. There may be two reasons for such irregularities. First, since our beats are produced based on conditional probabilities, there will be certain times in which certain beats sound better than others and certain irregularities might be present. One other source of issue may be from the dataset itself since the input data is scraped from casual beatmakers on the Internet submitting their beats. Hence, the input beats themselves may already have irregularities in them. However, generally looking at all the beats produced, the beats produced by our algorithm are not significantly different from those produced by casual beatmakers.

Conclusion

Our algorithm produced beats at a quality that is at least comparable to that of an average casual beatmaker. While the beats produced by our algorithm did score lower in terms of the ratings compared to the beats produced by a casual beatmaker, the ratings did not diverge far enough for people to tell the difference between beats generated by humans and the algorithm.

One finding that resulted from our survey is also that by using K-means to classify dataset into different densities of beats, our beats sounded better. Furthermore, beats trained on low-density dataset scored higher ratings compared to beats trained on datasets of other densities. This means that the gap between the beats produced by humans and by our algorithm can further be diminished if we were to survey people only beats of low density produced by our algorithm. More future work can be done in terms of looking into different densities of beats.

Future Work

In terms of the future work, according to our results and analysis, there seems to be promise in using MLE, and factor graphs to generate loops of beats and even melodies.

One area of future work is in delving deeper into gathering more information and data into finding how or if different densities of beats produce different ratings. Though we were able to gather some data and ratings on low, medium, high densities of beats, we still needed more data to further conclude that this finding of training beats on low-density data set creates higher ratings is true. One future area of work can be to delve deeper into finding what might be the potential causes of finding why different densities of beats show disparities in terms of ratings and may help in creating finer-tuned factor graphs (or constraints) for better sounding beats.

Another area of future work is also to apply the algorithm on pieces of music other than beats. Right now, we are focusing on repeatable beats, but we can also try this algorithm out with different genres of music and also melodies.

Acknowledgment

Thank you to our project advisor, Helen Jiang, and to all of the CS221 staff and professors!

References

[1] Huang, K., Jung, Q., & Lu, J. Algorithmic Music Composition using Recurrent Neural Networking.

[2] Pesto, C. Music Generation.

[3] Huang, A., & Wu, R. Deep Learning for Music.