CSC111 Project Report: Generating New Music Based on Pre-Written Music

Max Wang

Friday, April 16, 2021

Problem Description and Project Goal

Music is one of my interests outside of computer science, and I have been interested in the possibility of music being written by AI. A recent example of AI-based music is the Jukebox project by OpenAI, a neural network which generates new music using audio from any musician's discography. The approach for this project is much simpler, however. The main focus is music composition, or the actual written music similar to what is contained in sheet music, rather than generating new audio. **I am using the chord progressions of pre-written music to generate new pieces of music.** I worked with MIDI data, which consists of the raw data based on each note of a song. I used a dataset with MIDI files of hundreds of songs and analyzed the chord progressions of each. These chord progressions were used to randomly generate new chord progressions and melodies on top.

Dataset

This project uses the POP909 dataset, which can be found on Github at https://github.com/music-x-lab/POP909-Dataset (Wang et al.). This is a dataset consisting of the MIDI information from 909 songs. The dataset contains both MIDI files and text files which lay out the specific chords of each song. The text files contain the start and end points of each chord in the first two columns, but only the third column is necessary as it contains the name of the chord itself.

Computational Overview

The analysis of music from the dataset begins in process_chords.py, which extracts lists of chords for each song in the dataset. The chords in the dataset's text files are first read using the extract_all_chords function, which uses extract_chords as a helper function. The chords are filtered during the file reading by removing any slashes or parentheses from chord names, since I judged those chords to be too complicated to handle. This function also has an optional feature to simplify chords by reducing more complicated or dissonant chords to a closely-related, simpler chord.

The next step is to use these collected chords to build a tree of possible chords. The ChordTree data type in chord_tree.py is largely borrowed from Assignment 2, with modifications to match the new domain. When inserting a chord progression into the tree, the list of chords is sliced in order to limit the depth of the tree. The chords from each song are added to the tree one-by-one.

Once this tree is created, the song can be generated in create_song.py. The first chord is randomly chosen, and each following chord results from randomly choosing another chord from the current chord's subtrees. Once the end of the tree is reached, the process restarts at the top of the tree. This loop continues until reaching a specified length, then the first chord is appended to the end. At this point, the chords can be converted into MIDI information using the MIDIUtil module (Wirt). A helper function, chords_to_degrees, first converts each chord into specific pitches based on the root note and quality of the chord. In create_midi, a MIDIFile object is initialized with basic parameters, like tempo. Each chord is added to the file using the addNote function, specifying the starting point of each note and its duration. A melody can be created by choosing random notes from each chord and setting a random duration for each note. A bassline is also added by simply duplicating the root note of the chord an octave below. Finally, this function outputs the completed MIDI file using the writeFile function. Then, the file can be opened using the subprocess module, which will open the file in the device's default media player.

Instructions

The POP909 Dataset can be picked up through UTSend:
Claim ID: 6pabUofhMUT4s5RS
Claim Passcode: 9889zNkjqCsFxABk
The dataset can also be found through the Github link. Extract the POP909 folder into the folder containing main.py. When running main.py, the process should only take several seconds, but there are still print statements indicating the start and end of the process if needed. The song should automatically open in your default media player using subprocess. If it does not work, the MIDI file can be found in the same folder as main.py. To create a new song, close song.mid first to avoid an error, then re-run main.py. Freely experiment with the variables at the top of main.py, which can create different results.

(Bonus: Listen to example.wav to hear one of the resulting songs played through a synthesizer)

Changes

For the most part, the end result was close to what was vaguely described in the plan. The plan of the project did not undergo much change aside from neglecting a few off-hand ideas. I planned to experiment with multiple genres or artists, but I favored improving the chord generation process as it would provide a satisfactory result. The original plan for creating melodies involved using a separate tree, but I instead opted to randomly choose notes from the chords, which was a simpler and just as sufficient alternative.

Discussion

The main problem I encountered was figuring out how I would avoid adhering to the pre-written music. As mentioned in the computational overview, I created a limit to the depth of the chord tree. Simply following a chord tree without a limit would result in songs which are too similar to the pre-written chord progressions. My solution was to limit the depth of the tree, so any chord progression could only borrow a certain number of chords from one song. On the other hand, choosing a very small depth would result in disjointed, loosely-connected chord progressions. Thus, I attempted to balance this by keeping the tree at a depth of around 5.

Another issue was the overly large number of possible chord qualities. Every unique chord quality combined was too much to handle, and the list of possibilities included many chords which I was unfamiliar with. My solution was to disregard any slashes or parentheses in chords, which narrowed the number of possible chords to 18. Even then, some of the resulting chord progressions were too dissonant or vague, which inspired the idea of the optional use_simple_chords variable.

When choosing chords at random, the generated progression will continuously wander without reaching any conclusion. A very basic solution was to attempt to resolve the progression to the beginning chord, which would hopefully create a sense of finality and even allow the progression to loop. In music, the "dominant" chord is the most common for resolving a progression. In the generate_chords function, I force the second last chord to be the dominant chord, followed by the beginning chord. This solution is simple and not always seamless, but it provides a better result than abruptly ending the progression.

Aside from the occasional difficulties during the project and imperfect chord progressions, I was able to improve the chord generation to the point that many of the results are now enjoyable. The outcome of this project was ultimately satisfactory, even if results are not perfect. If this project were to continue, I could attempt to improve the chord generation so it is less dissonant and not purely random, and I may revisit the idea of attempting different genres and artists.

References

Wang, Ziyu, et al. *POP909 Dataset for Music Arrangement Generation.* 2020, github.com/music-x-lab/POP909-Dataset.
Wirt, Mark Conway. *MIDIUtil.* midiutil.readthedocs.io/en/1.2.1/.