Music Generator

By

Terry Wang, Rima Mittal, Joshua Goldberg

Supervisor: Yuri Balasanov

A Capstone Project

Submitted to the University of Chicago in partial fulfillment
of the requirements for the degree of

Master of Science in Analytics

Graham School of Continuing Liberal and Professional Studies

March, 2020

The Capstone Project committee for Terry Wang, Rima Mittal, Joshua Goldberg
Certifies that this is the approved version of the following capstone project report:

Music Generator

Approved by Supervising Committee:

_____

Yuri Balasanov

Abstract

We propose the application of Deep Learning models to help individuals generate interesting music ideas. The platform has a GAN-based generator. The data used to train the generator are 130,000 midi files across many different genres of music. The algorithmic process is expected to be able to explore more music combinations than any individual, and thus provide a more holistic and creative approach to music creation.

Our work includes a modified version of the python package pretty_midi, which serves as a midi pre-processing starter package for Deep Learning. Using Generative Adversarial models, we provide insights on predicting pitches and duration to ultimately generate music notes.

Keywords: Deep Learning, Music Generation, Generative Adversarial Model, Midi Processing

NOTE: Do not use "#" or "##" symbols to start new sections in the abstract section, as one typically would in other r markdown documents. Doing so will result in generating a table of contents entry *prior* to the Introduction, which is not desirable.

Executive Summary

We propose the application of Deep Learning models to help individuals generate interesting music ideas, with the goal of aiding musicians to use these computer-generated musical ideas to enhance their music writing process. The results show limited success in reaching our goal of meaningful musical output, and we include a few ideas for future improvement on the model.

Our main model is a Generative Adversarial Network with two components: the generative model, which takes a Gaussian noise input of length 128 and outputs a numpy array representing a 20-note midi sample; the discriminator, which takes an input of the numpy array representing a 20-note midi sample and outputs a float between 0 and 1, with 1 representing real and 0 fake. For real samples, we used a midi data repository totaling 130,000 midi files across many different genres of music, but for this project specifically we used mostly Baroque music. For generated samples, we use a deep neural network structure that terminates in separate branches for pitches, whose output is a length 128 softmax vector representing the probabilities of the pitches, and duration, whose output is a vector of 2 positive numbers representing the start and end time of the note, for each of the 20 notes generated. This structure seems to perform the best out of the many structures we tried.

Our work includes a modified version of the python package pretty_midi. We adapted its main engine to pre-process midi files with varying tempo and key changes and write midi files into numpy array, as well as putting the generated numpy array back into the midi format. In addition, we provided an assortment of utility functions and model selections, which we made into a python package together with the pretty_midi.

NOTE: Like the abstract, do not use "#" or "##" symbols to start new sections in the executive summary section. Doing so will result in generating a table table of contents entry *prior* to the Introduction, which is not desirable.

Table of Contents

List of Figures

List of Tables

Chapter 1

phoenixdown::capstone_gitbook: default

Placeholder

Problem Statement

Research Purpose

Variables and Scope

Writing Tips

R Markdown Basics

Lists

Line breaks

Background

Placeholder

Code chunks

Linked tables and List of Tables

More than R: Other Languages

Including plots

R Markdown Tables, Graphics, References, and Labels

Inline code

Figures

Footnotes and Endnotes

Methodology

The methodology section may include the following subsections:

- Data
- Descriptive analyses
- Modeling Framework

Data

The training data used to train our model is a collection of midi files that was compiled by a Reddit user see reddit page and put up for download. There are about 130,000 midi files from a wide range of genres included in this collection. Our training, however, centers around a small subset of the files that are of 4/4 rhythm, in order to reduce the chance of complication from training with a wider range of rhythmic patterns.

In addition, in our pre-processing script, we took steps to "normalize" the samples as much as we can, in terms of making the tempo constant as well as transposing all songs to the key of C major (or A minor if in minor key). The efforts to normalize keys is not always successful. In some midi files, tempo and key changes are included as part of the metadata, which we can then extract and edit. But in others, the metadata is not included.

Before diving deeper into the details of our data, it is necessary to introduce the basic structure of midi files. Midi files (.mid) are digital records of musical "events" divided into various tracks, where each track contains a voice, an instrument, or a line of melody. For example, a midi file containing piano music might have two tracks representing the notes played by the two hands of the human piano player. Each time the human player plays a note, the midi file would record the following information: at what time is the piano key pressed; at what time is the key released; how hard is the key pressed; what is the pitch bend. After collecting these information for each and every note in a song in a midi file, we will have most of the information needed to reproduce the song in its entirety down to some basic expressive elements. In the midi metadata, some other relevant information is stored, such as tempo, key, tempo changes, etc. Midi files are not music by themselves because they only contain the information needed to produce music - think of it as the digital equivalent of the sheet music. Midi decoders and synthesizers are needed to translate midi files into music.

If we strip the midi file structure down to its most basic elements, it would look something like this:

Insert graph 1-1, unable to get DiagrammeR to work for now

We can reimagine the midi file to be like a dictionary, where each track is indexed by its name, and in each track we have a nice tabular structure of data consisting of the following columns: pitch number, start time, end time, and pitch velocity (we will ignore pitch bend for now). For example, if we look at the first 3 notes played by the right hand in Beethoven's song Moonlight Sonata, we can tabulate them like the following:

Table 1.1: Midi File Structure Example

Pitch	Start Time	End Time	Pitch Velocity
56	0.00	0.40	33
61	0.40	0.81	26
64	0.81	1.21	26

The above table tells us the following: at time 0.00 seconds pitch 56 is played with velocity 33, and the note stopped at time 0.40 seconds; then at time 0.40 seconds pitch 61 is played with velocity 26, and the note stopped at time 0.81 seconds; and so on. We can see clearly how a piece of music can be represented digitally via the midi file structure.

The other thing to note about midi is the data types of the columns above. The `Pitch` and `Pitch Velocity` columns take only an integer between 0-127. The `Start Time` and `End Time` columns are positive floats to represent the number of seconds since the start of the song. These things will have an impact on the decisions we made about the model structure.

Since the goal of the model is to generate novel melody ideas, we decided to focus only on pitch, start time, and end time from the midi file information, as well as limiting the output of the generator to 20 notes at a time.

For data fed into the model, we use a processing script to rewrite a pre-selected subset of these midi files into numpy arrays in accordance with the above-mentioned structure: each song will be represented by a series of numpy arrays, each representing a track, and each array will have the tabular structure of 130 columns: a 128-length one hot vector for pitch, start time, and end time. We made the decision to only focus on pitch, start time, and end time in order to focus on learning the melody and not expressiveness, as well as to cut down on complexity. We then use another training script to randomly make 20-note samples from the training data and arrange them into a numpy array. We need 20-note samples in order to make sure that the real and fake samples have the same dimensions[1]. Therefore, our training dataset has dimensions of (x, 130, 20) where x represents the number of real

[1]For our discriminator model this is necessary, but there are other model structures available to overcome this problem.

samples generated for training, the number 130 represents the three columns (pitch, start time, end time), and 20 represents the 20 notes sampled.

maybe insert a chart with sample data structure

One early choice we made to make the pitch a one-hot vector was done out of the observation about the nature of pitches in the Western musical tradition: the meaning of pitches in the musical sense do not exist in a continuum despite their origins in the frequencies. Therefore, it does not make sense to treat them like so. Instead, one-hot encoding pitches will allow us to treat each pitch as independent objects; in addition, our model will be able to output softmax for pitches, therefore leading to pitch probabilities that can be further investigated. However, there are successful music generation models that treat pitches as one number.

Modeling Framework

Generator

Our generator is a deep neural network with the following basic structure:

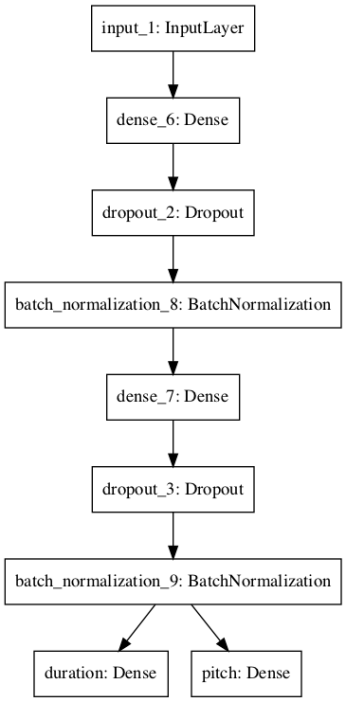

Figure 1.1: museGen Model Structure

The input of this model is a 128-length vector of standard normal Gaussian noise. The noise will go through 4 layers of dense structure (currently set at 256 nodes) with ReLU activation, dropout, and batch normalization. Then, the model splits in half, one each for pitch and duration generation. The pitch part of the model then upsamples the output from the last layer into 20*128 nodes, where 20 is for the 20 notes to generate, and 128 is for the one-hot vector of pitches. Similarly, we upsample duration from the same layer into 20*2 nodes, where 20 is for the 20 notes to generate and 2 is for start and end times. Afterwards, we have to reshape these nodes into the correct shapes, (20, 128) for pitches and (20, 2) for duration, and apply to the correct axis the corresponding activation functions: softmax for pitch (for pitch onehot), and ReLU for duration (for outputting positive numbers). The generated pitch and duration vectors are then concacted into one single array of dimensions (20, 130) as the output of the model.

Using this modeling structure, we ensure that pitch and duration of one generated music sample is generated by a single model and one single noise input. The dense layers preceding the split in the model will allow the model to learn latent features and rules of music before feeding that latent representation of the final product into the part of the model that turns latent vectors into pitch and duration.

Conventionally, music generation models are

Justification of model(s) selected. Identification of dependent and independent variables, per model as well as variable transformations. Feature extraction (if applicable). Discussion of model(s) functional form. Assumptions of model(s) and ways of insuring that assumptions are observed or tested. Assessing model(s) performance and validation.

Transform `dose` into a `factor`. Only three dosage levels are present.

```
data(ToothGrowth)
colnames(ToothGrowth) <- c("length", "supplement", "dose")
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
```

We are most interested in discovering which treatment leads to the optimal tooth growth. In this vein, we use `aggregate` function to transform our data and compute the average tooth `length` by both `supplement` type and `dose` size.

```
groupedTooth <- aggregate(ToothGrowth, by=ToothGrowth[,2:3], FUN=mean)[,1:3]

knitr::kable(groupedTooth, align = "r", caption = "Average tooth length",
      format = "latex", longtable = TRUE)
```

Table 1.2: Average tooth length

supplement	dose	length
OJ	0.5	13.23
VC	0.5	7.98
OJ	1	22.70
VC	1	16.77
OJ	2	26.06
VC	2	26.14

Math and Science notation

TEX is the best way to typeset mathematics. Donald Knuth designed TEX when he got frustrated at how long it was taking the typesetters to finish his book, which contained a lot of mathematics. One nice feature of *R Markdown* is its ability to read LATEX code directly.

Get around math mode's automatic italicizing in LaTeX by using the argument `$\mathrm{formula here}$`, with your formula inside the curly brackets. (Notice the use of the backticks here which enclose text that acts as code.)

So, $Fe_2^{2+}Cr_2O_4$ is written `$\mathrm{Fe_2^{2+}Cr_2O_4}$`.

The command below does what you'd expect: it forces the current line/paragraph to not indent. See below and examples of commonly used symbols:

Exponent or Superscript written as `$x^2$` becomes x^2

Subscript written as `$x_1$` becomes x_1

Infinity written as `$\infty$` becomes ∞

alpha written as `$\alpha$` becomes α

beta written as `$\beta$` becomes β

delta written as `$\delta$` becomes δ

epsilon written as `$\epsilon$` becomes ϵ

sigma written `$\sum_{i=1}^n f(x)$` becomes $\sum_{i=1}^{n} f(x)$

Math Examples

An Ordinary Least Squares model, from *Introductory Econometrics, 6th edition* by Jeffrey M. Wooldridge, page 27.

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

An infinite distributed lag (IDL) time series model, by Wooldridge, page 633.

$$y_t = \alpha + \delta_0 z_t + \delta_1 z_{t-1} + \delta_2 z_{t-2} \ldots + \varepsilon_t$$

A vector autoregressive (VAR) model, by Wooldridge, page 657.

$$y_t = \delta_0 + \alpha_1 y_{t-1} + \gamma_1 z_{t-1} + \alpha_2 y_{t-2} + \gamma_2 z_{t-2} \ldots,$$

Determinant of a square matrix:

$$\det \begin{vmatrix} c_0 & c_1 & c_2 & \dots & c_n \\ c_1 & c_2 & c_3 & \dots & c_{n+1} \\ c_2 & c_3 & c_4 & \dots & c_{n+2} \\ \vdots & \vdots & \vdots & & \vdots \\ c_n & c_{n+1} & c_{n+2} & \dots & c_{2n} \end{vmatrix} > 0$$

A regularization problem solved by Jerome Friedman, Trevor Hastie, Rob Tibshirani and Noah Simon, implemented in the R package `glmnet`.

$$\min_{\beta_0,\beta} \frac{1}{N} \sum_{i=1}^{N} w_i l(y_i, \beta_0 + \beta^T x_i) + \lambda \left[(1-\alpha)||\beta||_2^2/2 + \alpha||\beta||_1 \right]$$

From Lapidus and Pindar, Numerical Solution of Partial Differential Equations in Science and Engineering, page 54.

$$\int_t \left\{ \sum_{j=1}^{3} T_j \left(\frac{d\phi_j}{dt} + k\phi_j \right) - kT_e \right\} w_i(t)\, dt = 0, \qquad i = 1,2,3.$$

From Lapidus and Pindar, page 145.

$$\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta,\zeta) = \sum_{k=1}^{n}\sum_{j=1}^{n}\sum_{i=1}^{n} w_i w_j w_k f(\xi,\eta,\zeta).$$

Additional R Markdown and bookdown resources

- *Bookdown* Online Book - `https://bookdown.org/yihui/bookdown/`
- *Markdown* Info Sheet - `https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet`
- *R Markdown* Reference Guide - `https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf`

Findings

Placeholder

Results of descriptive analyses

Modeling results

Results of model performance and validation

Conclusion

This section includes a concise summary of the findings. Your summary might be organized by the research objectives or hypotheses. Make sure you address the extent to which research objectives are achieved, and if they are not achieved, explain why. Make sure to interpret your findings in a way that acknowledges the limitations of the research. That is, do not extrapolate the insights derived from your research to situations you have not examined.

While increasing dosage leads to larger incisor length, the choice of delivery mechanism between Orange Juice and Vitamin C does not seem to make a difference. However, at very low levels, Orange Juice appears more effective, displaying higher average growth.

Recommendations

Includes guidelines as to ways in which your results should or could be used in practice. You may discuss other uses of your results, if there are any. The ways to extend your analysis and the benefits of doing so might be included in this section as well.

Appendix A

The First Appendix

This first appendix includes all of the R chunks of code that were hidden throughout the document (using the `include = FALSE` chunk tag) to help with readibility and/or setup.

In section 1:

In section 1:

Appendix B

A Second Appendix, for example

References

Placeholder