Clara: Generating Polyphonic and Multi-Instrument Music Using an AWD-LSTM Architecture

Christine Payne, OpenAI Scholars Program
August 29, 2018

Clara is an LSTM that composes piano music and chamber music. It uses either a chordwise or notewise encoding for midi files, using a 62 note range, and allowing any number of notes to play at each musical time step. This is in contrast with many generation models that insist on having always exactly 4 notes at a time, or on having much smaller note ranges. Models were trained on a large corpus of piano and chamber music, drawn from Classical Archives[1].

Introduction

Clara is an AWD-LSTM that composes piano music and chamber music. While many models insist on a small note range, or in having a limited or fixed number of notes per step, this model uses a 62-note range (which encompasses most of classical piano and violin music), and allows any number of notes and instruments at a time. It uses PyTorch, MIT's music21, and the FastAI library. The model was trained using midi samples from Classical Archives, although there is nothing in the model specific to classical music, and other piano or instrumental midi files should be compatible.

Methods

A primary factors in the quality of the musical generations is the method of encoding the midi files into text formats. The midi files provide the timing of when each note starts and stops, along with information about the instrument, the volume, and tempo markings for the piece. These note events can happen at arbitrary times, there can be any number of notes playing at once.

Time discretization

Music often has a quarter note as a basic measurement of time. It is very common to see notes played 3 times or 4 times per quarter note (triplets, or 16ths, respectively). In discretizing the midi timing information, there is a tradeoff: sampling too frequently will make it hard for a model to learn a longer scale musical pattern, whereas sampling too infrequently will distort some of the rhythms. I chose to offer two options: sampling either 12 times per quarter note (to allow for both triplets and 16th to be encoded faithfully), or 4 times per quarter note (triplets are distorted slightly, but it makes it easier for a model to learn a longer scale musical pattern). The user can also specify a different sampling frequency.

Chordwise and Notewise Encodings

One option would be to ask the model to predict yes/no for each of the 88 piano keys, at every musical time step. However, since each individual key is silent for the great majority of the time, it would be very difficult for a model to learn to ever risk predicting "yes" for a note. I offer two different possible solutions for this problem. In language models, we often work at either the word-level or character-level. Similarly, for music generation, I offer either chordwise or notewise levels.

For **chordwise**, I consider each combination of notes that are ever seen in the musical corpus to be a chord. There are usually on the order of 60,000 different chords (depending on the note range and the number of different composers). Then I treat the chord progression exactly as a language. In the same way a language model might predict "cat" when prompted with "The large dog chased the small ...", I ask the musical model to predict the next chord when prompted with a chord sequence. Note that a "chord" is used generally here - it is any combination of notes played all at one time, not necessarily a traditional musical chord (such as "C Major").

For **notewise**, I consider each note start and stop to be a different word (so the words might be "p28" and "endp28" for the start and stop of note #28 on the piano). I then use "wait" to denote the end of a time step. In this way, the vocab size is around 100.

In both cases, I use a slightly decreased note range (62, instead of the full 88 on the piano). For notes that fall outside this range, I move them up or down an octave, so that they fall into this range. The vast majority of classical music falls into this note range (for example, in the entire piano works of Brahms, only 3 notes were outside this range).

Rest Encodings

Chordwise encoding has the problem that for any given musical timestep, there is a high likelihood that no notes are played (ie, that the chord is a rest). Even when sampling only 4 times per timestep, around 40% of all chords across the musical corpus are rests. So, in training an LSTM model to predict the next chord given an input sequence of chords, it is very easy for the model to fall early on into only predicting rests.

There are multiple ways to avoid this problem. One might be to shift the loss function so that a model is penalized less for not correctly predicting a rest. I took a different approach: I re-encode the rests, so that each rest represents the number of notes played in the previous 10 time steps. This serves two purposes: first, the model cannot blindly predict “rest” and be correct 40% of the time. Secondly, the model is given the auxiliary task of keeping track of the number of notes. Since this task is compatible with the main goal of learning musical structure and being able to predict upcoming chords, the task helps to push the model to learn meaningful embeddings.

Another approach might be to consolidate the rests (ie, to signal “rest 1 time step”, “rest 4 time steps”). I chose not to do this for two reasons. First, there are still a large number of 1-time-step rests (frequently in the corpus there are patterns such as: “chord” “rest” “different chord” “rest”). So there would still be the problem that 1-step rests are over-represented. Secondly, this destroys the timing structure in the input to the LSTM. Unlike notewise representations, chordwise has the large advantage that each LSTM step corresponds to one musical timestep. This allows the LSTM to learn longer scale musical and rhythmic patterns.

Wait Encodings

Notewise encoding has a similar problem, in that there may be several “wait” steps in a row. Here, I do consolidate these. For example, with a sampling frequency of 12 per quarter note, two eighth notes in a row might appear: “p12 wait6 p18 wait6” (play piano note 12, then wait half a quarter note, then play piano note 18 and then wait half a quarter).

Chamber Music

The notewise representation lends itself relatively easily to adding in multiple instruments. The main difference is in encoding the midi files. Here, I mark violin notes with the prefix "v" and piano notes with the prefix "p", but otherwise train the music generator in the same way. One problem is there are much fewer violin/piano pieces than pure piano solo pieces. I included in the data set any pieces that are a solo instrument and piano (such as cello/piano, horn/piano), and I transpose the solo instrument parts to be within the violin range. In the future I may explore ways to use the piano solo music to pretrain the network.

Data Augmentation

For all the encodings, I also transpose all the pieces into every possible key. The test set is kept separate: a piece in the test set cannot be a transposition of a piece in the training set. Performing this transposition is a matter of shifting all the notes in a piece up or down (I use the range -6 to 5, to represent all 12 possible keys), and then checking that no notes now fall outside the overall allowed range.

In the future, I may work more carefully on the violin transpositions. The violin has much more specific requirements about which notes it can play at once, and which notes are physically impossible pairs. One way to handle this may be to train first on all the transpositions (whether physically possible or not), and then to fine tune the model only on the original keys.

Musical Generator

I used an AWD-LSTM network from the FastAI library as the generator model. In *train.py*, one can easily modify the hyperparameters (changing the embedding size, number of LSTM layers, amount of dropout, etc). I train the model by asking it to predict the next note or chord, given an input sequence.

Once the model is trained, I create generations by sampling the output prediction, and then feeding that back into the model, and asking it to predict the next step, and so forth. In this way, generated samples can be of arbitrary length. With notewise encoding, I achieved much better results by not always sampling the most likely prediction by the model. Taking the top prediction only usually leads to generations that fall into a short musical pattern that repeats over and over. To handle this, I use *truncated sampling,* where I sometimes sample from the top *n* most likely guesses (according to the probabilities predicted by the model). In this way, the generations are more diverse, and a single prompt can produce a large variety of samples.

Critic and Composer Classifier

I created a critic model that attempts to classify whether a sample of music is human composed or neural net composed. "Real" samples come from the original human-composed midi files (encoded into text files). "Fake" samples are created by the generator LSTM. This can be used as a way to score musical generations. In this way, I can select the best samples from a batch created by the generator.

I also created a composer model that attempts to classify which human composer created the music sample. Training samples come from the original human-composed midi files (again encoded into text file). This can be used to create an Inception-like score for the models. Models score highly when each individual generation is very much like a specific human composer, and when on average there are generations that match all of the different human composers. A model would score poorly for generating only pieces like Chopin, and never generating ones like any of the other humans.

Alternately, it is interesting to combine these two model scores. One could create a novel musical style by finding samples that the Critic considers "real", but which the Composer Classifier does not match to any human composer.

Results

Chordwise vs. Notewise

The chordwise and notewise encodings each have some interesting advantages and disadvantages. In general, the chordwise version is better at learning long-scale musical patterns, but does not generalize well beyond the training set. The notewise version creates samples that sound very good in the short term, but which do not have longer scale (15-30 second) structure.

Chordwise Representation	Notewise Representation
● Able to memorize and recreate 45 seconds of Mozart (sometimes with chromatic distortion) ● More difficult to generalize beyond training set ● More variety in quality of generations ● Chromatic Distortion ● Currently unable to handle different durations of notes ● Usually more effective to sample top prediction at each timestep	● Creates musical sounding samples even when prompted with music it has never seen before ● Handles 12-step sample frequency well (able to generate both 16ths and triplets smoothly) ● Able to create musically coherent rhythmic patterns ● Easily extends to chamber music ● Does not create long scale patterns well ● Easily handles long note durations ● Usually most effective to sample from the top 3 predictions at each timestep