Imitative Classical Music*

APS360: Project Final Report

August 15, 2019

Introduction

This project investigates how RNNs and other ANNs may be used to generate single-instrument classical music. RNN-based architectures were the de facto foundation for text generation ANNs for many years since its inception. [1] At least ostensibly, the same techniques should be largely transferable for generating other sequential data of similar nature such as music. For instance, each note in a musical piece can be thought of as a token in a character-based RNN.

The project is interesting in that it involves an intricate data processing stage whereby the choice of encoding greatly affects the quality of the generated results. Also, the model must interface aural comprehension, a part of the human cognition that has not been dealt with in the course thus far.

Code Repository

Part 1-2:

`https://colab.research.google.com/drive/1Ohr5fhHgW2NNrSn7i7Pi5dtnrON6K7gf`

Part 3-14:

`https://colab.research.google.com/drive/1odx8YPtpidmqH_fH6uqcKr-QWM-owPIx`

*Word count: 1977. Penalty: 0%.

Contents

Project Difficulty

Background & Related Work

A rich profusion of papers on NN-based music generation have been published in recent years, many of which overlap in content. In this section we briefly introduce two high-impact works.

Modeling Temporal Dependencies in High-Dimensional Sequences (2012)

This influential paper Bengio et al. [2] investigates how polyphonic music may be represented in "piano roll" representation while preserving musical richness. Further, it introduces the RNN-RBM (restricted Boltzmann machine) model for extracting temporal patterns from high-dimensional sequential data.

BachBot (2016)

The BachBot [3] synthesizes harmonized chorales that closely resemble Bach's original compositions. The BachBot draws ideas from music theory and employs a probabilistic sequence model (LSTM-RNN). An online demo is available at `https://bachbot.com`.

Data

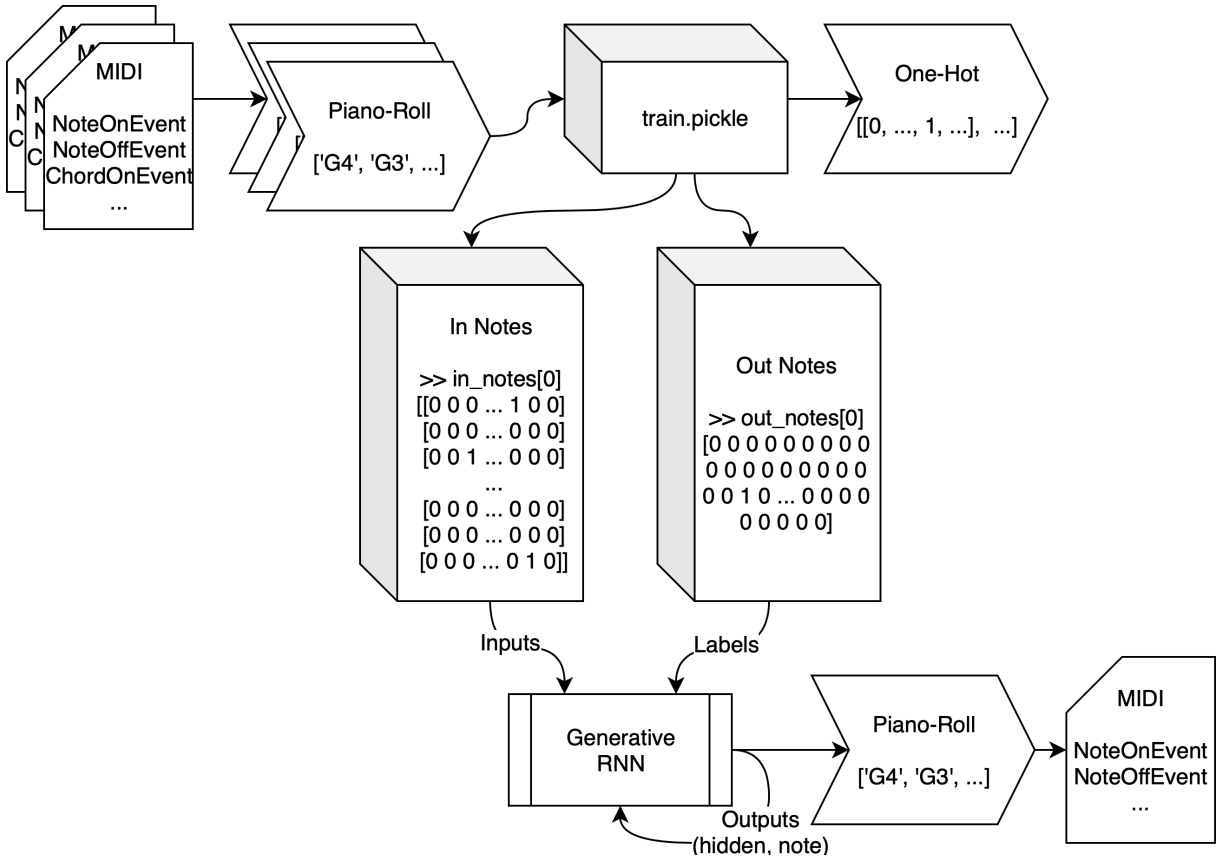

Figure 1: A high-level overview of the data pipeline for model training. Refer to *Terminologies* for explanations of the nomenclature. See *Structure of the MIDI File Format* for an overview of the MIDI file format. See *Preprocessing the MIDI Files* for details on the encoding process. See *Architecture* for details on the RNN architecture.

Terminologies

We first introduce some terminologies and concepts from music theory.

- A **note** is the most elementary unit of sound in musical compositions. A note is comprised of its **pitch** (how high the frequency is) and its **duration** (how long the sound is held for).
- A **pitch class** is the primary measure in the textual representation of a pitch, designating a set of all pitches that are separated by an **octave**. There are 7 fundamental pitch classes: A, B, C, D, E, F, and G.
- An **octave** is the secondary measure of a pitch. An increment in octave corresponds to doubling of the frequency. For instance, C4 is 260Hz, and C5 is 520Hz.

- A **chord** represents multiple notes are played simultaneously.
- **Tempo** indicates the pace of a given piece.
- A **rest** corresponds to an interval of silence where no notes are played.

Structure of the MIDI File Format

The **Musical Instrument Digital Interface (MIDI)** technical standard defines a binary file format of the same name, which can be used to concisely represent musical pieces. MIDI files greatly differ from familiar audio formats such as MP3, in that it only contains abstract schemata about how the notes should be played, as opposed to an actual recording of how the audio sounds. *Figure 2* shows a decoded MIDI file.

```
224   |     midi.NoteOnEvent(tick=0, channel=0, data=[77, 49]),
225   |     midi.NoteOnEvent(tick=0, channel=0, data=[49, 35]),
226   |     ...
228   |     midi.NoteOffEvent(tick=141, channel=0, data=[49, 0]),
229   |     midi.NoteOnEvent(tick=0, channel=0, data=[56, 30]),
230   |     midi.SetTempoEvent(tick=120, data=[12, 203, 57]),
231   |     midi.NoteOffEvent(tick=0, channel=0, data=[77, 0]),
232   |     ...
```

Figure 2: A snippet of a programmatically decoded MIDI file.

Only two pieces of information in *Figure 2* are relevant to us:

- **NoteOnEvent/NoteOffEvent** respectively corresponds to pressing down on a piano key, and releasing a pressed piano key.
- The first elements in the data parameters (ex. 77, 49, 56) are the **MIDI Note Numbers**. These are discretized representation of pitches as defined in the MIDI standard. For instance, 77 refers to F5.

Gathering the Raw Dataset

The model is trained from 924 MIDI files of classical piano pieces sourced from `http://www.piano-midi.de`. A few files were malformed and were discarded by filtering for

the MIDI file header (0x4d 0x54 0x68 0x64). The files were randomly split into train (0.8), validation (0.1), and test (0.1) sets.

Preprocessing the MIDI Files

Several libraries were investigated for working with MIDI files. The initial candidates were `mydy`[1], `pypianoroll`[2], and MIT's `music21`[3]. We opted for using `music21` as it had the most reliable documentation and APIs.

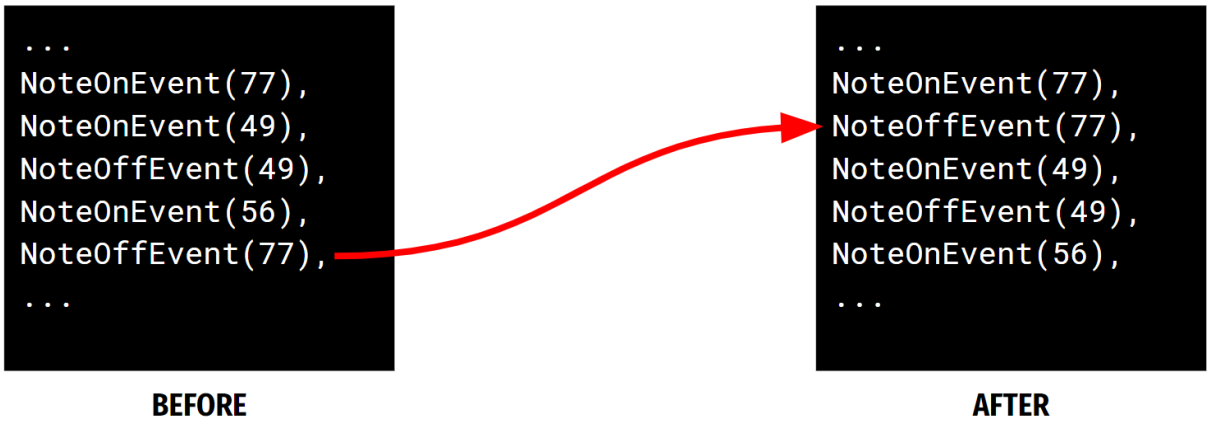

Figure 3: Visualization of the duration simplification process.

A few simplifying assumptions were made to make the task easier to solve:

- Rests were discarded as they seemed to have minimal impact on the sense of melody in single-instrument monophonic music.
- The lengths of the notes are also disregarded for a similar reason. There are two cases to consider. In the first case, a single pitch is held for multiple time steps without new notes being played. This would somewhat impact the melody, but was found to be statistically insignificant (less than 0.5%).

  In the second case, a pitch is held for multiple time steps in the background, but new notes are being played simultaneously. With piano, unlike string instruments, holding the key in the background while playing new notes does not impact the melody as much, as the volume of the background pitch will attenuate quickly.

  Refer to *Figure 3* for a visual example of the simplification process.

[1] https://github.com/jameswenzel/mydy
[2] https://pypi.org/project/pypianoroll/
[3] https://web.mit.edu/music21/

- Chords, in which several notes are played in the same time step, were substituted by the corresponding highest-pitch note within each chord. While chords are important, this simplification curbs the combinatorial explosion that would otherwise make the input encoding much more cumbersome in the naive one-hot encoding approach that we adopt in the project.

Since parsing the entire dataset is computationally expensive, the notes are extracted once and saved on disk for later use using python's `pickle` module.

Each dataset is then simply a sequential array of pitches. All of the musical scores are concatenated together within each dataset to make the sampling process easier. This was deemed acceptable as the boundaries are statistically insignificant (1,000 or so notes out of 700,000). Finally, the data is one-hot encoded using the `pandas` library.

There are 87 possible notes in the entire dataset. A snippet of the training data is shown in *Figure 4*.

```
>> train_data[:10]
# The piano-roll representation is obtained after
# preprocessing the MIDI files.
['G4', 'G3', 'G4', 'B3', 'G4', 'D4', 'A4', 'D5', 'C5', 'B4']

>> notes_in[0]
# 2D list of shape (87, sequence_length).
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]

>> notes_out[0]
# 1D list of shape (87).
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Figure 4: A snippet of the training data.

Note that the data resembles that of a text generation problem, hence it may now be modelled similarly as one.

Architecture

In total, we trained and compared approximately 30 different models. Disregarding hyper-paramter tuning and trivial modifications, we outline 4 architectures that yielded the most interesting results.

All models were trained for 128 epochs with a learning rate of 0.001.

Result Criteria

- **Octave Accuracy** measures whether the first predicted pitch is in the same octave class as the ground truth.
- **Exact Accuracy** measures whether the prediction is in the same octave class and the pitch class.
- ± 1 **Accuracy** measures whether the prediction was exact, or the neighbouring two pitches were predicted.
- ± 3 **Accuracy** measures whether the prediction was exact, or the neighbouring six pitches were predicted.

Justifications for and implications of using these criteria to measure the quality of generated music are further discussed in the sections that follow.

C5 as Baseline

C5 is the most common note in the dataset, comprising of approximately 3.97% of the 700,000 notes. We use this note as the bare-minimum baseline for prediction accuracy.

Results

	Octave Accuracy	Exact Accuracy	± 1 Accuracy	± 3 Accuracy
C5	0.3295	0.0397	0.0942	0.2237

MLP as Baseline

Since there are only 87 possible notes, one observes that it may be possible to model this as a simple multi-class classification problem, and attain reasonable performance with a multilayer perceptron (MLP). A visualization of the data flow is shown in *Figure 5*.

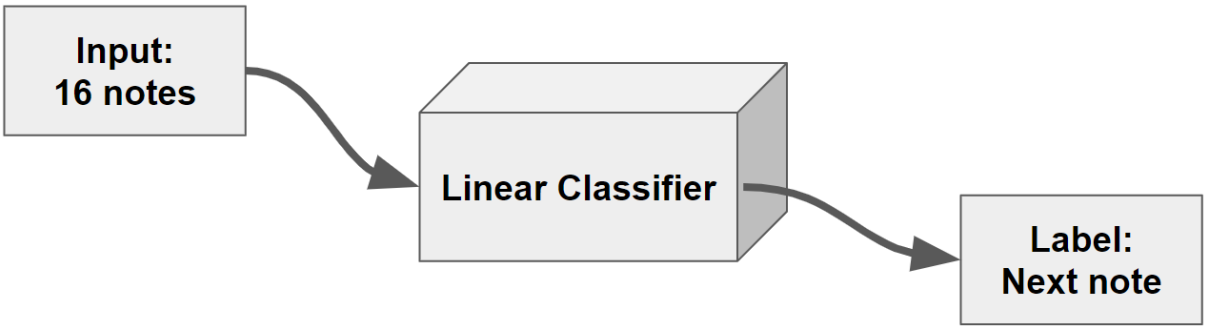

Figure 5: A visualization of the MLP baseline model.

The linear classifier consists of 3 `nn.Linear` modules. The input of 16 notes are expanded into 256 then 512 hidden units, goes through an `nn.Dropout` layer of 25% before outputting the 87 probabilities for the next note.

Results

	Octave Accuracy	Exact Accuracy	± 1 Accuracy	± 3 Accuracy
C5	0.3295	0.0397	0.0942	0.2237
MLP	0.4736	0.0641	0.1537	0.3643

The performance improved about 50% over the C5 model, indicating that the model is working as intended.

Basic GRU

The improved model primarily employs the GRU architecture, as satisfactory results were demonstrated with LSTM RNNs in previous related works. GRU-RNN is generally expected to perform on par with LSTM-RNN but with shortened training time.

Using an RNN makes sense, because melodies by their very construction require harmonic congruence (i.e., how certain notes sound “good” together as studied in music theory) in conjunction with temporal proximity. The simplified, temporally encoded MIDI data are fed

into the RNN, as outlined in *Preprocessing the MIDI Files*. The model makes unidirectional predictions in which successive pitches are generated based on the temporally preceding pitches.

The basic GRU model consists of the built-in `nn.GRU` module with 5 layers, dropout of `0.25`, hidden-state dimension of 128, sequence length of 64, followed by a linear classifier.

Results

	Octave Accuracy	Exact Accuracy	± 1 Accuracy	± 3 Accuracy
C5	0.3295	0.0397	0.0942	0.2237
MLP	0.4736	0.0641	0.1537	0.3643
Basic GRU	0.5245	0.1343	0.1848	0.4119

The exact accuracy improved significantly over the MLP model, which is in line with the expectation that RNNs are expected to perform better with sequential data.

GRU with MSELoss

We observe that a valid strategy may be to minimize the error in the frequency domain. That is, failing to predict the correct note, we generally want the prediction to be close in frequency to the ground truth.

There are some caveats to this claim. What sounds "good" to the human ear is nontrivial to quantify numerically. For instance, a C#4 in place of C4 may sound very unnatural although the pitches are very closely located in the frequency domain. On the other hand, C5 may better maintain the melodic quality although it is a full octave away from the ground truth.

Nevertheless, we test out this theory with an alternative architecture employing MSELoss instead of CrossEntropyLoss. We devised the following function which is evaluated for each of the 87 MIDI notes:

$$\delta = \alpha^{\log_2 b - \log_2 a}$$

Where:

- α is a hyperparamter that was found experimentally
- a is the maximum of the target frequency and the frequency of the MIDI note

- b is the minimum of the target frequency and the frequency of the MIDI note

Derivation was mostly guessed, but the assertions $\alpha > 0$ and $\log_2 b - \log_2 a < 0$ guarantee mathematical stability. Further, $\log_2$ was derived from the fact that pitch classes are on an exponential scale (c.f. *Terminologies*).

Results

	Octave Accuracy	Exact Accuracy	± 1 Accuracy	± 3 Accuracy
C5	0.3295	0.0397	0.0942	0.2237
MLP	0.4736	0.0641	0.1537	0.3643
Basic GRU	0.5245	0.1343	0.1848	0.4119
MSELoss GRU	0.5600	0.1093	0.2582	0.4688

Here, the ± 1 and ± 3 accuracies had notable improvements, while the exact accuracy had regressed. This is in line with the discussion in the beginning of the section about the error in frequency not necessarily being a good measure of the quality of the generated melodies.

GRU with Teacher Forcing

We implemented teacher forcing as the final improvement to the model, with losses being evaluated for every prediction in the sequence as opposed to only at the last note. Also, the sequence length was increased from 64 to 128.

Results

	Octave Accuracy	Exact Accuracy	± 1 Accuracy	± 3 Accuracy
C5	0.3295	0.0397	0.0942	0.2237
MLP	0.4736	0.0641	0.1537	0.3643
Basic GRU	0.5245	0.1343	0.1848	0.4119
MSELoss GRU	0.5600	0.1093	0.2582	0.4688
Teacher GRU	0.6573	0.3800	0.4411	0.5759

The use of teacher forcing allowed for a better convergence rate with most criteria improving significantly across the board.

A demo of the final model is available at `https://soundcloud.com/user-702429992/demo`. The best sample out of 5 was selected by the authors for a survey. From a survey

pool of 5 people, the generated sample was found to be generally pleasing to the ear and passable as a human creation.

Discussion

- The final model is performing beyond our expectations. Across multiple samples, it is generally able to learn features from music theory such as harmonics and repetition.
- Some models experienced output notes getting stuck at the same pitch at times. This likely indicates some flaws in the architecture or programming errors that we were not able to figure out at the time of writing.
- Data representation has a great impact on the quality of the generated results, as seen from the alternative frequency-based approach.
- The model still lacks interpretability. It is difficult to know whether the model is memorizing and stitching together short segments of training data, or is genuinely creating new compositions from the learned abstract features about music theory.

Ethical Considerations

- *Copyright and ownership of training data and generated content*. In the United States and Canada, the use case largely falls under the fair use (or fair dealing) doctrine, hence legally inculpable. Thus the problem pertains to ethics than lawfulness, and one should aim to err on the side of ethical transparency when using the model. For instance, the model may be used for reconstructing portions of damaged or lost recordings, but could also be used to mass-produce infringing music that exploits the legal ambiguity for profit.
- *Superannuation of human-composed music*. While current state of the art still seems distant from inventing a musical style that matches the precision and facility of human composers', the edgy hypothetical question remains—what if AI could match or exceed human creativity? However vapid the hypothesis may seem, unimpeded numerical abstraction of human creativity may make life unfulfilling for composers and listeners alike.

Project Difficulty

The most notable challenge we faced was choosing the correct data representation. In this project, we implemented and compared two plausible approaches—one modelled as a regression problem using frequency domain metrics, and the other using naive textual representations of the notes and modelling the task as a text generation problem.

References

[1] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, 2019.

[2] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," *arXiv preprint arXiv:1206.6392*, 2012.

[3] F. Liang, "Bachbot: Automatic composition in the style of bach chorales."