David Wang

Dr. Zeynep Özcan

ENGR 430

29 April 2021

Capstone Final Report:

Generating Pop Songs through Machine Learning and Algorithms

1. Introduction

My interest in both music and computer science inspired this capstone project. After listening to so many pop songs, I noticed a set of shared characteristics between them. These characteristics include having a clearly defined chord progression or having repetition. It's as if an algorithm could generate them. Therefore, I decided to explore how machine learning and hard-coded algorithms can recreate the vocal melody of pop songs.

There were two goals I had for this project. First, explore how Google Magenta's models could generate the vocal melody of pop songs. Magenta is a library filled with machine learning models that specialize in music generation. After reading Qibin Lou's report [1] and Sam Agnew's blog post [2], I was inspired by their work to investigate Magenta's potentials. I aimed to train Magenta's models with my pop song MIDI files. If successful, it may learn how to repeat and continue melodies logically. My second goal was to look for alternative ways to generate pop songs melodies if the first approach wasn't successful. It eventually evolved into the second half of my project, where I wrote a Python script with hard-coded algorithms for generating melodies. I would discuss both parts of the project extensively in this report.

2. Project Questions

In my original project proposal, I listed five questions that I aimed to answer by the end of this project. This section would provide brief answers to these questions. It could serve as a preview for the upcoming sections of this report.

Question 1: How does Magenta's melody continuation feature work?

Answer: Magenta uses a recurrent neural network (RNN) to continue melodies. RNN is a type of artificial neural network commonly used to solve ordinal or temporal problems such as natural language processing [3]. Magenta uses a special RNN called Melody RNN to predict and

generate notes [4]. The two models I'm exploring in this project, Attention RNN and Lookback RNN, are Melody RNN under different configurations.

Question 2: Does training Magenta's model with pop song MIDI files help improve its performance?

There is both a yes and no answer to this question. One key takeaway is that Magenta's output takes on characteristics of its training MIDI files. When I trained Magenta with a dataset filled with Nintendo video game music, it outputted some rapid 16th notes. Theoretically, Magenta's output will take on the characteristics of pop song vocal lines when trained on them.

As for the no part of my answer, Magenta cannot learn the repetitive quality of pop songs. Due to a structural problem with Magenta, it is unable to reuse the bars it previously generated. Hence, there's barely any repetition in the output. If we look at 1 or 2 bars, it might look similar to a pop song melody. Yet, if we assess the entire output, it looks nothing like a pop song.

Question 3: What structural problem with Magenta's model may contribute to its inability to produce pop song melodies?

As it outputs each note, Magenta's model only looks at the notes one and two bars before the note's position. It does make repetition easier to occur. However, the 2 bar limit also stops the model from repeating melodies it hasn't seen in two bars. This goes against the structure of pop songs, where multiple sections are repeated throughout the piece.

Question 4: How could I modify one of Magenta's models to fit my needs?

There is a chance that the model could learn the concept of repeating entire bars if we increase the number of notes it looks back on. However, Magenta's strength lies in producing logical continuations of a melody. Repetition, on the other hand, could be achieved by simple Python scripts. Instead of modifying the model, it would be easier to hard-code a Python script that fits my needs.

Question 5: Are there any solutions besides modifying Magenta's assets?

For the second half of this project, I hard-coded an algorithm for generating pop song melodies. It overcame Magenta's shortcomings in repeating melodies. However, the algorithm's output was still far from recreating the style of actual pop songs.

3. Methods and Results:

In this project, I took two approaches to generate music. The approach was machine learning, while the second one was hard-coded algorithms. In this section, I'll discuss the steps for each approach and as well as the results.

3.1 Methods: Machine Learning

3.1.1 Finding Suitable Models

My goal for this approach was to explore if I could train Magenta's models to fit my needs of generating pop songs. I started by first investigating which models from Magenta to use and how to use them. A Magenta blog post titled "Generating Long-Term Structure in Songs and Stories" served as the starting point for my research [5]. The blog post described two recurrent neural networks developed by Magenta, Lookback RNN and Attention RNN.

Lookback RNN has two custom labels: repeating an event from 1 bar ago and repeating an event from 2 bars ago [5]. With these custom labels, each event Magenta outputs could be a direct copy from 1 or 2 bars ago. Repetition is a huge part of most musical pieces, not just pop songs. Theoretically, Lookback RNN's custom labels should make its output sound more musically logical. The second model, Attention RNN, is also structured to generate musically logical melodies. It uses a unique "attention mechanism" to look back on previous steps. According to Magenta, this makes the model easier at learning long-term dependencies [4]. Since Magenta provided reasonable explanations for their music-generating capabilities, I decided to pick these two models for training.

3.1.2 Collecting MIDI Files

A model could not be trained without MIDI files. A MIDI file contains information on all the events in a musical piece. These include the placement and pitch of each note. As mentioned before, Magenta could convert MIDI files into note sequences to train its models. The problem lies in how to collect these MIDI files.

The first dataset I found was a 1GB MIDI dataset online [6]. It contains over 130,000 MIDI files, many of which are pop songs. After downloading the dataset, I browsed through MIDI files and located several pop songs. For each file, I identified the track for vocal melody and saved it as a separate MIDI file. Due to Magenta's configurations, it couldn't process tracks that I directly

saved. To overcome this issue, I had to copy and paste all the notes of an existing track into a new track. I ended up collecting 67 vocal tracks for my training set [7].

In addition to the vocal tracks, I also found a second MIDI dataset from Sam Agnew's blog post. It contained 2230 MIDI files of Nintendo video game music [2]. I used them to test if Magenta's models could fully adapt to the style of its training files when given a large training dataset.

3.1.2 Training Magenta's Models

Magenta's models are fairly easy to find and interact with. Lookback and Attention RNN could be found on Magenta's GitHub repo under Melody RNN [4]. The two models could be accessed under different configurations of Melody RNN. To train the model, first, convert the MIDI dataset into note sequences. Next, split them into the training and test set. Then, train the model with these two sets. Finally, save the last weights as a bundle file. These steps could be accomplished through terminal commands. By the end of this project, I have trained both Lookback and Attention RNN with my two datasets.

3.1.3 Generating Melodies

By feeding an input melody and a bundle file to Magenta, it would generate a continuation of the input melody according to the weights inside the bundle file. Magenta provides default weights for its models on its GitHub repository [4]. Combined with the weights I obtained by training Lookback and Attention RNN, I was able to generate melodies with these six bundle files.

1. Magenta's default weights for Lookback RNN [4]
2. Magenta's default weights for Attention RNN [4]
3. Weights from training Lookback RNN with pop song vocal tracks [8]
4. Weights from training Attention RNN with pop song vocal tracks [9]
5. Weights from training Lookback RNN with Nintendo video game music [10]
6. Weights from training Attention RNN with Nintendo video game music [11]

3.2 Results: Machine Learning

I generated multiple MIDI files using the 6 bundles I mentioned above and two input melodies. The first input was the first four notes of Twinkle Twinkle Little Star (Fig.1) [12]. The second input was the first two bars of Maroon 5's Payphone (Fig.2) [13]. In this section, I would describe how each variable, such as the input melody, training dataset, and model type, would affect the model's output. I would also explain why Magenta's models may not be ideal for generating an entire pop song.

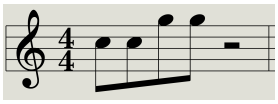


Fig.1 Input 1

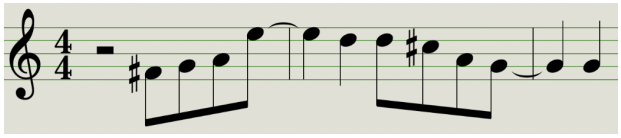


Fig.2 Input 2

3.2.1 Input 1 v.s. Input 2

My first observation was that Magenta's output would reflect the complexity of its input melody. From the figures above, we see that input 2 had a wider pitch range (Fig.1, Fig.2).

Using the weights from training Attention RNN with pop song vocal tracks, I put both inputs through an Attention RNN and generated a continuation for each. Output 1 was Input 1's continuation and Output 2 was Input 2's continuation. Output 1's use of note still seemed very restricted (Fig.3) [14]. It mainly used C and G from input 1 or their surrounding pitches. Output 2's use also used pitches close to those from input 1. However, since input 2 had a wider pitch range, so did output 2 (Fig.4) [15]. This comparison showed that the input melody's content would affect the content of Magenta's output.

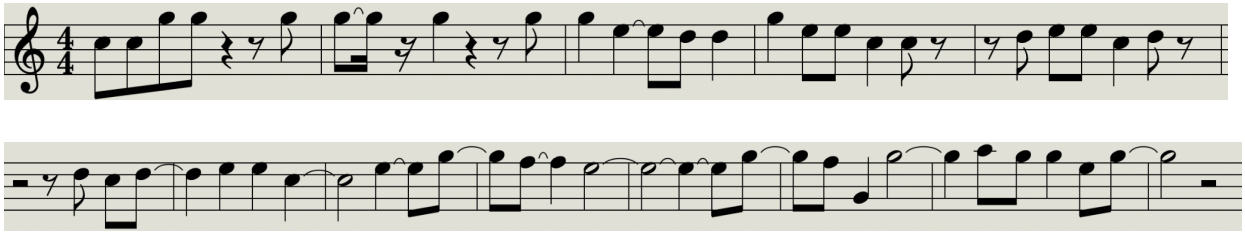


Fig.3 Output 1

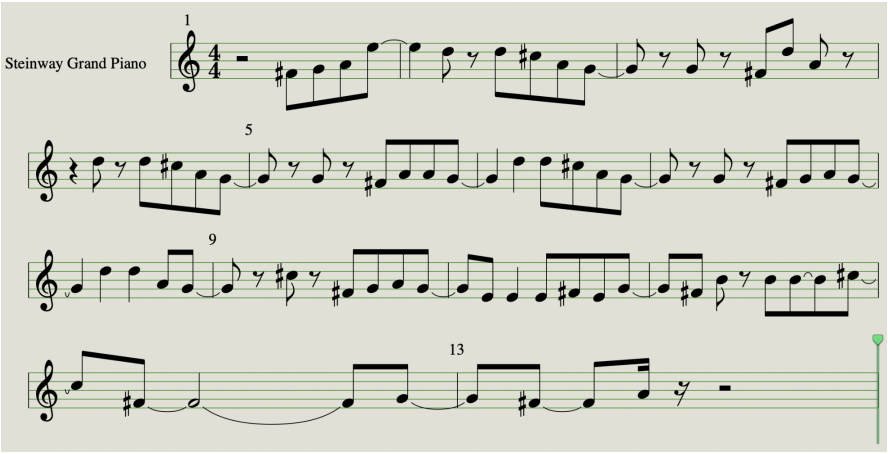


Fig.4 Output 2

3.2.2 Pop Song Vocal Tracks v.s Nintendo Video Game Music

My second observation was that the characteristics of the training dataset would affect the model's output. Output 3 was generated using the weights obtained by training an Attention RNN with the Nintendo video game music dataset (Fig.5) [16]. Similar to output 2, it used input 2 as its input melody. However, several 16th notes appeared in Output 3, which were not present in Output 2. Although not common in pop songs, 16th notes do appear in video games during intense battles.

Aside from rapid 16th notes, I found another characteristic of the training dataset affecting the model. Output 4 was generated using weights obtained by training a Lookback RNN with the Nintendo dataset (Fig.6) [17]. Starting from bar 5, the model produced the same rhythm for each bar, consisting of 4 eighth notes. This consistent rhythm likely belonged to the accompaniment for a piece. In the video game music dataset, the main melody and accompaniment were separate tracks. It was likely that Lookback RNN trained itself on an accompaniment track and learned its format.

From these two examples, we learned that both Attention and Lookback RNN generated music with characteristics from the video game music dataset. Hence, it proved that the training files do play a part in affecting the model's output.

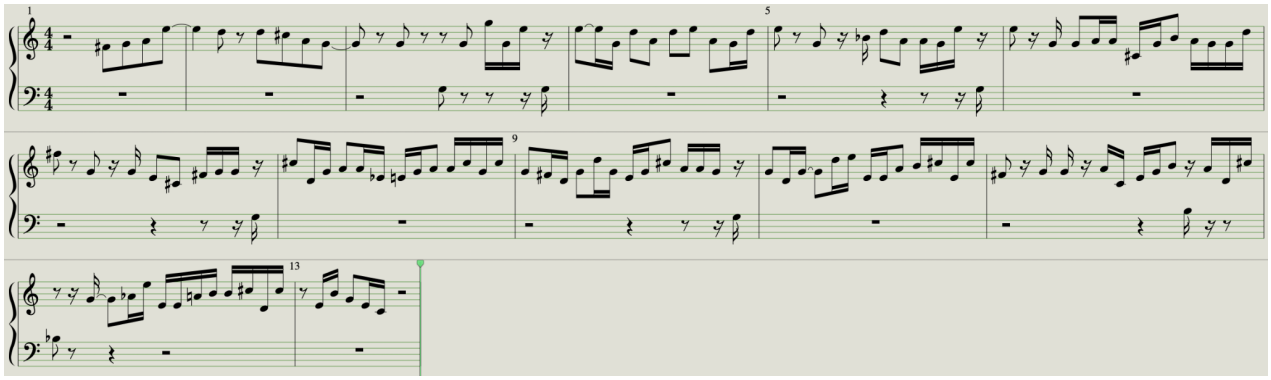


Fig.5 Output 3

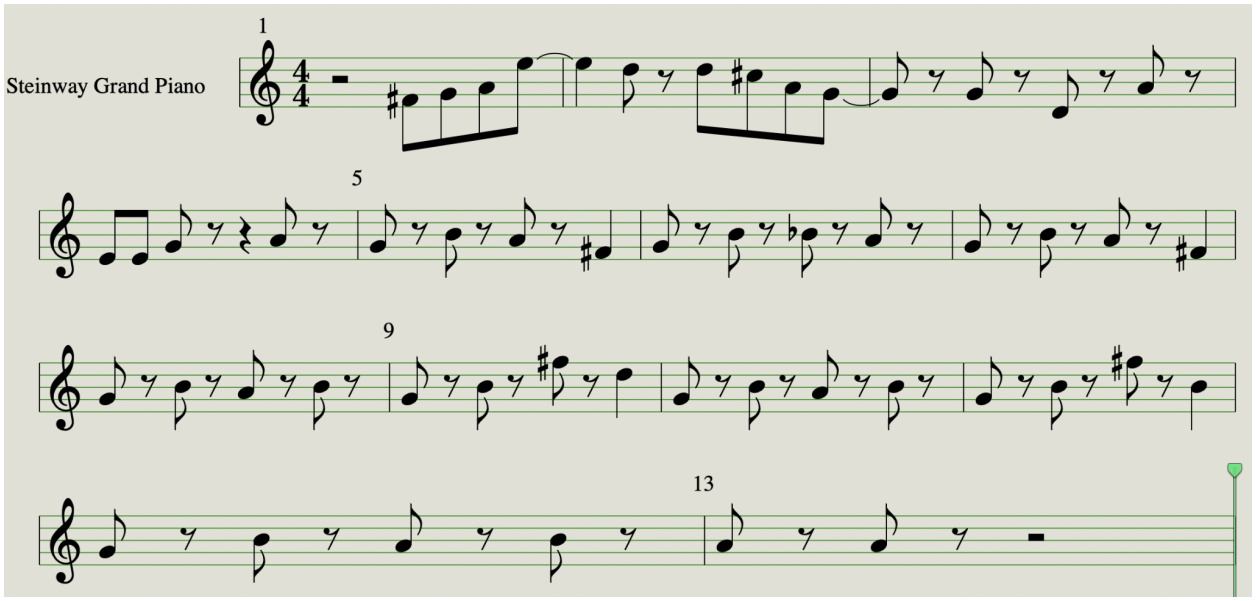


Fig.6 Output 4

3.2.3 Lookback RNN v.s. Attention RNN

Magenta noted that Attention RNN could learn longer-term dependencies more easily. It would result in Attention RNN generating melodies that have long arching themes []. I found one pair of outputs that agreed with their statement.

Both Output 5 and Output 6 were generated using Input 1 as their input melody. However, Output 5 was generated by Attention RNN (Fig.7) [18]. Output 6 was generated by Lookback RNN (Fig.8) [19]. In Output 5, the same rhythm was used from bar 1 through bar 11. The first four notes of each bar even followed an ascending pattern. On the other hand, Output 6 switched

rhythm every 2 bars. Since Attention RNN's held a constant rhythm for more bars than Lookback RNN did, it showed that Attention RNN was better at generating long arching themes.

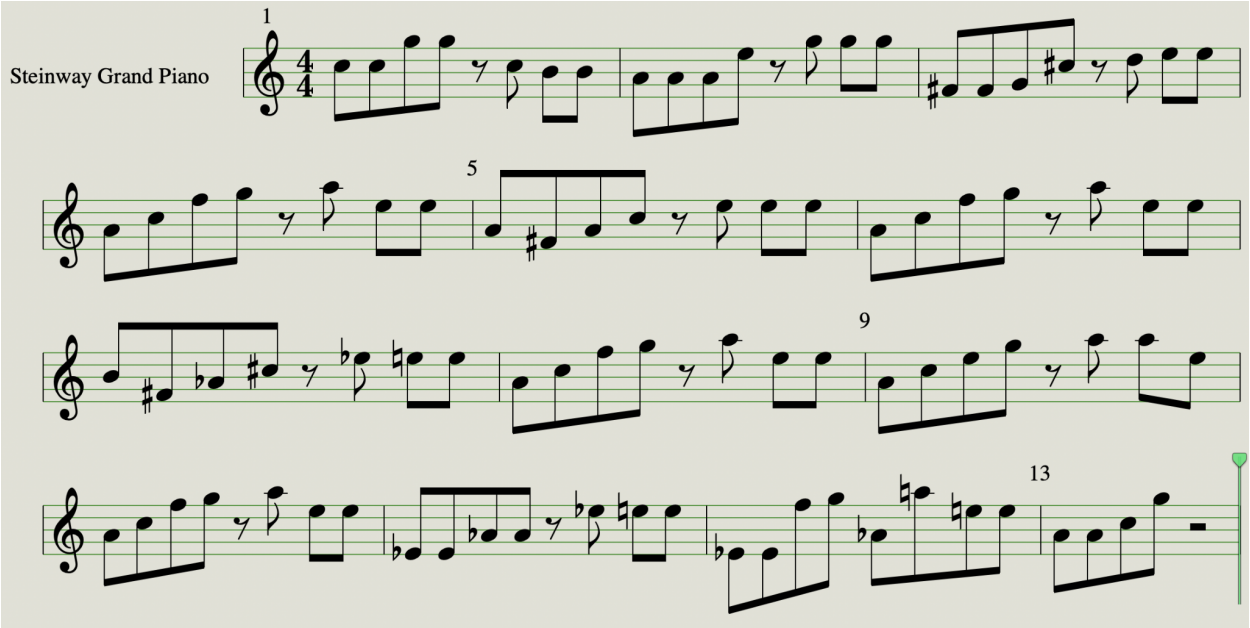


Fig.7 Output 5

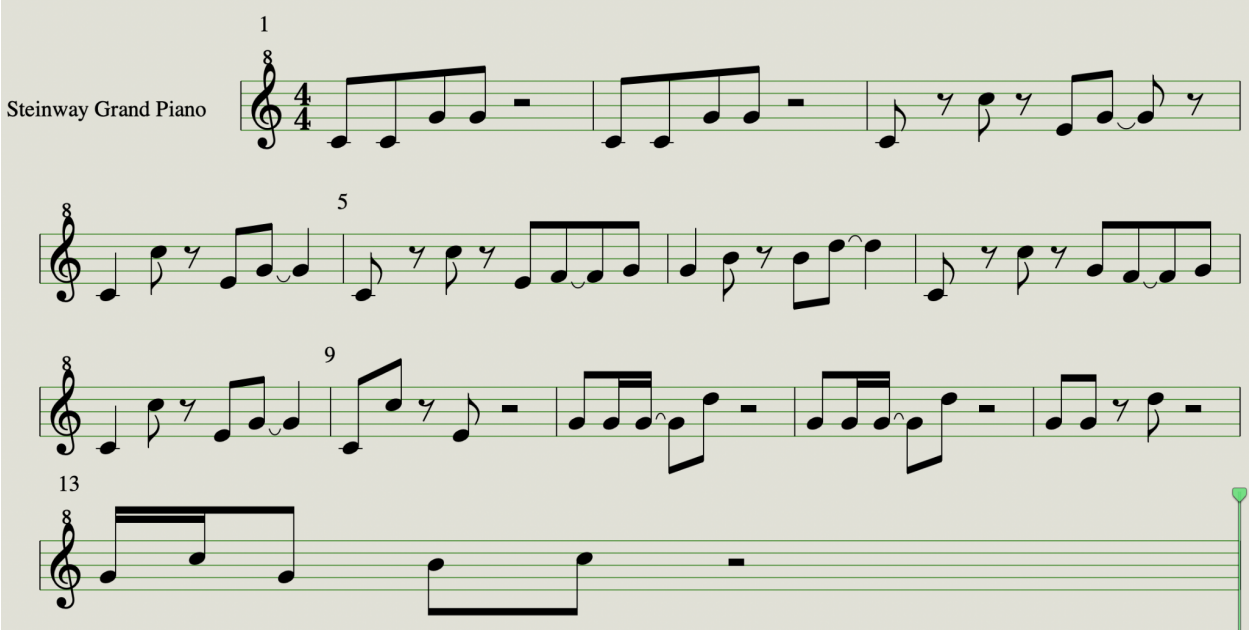


Fig.8 Output 6

3.2.4 The Limits of Magenta's Models

From the previous subsections, we showed that both Lookback RNN and Attention RNN could learn the characteristics of their training data. They would also take into account the input melody's range when generating new tunes. These traits should prove their usefulness in adapting to and producing pieces. However, I observed a limitation that stopped the models from achieving my goal.

Originally, I expected that the two models would learn the structure of pop songs. Learning which rhythm and combinations of pitches to use was just the basics. I hoped that it could know when to reuse entire bars from before. It would better mimic the repetition seen in pop songs. However, I noticed a pattern between all of their outputs. They couldn't repeat multiple bars that they previously generated. Instead, the two models tend to modify a melody across the piece continuously. Their inability to repeat numerous bars defied went against the repetitive structure of pop songs. Hence, I had to admit Magenta's models weren't fit for my needs.

Although I identified their shortcomings, I still had to investigate why Magenta's models couldn't repeat multiple bars. After searching through the source code on GitHub, I found that the models set their default lookback distances as notes one bar and two bars before the next output note [20]. It meant that the model could repeat a note across several bars. Yet, once a note stopped appearing for more than two bars, it wouldn't reappear. Entire bars could not be repeated when even one note may not reappear. Since Magenta could not repeat previously appeared bars, it couldn't replicate the repetition-heavy style of pop song melodies.

3.3 - Methods: Hard-Coded Algorithms

This part of the project took place after I realized Magenta's output didn't perform as I expected. I looked for alternative ways to compose music. David Cope's algorithmic approach to music composition did inspire me [21]. However, instead of recreating his work, I decided to implement my own hard-code rules for writing pop song melodies. *Music21* is a Python library capable of generating MIDI files [22]. With the help of this library, I was able to hard-code algorithms based on my understanding of pop song melodies. In this section, I will describe the inner workings behind each of my algorithms.

3.3.1 Generating Rhythms

Rhythms in pop songs must be singable by the general public. Therefore, it usually wouldn't contain rapid 16th notes. I designed my algorithm to generate some of the more common notes seen in pop songs [23]. These include eighth notes, quarter notes, dotted quarter notes, and half notes.

Here is a list of all the possible note combinations that I used to create my rhythm.

1. Eighth, Eighth
2. Quarter
3. Quarter, Quarter
4. Dotted Quarter, Eighth
5. Eighth, Dotted Quarter
6. Half

For each bar, I would keep pick combinations from this list until its total length reaches the max length of 4 quarter notes. If the total length exceeds the max length, I will reduce the length of the last note. By assigning a single pitch to all of the notes, I could obtain one bar of rhythm. The image below is an example of a generated rhythm (Fig.9). The chosen pattern was "Eighth, Eighth, Eighth, Eighth, Dotted Quarter, Eighth."

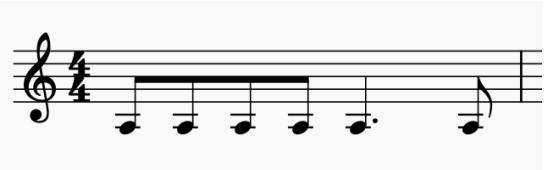


Fig.9 Rhythm generated by the hard-coded algorithm

3.3.2 Assigning Pitches

Chord progressions are a vital component of music. By switching chords throughout the piece, it gives the work a sense of progression. For my algorithm, I assign a chord to each bar [24]. In order to fit the rhythm of each bar to a chord, we must first list out all the notes that make up that chord. For instance, the figure below shows the notes C, E, and G (Fig.10). These are the chord tones that make up the C chord. If we successfully allow these notes to dominate the entire bar, we fulfill our chord assignment.

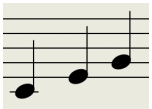


Fig.10 Three chord tones of the C chord

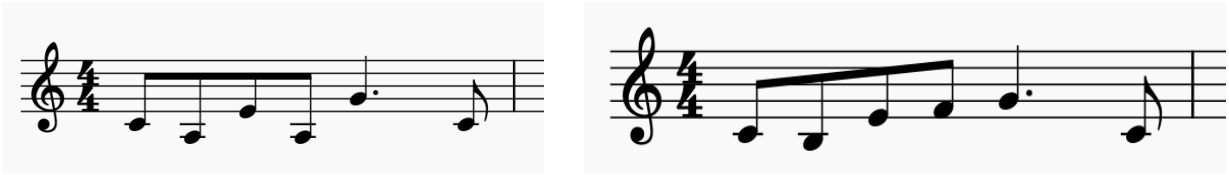
I will demonstrate how to fit one bar of rhythm to the chord of C (Fig.11). The first step is to ensure all three of C's chord tones exist within the bar. By placing C, E, and G, at the first three

downbeats of the bar, the algorithm ensures that the bar would sound like a C chord. Downbeats are prioritized to have a chord tone assignment. If there is no downbeat, such as the last beat in the example, the upbeat will get the assignment.

Before After

Fig.11 Assigning notes in one bar with the chord tones of C

As for each remaining note in the melody, I would assign it with a pick between the two pitches neighboring its previous chord tone. For instance, the second note in the example gets the assignment of B (Fig.12). This is because its previous chord tone, C, has neighbors B and D. By randomly picking between B and D, it gets the assignment of B.



Before After

Fig.12 Assigning remaining notes with neighboring tones

3.3.3 Repetition and Modification

In most pop songs, repetition exists to reinforce our memory of certain bars of the piece. In many cases, the second half of a chorus is just the first half, with the last two bars modified. This could be exactly mimicked with my hard-coded algorithm.

To mimic the structure of pop songs, my algorithm first creates a four-bar rhythm and fits it with a chord progression [24]. In the example seen below, I'm using CFGC as my chord progression (Fig.13). Next, my algorithm duplicates these four bars. The duplicated bars would be refitted with the same chord progression. However, only the last two bars would be modified, while the first two stay the same. By combining the original 4 bars and the duplicated 4 bars, the algorithm mimics the repetition and modification seen in pop songs.

```
for i in range(1,6):
    m0 = Melody(0)
    m1 = Melody(4)
    m1.fitChords(['C','F','G','C'])
    m0.combine(m1)
    m1.unlockBars([2,3])
    m1.fitChords(['C','F','G','C'])
    m0.combine(m1)
    m0.export('CFGC_{}_.mid'.format(i))
```

Fig.13 Screenshot of the code for repetition and modification

3.4 Results: Hard-Coded Algorithms

I produced 10 melodies following the chord progression CFGC with my algorithm [24] [25]. I observed both upsides and downsides of my algorithm. The output melody did follow my strict chord assignment. With all chord tones present within each bar, the audience could hear the chord progression easier. However, it also meant for some bars where only 3 notes are present, only the 3 chord tones would be used. This is the case for the second bar in the example below (Fig.14). Overall, the algorithm did create simple melodies that follow a clear chord progression and had repetition. However, it could still not reach the level of perfectly mimicking the style of pop songs. All 10 outputs that the algorithm produced are available on Google Drive [25]

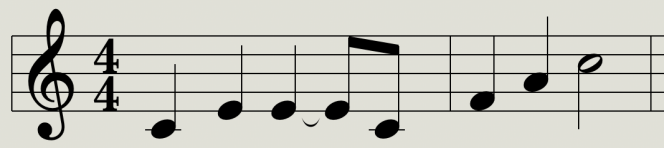


Fig.14 Case where the second bar has only chord tones

4. Discussion:

From this project, I learned both the strengths and weaknesses of Magenta's models and my algorithm. Magenta could generate logical continuations of a melody. It could help brainstorm 1 to 2 bars of a piece. Yet, it could not produce a complete piece due to its inability to repeat entire bars. My algorithm, on the other hand, is capable of repeating entire bars. Yet, it constructs very rudimentary pieces based on chord tones and neighboring tones. Its approximation is nowhere near the melodies of actual pop songs. If I could continue this project in the future, I'd like to combine Magenta's models with my algorithm. Let Magenta generate new short bars of melody while my algorithm takes care of repeating bars throughout the piece. It's only through a

combination of these two tools that my approximation of pop song melodies could be more accurate.

My biggest takeaway from this project is learning how to work with Magenta. I took notes on the command lines I used to interact with Magenta. These notes would be helpful in future Magenta-related projects. An improvement I should make is to write a Python script for extracting MIDI tracks from my dataset. Having spent an entire afternoon extracting only 67 MIDI tracks proved that manual extraction was efficient.

To those interested in knowing more about my project, I have uploaded the algorithm's source code and a few of Magenta's outputs onto my Google Drive [26]. I hope that the work I left behind could give you an understanding of how Magenta and hard-coded algorithms generate music.

5. References:

[1] Qibin Lou, "Music Generation Using Neural Networks," *Stanford University*, 2016. [Online]. Available: Semantic Scholar, http://cs229.stanford.edu/proj2016/report/Lou-MusicGenerationUsingNeuralNetworks-report.pdf. [Accessed: Apr. 29, 2021]

[2] Sam Agnew, "Training a Neural Network on MIDI data with Magenta and Python," *Twilio*, Oct. 4, 2019. [Online]. Available: Twilio, https://www.twilio.com/blog/training-a-neural-network-on-midi-music-data-with-magenta-and-python. [Accessed: Apr. 29, 2021]

[3] IBM Cloud Education, "Recurrent Neural Networks," *IBM Cloud Education*, Sep. 14, 2020. [Online]. Available: IBM, https://www.ibm.com/cloud/learn/recurrent-neural-networks. [Accessed: Apr. 29, 2021]

[4] Magenta (2021) Melody RNN, [Source Code]. https://github.com/magenta/magenta/tree/master/magenta/models/melody_rnn.

[5] Elliot Waite, "Generating Long-Term Structure in Songs and Stories," *Magenta*, Jul. 15, 2016. [Online]. Available: Magenta, https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn. [Accessed: Apr. 29, 2021]

[6] Fabian Gleeson, "Free MIDI files – The Ultimate List," *Off The Beat*, 2021. [Online]. Available: Off The Beat, https://www.off-the-beat.com/free-midi-files/. [Accessed: Apr. 29, 2021]

[7] David Wang, pop_midi_tracks, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/drive/folders/1LTMlwB_Iy6YZRMoxj-1Jlp2bNib9a3Wc?usp=sharing.

[8] David Wang, lookback_pop.mag, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/file/d/1Y9RUJhH9BHcFyNvU3s0kV7YvlWlpei1O/view?usp=sharing.

[9] David Wang, attention_pop.mag, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/file/d/1jXDVhfNPVt0pV3y0mD9p35UjZgc3oTBq/view?usp=sharing.

[10] David Wang, lookback_nintendo.mag, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/file/d/12aCVlr_d2-gwfCtcX7Ee3hM2za_BO18-/view?usp=sharing.

[11] David Wang, attention_nintendo.mag, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/file/d/1xMDxPHLp59u0eq_Nxdnmp5ahTeMgQBk8/view?usp=sharing.

[12] David Wang, input_1.mp3, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/file/d/1W-dLkzEYkCS9ueApk-PIUDjY3mdnGp38/view?usp=sharing.

[13] David Wang, input_2.mp3, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/file/d/1ykryUUScPl0M524IYMl7dx2WGKnw9jMO/view?usp=sharing.

[14] David Wang, comp_1_input_1.mp3, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/file/d/1oq1n0Z3kjy7D8jSGbJAkvWDJ_uCvew9V/view?usp=sharing.

[15] David Wang, comp_1_input_2.mp3, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/file/d/1rJp5z2csGPToSryZ4TTe7ZrhL_aRYaap/view?usp=sharing.

[16] David Wang, comp_3_video_game.mp3, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/file/d/1wnGVvaWHTPV4xc-UMrI8GhEhgjSWEUy2/view?usp=sharing.

[17] David Wang, comp_3_video_game_2.mp3, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/file/d/1gFK0yyNgJp8f1609y7aGim9Ewi_1vie-/view?usp=sharing.

[18] David Wang, comp_2_attention.mp3, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/file/d/1ZLPwU3vc-d-DdmN3U7DM0V1eABA-XCZV/view?usp=sharing.

[19] David Wang, comp_2_lookback.mp3, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/file/d/1meHuH3CjzvV30DIb_CLVHSsT0Fpbqz3k/view?usp=sharing.

[20] Magenta (2021) note_seq/encoder_decoder.py, [Source Code]. https://github.com/magenta/note-seq/blob/master/note_seq/encoder_decoder.py.

[21] David Cope, "Recombinant Music Composition Algorithm and Method of Using the Same," *Recombinant Inc*, Apr. 13, 2010. [Online]. Available: Google Patents, https://patents.google.com/patent/US7696426B2/en. [Accessed: Apr. 29, 2021]

[22] CuthbertLab (2021) Music21, [Source Code]. https://github.com/cuthbertLab/music21.

[23] David Wang, make_rhythm.py, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/file/d/1zjD-KvR3v10xA2s-GzNHdyZdP6US0ya7/view?usp=sharing.

[24] David Wang, hard_coded_algorithm.py, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/file/d/1W3qYoVocU06BkG0qobA9BhRHk-dbttNC/view?usp=sharing.

[25] David Wang, algorithm_mp3_outputs, *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/drive/folders/1SaYO0wkm39On67wAkSVIMIuMKqK8SdgZ?usp=sharing.

[26] David Wang, David Wang Capstone Publicly Available Files (2021), *University of Michigan*, Apr. 29, 2021. [Online]. Available: Google Drive, https://drive.google.com/drive/folders/17rYer6xncRgDK_hUyX3RduFuHWFTM4YC?usp=sharing.