221 P-Progress: iMGM - Improved Music Generation with Magenta

Suraj Heereguppe[a], **Nathan Dalal**[b], **Akash Mahajan**[c]
[a]ICME, Stanford University hrsuraj@stanford.edu
[b]CS, Stanford University nathanhd@stanford.edu
[c]MS&E, Stanford University, akashmjn@stanford.edu

1 Task Definition

We aim to generate melodies as well as possible. By reading melodies of particular genres, can we create melodies that make sense and sound good? We will explore algorithms to generate such sequences and will explore how to evaluate these algorithms. We start with baseline techniques like Pink Noise, expand to Markov processes, and finally plan to experiment with extending ideas from previous work using RNNs (recurrent neural networks) either for conditional generation of melodies on chords / harmonies.

2 Approach

Since music generation is very subjective, we built many simple baseline algorithms and worked on building a good evaluation function, based on a few heuristic music theory rules. Our approach during this initial phase has been to get a better understanding of different ways we can formulate our problem in terms of states (for our Markov Process) and encodings (for the RNN-based approach) through many small experiments.

We spent substantial time encoding MIDI files as states and vectors for future algorithms. For our baseline we implemented a standard baseline in music generation, pink noise[1]. We then moved to a more advanced approach of Model-based Monte Carlo as an above-baseline performer. We also implemented some standard way to evaluate performance of our music generation models, inspired by the Google Magenta Project[2].

2.1 Note Representation

We are working with MIDI files to read notes. The notes in these structured files contain information about their duration (start time and end time), their velocity (how hard the key is pressed on a piano), and their pitch (a note in a piano scale).

For our initial milestone analysis, we represent a note as a one-hot vector of pitch. Thus immediately, we disregard velocity and we disregard duration, and represent each note as a vector where all elements are zero except the note pressed, which is a one. A guide on how piano notes map to numbers can be found here (there are 128 possible values in a pitch element of a MIDI note).

[1]Douglas Eck - Talk at MUSIC 421n seminar at Stanford Oct 2017
[2]https://magenta.tensorflow.org

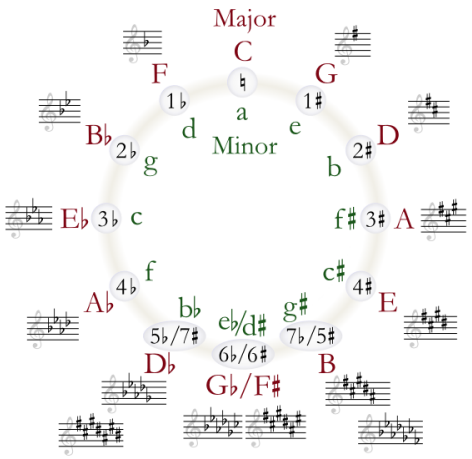

Fig 1 Circle of fifths, showing all 12 notes and related intervals

We encoded these 128-length note vectors by sampling the MIDI file to get notes and rests across tracks. A MIDI file may have multiple tracks containing multiple instruments, and what we do is merge these instruments into one track. Segmenting these notes into multiple chords (by estimating the beat length of a MIDI file), we obtain a matrix of dimension $(n_{beats}, 128)$. These are many hot vectors where each beat represents the chord at that beat (can be a many-hot vector, can be all rests as well).

For most of our baseline and basic milestone algorithms we converted the many-hot algorithms to one-hot algorithms by picking the lowest note in the chord arbitrarily. Another useful method of thinking of representing musical notes is through intervals, as visualized in the figure above, Fig 1

2.2 Evaluation Metrics

While we can subjectively evaluate tracks, we constructed an initial version of evaluations functions, that given a sequence of notes, compute some statistics related to music theory constraints. We will continue to iterate over these metrics, however for the initial experiments conducted we used the following basic metrics:

i The fraction of notes that are played 'in key' relative to the root note that was used to seed the sequence. Music typically has several different scales and variations of scales, and we used the Major scale notes (the most commonly used for this purpose)

ii The distribution of "intervals". While music consists of 12 equally spaced notes (Fig. 2.2), songs typically start in different 'keys' such as C/D etc. that related to the starting point. What is common to all music however are the relative jumps made (called intervals), and there are some commonly used 'pleasing' intervals. We generate a distribution of intervals in our generated sequences.

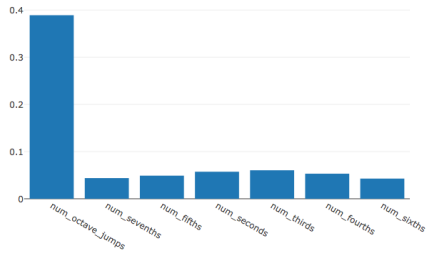

Fig 2 Distribution of intervals in Pink Noise

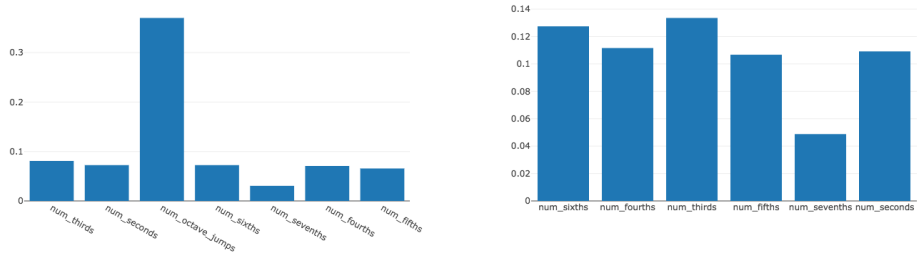

Fig 3 Distribution of intervals in Markov model generated sequences. (Left) - unrestricted sampling (Right) - restricted sampling to reduce big octave jumps

3 Data and Experiments

Our main source of data for all of our experiments is the Reddit Unique MIDI Dataset. This dataset has different genres of music. Any model may be trained either on a particular genre or a mixture of genres, making it ideal for segmenting our models by generating music from a given genre.

3.1 Baseline - Pink Noise

Pink noise samples from an inverse frequency generator. This means that the notes sampled produce a random distribution preferring notes with a lower frequency. This allows more notes to be in the low frequency ranges with interesting higher notes appearing less often. It offers some continuity in creating random music, but at its base it is still random.

3.2 Algorithm 1: Markov model-based Monte Carlo

Monte Carlo learns transition probabilities by looking at a large number of samples. We defined probabilities as a normalized count of transitions from one note to another note, a state space mapping between a note and every other note for all notes (state space = 128). We trained this type of model on a variety of genres to generate different kinds of music, for example classical, jazz, and metal. During training, the model goes over MIDI files one at a time and iteratively updates a running value of the transition probabilities. During the music generation phase the model is

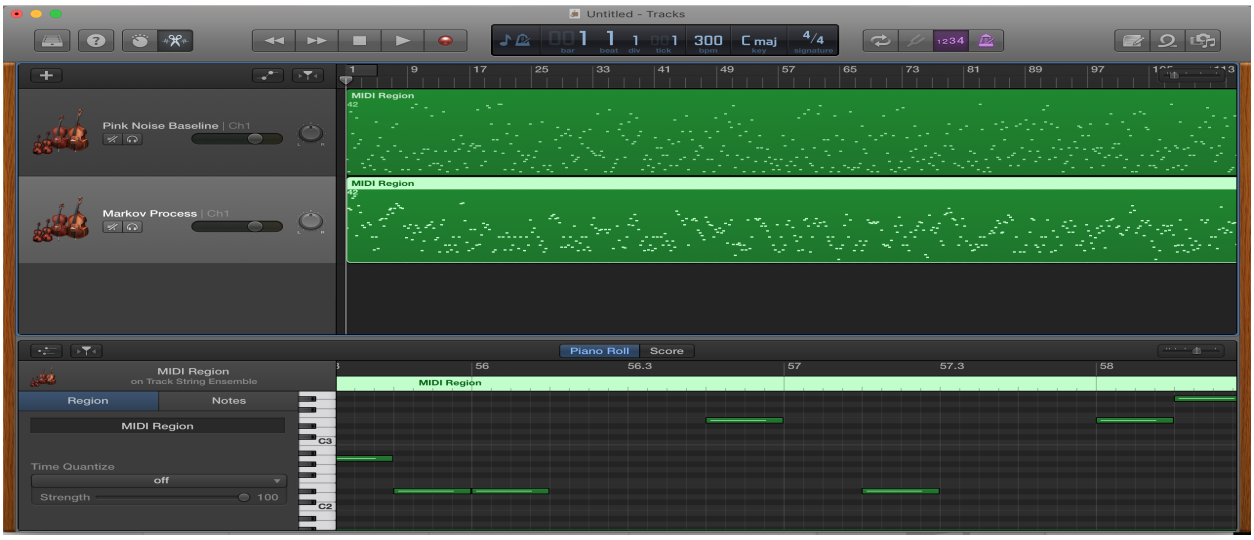

Fig 4 A sample of a subjective evaluation comparing Pink Noise (above) and Markov-generated MIDI sequences (below). We can see the structure of the intervals created by the probabilistic algorithms. Pink noise visually seems to have less 'structure' and a much larger spread of notes.

initialized to a random note in the middle octave. From here on, the generation of music is self-sustaining because the state for this model is defined as a single note.

As a slight modification/improvement to the model, we incorporated a minor rule during the generation phase. Typically, no music has note transitions that span an octave or higher. Thus, the modification made to the code during the music generation phase is that a note to be generated is constrained to be within an octave of the previous note generated. In this way, we add more structure to our music and make it more realistic, resulting in a better subjective sound as well.

4 Analysis & Future Work

Aside from a subjective evaluation of the sequences generated via Fig. 4, the evaluation metrics defined also give us an objective way differentiating between sequences. As we mentioned in our Approach section, our representation has been intentionally simplistic, modelling just pitch (without rhythm or dynamics for now). Some insights noticed:

i We notice that generated melodies modeled via a Markov process $p(s_t|s_{t-1}$ can generate phrases that are 'locally' following some structure. However generating 'phrases' and recurring 'motifs' that typically occur in music need models that have some longer-term structure, not just one-note ahead.

ii Including rhythm into our states, can be done either via discretizing our sequences, or the explicit duration of note held, which we need to explore

iii In parallel, we aim to move on to more complex models such as RNNs, where we can encode rhythm information such as holding certain notes for a longer time, introducing silence in the middle of a melody, etc. We already have the architecture for the RNN set up. We need to run rigorous tests on it like we have done with the model-based monte carlo method.