Dilated Convolutions for Music Generation

Kaleb Morris, Hannah Leou
Department of Computer Science
Stanford University
CS 230 Spring 2018
SUNetIDs: `kalebm,hleou`

Abstract

Art and science are often considered to be polar opposite disciplines. Many debate that the creativity and natural fluidity behind music composition is the sole product of human generation. They argue that technology, is far too cold and rigid a field to produce the freeform and melodic arrangements. We would like to challenge this notion and investigate the intersection of these two fields through computer generated music. One may be skeptical of a computer's ability to generate something so subjective as music, after all how would a computer know if it was producing something that sounds pleasing? While the quality of musical piece is subjective, we see that pieces from the classical genre follow fairly rigid principles of music theory and composition. Unlike pop or other modern genres, we see that classical melodies are highly structured and thus are perfect for training a model that can intelligently generate music. Within our project, we sought to learn the principles and recurrences encoded in classical music, utilizing them to generate pieces through a Dilated Convolutional Neural Network.

1 Task Definition

For our project, we are using a Dilated Convolutional Neural Network to sequentially generate music. Given MIDI files, we extract the primary melodic voice from the track. We then encode "active note" vectors for each time step in our training sample compositions. For our model, we emulate Google DeepMind's WaveNet for raw audio generation. Our model follows the general Wavenet architecture of a Dilated Convolutional Neural Network, but is optimized for the task of music generation rather than speech generation. Furthermore, our model is restructured such that it is designed to train on and generate MIDI files. Our model produces output by means of generating an "active note" vector for each time step. In our post-processing step, we transform these encoded vectors into MIDI events and conjoin the successive MIDI events to generate a complete MIDI file.

As music is very subjective, it's difficult to quantify the results of each model to guide the development of our final model. We therefore constructed a survey to determine the quality of a generated piece. In this survey, we focus on evaluating the flow, rhythm and repetitiveness of the piece. We also gauged listeners on how musically correct and classical the piece sounds. While the last two metrics are good general guideposts, they are unreliable due to their subjectivity.

2 Infrastructure

We created a class called midiGenerator, that given the training data set outputs a generated MIDI file. Our first goal in designing the class was to build an interface so that we can directly compare the results of training the Dilated Convolutional Neural Network on different training sets. Our second goal in designing the class was to build an interface so that we can experiment with different tempos (ticks per beat) to generate the most sonically pleasing musical compositions. When running

Submitted to CS230 Spring 2018

the midiGenerator.py file, you can easily specify the desired model training set and output tempo following the usage outlined below:

Usage: [# MIDI events to generate][path to training examples][ticks per beat]

path to training examples - The algorithm will use only the MIDI files in the specified directory path to train the Diluted Convolutional Neural Network

ticks per beat - The algorithm will use the given ticks per beat to set the generated MIDI file's tempo in the Meta Messages of the MIDI File

3 Approach

We begin by reading in a MIDI file, extracting the primary melodic voice, and then for each time step, encode the active notes as a vector. We stack these "active note" vectors on top of one another to create a matrix representation of the MIDI training file. After creating our matrix, we train our model using the sequence of encoded "active note" vectors. We frame our training in the context of an input x and a target y, where x is a series of "active note" vectors and y is another set of "active note" vectors with overlapping time-steps.

3.1 Data Encoding

For our "active note" vector approach of data encoding, it was necessary to pair down the robust MIDI files. MIDI files, especially those of classical compositions typically consist of many melodic and harmonic voices represented by MIDI tracks. To single out the primary melodic voice in the MIDI file, we visualized the MIDI files using GarageBand. From a simplified MIDI file, we encoded the active notes at each time step with a modified "one-hot" vector. There are 128 note classes in MIDI files and each of these classes is represented by a value of 1 (ON) or a value of 0 (OFF).

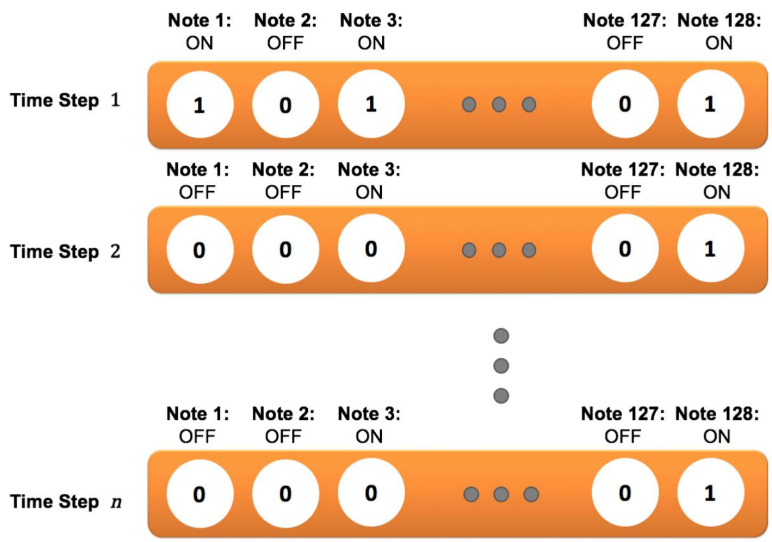

Figure 1: Visualization of the matrix composed of "active note" vectors encoded for 128 note classes

For a training piece, we stacked the sequential "active note vectors". The first row of the matrix is an "active note" vector that corresponds to the first time step of the piece, the second row of the matrix is an "active note" vector that corresponds to the second time step of the piece, and so on.

3.2 Model

We modeled our own architecture after the Google DeepMind's WaveNet, a dilated dilated convolution (convolution where the filter is applied over an area larger than its length by skipping a constant number of inputs)

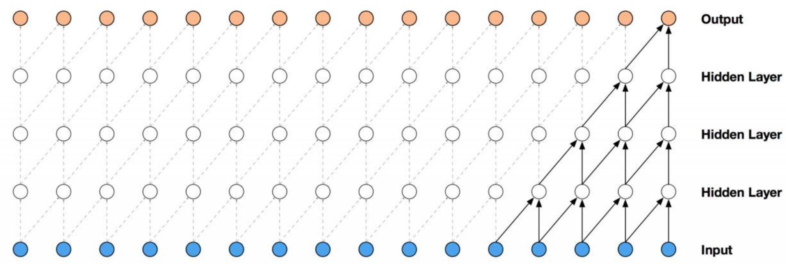

Figure 2: Figure 1: Visualization of a stack of causal convolutional layers.

The model we created takes as input MIDI files and generates as output MIDI files. In order to generate MIDI files rather than raw audio output, we replaced the final activation function in the original WaveNet with a sigmoid function in order to map the values for 128 note classes into a range between $[0, 1]$. We fine tuned the threshold such that note classes with values above the threshold would be switched off in the generated encoding to a value of 1 and classes with note values below the threshold would be switched on in the generated encoding.

Dilations: The dilated convolutional layers incorporated to our model greatly improved the receptive field. Our initial assumption that a larger receptive field would help generate better music led us to add many dilated convolutional layers to our model. However, we observed that additional layers minimally brought down training loss and significantly brought up training time. We valued the ability to iterate on and redesign our model, so we chose to incorporate a modest 5 stacked dilated convolutional layers to our model.

Activation Functions: For the activation functions within our network, we tested a number of different activation functions. The activation that we settled on for the majority of our layers was an ELU function, as this appeared to be giving us the best results. The final activation function we utilized for our model was the sigmoid function. We utilized the sigmoid function because our model was outputting "active note" vectors with note class values varying from low negative values to low positive values. We chose to use the sigmoid function in our final layer to recenter our data such that all class values ranged from 0 to 1 so that we could transform the final vector to be representative of a note, chord, or silence.

Learning Rate: We tried a few values for our learning rate, but it appeared as though the default of 0.001 in PyTorch was best suited to our task. Increasing the learning rate from 0.001 resulted in our loss being greatly increased, while decreasing the learning rate from 0.001 resulted in similar loss with much greater time to train.

4 Literature Review

Music generation is a very well-explored problem in the realm of artificial intelligence. There is a diverse variety of models which have been used to model music composition. Among the most popular models for music generation are Recurrent Neural Networks and Markov Chains.

Recurrent Neural Networks (RNN) are particularly well equipped for music generation because they are not constrained by direction like an archetypal directed Neural Network. The RNN is a superior model because it has a degree of memory with each layer receiving input from the previous layer and input from the previous time-step. In order to expand the memory of this model, many music generation models employ Long Short-Term Memory (LSTM) nodes which incorporate a memory cell which passes inputs from one layer's time step down to multiple layers' proceeding time-steps. Google's deep learning music project, Magenta, utilizes an RNN and two LSTM's to generate a single melodic voice. Music composition is a highly methodical process which contains a plethora of recurring patterns. Music generation can be artificially initiated by breaking down musical pieces into common subsequences and reordering the subsequences in innovative ways. The Markov Chain Model is particularly apt for this task as it functions under the assumption that there is a hidden structure or underlying relation between successive notes. Typically, Markov Models for music generation will take into account time, pitch, and duration when crafting the state space. While Markov Models are advantageous in the sense that they can be quickly trained, they come with the major disadvantage of having "memorilessness" and inability to pick up on dependencies between hidden layers.

For our project, we chose to model music generation using a Dilated Convolutional Neural Network. We chose this model for two reasons: characteristics of classical music and MIDI input/output. The classical music genre is characterized by its regimented, constrained structure and contains a good deal of recurring note sequences. This underlying structure of classical-style music can be exploited by Dilated Convolutional Neural Network. Moreover, the encodings for MIDI files is very befitting for a Dilated Convolutional Neural Network. MIDI files are encoded using messages with each message containing note, note velocity, and note duration. These messages can be reinterpreted as a modified version of a "one hot" vector (per time step) in which all active note classes are populated with a value of 1 and all inactive note classes are populated with a value of 0.

We chose to utilize a Dilated CNN because of the incredible breakthroughs Google DeepMind has made with its WaveNet, a deep generative model of raw audio waveforms. WaveNet is "a fully convolutional neural network, where the convolutional layers have various dilation factors that allow its receptive field to grow exponentially with depth and cover thousands of timesteps." DeepMind's model, originally developed for applications in speech generation is now being expanded to applications in music generation through the DeepSound project.

Currently, the WaveNet model is trained on raw audio input and generates raw audio output. In our project, we decided to modify and restructure the WaveNet architecture such that the model is trained on MIDI files as input and generates MIDI files as output. This way, the CNN will able to capitalize on learning from focused information concentrated in MIDI files rather than noisy data found in raw audio. Ultimately, the CNN we are constructing will have to do significantly less work to generate equally natural music to that of a CNN operating on raw audio, ideally making music generation more feasible for those without significant computational power.

5 Error Analysis

Although the nature of the subject matter is highly subjective, we can measure error via our population survey results, explained below.

5.1 Population Survey

The best measure of quality of a musical composition is the human ear. Therefore, we surveyed individuals and asked them to score the training music compositions and generated .mid files produced in five categories. We surveyed 30 students.

1. Score the flow of the song (1-5)
2. Score the rhythm/beat of the song (1-5)
3. How classical does this song sound? (1-5)
4. How repetitive is this song? (1-5)
5. How musically correct does this song sound? (1-5)

Using this model, we received the following scores throughout the different iterations on our model trained on composer-specific data sets.

Throughout the process, our team managed to more closely bridge the gap between the aspects of human-composed music and machine-composed music. Despite stark differences in musical structure, which is primarily what allowed our survey responders to distinguish human-composed music from machine-composed music, certain aspects generated by our model more closely resembled those of real music.

Project Stage	**Flow**	**Rhythm**	**Classical**	**Repetitive**	**Correct**
Training on ONLY Mozart	3.2	2.0	3.3	1.0	1.8
Training on ONLY Haydn	3.1	2.1	3.0	1.1	2.1
Training on ONLY Beethoven	3.0	1.9	3.2	1.3	2.3
Training on ALL composers	2.6	2.2	2.9	.9	1.7

5.2 Flow and Rhythm

We found that flow and rhythm scores were fairly equal for the model when being trained on exclusively Mozart, Haydn, or Beethoven. We believe that this is due to the fact that composers have specific musical styling that translates to unique flow and rhythm patterns our models were able to pick up on. Therefore, we observed that although all composers fall under the classical genre, the model, when trained on the aggregate of Mozart, Haydn, and Beethoven compositions performed significantly worse due to the discrepancies in musical styling.

5.3 Repetitiveness and Correctness

We found that none of our trained models generated pieces that were characterized by a lot of repetition. We believe that our model may have generated too large a receptive field with the multiple dilated convolutional layers. Perhaps if we generated longer pieces a repetitive quality may have been more apparent. Ultimately, we attribute this to the fact that the model was trained on much longer classical pieces than we generated.

The correctness of our generated pieces was better among the models trained on one exclusive classical artist. We believe that this is valid because each specific artist has an underlying structure and style for their compositions. It is logical that generated pieces from the "Beethoven ONLY" trained model would generate music resembling the Beethoven's actual compositions. Further, a generated piece from the amalgamation of classical composers would struggle to generalize and pinpoint specific patterns and structures among varied artists.

5.4 Classic Element

Our results seemed to somewhat capture the essence of classical music. However, we would have liked to see our results more closely resemble the style of the input that our network was trained on. This likely occurred due to the limitations of our model in our attempt to simplify the music generative process. In the future, we would add more complexity to the modeling of MIDI notes. Particularly, we could consider the velocity or duration of the notes more accurately.

6 Conclusion

The music generated through our model did not sound as natural as we had initially thought it would, but certain aspects of the music definitely improved upon altering our model. Notably, the model's capability to generate more complex pieces consisting of notes, chords, and silences came about after altering how we were processing and feeding data into our model. In future iterations, we would train on a much larger dataset, and we would test with different architectures, specifically deepening our network and possibly adding residual and skip connections to improve our model's memory capability.

References

1. Oord, Aaron van den, et al. "Wavenet: A generative model for raw audio." arXiv preprint arXiv:1609.03499 (2016).
2. Brinkkemper, Frank. “Analyzing Six Deep Learning Tools for Music Generation.” The Asimov Institute, 7 Oct. 2016, www.asimovinstitute.org/analyzing-deep-learning-tools-music/.
3. Johnson, Daniel. “Polyphonic Music Generation Using Tied Parallel Networks.” Polyphonic Music Generation Using Tied Parallel Networks, 12 Dec. 2017, www.cs.hmc.edu/ ddjohnson/tied-parallel/.
4. Merwe, A. and Shulze, W., "Music Generation with Markov Models," in IEEE MultiMedia, vol. 18, no. 3, pp. 78-85, March 2011.
5. Steinsaltz, D., Wessel, D. "The Markov Melody Engine: Generating Random Two-Step Markov Chains." Department of Statistics, California Berkeley, http://www.steinsaltz.me.uk/papers/melody.pdf.