Fix It Till It's Baroque'n:

generating compositions in the style of the Baroque period

Ivan Leung

Introduction

The ability for machines to create aesthetically pleasing works has been the subject of many debates. In the past few decades, many machines have been designed to produce original artworks in an attempt to demonstrate the potentials for machines to learn and to create. Notable AI machines include AARON, a 'Cybernetic Artist' developed by Harold Cohen that paints[1], and Emily Howell[2], a computer program written by David Cope to compose music. This project aims to add to the community of AI artists in the realm of music composition. In particular, this project will learn from the wealth of works in the Baroque period in to produce pleasing chorales and fugues.

Task Definition

This project will focus on music compositions in the Baroque Period. In particular, this project will attempt to generate works in the forms of fugues. The Baroque Period is chosen because many composition rules are strictly observed; the consistency of compositional conventions will aid the learning process of the AI program. Further, the results of the AI compositions have more objective evaluation compared to romantic or contemporary works, since most of the composition rules in the Baroque period are well-known and easily calculated, whereas the aesthetics of a romantic or contemporary work is harder to evaluate. Chorales and fugues are among the most common genres of Baroque compositions, and Bach, among other great composers of his time, has graced us with a wealth of excellent chorales and fugues that can used as the training dataset.

Literature Review

Emily Howell

Perhaps one of the most notable development of AI musical composition, Emily Howell was developed in the 1990s[3]'. She was continuously trained by her creator David Cope to tune her compositions to human taste.

Melomics

Melomics sprouted from a research project in University of Malaga. Using evolutionary algorithms that encode melodies as genomes, Melomics was able to generate contemporary original work without requiring human input in the training stage[4]. In 2012, Melomics published their first album Iamus

---

[1] http://www.kurzweilcyberart.com/aaron/history.html
[2] http://artsites.ucsc.edu/faculty/cope/Emily-howell.htm
[3] http://artsites.ucsc.edu/faculty/cope/Emily-howell.htm
[4] http://geb.uma.es/melomics/melomics.html

[5], which was recorded by the London Symphony Orchestra and was lauded by Time Magazine to be "[writing] contemporary music without human help"[6].

Music21

Music21 is a python package developed at MIT for "computed-aided musicology"[7]. This versatile toolkit includes capabilities to load large music datasets, parse and generate music XML files, and perform musical analyses on corpuses. The functionalities of Music21 will be immensely valuable for this project, because it has the ability to parse existing musical works into python objects and convert AI compositions to readable scores.

Algorithm: Recurrent Neural Network

Eck and Schmidhuber (2002)[8] used an RNN to train compose blues music. Long short-term memory (LSTM) was used to learn blues melodies and chords. Eck and Schmidhuber also brought up an important note often raised in AI art research: that evaluating the performance of the result is difficult to evaluate objectively. Fortunately, given the specific format of the composition I aim to produce, there are some, albeit not many, objective parameters that one could reference to evaluate the quality of the compositions.

Baseline and Oracle

As a baseline, chorale, another popular genre of Baroque compositions, is used because of its relative structural simplicity. The chorale is rhythmically much simpler than the fugue, in that the vast majority of the notes are played on beat and have the same note duration. Thus, the chorale can be viewed and modeled as blocks of harmonies moving together at a constant tempo (speed); a simple markov chain that predicts the next harmony as a function of the previous k harmonies actually suffice to yield reasonable results.

I used the Music21 package to parse Bach chorales (provided in the package) and generate baseline 'chorales'. Chorales are chosen for the baseline implementation because its form is much simpler than the fugue: a chorale typically has 4 voices (soprano, alto, tenor, and bass) singing in parallel, and the majority of the notes quarter notes that are on beat. The baseline algorithm is simply a counter object for all pairs of chord progressions that has appeared in the training sample of Bach chorales. Given a chord (with 4 voices), the next chord is generated randomly from the set of chords that has appeared in the training samples. For programming convenience, the algorithm only considered chords that are on beat, and assumed all chords would have the duration of a quarter-note. An excerpt of sample output is shown below:

[5] http://geb.uma.es/melomics/melomics.html

[6] http://techland.time.com/2013/01/04/finally-a-computer-that-writes-contemporary-music-without-human-help/

[7] http://web.mit.edu/music21/doc/about/what.html

[8] http://people.idsia.ch/~juergen/blues/IDSIA-07-02.pdf

The baseline overall does quite well, given that all paris of chords have appeared in the training samples. However, more subtle voice-leading rules are not observed, for example, the bass part of measure 2 and 3 contains multiple downward leaps (F4 → D4 → Bb3 → Eb3), which is disallowed in chorales. A study of these 7 bars would pick up at least 11 violations of chorale rules.

The oracle, of course, would be a chorale written by an accomplished human composer than contains 0 violations of chorale rules. Any chorale by Bach or in fact, a short chorale written by an excellent student taking the AP Music exam, would contain close to 0 or even 0 mistakes.

Infrastructure

XML files for all 24 fugues in Bach's Well-Tempered Clavier Book 1 and 2 are offered as an open source project by the Open Well Tempered Clavier project[9] and is availble for download on https://musescore.com/. Music21 python package is also used to process XML files into special note objects. Using a python script, the note objects are aggregated and sorted by the voice part and they belong to and the time point they appear in the composition.

For the data representation, each fugue is divided into *time epochs* with length equal to a 1/32 musical note. Each fugue is then represented as an array of *time epochs*. Within each *time epoch* is an array of *voices*, with each element in this array of *voices* representing the *musical event* happening at that particular *voice* at the given *time epoch*. The *musical event* is either: 1) Null, meaning that no musical note occupies that *voice* at the given *time epoch*, or 2) a *NoteEpoch* object, which contains the essential information of the musical note that occupies the given *voice* and *time epoch*, such as the pitch value and duration.

[9] https://www.welltemperedclavier.org/

Approach

Part 1. Strategy

The composition of a new fugue is modeled as a markov chain: the next set of notes composed is distributed with respect to the set of k previous notes for each voice. This vague description of the markov chain model in fact summarizes the major challenge of composing a fugue, or any polyphonic (multiple voices) musical work: each voice, when viewed independently, forms a pleasing melody; a good polyphonic composition would stack up individual voices in such a way that the voices sounds harmonious together. For example, an acapella performance is a form of polyphonic music: the voices are in harmony with each other, and each voice is also a singable melody.

Thus, a markov chain that models a polyphonic composition process cannot treat each voice separately; rather than simply combining multiple independent markov chain to give multiple voices, the model must consider at the events for all voices to simulate a new set of notes.

Part 2. Model

With this understanding of polyphonic music in mind, let us refine our markov chain model:

Suppose we wish to compose a fugue of *v* voices for *N* time epochs, and our markov chain takes into account of the previous *k* notes in each voice.
Let X_i denote the set of event for all voices at some time epoch *i*, and
Let $X_{i,v}$ denote the event for voice *v* at some time epoch *i*.
For notation convenience, let us also define an event, $X_{i,v}$, to be non-null if and only if a new note is inserted at time epoch *i* at voice *v*, in which case the $X_{i,v}$ = the new note
Given our markov chain model, we see that:

$$P(X_{i,1}, \dots, X_{i,j}, \dots, X_{i,v}) \propto P(X_{e1,1}, \dots, X_{e1,j}, \dots, X_{e1,v}, \dots, X_{ek,1}, \dots, X_{ek,j}, \dots, X_{ek,v}),$$

where $e_1, \dots, e_k$ are denote the 1, ..., kth previous epochs where event is not null

Note that the above notation is incomplete: we actually do not know the value of $e_1, \dots, e_k$. The modeling would almost be impossible if we add $e_1, \dots, e_k$ to the joint distribution on the right side as well, as we will end up with too many parameters to model at the same time. Fortunately, it is reasonable to make the simplifying assumption that the duration of the next note in voice *v* is powered solely by the duration of the previous note in voice *v*, and are independent of other voices. Thus, the complete notation, with the above simplifying assumption, gives:

$$P(X_{i,1}, \dots, X_{i,j}, \dots, X_{i,v}) \propto P(X_{e1,1}, \dots, X_{e1,j}, \dots, X_{e1,v}, \dots, X_{ek,1}, \dots, X_{ek,j}, \dots, X_{ek,v} | \{e\})\, P(\{e\}),$$

where $P(\{e\}) = \prod\prod P(e_j)_l$ (product of P(e) for each j in (1,...k) for each *l* in (1,...,v)

The term $P(X_{e1,1}, \dots, X_{e1,j}, \dots, X_{e1,v}, \dots, X_{ek,1}, \dots, X_{ek,j}, \dots, X_{ek,v} | \{e\})$ is being modeled by a few set of features that reflect the stylistic conventions of fugues in the Baroque period. Thus, we replace:

$$P(X_{e1,1}, \dots, X_{e1,j}, \dots, X_{e1,v}, \dots, X_{ek,1}, \dots, X_{ek,j}, \dots, X_{ek,v} | \{e\})$$

with

$$P(\Phi(X_{e1,1}, \dots, X_{e1,j}, \dots, X_{e1,v}, \dots, X_{ek,1}, \dots, X_{ek,j}, \dots, X_{ek,v}) | \{e\}),$$ where $\Phi(\{X\})$ is the feature for set X.

Part 3. Feature Extraction

3 sets of features that are particularly important to Baroque music are extracted:

a) voice-leading features.
   Fugue grows out from counterpoint, a strict methodology that governs how each voice in a polyphonic musical text should move with respect to other voices. The fugue incorporates many counterpoint conventions but also relaxes the rules significantly. By noting these voice-leading features, the algorithm can learn what voice-leading patterns are preferred and allowed. Specifically, these voice-leading features quantifies how one note transition to the next. For example, if the current note is one semi-tone above the previous, it is called a chromatic step; a current note that is more than a perfect-fourth above the previous (equivalently, more than 6 semi-tones above) is called a leap. Essentially, one can view these voice-leading features as a way to quantize the tone difference between two neighboring notes.
   As we aggregate the voice-leading features for all voices, we can begin to understand what combination of note movement is preferred
b) interval features.
   An interval is the tone difference between two notes that are played simultaneously. Multiple intervals form chords and harmonies, and the fugue places great importance in harmonic progressions.
c) pitch features.
   Some pitches reside within a key (e.g. C major contains all the pitches that correspond to the white keys on the piano), and some do not; naturally, two neighboring pitches are more likely to reside in the same key. And observing the transition between pitches help the composition anchor to the proper notes

Part 4. 'Gibbs sampling'

In the progress report, I performed KNN clustering and use the clustered groups as the latent markov states for the model. However, the results (from the progress report) show less than ideal performance. Because there are many subtleties in voice-leading, it is more accurate to a dictionary that maps a set of previous features to the next set of features. Thus, the current model directly looks into the feature set of each *time epoch*, and constructs a dictionary that maps a feature set of the previous event to the next.

However, the space for the cross-product of voice-leading, interval, and pitch would be too large to compute. For example, for a given interval for two voices, there are >50 sets of pitches that correspond to this given interval. In practice, the algorithm leverages the idea of CSP and explores the possible feature space inspecting the most constrained feature set first.

The duration of a note is the most straightforward variable to model, since each voice is assumed to be independent. Given the desired number of *time epochs* and *voices* to be composed, a 'scaffold' that consists of note durations is sampled from the duration transition probability.

The algorithm scans through each *time epoch* after this note scaffold is constructed. The next most constrained feature set is the voice-leading features. It uses the voice-leading feature set of the previous set of notes to search for possible voice-leading feature options.

Each possible voice-leading feature set covers a set of pitchSet (one pitch for each voice) (recall that voice-leading feature is only a quantized version of tonal movement, so each voie-leading quantity implies at least one possible tonal movement). Using the previous pitchSet, each newly generated pitchSet is evaluated by its transition probability based on the interval and pitch features.

In this way, the *time epochs* are filled up epoch by epoch by 'sampling' one transition on each new *time epoch*.

Results

This simple model of sampling from transitions produces surprisingly good result. An excerpt of a composition after learning 24 Bach's fugues is presented below. One could see each individual voice has a reasonable melodic line, and the vertical harmonies actually form chord progressions common in the Baroque period. For example, the first 3 beats in measure 1 consists of the A-major chord (A, C#, E) which transitions to E-minor chord (E, G, B) on beat 4 of measure 1, which is a common five-chord to tonic-chord transition of the era.

A 4-voice composition

Another 4-voice composition is shown below. Beat 3 of the first measure is a d7 chord (D, F, A, C), which transitions to D chord in beat 1 of measure 2 (D, F, A), which then transitions to a E minor, then E major, finally a brief cadencial conclusion to A minor before setting of in C major; this kind of chord progression is again a common technique of the Baroque period, and this algorithmic composition manages to capture that.

Error Analysis

It is difficult to have an objective evaluation of the algorithm's performance. However, one good reference is to inspect the number of invalid voice-leading features it has made. Because the voices interleave with each other, it is actually possible to end up having a voice-leading feature set that is not present in our transition probability, which we learned from Bach's fugue. (imagine each voice entering one after another, so at each point the voice-leading feature consists of at most 1 voice, since only 1 voice is moving at a time; but if all 4 voices decides to stop at the same point and immediately create a new note each, suddenly we have 4 voices to be generated in the next *time epoch*, and the previous voice leading feature set is now the union of all the individual voice-leading feature for each voice; and it is possible that this new set has not been observed in the training data, even though we have only made valid voice-leading transition in the past)

In fact, using the plain transition probabilities often lead to these invalid voice-leading transitions because of the reason illustrated above. These invalid moves partially come from the fact that we are sampling from transitions in the training data, meaning that sometimes we would end up selecting certain more obscure voice-leading transitions that lead to eventual invalid moves. To correct for this, I raise the transition counts to a positive power y, so that the more popular transitions now have higher normalized probability with respect to the obscure ones. For $y = 1$, the number of invalid epochs is typically ~ 30, and the result sound less pleasing and less like a Baroque period music (in my humble opinion), such as the one below:

For this particular example, the generous amount of accidentals practically destroyed the harmony.

When the power is raised to >= 5, the number of invalid transitions decreased to 20, for a training sample of 4 of Bach's fugues.

power	1	3	5	7
# invalid transitions	29.00	30.67	24.67	20.00

Overall, as we see in the results section, the algorithm manages to produce some aesthetically pleasing works that employs stylistic conventions of the Baroque period. One flaw in the algorithm is that it is only able to compose locally: the transitions and harmonic progressions are mostly correct, but it certainly does not have the compositional flair of a human composer. This is expected as the features are centered on local transitions. Thus, it is strictly not a fugue without all the compositional developments, e.g. a single-voice melody begins, followed by a counter-melody, enters a development phase, and returns to the starting melody.

Future Work

The end of the analysis section sets up many opportunities for future work. The compositional developments mentioned in the analysis section is actually quite straight-forward: with the ability to compose phrases of arbitrary number of voices, the algorithm can naturally compose a 2-bar melody that has only 1 voice. The counter-melody simply transposes the melody, while we are quite free to explore the development section by incorporating bits and pieces from the starting melody. The ending recaps the starting melody, and can be easily inserted to the composition.

Appendix. selected compositions

Sample 1

Music21


8
11
13


15
18

Music21

Sample 3

Music21

Sample 4

Music21