CHIPTUNE EXTENSION AND GENERATION USING MARKOV CHAINS

Oscar Sandford
University of Victoria
oscarsandford@uvic.ca

Colson Demedeiros
University of Victoria
cdemedeiros@uvic.ca

Jae Park
University of Victoria
jaehyunpark@uvic.ca

Do we understand what we are doing?

George Tzanetakis

ABSTRACT

Music generation is a well-studied field. Previous attempts at music generation have succeeded in replicating training tracks using various methods involving complex time inference models. In this paper we show that simple, but appropriately designed, first and k-order Markov models are a sufficient enough machinery to both replicate and generate classic 8-bit video game music. [1]

1. INTRODUCTION

Music is typically constructed by humans, for humans. However, machines are becoming more adept at revealing patterns in the way musicians craft their chords. This enables humans to create their own music, and then let the machine take over the task of composer. Our project implements models for simple 8-bit music generation using temporal inference techniques. In the following section, we explore previous implementations and existing literature regarding music generation. Later, we will present techniques for designing generative Markov models to extend music as well as create derivative musical works.

2. BACKGROUND

2.1 Chiptunes

Chiptunes, also called 8-bit music, originated in the 1980s and the 1990s, when hardware limitations for video game devices allowed only simple waveforms such as square, triangle, and saw waves [19]. As a result, the necessary waveforms can be composed relatively easy in a computer.

Some examples of chiptunes come from various classic video games, including Super Mario Brothers [10], Undertale [8] and Legend of Zelda [14].

[1] This abstract will be expounded in time, pending study completion and results organized.

2.2 Markov Chains

Markov chains model a *sequence of states.* [2] Changes from one state to another are called *transitions* [3]. If one is to predict by hand which state will come next, given a certain state sequence, one might consider patterns in the sequence up until this point. That is, the probability distribution of a current state at time t is conditioned on all the previous states.

$$P(X_t|X_{0:t-1})$$

However, the number of previous states to consider from the beginning will be unbounded as the number of states grows. To this end, we use the **Markov assumption** that the current state is only dependent on a fixed number of k previous states [17]. This is the backbone of the so-called Markov processes (also called Markov chains).

$$P(X_t|X_{0:t-1}) = P(X_t|X_{t-k:t-1})$$

The variable k reflects what is called the **order** of the Markov process. First-order Markov processes have the current state depend *only* on the previous state, making transitions completely independent from state before the one directly before the curren state [17].

Any given set of k states can have an arbitrary number of possible next states, each assigned its own transition probability. The sum of the transition probabilities for each option must sum to 1 [7].

Markov chains are generative models, consisting of a set of states of size N. First-order models use a transition matrix of size $N \times N$ to store transition probabilities, but transition matrices for higher-order (i.e. $k > 1$) processes may be larger.

Hidden Markov models incorporate "hidden" states. Consider an unstable coin that has two states: fair (50-50) or biased (90% tails) [9]. One cannot necessarily discern the coin’s state simply from appearance, but we can infer the state through observations "emitted" by the hidden state. This variation on the Markov model is very useful, as we will see in the following section on related works.

3. RELATED WORK

3.1 Inference Models

Yanchenko and Mukherjee explored the use of Hidden Markov Models (HMMs) to compose classical music, finding proficiency in generating consonant harmonies, but

[2] In this study, we formulate the states as piano notes. Exact specification is revealed later.

lacking melodic progression [21]. Indeed, the models were found to learn the harmonic hidden states quite well, in some cases leading to overfitting. HMMs have also found use in chorale harmonization, where a given observable melody uses inference to derive hidden harmonies to complement it [2], as well as drum beat detection in polyphonic music [16].

Walter and Van Der Merwe's methods involve representing the chord duration, chord progression, and rhythm progression with first or higher-order Markov chains, whereas the overlaying melodic arc is represented by a HMM [20]. This separation works well to reduce the processing power needed for music generation, but the independent learning of each component leads to less cohesive compositions. Generating music is generally done by sampling a statistical model [5]. However, we want to create music that does not only simply replicate the training data, but also creates cohesive pieces in a more natural way.

Shapiro and Huber's approach to music generation simply uses Markov chains, no hidden states [18]. In their work, the states represent sound objects with attributes such as pitch, octave, and duration. Their results show that human-composed pieces can be closely replicated using the simpler Markov chains. Further, they have attached their implementation in their paper. We will consider this work when constructing our own implementation.

Corrêa and Jungling suggest using Markov chains of different orders to predict classical music. By using stochastic models to analyze different classcal songs and styles, the computer can even capture subtle and intuitive features such as style of the composer. Although this paper focuses on classical music and its prediction, the authors stress that its application to other music genres should be straightforward [6]. The only requirement is a MIDI file with a good quality, as it should be used for Markov chains that can properly estimate the music's pattern.

3.2 Data Format

The papers previously mentioned use the MIDI file format to write digital music. This format appears to be the standard for digital music creation [11]. One of MIDI's drawbacks is that it cannot store vocals [4]. This is of no concern to us, as we will only be attempting to generate instrumental compositions. Additionally, successful approaches to melody extraction from MIDI files [15] make assurances that this will be an adequate medium for the music our models will generate. MusicXML is a standard file format for storing sheet music, just as MP3 is for recordings [13]. Both are commonly used standards, and after experimenting with MusicXML for input data, we decided to use MIDI instead.

We decided to map MIDI to RTTTL (RingTone Text Transfer Language) notes. RTTTL is format used to specify ringtones for Nokia phones, and is composed of a comma-separated string of notes of a given format [12]. RTTTL is perfect for this study because notes (states) come one after another, in a sequence.

4. INITIAL APPROACH

After considering the differences between HMMs and simpler Markov models, we decided to design our generator as a k-order Markov process without hidden states, since these methods have seen success in recent works [6, 18].

Initially, we had two designs in mind. The first approach involved representing a song with a single chain of sound object states, including pitch, octave, duration, and other attributes. An alternative used m separate k-order Markov processes, one for each key on a piano used in the song. It would be prudent to consider extending the order of the processes in this case, in order to capture the musical patterns more clearly. First-order probabilistic transitions will inevitably make the generated notes very random. Extending the process order would alleviate the generative faults coming from independently trained Markov chains.

4.1 Timeline

We expect to have 3 milestones for this project:

1. **Milestone 1** (February 18th): Convert music to MusicXML files using AnthemScore. Devise sound object structure for Markov states or other architecture. Write a parser to transform the MusicXML data to sound objects.

2. **Milestone 2** (March 21st): Apply Markov chain algorithm to the music samples in order to train the model. We plan to refer to various works that we found. By this point, our models should be able to replicate the music tracks given as input. By this point our program will be able to extend music.

3. **Milestone 3** (April 7th): Tweak the models to generate more original variations. Enhance the quality of the music. Possibly a GUI for aesthetics as well, if time permits.

4.2 Task Delegation

In order to figure out the best approach and gather a plethora of sources, each team member researched various sources related to music data parsing and music generation, from theoretical papers to Python libraries. Colson and Jae found classic 8-bit tracks for use training and testing our models. Oscar set up a GitHub repository to include written work as well as source code, and made outlines for the final report. Colson would then be responsible for finding mumsic, and designing a parser to turn notes into data. Jae and Oscar would work on figuring out the best state attributes and designing the generative Markov models.

4.3 Resources

4.3.1 Tools

Stacher [3] is a frontend GUI for YT-DLP, which is a command line downloader that can be installed for converting Youtube videos to MP3 files. While YT-DLP works just

[3] https://stacher.io

fine, Stacher makes it very easy to convert YouTube links to various file formats (MP3, WAV, AAC, etc.) and save them on your current device. This is software we could use to get the songs as MP3 files saved and put into AnthemScore to be converted into XML files.

AnthemScore [4] is a music transcription software that converts WAV, MP3, and other audio formats into sheet music using an advanced neural network. The sheet music can then be exported to various other formats such as PDF, MIDI, or XML. The main use of this software is to easily obtain XML files that we would have used for data in the Markov chains. While AnthemScore is a great tool to aquire relatively true XML files of songs, the artificial intelligence that powers it does occasionally miss notes. What this means for us is that without changing the notes that are detected and placed by the AnthemScore software, it may miss notes or sounds that exist within the original song. Despite this, as long as it is mostly accurate and the majority of notes are in place, it shouldn't affect the overall process of the project. We avoided songs that don't transform accurately into MusicXML. Preliminary coding will be done in Jupyter Notebooks using Python.

5. FINAL APPROACH

5.1 Initial Steps & Restrictions

The first step we took was the decision to move away from the MusicXML format we started with, for a couple of reasons. For one, it required a software called AnthemScore, which is a paid service that works by converting MP3 files to MusicXML format. We had originally downloaded it with a one month free trial, and had intended to find a selection of songs in that time, and convert them to MusicXML immediately. We decided that this approach was not extensible, as any future progress using this method would require the software again to access new data. From here, we landed on MIDI files to use as data instead, as the existing Mido Python library makes accessing the files and data easy. This also removed Stacher, the software we had used to turn YouTube links into MP3, as a requirement.

Another thing we had to consider was the complexity of the MIDI files we used as data. For the most part, the code we wrote to interpret the MIDI file data is quite simple, capturing things like the time signature, key signature, tempo, as well as the tracks turned into a list of notes. MIDI files can be quite complex, using things like system exclusive messages, control change messages, and more.

The second issue was related to the creators of the MIDI tracks, and how they decided to implement simultaneous notes. Due to the nature of the RTTTL format, notes and rests play in sequence, from start to finish. We intended to have each MIDI track be a different part of the song, which would then each have its own RTTTL track that would be analyzed. Unfortunately, track creators can also have notes play simultaneously in one MIDI track by having a new note on message at $time = 0$ following another note on message of a different note. For these

[4] https://www.lunaverus.com/

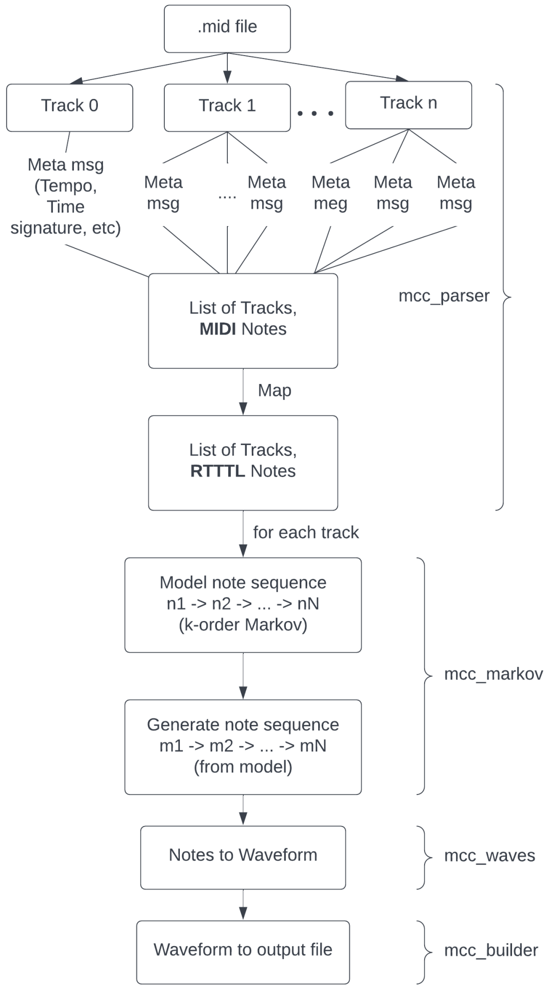

Figure 1. *The implementation pipeline.*

kinds of tracks, we simply have to drop one of the notes that is played, since we cannot easily separate them, and we cannot have them play simultaneously in RTTTL format. This, along with the more unique MIDI messages present in some songs, are the two main restrictions to our implementation and the tracks we have used as data.

5.2 Implementation

The source code for the implementation can be found here: https://github.com/oscarsandford/chiptune-generation.

5.2.1 MCC Parser

The mcc-parser module parses a MID file, and converts MIDI notes to an RTTTL string. Colson worked on the most of the parser module functions, which takes a MIDI file and tracks as inputs, and converts them into playable RTTTL strings. Jae implemented the MIDI-to-RTTTL function, which converts each MIDI note into a corresponding RTTTL note, which must take into account the given beats and tempo of the song. We used some Python code we found on Github by user devxpy [1], which converts MIDI note and instrument numbers to the appropriate instrument names. This function was used to attach the instrument name to each note, so that in the future, we could attempt to apply that instrument type to each track when generating music.

5.2.2 MCC Markov

Oscar wrote two classes for Markov models in the mcc-markov module.

- $SimpleMarkov$ is a naive construct for only modelling first-order Markov processes, as well as biasing the probability of the most likely action with a greedification factor ε.
- $KMarkov$ is designed to model first- and higher-order Markov processes, and is far more configurable and accurate in its output than $SimpleMarkov$ models. Its success has reduced $SimpleMarkov$ to a stepping stone - a viable, but less capable solution.

Both classes have two methods:

- **Fit**: takes a sequence of states as an argument, and builds a suitable model.
- **Predict**: samples a fitted model for a given number of samples n.

While both classes are implementated generically (i.e. they can take most hashable Python types as states), they are designed to model sequences of RTTTL notes as states.

Each track of a MIDI file is modelled as a separate chain, and sampled independently.

Higher-order Markov chains (i.e. $k > 2$) generate reasonably good-quality music that, with a high enough k-value (e.g. $k > 50$), come to resemble the original track. By itself, a single extended RTTTL track may not sound great, but we assume that once we gather all the MIDI tracks in a file and combine them all, the quality of the music will be much better.

5.2.3 MCC Waves

This module is relatively simple at a high level. It offers functionality to create square and triangle waves, as well as create ADSR envelopes for notes.

5.2.4 MCC Builder

The next step would be to combine tracks after generation. This is what the mcc-builder module is designed to do. Issues occur with the starting times of each track.

All of the MIDI tracks we have collected are type 1, which means they have multiple tracks with notes, but they all start simultaneously. The way the time attribute works in MIDI files is that it is a reference of time elapsed from the previous message, rather than a time of that message being played. In other words, A message with $time = 0$ starts immediately, and the following message that has $time = 100$ plays 100 ticks (MIDI units of time) after the previous message. Herein lies the issue with multiple tracks.

Often the melody, harmony, bass, and so forth, don't play immediately when the track starts. This causes an issue when combining the generated tracks back together, as the starting time is not saved anywhere. A solution we will explore is to look at the first message of each track and analyze the time value. This value can be converted to seconds, and saved as a value for each track. After generating the notes for each track, that value can be attached at the start, so that each track picks up at the same appropriate start time.

We have completed a WAV export function for mcc-builder, which converts the sound data into a WAV file. After all those above tasks are over, If we have time, we would like to implement a GUI as well, as it can makes the experience more intuitive for an end-user.

5.3 Reflection

Communication within the team was well done. After task delegation, each team member mostly worked on each of their task individually. Whenever someone was stuck or had a question, there was another person who could help out. We made steady progress as well, which is why we are keeping up with the deadlines that we set up initially. We expect to at least complete our project's "expected" goal by the end of the semester. See below.

One challange that we faced was the format of MIDI file was not so intuitive. Especially, the time attribute is not the time for that given message, rather, it represents time elapsed since the last note. We found some conflicting documentation on this matter, causing roadblocks of confusion. In order to achieve our goal, we had to make some compromise when parsing the MIDI notes, such as ignoring overlapping notes. Another challenge was we did not make a lot of progress during the reading break, due to copious reading, as well as playing video games.

The followings are our scenarios for possible outcomes:

1. **Basic/minimum**: single track RTTTL notes to chiptunes generator/extender. (Accomplished!)
2. **Expected**: MIDI file input of chiptune music with multiple tracks, generates and extends each track, outputs a WAV.
3. **Stretch goal**: tunable model with GUI.

6. REFERENCES

[1] Dev Aggarwal. Midi instrument converter, 2018.

[2] Moray Allan and Christopher KI Williams. Harmonising chorales by probabilistic inference. *Advances in neural information processing systems*, 17:25–32, 2005.

[3] Charles Ames. The markov process as a compositional model: A survey and tutorial. *Leonardo (Oxford)*, 22(2):175–187, 1989.

[4] Zehra Cataltepe, Yusuf Yaslan, and Abdullah Sonmez. Music genre classification using midi and audio features. *EURASIP Journal on Advances in Signal Processing*, 2007(1), 2007.

[5] Darrell Conklin. Music generation from statistical models. *Journal of New Music Research*, 45, 06 2003.

[6] Débora C. Correa, Thomas Jungling, and Michael Small. Quantifying the generalization capacity of markov models for melody prediction. *Physica A: Statistical Mechanics and its Applications*, 549:124351, 2020.

[7] Luisa de Francesco Albasini, Nicoletta Sabadini, and Robert F. C. Walters. The compositional construction of markov processes. *Applied categorical structures*, 19(1):425–437, 2010;2011;.

[8] Toby Fox. Megalovania, Nov 2017.

[9] Jasleen K. Grewal, Martin Krzywinski, and Naomi Altman. Markov models - hidden markov models. *Nature methods*, 16(9):795–796, 2019.

[10] Koji Kondo. Super mario bros (nes) music - underwater theme, Dec 2009.

[11] Gareth Loy. Musicians make a standard: The midi phenomenon. *Computer Music Journal*, 9(4):8–26, 1985.

[12] MeRWiN. Rtttl format specifications.

[13] MusicXML. Musicxml for exchanging digital sheet music, Jan 2022.

[14] Akito Nakatsuka. Zelda ii the adventure of link music: Overworld theme, Jul 2009.

[15] G. Ozcan, C. Isikhan, and A. Alpkocak. Melody extraction on midi music files. In *Seventh IEEE International Symposium on Multimedia (ISM'05)*, pages 8 pp.–, 2005.

[16] Jouni Paulus and Anssi Klapuri. Drum sound detection in polyphonic music with hidden markov models. *EURASIP journal on audio, speech, and music processing*, 2009(1):497292–497292, 2009.

[17] Stuart J. Russell and Peter Norvig. *Chapter 14: Probabilistic Reasoning over Time*, pages 867–868. Pearson, 2022.

[18] Ilana Shapiro and Mark Huber. Markov chains for computer music generation. *Journal of Humanistic Mathematics*, 11(2):167–195, Jul 2021.

[19] Shih-Yang Su, Cheng-Kai Chiu, Li Su, and Yi-Hsuan Yang. Automatic conversion of pop music into chiptunes for 8-bit pixel art. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 411–415, 2017.

[20] Andries Van Der Merwe and Walter Schulze. Music generation with markov models. *IEEE MultiMedia*, 18(3):78–85, 2011.

[21] Anna K. Yanchenko and Sayan Mukherjee. Classical music composition using state space models. *CoRR*, abs/1708.03822, 2017.