LILAC

Manipulating Data to Generate Music

Elijah Foreman, Kevin Liu

CS4100 | 4/26/19

Abstract

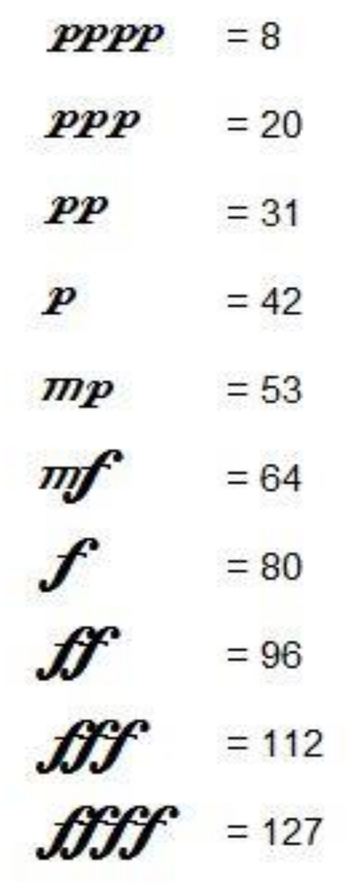
Music is an art that can be enjoyed at any point in one’s life. However, what makes music “appealing” to us and unique? In this paper, this question guides our research and conclusion. In order to determine what makes a piece of music unique, we utilize the state-of-the-art machine learning platform TensorFlow and its music manipulation library-Magenta. Magenta helps artist and software engineers generate art and music. We researched the underpinnings of Magenta and created our own MIDI manipulation model called Lilac. Using Magenta’s built in data structures, we were able to figure out what makes certain songs unique.

Keyword: Music, Machine Learning, Variational Autoencoder.

Music can be enjoyed by nearly everyone alive; no matter what race, religion, political affiliation, or age category. As avid fans of electronic dance music, Kevin and I realized that there were a few patterns some of our favorite music producers repeatedly used. Whether it's putting the emphasis on the down beat, or making use of layered chords, producers seemed to be utilizing the same tricks. However, if this was really the case, all electronic music would sound very similar or exactly the same. This, luckily, isn't the case and a track produced by Marshmello is significantly different than a track produced by Jai Wolf. Our goal was to extract what features make certain songs unique, and time permitting, use this knowledge to make our own compositions.

In this project, we set out to combine the machine learning and music production/analysis domains. However, this required us to brush up on domain specific knowledge. We had varying understandings of music theory, so we both made sure we were on the same level of understanding before starting the project. Important concepts we learned included: tempo, note velocity, pitch, note duration, measures, time signature, melody, harmony, and chords.

Another thing that was important for us to understand was the concept of MIDI, since it was how we were feeding our model data. MIDI, or Music Instrument Digital Interface, is a type of file that carries event messages and data that specify instructions for a piece of music. By using these MIDI files, we were able to instruct our model what it should sound like. MIDI files make use of "MIDI code" which make use of numerical mappings for different musical attributes. For example, in Figure 1, the MIDI codes corresponds to the velocity of each note.



pppp	= 8
ppp	= 20
pp	= 31
p	= 42
mp	= 53
mf	= 64
f	= 80
ff	= 96
fff	= 112
ffff	= 127

Figure 1: MIDI Code that corresponds to a note's velocity

The velocity value goes from 1 to 127, and it covers the range from an inaudible note up to a maximum note level. Using MIDI codes to describe musical attributes is useful because it allows us control them with a certain level of nuance. Pitch, time, instrument type, tempo, and measure length all have corresponding midi codes.

In order to combine MIDI technology with our machine learning library and work with it efficiently we needed to have a way with interacting with the data inside the MIDI file. Our solution to this problem was to use Magenta's MusicVAE model and it's NoteSequences data structure. NoteSequences are an abstract representation of a group of notes. Each note has a MIDI code corresponding to a unique pitch, start time, end time, velocity, and instrument type. NoteSequences can also be trimmed, concatenated, expanded, and converted to other file types. MusicVAE has a MIDIConversion class that allows the user to convert midi files into NoteSequences and vice versa.

One of the main issues with training models using MIDI files is the fact that music is an extremely dynamic art form. Using the piano as an example, if only one key is pressed down, 90 different events could have happened. That's because a normal piano has 88 different keys and including the events for pressing the key and releasing it, there are a large amount of different possible event combinations. Even when we ignore tempo, only use 16th notes, and specify that a piece of music is just two measures long, you would still end up with 90^{32} different event combinations. If you extended this to just 16 bars, the length of the average verse in a song, the possible amount of combinations balloons to 90^{256}, and this is just for one instrument- the piano. However, out of the 90^{256} different combinations, only a select few would sound musical. Trying to find usable combinations by simply going through the different possibilities is not feasible. Therefore, using a latent space model, like MusicVAE, is the ideal option.

Latent space models can learn the fundamental characteristics of a data set and exclude any unusable combinations. The models do this by clustering fundamental qualities in the data set together and laying out variations along vectors defined by three properties. These properties are Expression: where data can be mapped to some point in the latent space and reconstructed from it, Realism: Any point in the latent space represents some realistic data., And Smoothness: Data from nearby points in the latent space have similar qualities to each other. For our project the most important properties are Realism and Smoothness, because they allow us to randomly sample new examples that are similar to those in our dataset by grabbing them from a randomly selected data point in the latent space.

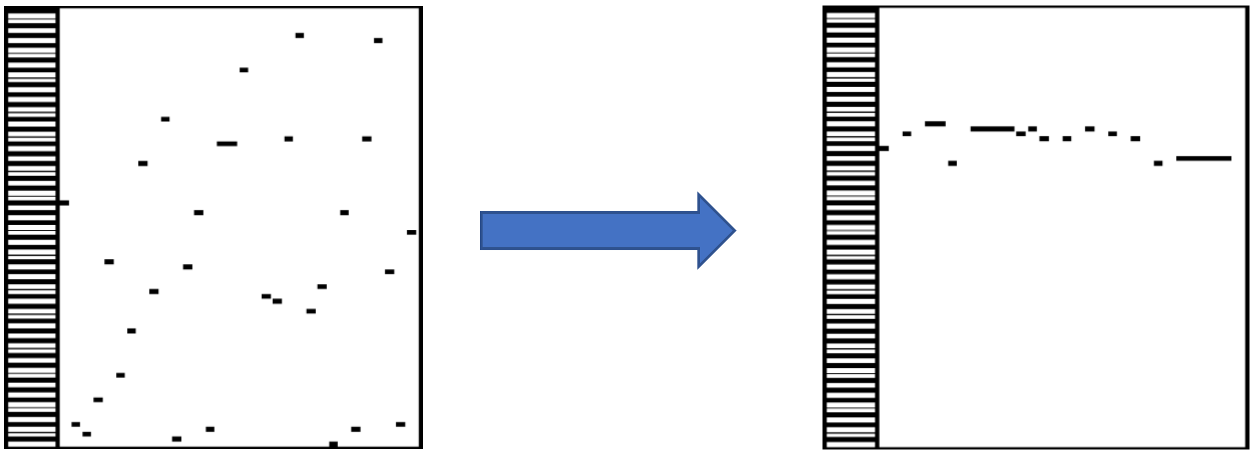


Figure 2 (left) A piano roll showcasing possible melodies, without using any sort of model that gets rid of "bad" variations. (right) A piano roll with no bad combinations, that has been trained with the "mel_2bar" latent space model

On a deeper level, the latent space uses these properties to form constraints. In the paper, “Latent Constraints: Conditional Generation from Unconditional Generative Models”, the authors use these concepts to condition generation without retraining their model. By using the constraints, the authors create a value function that can identify regions in the latent space that generate desired outputs. These constraints create the bottleneck called the encoder.

As Shown in Figure 2, using latent space models to get rid of bad combinations is extremely useful. MusicVAE does this by utilizing an autoencoder model. As Shown in Figure 3, The autoencoder process is broken down into two steps: compression and reproduction. The model creates the datasets latent space by compressing, or encoding, each data point into a vector of numbers. These numbers are called the latent code. These numbers then go through a filtering processes called the bottleneck. The bottleneck uses fewer dimension than the actual data it’s given, which forces the model to learn an encoding scheme. In this process the bottleneck finds the qualities that are common throughout the given dataset. The model is then readily available to reproduce, or decode, new data based on the previous data and the information gleamed from the bottleneck.

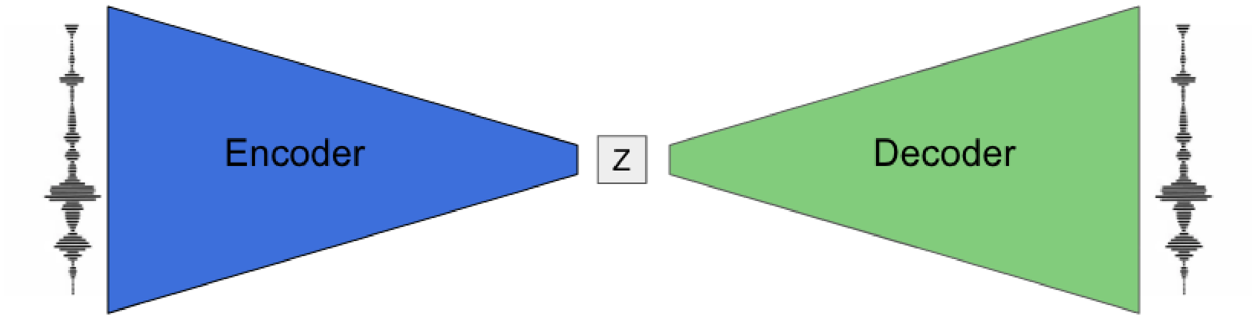


Figure 3 The Autoencoder Process

For this project, we created our own dataset. We did this because the model required us to use midi files that only contained piano samples. We took popular songs from the Chainsmokers off of YouTube, converted them into mp3 files, and then converted them into MIDI files. This processed created a few problems, because the MIDI files for some of the songs were corrupted with random breaks and notes. Since we converted from mp3 to MIDI, there was a high signal to noise ratio. The optimal dataset would be to produce directly in MIDI form to keep the original melody. Because of these reasons, we could only use a smaller subset of our original dataset when training the model.

Our model was also trained from the lakh midi database, which contains about 1 million different midi files. However, the sample model we used was trained by the magenta team with the "mel-2bar" dataset. This differed because, their dataset contained music that was only 2 bars long, while we used whole songs. For our model, we first trained with Lakh midi dataset to obtain a baseline for what "music-like" would be. From there, we would train the model with our own dataset of Chainsmokers songs. Since there are only so many usable Chainsmokers songs, we wanted to change the learning rate for our model when we trained using Chainsmokers songs. We did this by setting the learning rate for The Chainsmokers songs to be 100x the learning rate for the lakh midi songs.

Our hypothesis for this experiment was that because we used songs by the Chainsmokers to train the models, the resulting output files would skew toward sounding more like the Chainsmokers. We conducted our experiment by first training our model with the dataset. We then had one person pass a song into the model, without the other persons knowledge. The clueless person then had to guess what song it was based on the MIDI files that were created. We did this process three separate times, which resulted in correct guesses on two different occasions. The songs produced by the model was only recognizable from its distinct melody. That is exactly what we were looking for “sound like Chainsmokers” music generation.

Throughout this project, we faced many difficulties but learned just as much. First off, we underestimated the transition from theoretical machine learning, which we learned about in class, to practical machine learning utilizing TensorFlow and magenta. In class, we focused on supervised learning, whereas this project is drastically different. Setting up TensorFlow for the first time has been difficult. Applying magenta tutorial and making sure it works on local machine was also difficult.

It is also hard to apply machine learning principles without fully understanding the domain. Especially for music, domain knowledge is crucial. Magenta’s pre-trained models have strict requirements that includes specific number of bars, the type of instrumentals, and melodies. This additional domain knowledge piled on the already steep learning curve.

Some unexpected setbacks we had includes the midi file format and training the dataset. Originally, we expected a direct conversion between mp3 files, which represent songs from a certain artist, to midi files. After performing conversion using online tools and python libraries, it seems that going from mp3 to midi is difficult and we cannot obtain the original sound. Mp3 to midi conversion creates a lot of noise that is not included in the original midi file. Some file conversions were so corrupt that the original melody and pattern was lost. Even with the original midi file for songs we were looking for, we ran into the problem with training the model. To train the model, we need midi files. We obtain lots of midi dataset from an online depository named “The Lakh MIDI Dataset v0.1”. The problem now becomes how can we train our model to be music-like while leaning heavily towards features from the artist we want. Because the magnitude from the Lakh midi dataset is significantly larger than midi files we can obtain for any artists, we need to figure out a way to be biased towards the artists. There are close to a million midi files from the Lakh Midi dataset. But even famous artists like The Chainsmokers have less than 100 songs. This was a major setback we faced during the project.

Although we had many difficulties and setbacks, we enjoyed the process of this project. We believe that exploration on an interesting domain was crucial in our enjoyment. So, one advice to future CS 4100 students, pick a domain you're interested in exploring so that even when you're struggling, you're having fun struggling. Also, don't be afraid to explore. When we first researched into the possibility of this project, we were faced with mountains of unknown terms and ideas. Even though it's been an uphill climb, we had fun doing it.

Resources

"MIDI Tutorial." *MIDI Tutorial for Programmers*, www.music-software-development.com/midi-tutorial.html.

"MusicVAE: Creating a Palette for Musical Scores with Machine Learning." *Magenta*, 15 Mar. 2018, magenta.tensorflow.org/music-vae.

TensorFlow. "TensorFlow/Magenta." *GitHub*, github.com/tensorflow/magenta/blob/master/magenta/music/midi_io.py.