CS230

Modular Music Synthesis

Eden Y Wang
Department of Computer Science
Stanford University
eyw@stanford.edu

Drew Gao
Department of Mathematics
Stanford University
drewgao@stanford.edu

Abstract

Music generation is an emerging field of generative modeling. We worked to bridge the gap between symbolic and waveform music representations by modularizing the generation process into a symbolic generation step and a style transfer augmentation step. We find that we are able to tractably iterate on the generation and style transfer step, leading to compositions with comprehensive melodies that can be accented with non-symbolic waveform features. We find that a modular approach to music generation is promising for work down the line.

1 Problem

Advancements in machine learning based methods for music generation is an emerging field with novel research in a variety of tasks, including melody extraction, midi representation, and feature extraction. Generative models of music currently work in two broad domains: raw audio (ie time-frequency) representations and symbolic representations (often in the form of Midi files). While less computationally intensive than working with raw audio, working with symbolic representations of music tends to result in rather "flat" sounding outputs which do not fully capture timbre/texture or other atmospheric (as opposed to melodic) features in music. This issue is especially prominent when considering certain sub-genres of electronic music, including but not limited to noise, ambient, breakcore and techno. These genres tend to feature simple melodies (if any melody is present at all) while the most identifiable features tend to be textural, atmospheric, rhythmic or otherwise atonal.

Our project seeks to address some of the limitations of working with symbolic representations of music by first generating symbolic representations of music then using neural style transfer to reintroduce texture, atmosphere and other features not capture by symbolic representations. As a test case for our model we will be attempting to generate realistic sounding tracks from the aforementioned genres. These genres in particular make good test cases for our approach due to the relatively simple melodic structure and extensive use of non-melodic features, but we hope insights from this work will be useful for other music generation tasks as well.

More specifically we hope to experiment with tractable and efficient music generation methods by using an ensemble model comprised of two modules: a midi melody generation model and an style transfer model. Our ultimate goal is to be able to feed in the same inputs (music / genres) through both modules to generate music from the provided distribution. Through this composite model, we hope to circumvent traditional difficulties with working with high sample rates and computationally intensive models, allowing us to modularize the music generation process.

2 Related Work

Researchers have explored a variety of neural network architectures for music generation—both in the time-frequency domain as well as the symbolic domain (1). Modular approaches have also been previously explored, as can be seen here (2).

There is less existing research about audio style transfer/texture synthesis. However, style transfer and texture synthesis methods from computer vision show promise in being applicable to audio domain data as well (3) (4). In particular, (5), showed the potential of a single convolution layer with a large amount of randomized convolutional layers to outperform even more advanced audio style transfer models such as those based around the VGC-19 network.

Thus, in order to achieve our goal of generating realistic pieces of dissonant/ambient electronic music in a computationally efficient manner, we will implement a RNN based module for MIDI melody generation in conjunction with a shallow CNN based style transfer module.

3 Dataset

For our generative step, we used a set of publicly-available samples of short Lofi chord sequences and samples. These samples were gathered through previous work on zacharykatsnelson/Lofi-Hip-Hop-Generator and comprise a set of melodies and midi note patterns similar to those commonly found in Lofi tracks. These tracks are meant to represent the distribution of Lofi music, though it is limited in size and cannot be automatically generated. For the style transfer module we prepared excerpts from five textually complex electronic music tracks with varrying degrees of melodic prominence (obtained from Drew's personal collection) as examples of styles we want to apply to our generated MIDI melodies[1].

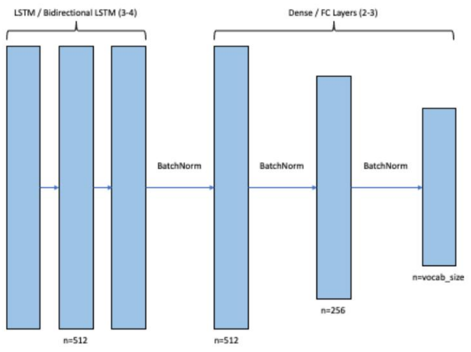
4 Methods

4.1 Code

Our current code for each module, as well as data files and additional results can be found at our Github repository[2]. Included in the folder are several samples outputted from our current generation step.

4.2 Midi Generation

For the generative step, we developed a RNN model based on Zachary Katsnelson's lo-fi hip hop generator (6). This model is comprised of several recurrent layers followed by a sequence of dense layers, with a softmax output and cross-categorical loss. Dropout and batch normalization is also used as regularization techniques. A diagram of our overall model architecture can be seen below:



[1]The tracks are as follows: "Eurocore MVP" by Venetian Snares, "Hungry" by Show Me the Body feat. Dreamcrusher, "Insular" by Operant, "No Rabbit No Life" by Blawan, and "Tha" by Aphex Twin

[2]https://github.com/eyw520/cs230-modular-music-synthesis

Training is performed on the lo-fi dataset above using an Adam optimizer. In addition to our above dataset, we experimented with various mp3 to midi converters to provide a richer dataset. Midi notes from our dataset are pre-processed, parsed, and compiled into sequences for use as input. The pre-processing step strips out outlier notes and determines what midi note properties (pitch, duration, intensity) will be relevant to our model. Notes are parsed into sequences, with the first k notes serving as the input and predictions being performed on the $k + 1$th note.

After completing 500 training epochs on k-note sequences from our midi dataset, we sample new midi notes and chords by generating a random input sequence and sampling notes, using generated notes as part of the next input sequence. We select the most probable note in the prediction, whether that be a chord, a rest, or a single note. Notes are written to a stream, then saved as a midi output.

4.3 Style Transfer

We based our style transfer step based on source code provided by (Ulyanov 2016). Given 2 audio files (either .mp3 or.wav) of equivalent length as input, this algorithm applies the short term fourier transform to both audio files, this gives us a 2-D spectrogram representation of magnitudes of audio frequencies over time. Since humans perceive audio intensity on a logarithmic scale, we then take the natural logarithm of all magnitude values and use the resulting spectrograms as inputs for the style transfer network. Each of these spectrograms is then treated as a 1x(Time)x(Frenquency) volume on which one-shot style transfer is applied using a 1 layer convolutional neural network with 4092 random filters each covering 11 time steps and the full range of frequencies.

More specifically we use gradient descent in order to compute a spectrogram, G that minimizes the following cost function:

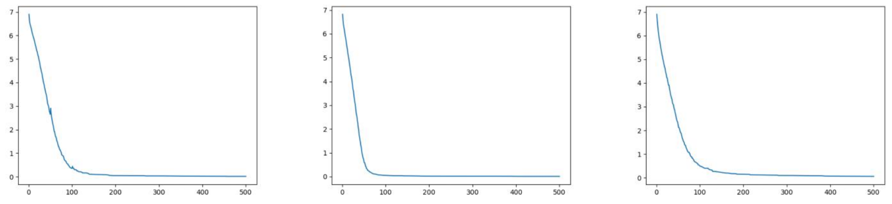
$$J(G) = \alpha J_{content}(C, G) + J_{style}(S, G)$$

where G represents the generated spectrogram with the content from one input, C and the style of another input G, and $J_{content}$ as well as J_{style} are defined in the canonical way given in (3). We achieved the best results when G was initialized to be equal to C with random noise added (rather than C itself or a purely random input). The alpha parameter roughly corresponds to the prominence of the original content in the final result.

5 Results

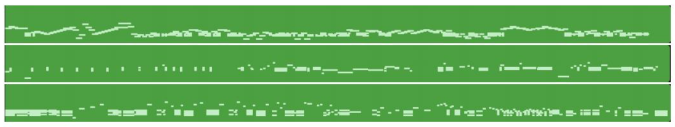
5.1 Midi Generation

The original RNN model, without any changes, sampled the most probable raw note sequences independent of note duration, rests, and velocity, leading to melodies lacking in compositional depth. Our implementation aimed to expand on this by accounting for more complex rhythms, in addition to a larger model architecture and the use of bidirectional LSTM layers. Hyperparameters, such as the number of notes preceding prediction and the incorporation of rests and note duration, were experimented with throughout the iterative process. Broader architecture structure, such as number of dense layers and types of recurrent layers, was also explored. The training loss for several 500 epoch runs are shown below (L to R: LSTM, no rests; Bidirectional LSTM with rests; Bidirectional LSTM, without rests).



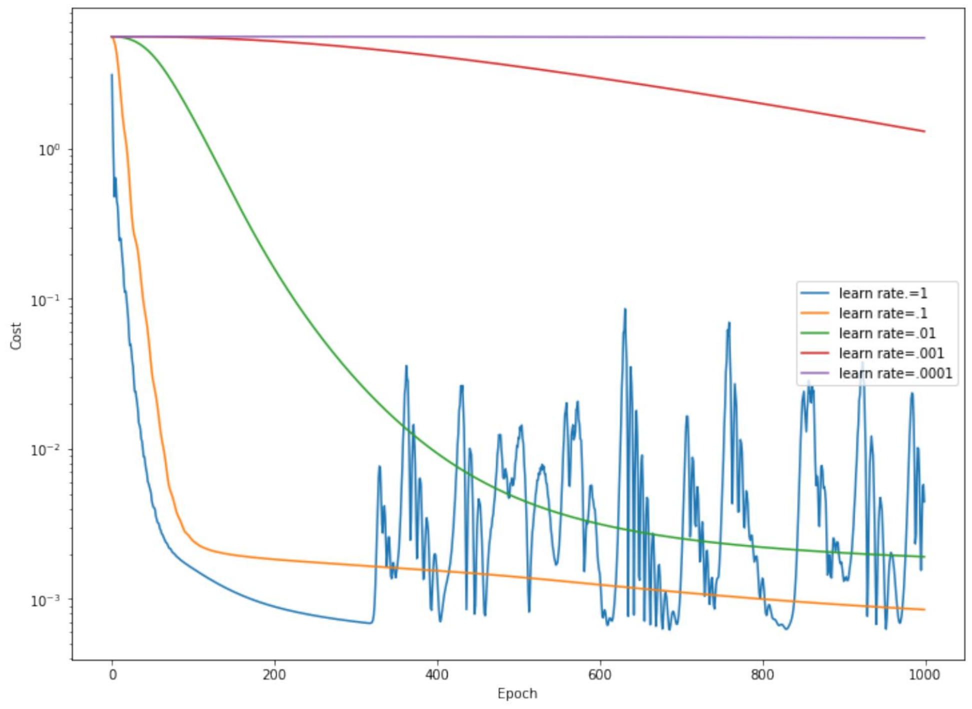
Output evaluation was performed subjectively; evaluation metrics in the realm of generative music models remains an emerging field with no clear consensus on the best approaches (7). Though reporting model loss remains an indicator of model performance, it is insufficient in determining how well our output fits our ideal distribution. And despite the Inception score having achieved recognition in the realm of image generation, no widely-accepted adaptation currently exists for waveform / music generation. As results varied greatly during our experimentation based on hyperparameter and preprocessing implementation, we were able to identify several aspects of improvement. We primarily focused on making improvements in the diversity of our output melody and rhythm.

We found that our changes were able to introduce a greater variety of melodic and rhythmic elements when incorporating both rests and complex beat duration. Beginning with uniform sequences of notes, we were able to experiment with both more complex rhythmns and melodies that took into account the full depth of the midi data. Samples can be found in our Github repository, but some midi tracks generated through our experiments can be seen below. Note the increasing complexity of the tracks through the generative process (displayed in Logic Pro; from top to bottom: constant note durations and offsets, an intermediate result with rests and varied note durations, our final output after 500 epochs of training).



5.2 Style Transfer

As with the MIDI generation module it is important to note that for this algorithm, lowest cost does not necessarily correspond to best subjective response. However, we did notice that for fixed values of alpha, it was indeed the case that outputs with the lowest cost tended to also be the least noisy output in which both subjective content and style features could be identified. The following plot shows the cost for various learning rates over the course 1000 epochs when applying style transfer.



We found that across all test cases, a relatively high learning rate of .1 resulted in the quickest convergence to the lowest cost. Thus we fixed this parameter while adjusting alpha for the rest of our results. We also fixed the number of epochs at 500 since improvements in the output are much less prominent beyond this point.

The structure of our cost function tells us that higher values of alpha result in the content loss being given increased weight when trying to optimize the output. This means that running this module with lower values of alpha results in outputs with more prominent style features, while running the module with higher values of alpha results in outputs with more prominent content features. This can be observed in figures 1 and 2 in the appendix:

Furthermore it is also important to note that the value of the objective cost function does not necessarily correspond to the subjective/human perceived quality. Thus in order to identify the best output we have to manually test various values of alpha for each content/style pair. Also of note is the fact that more texturally complex

Despite the relative simplicity, this algorithm is able to relatively successfully combine textures which are similar to the "style" input file and combine them with the main melody of the "content" file while ensuring the original content is still recognizable in the output. However, the generated textures are often noticeably derivative of the original file (oftentimes sounding like the original "style" input with a filter applied rather than a uniquely generated output). Furthermore when using "style" files with prominent melodic features (such as a loud, consistent tone), these features will remain present in the output in addition to the desired textural features.

5.3 Overall Results

Overall, we found that our greatest bottleneck was in our dataset; though midi provides a succinct and tractable representation for musical melodies, we found it difficult to find rich midi datasets that fully represented a given genre of music. Though we experimented with midi converters, we quickly found that existing converters were incapable of melody extraction and incorporated extraneous noise into the training process. Our dataset used pre-made, short sequences that were designed to capture lo-fi musical elements, but scaling our model to different genres would require manual data grooming to achieve comparable results. In summary, working with midi representations simplifies our model at the cost of requiring more domain knowledge in dataset curation.

6 Conclusion and Future Work

Although our model is far from perfect, our results do show the potential for a more modular approach to music generation where high level symbolic level features and low-level waveform level features are handled separately.

Model success and performance was limited primarily by our lack of domain knowledge when approaching the problem of music generation. Throughout the process of working on this project, we have developed a far greater idea of the tradeoffs associated with various waveform and symbolic approaches. We also have several ideas for natural extensions on this project; for instance, a third module capable of midi melody extraction can be used to create a comprehensive pipeline capable of generating the computational benefits described in our problem statement. In terms of pre-processing, ensuring that our data comes from the same distributions with the same compositions rules surrounding rests, note durations, and velocities can ensure a more replicable distribution. However, we believe that the underlying principle behind dividing the task into various modules remains logical, and that a modular approach reflecting the underlying data generation process has much promise in generative applications.

7 Contributions

Eden was the primary contributor for the implementation and tuning of the midi generation module while Drew was the primary contributor for the implementation and tuning of the style transfer module. Both authors contributed equally to report writing, literature review, analysis of results, and other high level components of the project.

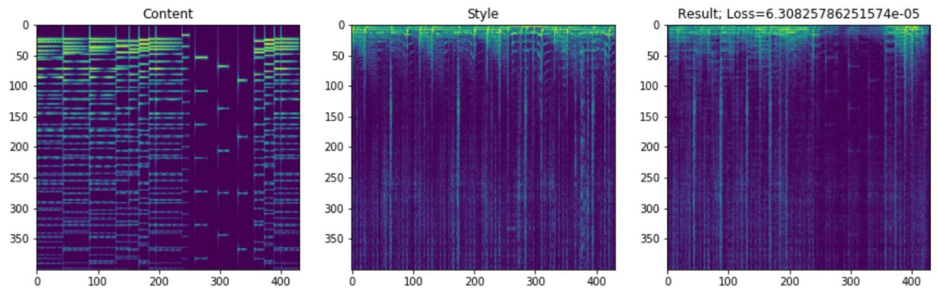


Figure 1: Style Transfer with Alpha=.001

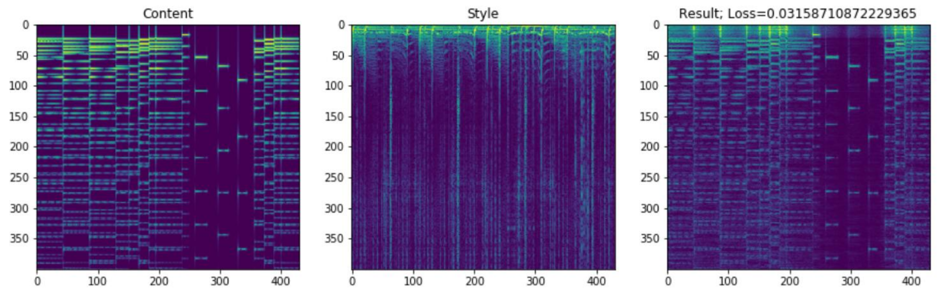


Figure 2: Style Transfer with Alpha=1

8 Appendix

References

[1] 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.

[2] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, “Jukebox: A generative model for music,” *arXiv preprint arXiv:2005.00341*, 2020.

[3] L. A. Gatys, A. S. Ecker, and M. Bethge, “A neural algorithm of artistic style,” *arXiv preprint arXiv:1508.06576*, 2015.

[4] I. Ustyuzhaninov, W. Brendel, L. A. Gatys, and M. Bethge, “Texture synthesis using shallow convolutional networks with random filters,” *arXiv preprint arXiv:1606.00021*, 2016.

[5] J. Chen, G. Yang, H. Zhao, and M. Ramasamy, “Audio style transfer using shallow convolutional networks and random filters,” *Multimedia Tools and Applications*, vol. 79, no. 21, pp. 15043–15057, 2020.

[6] Z. Katsnelson, “How i built a lo-fi hip-hop music generator,” 2020.

[7] L.-C. Yang and A. Lerch, “On the evaluation of generative models in music,” *Neural Computing and Applications*, vol. 32, no. 9, pp. 4773–4784, 2020.