

Data Driven Music Generation Using Neural Networks

Alexander Saad-Falcon, *Georgia Tech Student, BSEE*
Vertically Integrated Projects – Robotic Musicianship
March 13, 2015

Abstract – This design report outlines the reasoning behind and implementation of music improvisation software developed by the Robotic Musicianship team at the Georgia Institute of Technology. This report is directed towards future members of Robotic Musicianship with the intention of providing guidance for future attempts at generated music improvisation. The work done over the course of the semester had the goal of using jazz solo transcriptions and classical music data to train a neural network to improvise music. This project was sparked by the recently available machine learning library, TensorFlow, which was released by the Google Brain Team in November 2015. The music improvisation software developed hinges on parsing musical pieces into sets of phrases, which are then used to train a neural network using the library. To accomplish this, a very large amount of data is needed to produce musically interesting results. Thus, music data was pulled from several sources and converted into a common format to be added to the neural network.

Introduction

Music generation from statistical models is by no means new. There have been several methods in the past to generate music, including random walks, Gibbs sampling, and Metropolis sampling. Each of these methods has dealt with musical “events,” which were sometimes narrowly defined as individual notes. [1]

Random Walks

Random walks can be built from individual note events by giving each note a probability to proceed into another note. A table of probabilities can be built from any given piece, and then a piece can be

A transition matrix with notes









				
	33%	67%	0%	0%
	20%	25%	35%	20%
	80%	15%	5%	0%
	45%	45%	10%	0%

Figure 1. A table of probabilities of one note progressing to another note.

generated by choosing a starting note and randomly generating notes to follow using the probabilities. A random walk can be generated from the note transition matrix shown in Figure 1. As expected, the resulting piece is unlikely to have any significant musical meaning. [1]

Gibbs and Metropolis Sampling

Both Gibbs sampling and Metropolis sampling deal with events in pieces as well, but instead start with an existing piece and

attempt to change events in the piece in order to give a new piece that still has a high probability. [1]

Machine Learning Approach

All of these methods, however, have proved insufficient of producing sufficiently interesting musical results. Thus, with the recent advent of TensorFlow, a more sophisticated approach is in order. The methods used in the music improvisation software developed differ in two ways from these previous models.

Firstly, musical events are instead defined as groups of notes (analogous to words in language) which can then be grouped into phrases (analogous to sentences). Events are defined as a group of notes starting on a downbeat and ending right before the next downbeat that it does not carry into via

syncopation or a tie, as shown in Figure 2. This leads to a more sophisticated interpretation of music that is closer to treating it like a language than defining individual musical notes as events.

Secondly, TensorFlow

allows the usage of

machine learning

algorithms to be applied to

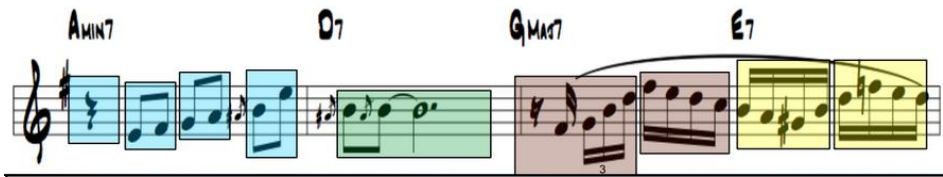


Figure 2. An example of part of a piece being divided into events (individual boxes) and phrases (groups of events with the same color).

music, so that a program can build up a large working knowledge of what events in music are likely to succeed others. Given an input, the neural network can give a reasonable musical output for a phrase to succeed the input. The data obtained to train this network, the parsing of this data, and the addition of the parsed data to the neural network, as well as having live inputs to the neural network, are discussed in the following sections.

Music Data to Train the Neural Network

For music data to be added to the neural network, the team looked at two genres: classical music from composers ranging from the Baroque to Romantic eras and transcriptions from jazz soloists. The former of these two options required much less work to prepare for the neural network; MIDI (Musical Instrument Digital Interface) data is available for almost any classical piece, and obtaining data for myriad composers did not present a challenge. Shown in the Appendix in Figure A-1 is the final list of composers whose pieces were chosen to be incorporated, with anywhere from 3 to 50 pieces being taken from each composer. The MIDI data obtained from each of these composers is readily convertible into the data format used in the neural network: a custom Google Protocol Buffer titled Musicproto.

For the jazz transcriptions, however, more work was required. Jazz transcriptions from famous soloists are not as readily available in the MIDI format as classical pieces are. Instead, they are mostly available in PDF. To address this, one of the group members purchased a PDF to MusicXML converter - MusicXML being a data format also being convertible into the Musicproto format. MusicXML is the standard format for digital sheet music, and it is formatted just like a regular XML document with specific fields and values. However, the PDF to MusicXML converter is far from perfect. With the large range of appearances sheet music can have, the converter makes frequent mistakes. For example, it interprets the chord progression over the blank measures at the end of transcriptions as a set of measures with null values that subsequently need to be removed. Additionally, the complexity of phrases of some jazz solos - which when notated results in septuplets, complicated chromatic phrases, grace notes, and accent marks - results in the converter misinterpreting many of the markings to give unnecessary rests, tuplets that do not carry through to the MusicXML, tuplets that did not exist in the PDF, more null measures, and often changes in key or time signature that are much more difficult and cumbersome to fix than other errors. A tuplet conversion error is shown in Figure 3.

The most practical solution to this problem was to divide the converted MusicXML files between the five group members and have each member fix their set of transcriptions. MusicXML is easily edited using any music notation editor, so the project members used the free but often bug-ridden MuseScore notation software. Over the course of the semester, the group obtained nearly 100 corrected jazz solo transcriptions in MusicXML format to be added to the neural network. These came from soloists

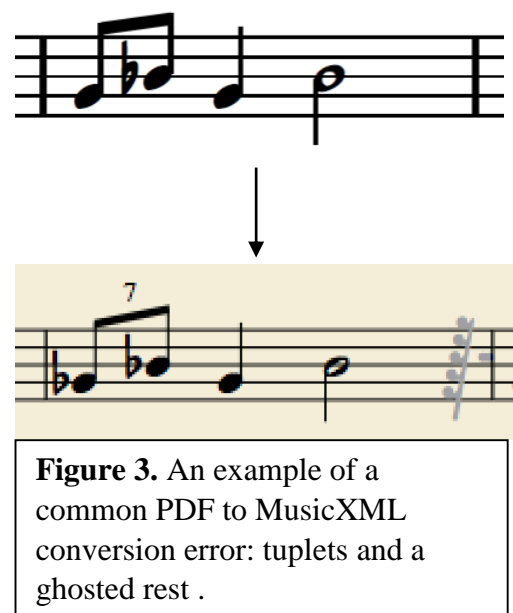


Figure 3. An example of a common PDF to MusicXML conversion error: tuplets and a ghosted rest .

ranging from Oscar Peterson on piano to Miles Davis on trumpet. With both types of data ready - classical music in MIDI and jazz transcriptions in MusicXML - the data can be parsed into the Musicproto format.

Parsing of Music Data

Once a set of accurate MusicXML files and MIDI files had been collected, they were both converted into a custom protobuf format. The music can then be fed into the neural net as training data regardless of what format it started in.

The MIDI standard states that MIDI files are a series of encoded messages that represent events in the progression of a piece. This encoding means that the files are small, but also difficult to read on their own. In order to start the parsing process, a python library called Mido¹ was used to convert the encoded files into a series of messages readable as a part of a parsing script, such as “note_on” or “note_off” messages. The parsing script then builds up to the Musicproto representation of the file message by message; notes are inferred from a note_on followed by a note_off on the same note value, and these notes are grouped into the team’s definition of musical events and phrases. Certain aspects of the music such as ties, voices and measure divisions are not explicitly noted in MIDI files. Because this information is crucial to distinguish events and phrases, it has to be deduced in the parsing script and included in the Musicproto.

Training of the Neural Network

Using the data obtained, a neural network can be trained to generate musical content. The basis for the content comes from how the music is organized (and parsed). By creating a neural network that sequentially models individual note transitions (similar to random walks), it is possible to generate new

¹ <https://mido.readthedocs.io/en/latest/#>

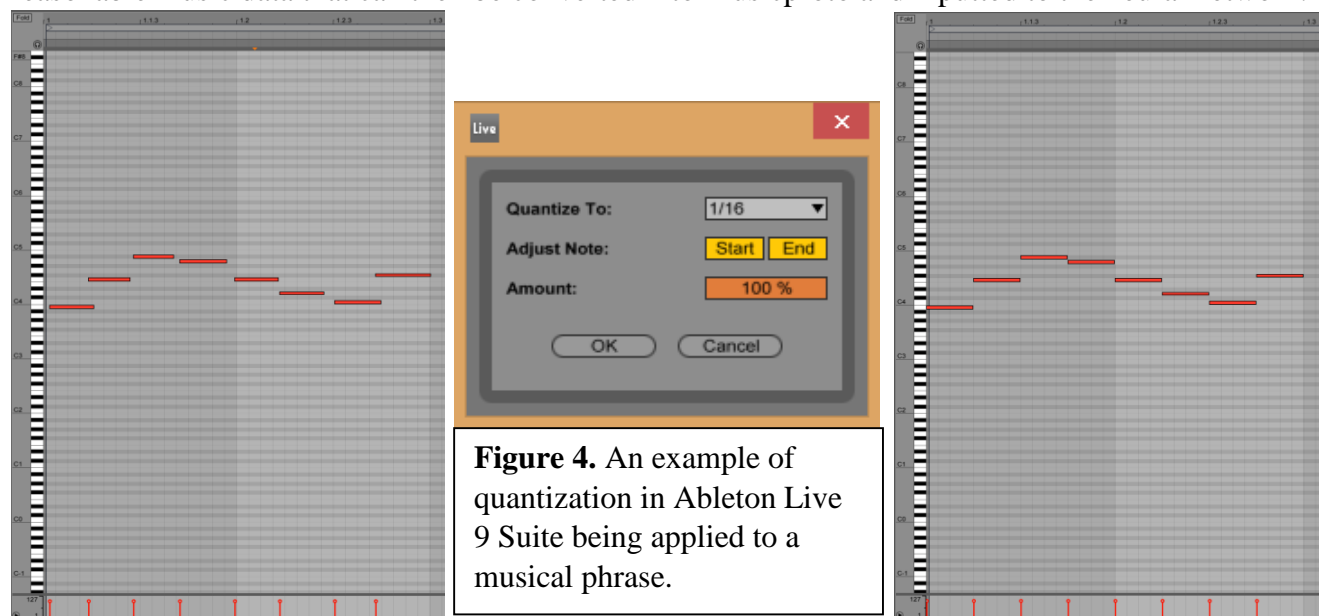
sequences. However, the quality of the music output is often poor and contains only local points of interest. Individual notes are the most frequently used input for music generation, but such systems are unable to model higher-level semantics and structure. This method is analogous to trying to generate a written story by training a model on the individual letter or character level. Though this is possible, an immense amount of data is necessary (on the magnitude of millions of documents). Like language, music exhibits a structural hierarchy from the morphological to the lexical and to the semantic. However, there is no single dataset of transcribed jazz improvisation to build a note level model capable of learning the hierarchical structure.

Instead, by creating musical features and models that describe these higher level structural components, the project will create more engaging music that has global plot structure, thematic development, and the build and release of tension. By parsing the input music data into events and phrases, the neural network can be trained to produce music that exhibits these higher level structures that would not be possible if it was working on a note-to-note basis.

Live Inputs

Inputs to the neural network can be created manually on a computer. Music notation software and especially music production software provide methods to create MIDI piano rolls that could then be parsed into Musicproto and inputted to the neural network. However, to add more flexibility to the usage of the software, live music input would be valuable. Luckily, most piano keyboards can provide output in the MIDI format. This MIDI can then be received by a computer, and parsed into Musicproto.

However, there needs to be an intermediate step. Live MIDI data from a keyboard is not quantized; its notes cannot necessarily be assigned to specific beats in a measure, since the input tempo is undefined. Even with a defined tempo, MIDI messages come with timestamps rather than beat values. Thus, in order to give reasonable input to the neural network, beat values have to be assigned through a process called quantization. A tempo and smallest note division are picked to quantize to, and then the starts and ends of notes are rounded to the nearest division in time, as shown in Figure 4. This leads to reasonable music data that can then be converted into Musicproto and inputted to the neural network.



Conclusion

Over the course of the semester, the Robotic Musicianship team has accumulated musical data from a myriad of classical composers and jazz soloists and parsed this data into musical events and phrases in the Musicproto format. By using all of this data and the machine learning library TensorFlow, a neural network will be trained that will hopefully produce meaningful musical improvisation when provided with either manual or live inputs. This musical improvisation will exhibit higher level structures because of the usage of events and phrases as opposed to individual notes. The repository can be found at <https://bitbucket.org/masonbretan/gatech-music-mind.git>, and the neural network will be trained and tested in the next few weeks. Future work includes the addition of more MIDI and MusicXML data.

Appendix

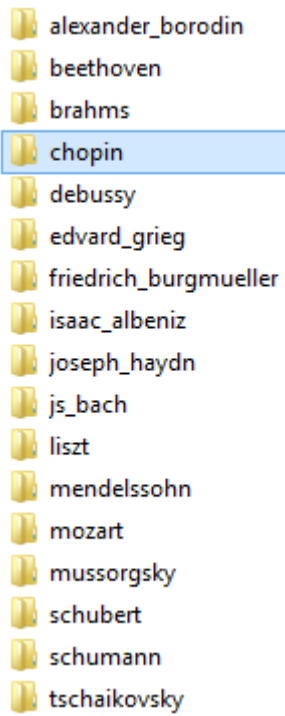


Figure A-1. The list of composers chosen for MIDI data.

References

- [1] D. Conklin, "Music Generation from Statistical Models", *Journal of New Music Research*, pp. 1-3, 2003.