Harmonic Neural Networks: Adapting Text Generation Techniques to Generating Harmonic Sequences Using Recurrent Neural Networks

Marina Cottrell - Music 258

December 12, 2018

Abstract

Both Markov chains and Recurrent Neural Networks (RNNs) have been shown to be a useful tool in generating musical sequences. This paper will look at two tasks–generating harmonic sequences and generating harmonic accompaniment given a melody–using both Markov chains and RNNs with Long short-term memory (LSTM) units. The differences between these implementations will show the usefulness of RNNs in understanding larger-scale structure and repeated patterns. Finally, ways of improving the models will be introduced, including suggestions for chord simplification and data enhancement.

1 Introduction

Harmony and harmonic patterns play a key role in how we understand and define musical style and structure. Many music theorists and musicologists use harmony to make broader statements about a musical work or a composer or even an entire genre. As music students, we learn to identify chords and harmonic functions and how they are used in different styles of music. This focus on harmony is indicative of its importance in our experience of music.

If we can come to a clear definition of what harmonic structure is, then we can start to automatically extract this underlying structure from encoded music files, such as midi files. Automatically extracting this structure can be used for a variety of different tasks, including the identification and imitation of specific composers or musical styles.

Techniques for generating text, such as character-based text generation or automatic translation of text, have already been shown to be particular effective when translated into the realm of generating melodic sequences. Both Choi et al. and Eck have shown how LSTMs can be used in music composition [1][2]. Some previous literature also exists on generating chord sequences[3][4], but so far not much has been researched on understanding or generating larger-scale harmonic structure.

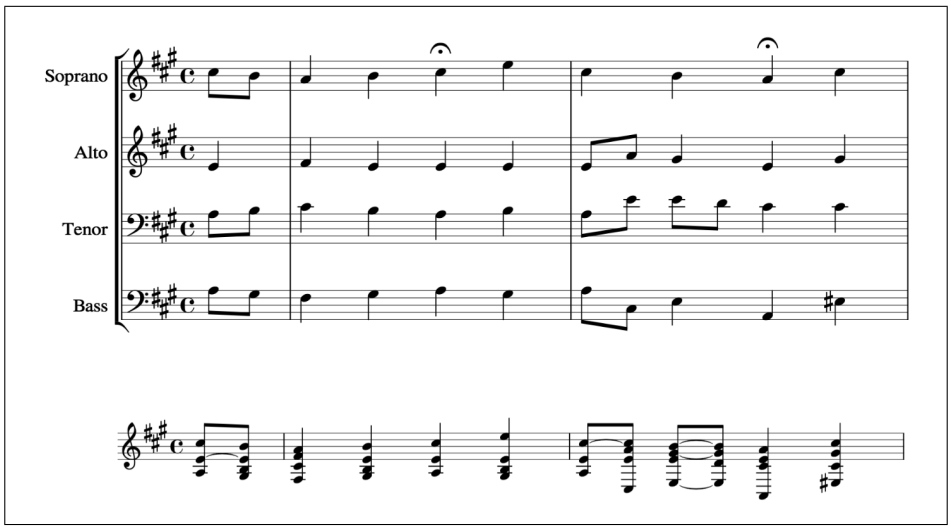

Figure 1: Example of "chordify" function input and output.

In this paper, I investigate some of the techniques for generating harmony, both independently and as accompaniment to a given melody. The purpose of this will be to understand what techniques are available and necessary for this task, and to see how moving from the surface-layer chords to an underlying harmonic structure can be achieved.

2 Methods

2.1 Data

For this project, I used all 370 Bach Chorales, available from the KernScores website, in the midi format as training data. Chords were parsed from the midi files using the music21 python framework and specifically the "chordify" function.

"Chordify" collapses all simultaneously sounding notes into single chords (See Figure 1). Notes that are being held over from a previous time step will be re-articulated, which, though somewhat inaccurate compared to what the composer wrote, preserves what we actually hear in its notation.

Once the pieces are processed by "chordify," the resulting chords were then translated into strings of different formats, depending on the use case. Generally, all chords were represented as a string of numbers, separated by spaces. These notes could either represent the midi value or the scale degree number of each of the notes in the chord. For a C major chord, for example, the string representation could look something like this: "48 52 55 60" in midi numbers, or "0 4 7" in scale degrees. Depending on the purpose, these strings can sometimes also contain tempo information added on to the end, represented as the

number of quarter notes in length (e.g. an eighth note would be represented as 0.5, because it is half the length of a quarter note). The C major chord with tempo information could then look something like this: "48 52 55 60 0.5" for an eighth note in length.

The final method I used to represent chords was using repeated sixteenth notes. Sixteenth notes are the smallest note length in any of the Bach chorales, so dividing notes into repeating sixteenth notes ensures that all the parts of the piece have the equal number of notes, allowing them to be lined up perfectly. The repeated sixteenth note chords were then given additional values to identify if they were the beginning of a chord (the onset), a continuation, or the end (the offset). These were represented with the strings "b" for beginning, "c" for continuation, and "e" for ending. A quarter note long C major chord would then be represented like this: "48 52 55 60 b", "48 52 55 60 c", "48 52 55 60 c", "48 52 55 60 e". This technique was especially useful in generating chordal accompaniment to a melody, because it meant that the chords could change at different times than the melody notes, while retaining the 1-to-1 mapping.

2.2 Generating Harmonic Sequences

2.2.1 Markov Chains

Using Markov chains is an easy way to probabilistically generate sequences, with each element depending on the n preceding elements. In this case, I decided to use 2nd-order Markov chains, which means each element depends on the 2 preceding elements. This was implemented in python, using dictionaries, where each key in the dictionary was a string that contained both chord elements, separated by a forward slash, and the value was a list of all the possible next chords, repeated depending on how likely they are to occur. Although a list makes more sense for storing two chords, lists cannot be used as keys in a python dictionary. One entry in the dictionary could look something like this: {"48 52 55 60 / 48 53 57 60": ["48 52 55 60", "48 52 55 60"]}.

For using Markov chains, I decided to represent my data as midi numbers without duration information, as in the example above. I also parsed the Bach chorales to exclude any chords on the offbeats. This meant I was only looking at chords on each quarter note beat of the chorale. This was done because Markov chains are generally not very good and would be able to learn sequences better if there is less information involved. Being too precise with timing information could lead to very strange and bad metrical decisions, and potentially even completely remove any kind of meter from the output.

As well as this particular representation of chords, I also included special begin and end tokens to show the generator where to start and stop. This meant that I always started a sequence by randomly picking among the possible keys with the word "begin" in them, and then looped to pick a next chord until the next picked chord was "end".

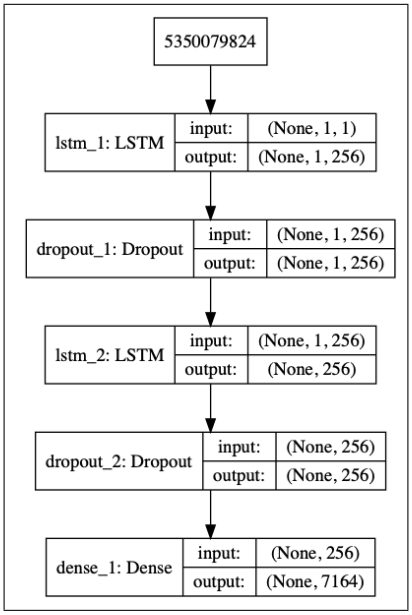

Figure 2: Plot of LSTM Generative Model.

2.2.2 RNN

LSTMs (Long short-term memory) are type of unit found in RNNs (Recurrent Neural Networks). RNNs are a structure primarily used in the prediction and classification of time series data, such as text and speech recognition. Although they do retain memory of past events, LSTMs were specifically designed to have a longer-term memory, and are therefore ideal for sequence generation in which there are longer lasting structures and patterns.

I based my RNN model on a couple of different models originally used for text generation. Brownlee's blog post[5] originally used a similar model to generate text character by character, trained on "Alice in Wonderland" and Campion's model generated text word by word, trained on his own book, a French fantasy novel. Both character-based and word-based approaches are common for text generation, and in my case I used chord-based generation.

Figure 2 shows a plot of the model's architecture. Each larger rectangle represents one layer in the neural network and the numbers represent the input and output shapes (how many features are being looked at in each layer). I use two LSTM layers, as each added layer adds further abstraction of the input over time. This simply means that larger-scale structures will be learned better. After each LSTM layer, I used a dropout layer. Dropout layers are ways of improving your accuracy by randomly removing training data. This makes the neural network more generalizable because it doesn't learn to rely on specific information.

For consistency between the LSTM and the Markov implementations, I also

trained this model on downbeat chords without including duration information. However, because I wanted to be able to control the length of the output sequences manually, I did not include "begin" and "end" tokens this time. Although this meant that I could output anywhere from 4 to 100 chords, it also meant that I had to generate one chord at a time. The model still retains a memory of what it has generated thus far, but because I had to control when to stop, I was unable to judge whether or not the model had learned how and when to end a chord sequence.

2.3 Generating Chordal Accompaniment to a Melody

2.3.1 Markov Chains

For the Markov chain implementation of accompaniment generation, I will refer to a project that I did two years ago. In this implementation, I used a Markov chain that encompassed the time-step preceding the chord I was trying to predict, which included three musical elements: the previous melody note, the previous chord, and the current melody, for which I was trying to predict a chord.

This implementation assumes a strict one-to-one relationship between melody note and accompanying chord, which means that for every new melody note, the chord will also change. Since this isn't how music generally works, I also filtered the input melody to look at only the downbeats, before passing it through the Markov chain generator. This made the output a little bit more natural, but it was unfortunate that it took so much preprocessing, and shows one of the Markov chain's huge disadvantages in this situation.

I also included a "begin" token, which meant the very first entry into the Markov dictionary contained two begin tokens, one for the previous note and one for the previous chord: "begin / begin / 72":["48 52 55 60"]. Unlike in the chord generation task above, starting a new sequence wasn't done by randomly choosing a key, but by creating one with the first note in the input melody. Although this makes the output accompaniment less random, it also means that every note in the given melody has to have existed in one of the training pieces. More specifically, any sequence of two notes in the input melody have to also exist somewhere in the training melodies. With 370 pieces to train on, the chances of this happening are fairly high, but it does mean that styles of melodies that are too different from Bach (e.g. 12-tone or atonal melodies) have a worse chance of working at all. Because the Markov chain's output depends so strongly on the melodic input, I didn't need to include an "end" token in this implementation, because the output would naturally end when it ran out of melody notes to use.

2.3.2 RNN

For this RNN model, I implemented a simple sequence tagging model, in which the input melody notes are considered the sequence, and the chords are the

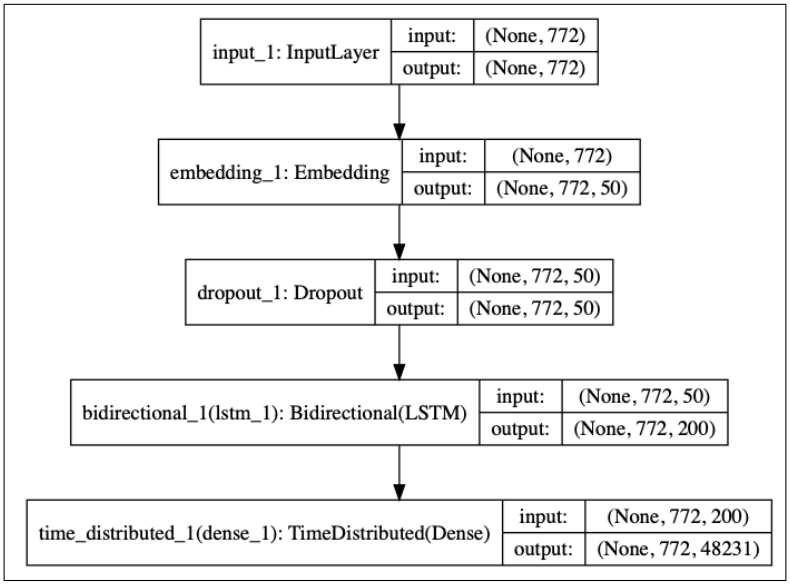

Figure 3: Plot of LSTM Labelling Model.

labels. Sequence tagging is often used in text as well, especially for labelling parts of the sentence (such as "noun", "verb" or even "location" or "person"). Some words always have the same label, but most of the time, labels depend on the overall context of the word in sentence. Although calling chords a "label" to a note is not very intuitive, the overall architecture does make sense. Just as with text, in music, although some chords go better with some notes, you can't accompany a note in isolation and need to be aware of the preceding harmonies.

To avoid the amount of pre-processing I needed to make the Markov chain implementation work, especially with regards to rhythm, I decided to train the RNN model on repeated 16th-note chords. This chord formatting meant that there was a one-to-one mapping of note to chord–required for sequence tagging. However, since both notes and chords contained timing information in the form of "beginning", "continuation", and "end" tokens, they could act completely independently of each other. While in the Markov implementation I opted to only look at the melody downbeats, using repeated 16th-notes meant that I could retain the full original melody, while still training it to learn the accompaniment correctly.

3 Results

3.1 Generating Harmonic Sequences

The differences between the Markov-generated harmonic sequences and those generated by the Neural Network are surprisingly subtle. For the most part, the

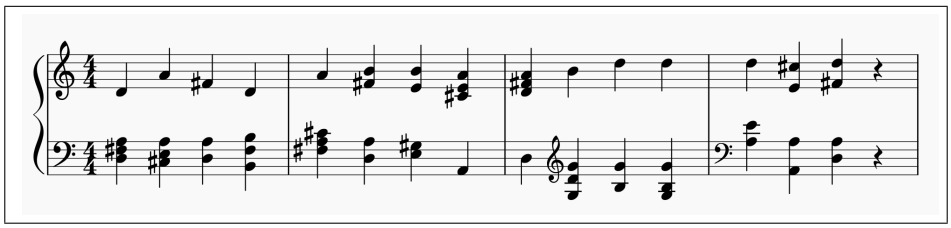

Figure 4: Example of a Markov chain-generated chord sequence.

Markov-generated sequences sounded pretty good. They seemed to make a lot of sense, especially on the local scale. Part of their success, however, stemmed from the fact that they were very simple–they tended to alternate between the tonic and another chord, but rarely stray much farther. As a whole, the sequences sounded more like ear-training examples taught to a beginner theory class, rather than musical pieces. However, when they did move away from their original tonic, they almost invariably modulated completely away and ended in a completely different key.

Figure 4[1] is one of the most successful of the Markov-chain generated chord sequences. While this one managed to begin and end in the same key, most of the others didn't do that. I chose this one however, because it demonstrates very clearly how tonic-focused the sequence is; aside from in measure 3, almost every third chord is the tonic chord. I think this highly uniform harmonic motion comes from the fact that motion to and from the tonic chord is statistically the most likely, even though within a piece as a whole, there tends to be more variation. It could also be that of the Bach chorales, the ones in D major tend to have less variation, meaning the Markov generator has fewer options to choose from and a higher chance of picking the simpler chord progressions.

The LSTM-generated sequences on the other hand generally contained more variation in harmonic motion away from the tonic, but without actually modulating out of the key. Figure 5[2] shows this really well in measures 9-12: the harmonic sequence in measures 9 and 10 repeats almost exactly down a half-step in measures 11 and 12. This is indicative of the Neural Network being able to learn larger-scale patterns like 2-bar repeating patterns, while still being able to stay in the tonic key.

3.2 Generating Chordal Accompaniment to a Melody

When it came to accompanying melodies, RNNs far out performed the Markov chain. It was much clearer here that the Markov chain-generated accompaniments did not have any understanding of longer harmonic patterns or key. Generally, each chord was consonant with the melody note it was accompanying,

[1] Listen: `https://ccrma.stanford.edu/~marina/audio_files/markov_generated_chords.wav`

[2] Listen: `https://ccrma.stanford.edu/~marina/audio_files/rnn_generated_chords.wav`

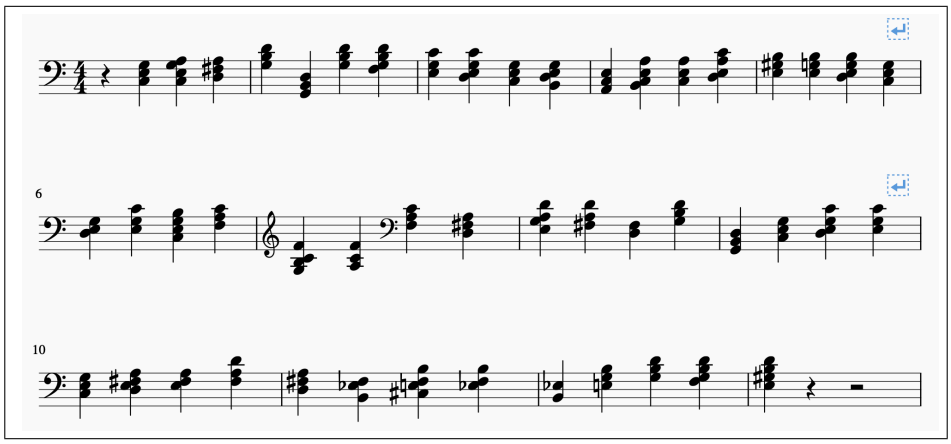

Figure 5: Example of an RNN-generated chord sequence.

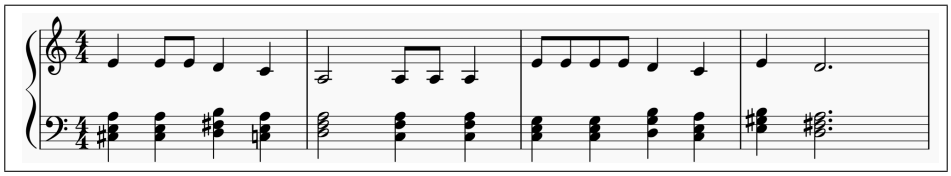

Figure 6: Example of an Markov chain-generated accompaniment.

and most pairs of chords made sense together, there was clearly no over-arching tonal structure. For the most part, it was also unable to find the correct key of the melody, meaning although the chords were "correct" in that they were consonant with their notes, it generally didn't feel correct.

In Figure 6[3], this melody was supposed to be in A minor, but the Markov chain harmonized the first chord with an A major chord, meaning it had to course-correct on the C natural at the end of the first bar. These problems all make sense, since this Markov chain had especially little context–it was only looking back one time-step.

The RNN implementation, on the other hand, always harmonized in the correct key, from beginning to end. Although there were strange modulations in some of the accompaniments, they never drifted too far and always ended back on the tonic. It also seemed to understand other cadential points in the melodies, signalling that it was beginning to understand larger structural elements to the music, rather than just the chord-to-chord relationships.

Figure 7[4] is the first two measures of "Ode to Joy" with RNN-generated accompaniment. The accompaniment manages to hit the key moments of the

[3]Listen: `https://ccrma.stanford.edu/~marina/audio_files/markov_generated_accompaniment.wav`

[4]Listen: `https://ccrma.stanford.edu/~marina/audio_files/rnn_generated_accompaniment.wav`

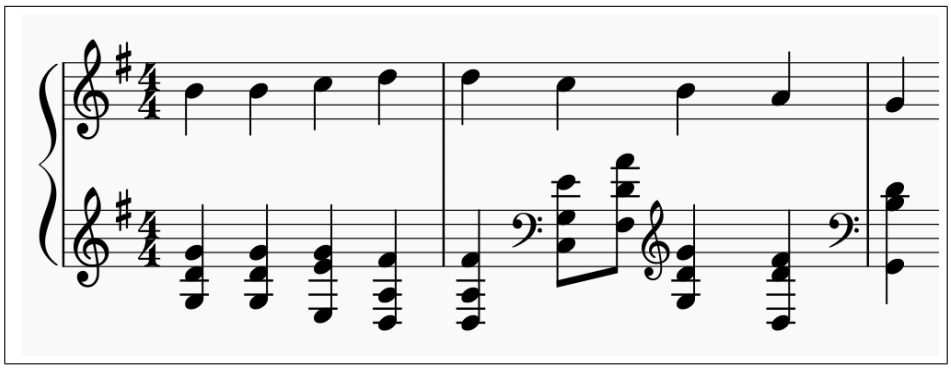

Figure 7: Beginning of "Ode to Joy" with RNN-generated accompaniment.

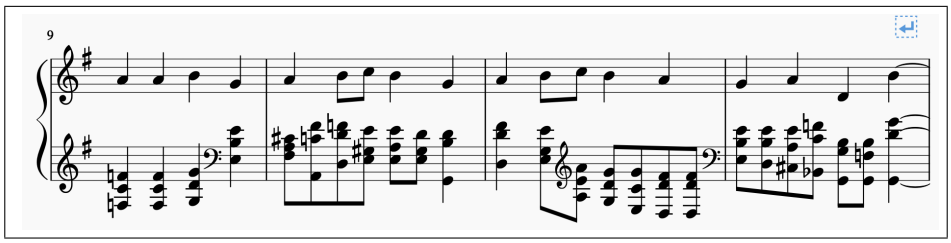

Figure 8: Middle of "Ode to Joy" with RNN-generated accompaniment.

original harmonization: the I chord at the beginning, the V chord at the beginning of the second measure, and the V-I motion into the third measure. The rest of the accompaniment varies somewhat in accuracy, but there are still key structural moments that the Neural Network manages to correctly identify.

Figure 8 shows measures 9-12 of the "Ode to Joy" harmonization. As you can see, there is a lot more variation going on in this part of the piece, and it strays a lot farther from the original key. Despite this, the upbeat to and the first beat of measure 11 are harmonized as expected with a V-I in the tonic. The last note in measure 12 also shows that the accompaniment managed to land on the correct harmony. This seems to indicate that, although there is room for improvement in some of the middle parts of the accompaniment, the Neural Network is starting to understand the over-arching structure of pieces and how to integrate harmony into that structure.

4 Discussion

The Neural Network far out-performed the Markov chains when it came to generating accompaniments. Although the difference between the two was not as big when it came to generating chord sequences, the larger-scale repetitions that the RNN managed to create are a sign that it had a deeper understanding of the

music than the Markov chain. While both RNNs seemed to have a good understanding of how to generate large-scale structure, they both struggled somewhat with the details of the chords in between those structural moments. Because I am interested in understanding and extracting the underlying harmonic structure of music, there are steps I could take here to ensure that the networks recognize more of the structural harmonies and don't worry as much about the chords in between.

Because I used one-hot encoding to encode the chords, the full output of my Neural networks was actually vectors of the probabilities of each chord at each point in time. The resulting chord chosen was always the chord with the highest probability. However, in non-structural moments (off-beats, non-cadential points, etc), the predicted probabilities were a lot lower than in key structural moments (beginning and end of the piece, cadences, etc). The fact that the RNN's accompaniment of "Ode to Joy" seemed to recognize cadential points and accurately harmonize them shows that those points probably had higher probabilities associated with them. Especially when compared to–a lot of them seem like strange and unusual choices, which signals some amount of randomness.

This could be used to our advantage: if you only output chords that have a predicted probability of over a certain threshold, it's likely that the output will be only the structural moments. In between, if none of the output probabilities are high enough, the output chords could be rests or "undefined" as placeholders.

Another option is to simplify the chords before training the Neural nets. Simple techniques, such as removing passing tones or other non-chord tones, choosing just one chord per beat (but not necessarily always the first chord in each beat), and writing all chords in the same way (e.g. root position) could help to reduce the number of unique chords the Neural net is trying to learn. This would lead to better results with less training, but also a more bare-bones harmonic sequence, more representative of the underlying harmonic structure.

For either of these options (probability threshold or simplifying chords), the output would be more sparse than the output above. The resulting sequences would have to act as a structural placeholder and probably be run through a second Neural net to get any kind of meaningful music out of it. Splitting up the work in this way means more training of Networks and more data processing, but would probably result in cleaner musical outputs, and would allow us to train different Networks on different kinds of music. For example, the bare-bones harmonic structure could be trained on Bach chorales, while the fleshed out melodic Neural network could be trained on Chopin. This would result in being able to combine different elements of different composers or styles of music in new ways.

While the Markov chain implementations probably can't be improved on very much, there are a number of things that can be done to improve the RNN implementations. One of the key aspects that makes Neural Networks perform well is having a lot of data. Although I am somewhat limited in that aspect because there are only so many Bach chorales, and I used all of them already, there are some techniques I can use to artificially increase my data set. One

option is to transpose all the Bach chorales into all 12 keys. This would allow me to have 4440 pieces to train on, rather than just 370. It would also balance out the learning across all keys. Right now, it is very likely that my Neural Network has learned certain keys better than others. If, hypothetically, 80% of Bach's chorales are in D major, then the RNN would have been trained on 296 pieces, while all the other keys would have been trained on only 6 pieces each. Transposing all pieces would allow all keys to be equally represented and improve the output.

Training the RNNs for longer would also allow for better results. All the Neural Nets here have been trained for 300 epochs, which in most cases took my computer all night. Because I needed a functioning computer the rest of the day, I couldn't run them longer. Similarly, I was not able to transpose all of my pieces because my computer couldn't run the networks with that much data. Moving forward with developing these networks, I will set up a GPU instance on AWS so that I can run everything faster and with all the data I need. This will also allow me to iterate more quickly and adjust and improve the architecture of the model as I work.

5 Conclusion

Both generating harmonic sequences as well as generating harmonic accompaniment had interesting results and showed a lot of promise. Although the Markov chain output was sometimes pretty good, there was a clear difference in the larger-scale structural understanding between the Markov chain and the RNN implementations for the same task. Using LSTMs seemed to be the right approach for both tasks, as they were able to not only stay in the right key throughout each generated sequence, but also generate large-scale repetitions over multiple bars.

Moving forward, I would like to experiment with using simplified chords as training data as well as looking at only predicted chords above a given probability threshold. Some combination of these approaches should give us a more skeletal harmonic structure to work with. Using this structure, new Neural networks can then be trained to re-introduce melodic and rhythmic patterns in various styles to create completely new pieces of music or even classify composers or styles of music.

References

[1] K. Choi, G. Fazekas, M. Sandler. "Text-based LSTM networks for Automatic Music Composition," *Conference on Computer Simulation of Musical Creativity*, 2016.

[2] D. Eck. "A First Look at Music Composition using LSTM Recurrent Neural Networks," 2007.

[3] G. Brunner, Y. Wang, R. Wattenhofer, J. Wiesendanger. "JamBot: Music Theory Aware Chord Based Generation of Polyphonic Music with LSTMs," *IEEE 29th International Conference on Tools with Artificial Intelligence*, 2017.

[4] H. Lim, S. Rhyu, K. Lee. "Chord Generation from Symbolic Melody Using BLSTM Networks," *International Society of Music Information Retrieval*, 2017.

[5] J. Brownlee. "Text Generation With LSTM Recurrent Neural Networks in Python with Keras," *Machine Learning Mastery*, 2016.

[6] D. Campion. "Text Generation using Bidirectional LSTM and Doc2Vec models 1/3," *Medium*, 2018.

[7] I. Sutskever, J. Martens, G. E. Hinton. "Generating Text with Recurrent Neural Networks," *International Conference on Machine Learning*, 2011.

[8] T. OBrien, I. Roman "A Recurrent Neural Network for Musical Structure Processing and Expectation".

[9] C. A. Huang, T. Cooijmans, A. Roberts, A. Courville, D. Eck. "Counterpoint by Convolution," *International Society of Music Information Retrieval*, 2017.