Purpose

To develop a blues solo improviser over a chorus

To re-iterate from proposal 0 on the type of blues I would like to generate:

- I would like to develop a solo improvisor i.e. a melodic line
- The blues I am referring to here is the classical blues format: 12 bars per chorus with a specific chord progression (1111-4411-5511). The time signature would be 4/4
- Since the nature of blues and jazz in general has a mixture of music theory and creativity, I would like to explore means to encode both in the improviser: i.e. both the algorithmic nature of music theory(be it classical or jazz) and randomness that is necessary in any blues/jazz performance.

Goal/Phases of Implementation

The following is also a re-iteration from proposal 0 with a bit of more details:
Here are the two phases that I would do. I am currently 70% done with Phase 1.

- *Phase 1:*

The main focus on this phase is to implement the probabilistic formal grammar proposed in the paper: A grammatical approach to automatic improvisation by Keller. I would have more explanation on the paper, what I have learnt from it and why I have chosen it in sections below. The goal of this phase, is to generate music (MIDI files) of 12 bars in a stochastic fashion, where the music is structured in a way that follows music theory.

- *Phase 2*

Using a deep learning approach (LSTM) to melodic line generation. The data used for training will come from Phase 1. The reason for using phase 1 data for training, is that I have control over the structure, length and grammar that is used to produce phase 1 data. This phase would heavily reference the deep jazz project(https://github.com/jisungk/deepjazz.), but modified in the following manners:

  - Training on multiple MIDI files that contains only a single melodic line
  - No need to extract chord structure and accompaniment as that's by default already set to a specific chord progression
  - Modifying the grammar helper functions that is more suited to the underlying probabilistic grammar used in Phase 1

I think it would be interesting to compare and contrast the grammar that's learnt by the LSTM model vs what is used in Phase 1.

Background information on Phase 1

This section is dedicated to the paper I have chosen to implement in Phase 1: A grammatical approach to automatic improvisation by Keller. With a brief explanation on some music theory concepts that are relevant to the paper.

Reasons that I chose this paper and to use a probabilistic CFG:

After surveying different methods to generate music during P0, I took particular interest in using probabilistic formal grammars due to the algorithmic and stochastic natures of the application of such grammars. This embodies both the theoretical and creative process of blues improvisation. This paper seems to be promising since there is an implementation on the proposed probabilistic formal grammar, with the purpose of aiding inexperienced jazz musicians to create licks (called the Imrpo-visor). That aligns with my purpose of focusing on generating melodies over a finite number of measures. I chose this paper over other methods since most of the other methods I have surveyed relies on some sort of database. Finding and constructing data to build a customized dataset is a manual and tedious process, while the use of a simple but powerful formal grammar seems to be a much more elegant approach.

An introduction to the formal grammar proposed

The set of terminal symbols of the grammar consists of both the rhythm and tone of a note. For example, a terminal symbol in this grammar would be "A4", where "A" denotes an approach tone, while "4" denotes a quarter note. The definition of the different tones will be explained further below.

The grammar consists of two layers: there is a layer that takes in n as a parameter, where n denotes the number of beats you wish to generate. And the grammar is simply a recursive set of production rules that gives you a set of non-terminals, that are going to be expanded by the next layer of the CFG. The following is the set of production rules for the skeleton of the melody to be produced, where each number inside the square brackets are the corresponding probabilities:

1) $P(0) \rightarrow$ empty [1]
2) $P(1) \rightarrow Q1$ [1]
3) $P(2) \rightarrow Q2$ [1]
4) $P(3) \rightarrow Q2\ Q1$ [1]
5) $P(n) \rightarrow Q2\ P(n-2)$ [.25]
6) $P(n) \rightarrow Q4\ P(n-4)$ [.75]

As 4 beats denote one bar, for the purpose of generating a 12-bars blues solo, the input to this would be 48 beats. One feature in jazz and blues is syncopation: instead of stressing on the strong beats, jazz musicians would create "swing" in the rhythm by stressing on the weaker beats. This could be created by allowing half-notes to straddle the mid-point of a measure (Keller). This is incorporated in the

above structure seen in rule 5. Where Q2 denotes half a measure.

Moving onto the next layer of production rules. The first part of a terminal symbol denotes a set of tones: e.g. C denotes a chord tone (musical notes in the current chord), L denotes a color tone (which are tones that are not in the current chord, but is harmonious to the current chord), R denotes a rest. The second part of a terminal symbol denotes the duration of the note. For example 1 is a whole-note.

The probabilities in the next layer of grammar doesn't necessarily add up to 1 for a particular non-terminal. Therefore, some normalization needs to be performed on the proposed CFG.

Since the generated sequence of non-terminals simply denotes a proposed tone, and not the actual note, some post-processing needs to be done to map the terminal symbol to an appropriate note. The note itself would be selected from a set of notes that is built according to the key and chord sequence that the user specifies.

Implementation on Phase 1

- See the following git repo for the current implementation of phase 1: https://github.com/pycsham/BluesSoloGenerator

- The CFG class:
  - Consists of two fields as of now, since the non-terminal symbols aren't really used in the generation of terminal sequences, it is currently commented out.
  - The constructor function maps each rule to a set of integers within the range 1-100, according to the probabilities specified in the paper by Keller. This is so that, if I want to adjust the probabilities, the function automatically assigns appropriate integers to each rule. These numbers are used in the generation process, where we will need to randomly select a production rule to substitute.
  - The two main functions in the class are: generate and generate skeleton
  - Generate recursively applies the production rules on each non-terminal from left to right, by randomly selecting a production rule that has the non-terminal on the left hand side, according to the specified probabilities.
  - GenerateSkeleton recursively applies the production rules for generating the high level skeleton/structure of the music piece. n is the number beats. Currently defaults to 48 for a 12 bars blues.
- The Main.py file: contains the main function, that creates a CFG instance, supplying the class with the set of terminals and production rules. Some normalization was done by hand in the CFG as mentioned in the section above.

- I have decided to generate all 48 beats at once, instead of generating one segment at a time according to chord sequences, since there is a correlation between the melodic sequence in each bar. Most jazz musicians who improvise on-the-fly, creates new changes in the melodic line based on the music that has been played so far and their knowledge on the underlying chord sequence. Therefore, to imitate that process, I am going to generate all 48 beats at once, independent of the chord sequence, then map the terminals to the correct notes according to the scale and chords.

Future Steps

- I am currently exploring music21 as means to map the generated sequence of terminals into MIDI files. I have briefly went over tutorials. I plan to create another class called musicMap that uses the Notes class in music21, along with a choice of scale (currently default to E major) and the default chord sequence to map the generated terminal sequence into MIDI files. I have written some code for this part, but since it's not completed, I have yet to commit them to GitHub.
    - In terms of which note to select from the set of notes built for each type of tone(e.g. Chord tone), there are some varieties to consider: the octave in which the note resides in and the mechanism in the selection of a note.
    - For the octave, I would currently only build a set of tones that has 2 octaves, since it is unusual for a blues solo to have a large range in terms of pitch. This also simplifies the data for the next phase.
    - As for mechanism in which the tone is selected, I would try to use a random selection process.
    - There are some common keys that are used in blues, and one of the most common is E major. This will currently be the default key.
- Then I would write a script to generate MIDI files to be fed into the LSTM model.
- I would then modify the DeepJazz git repository mentioned above and play around with training the model.

Concerns

- I am not entirely sure on how to evaluate/quantify the success of phase 1, apart from a subjective evaluation of the music quality.

References

Keller, R. M., & Morrison, D. R. (2007). A grammatical approach to automatic improvisation. In
Proceedings of the Sound and Music Computing Conference, pp. 330–337.