schuBERT & paGANini : Exploration of Deep Learning methods for Music Generation

Arunachalam Muthu Valliappan
Georgia Institute of Technology
arunachalam.mv@gatech.edu

Kunaal Ahuja
Georgia Institute of Technology
kunaal@gatech.edu

Abhishek Patria
Georgia Institute of Technology
abhishek.patria@gatech.edu

Sai Abhishek Pidaparthi
Georgia Institute of Technology
spidaparthi6@gatech.edu

Abstract

We explore different approaches to generate music in MIDI format using Deep Learning methods. Initially we explore simple methods for transferring one style of music to another in a very similar way to Style Transfer in Images using different loss functions and gradient ascent. We also explore if a Transformer trained to classify music to its composer can learn enough about music to be used generatively analogous to the generative capability of ImageNet models. In the second phase we use VAEs with LSTMs and GANs to generate music or to transfer music styles.

1. Introduction

We systematically explore the strengths and weaknesses of different generative models for the purpose of Music Generation. We do this in 3 levels. In the First level, we explore the generative capabilities of models trained only on supervised tasks like the classification problem to identify the composer of the piece. We use Transformer encoder stacks to treat an entire piece of music as one data point to generate music in another style. In the Second level, we explore the generative capabilities of VAEs and LSTMs on Music generation and Style Transfer tasks. Finally, we explore the state of the art approach of using GANs for Music Generation.

1.1. Current Approaches and Our Work

Google's Magenta project which uses the Music Transformer with Relative Attention and OpenAI's MuseNet which uses the GPT-2 Decoder to generate MIDI files are the two most recent approaches for Music Generation. These models are fairly large and have too many parameters. [9] [2]

Melody RNN is a light weight approach which is computationally feasible with little resources but it only focuses on Melody elements of music and would be able to represent only one Note per time interval.

We have built Music Generation methods from scratch right from Data Representation to Modeling and Output Generation. Our approach tries to capture both the Melodies and Harmonies simultaneously and apply the same to smaller models for generation of Music. We have also explored methods methods for bridging two songs together which takes a piece from its original style and smoothly makes a transition to another song with elements of both songs present in it.

2. Approach

2.1. Data and Data Representation

We collected MIDI files from both Classical composers and Modern pop musicians to transfer styles since it would be easier to identify transitions between Classical and Pop for the general populace compared to differences in styles between classical composers. The data set we collected includes works of classical composers like Bach, Beethoven, Brahms, Chopin, Debussy, Haydn, Liszt, Mendelssohn, Mozart, Paganini, Rachmaninow, Schubert, Schumann and Tchaikovsky. Our pop data set includes Backstreet Boys, Beatles, Britney Spears, Coldplay, Queen and Nirvana. In total we have 20 different classes for training our classification and other models (14 classical and 6 pop).

The objective of the project is to capture the styles of both the Melodies and Harmonies of different composers. Since multiple Chords, Notes might be pressed simultaneously, we introduced a new approach to represent Data. MIDI files usually capture Notes of 128 different pitches spanning close to 11 octaves compared to only 88 Notes of a Full length Piano. We represent our data as a binary array

of 128 0/1s for each time beat of the song where 1 represent a pressed key of a piano. Each song is a Tensor of Shape (Song length x 128). This input data format notably does not capture the velocity of pressed keys, and the different instruments in the original MIDI file. [4] [8] All output is generated with Piano as instrument.

2.2. Generative Capabilities of Supervised Models

The inspiration for this approach stems from the impressive generative capabilities of ImageNet models which were trained only on Supervised Classification tasks. To replicate the same effect, we obtain a data set of about 1500 songs from 20 different composers (both classical and pop) of varying length and generate 45 second snippets of them to have a fixed data size. We then train a model with a 6 layer Transformer encoder stack and an FC layer to predict which composer class each song belongs to. We obtained very good classification performance for this model on the composer prediction task. The model reached a CE loss of about 0.6 for 20 classes and had an average accuracy of about 85%.

Similar to image generation in DeepDream or in Neural Style Transfer, we added an L2 regularization loss over the entire song and introduced a continuity loss which penalizes changes from the previous state of the piano to ensure continuity and smooth transitions. The final 0/1 output was generated by choosing thresholds over outputs which leads to an optimal average of 2-3 notes per beat.

The Optimizer tried to maximize the score of the song for a different composer with the addition of the Regularization and Continuity losses mentioned above. Different constants were multiplied to these 3 losses to ensure that they were comparable to each other and all the loss parameters contribute to the total loss and gradient flow.

Though the scores were maximized enough to predict the new composer (analogous to adversarial attacks to create fooling images), the generated music sounded like random white noise. Our hypothesis is that, discriminative models can learn enough to be used generatively only when they are forced to, in situations where there are multiple input which look very close to each other but have subtle difference which would force the model to learn about the world view and learn these subtleties. The presence of only 20 classes and far fewer data points in comparison to images has proved that such an approach is not viable in the application of Music Generation.

2.3. Variational Autoencoders with Transformer Encoders

This VAE model was built with an Encoder made up of 6 stacked Transformer Encoder Layers and a Decoder made up of 6 stacked Transformer Encoder Layers. A significant difference between this model and the one used by Magenta and OpenAI is that we generate the entire fixed length song simultaneously instead of generating one Output at a time using decoders. Since the input is a (n x 128) binary vector, it is already in a valid format for embedding. Learn-able Positional Embeddings were generated for each position in the song and were added to both the Encoder and Decoder Inputs.

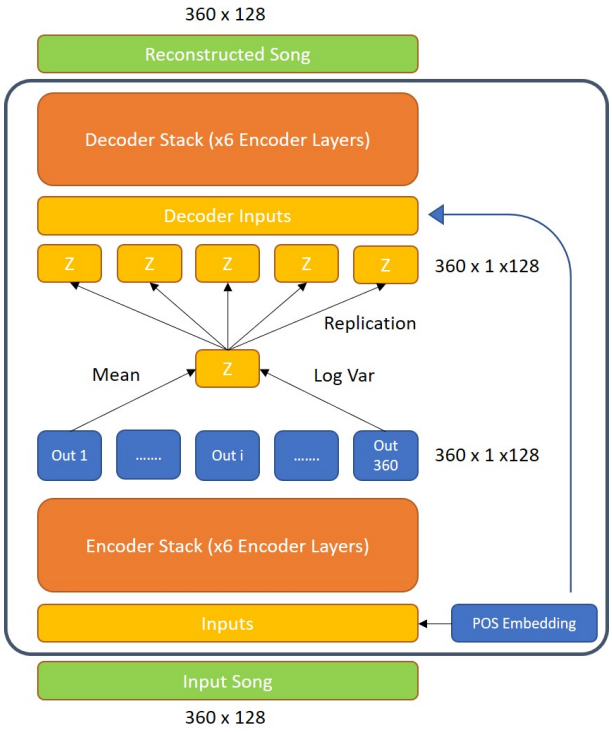

Figure 1. Transformer VAE Architecture

The dimensions of the encoder layer were maintained the same as the input dimensions and the output of the Encoder layer is a tensor of shape (n x 128). The first output and last outputs of the encoder layers of shape (1x128) were taken as mean and variance of the latent variable distribution and the generated z was replicated for all n positions to feed as input to the decoder output. A sigmoid function was applied on decoder outputs to convert them into probabilities and BCE loss was used along with KL divergence to train the model.

Upon inspecting the samples generated, we were able to determine that the outputs were better than our first approach and did not involve any random noise. But the outputs did not resemble the input and the model plateaued with high reconstruction losses. The outputs generated were mostly loops of the most commonly used notes in the middle Octave.

We were able to identify 2 main problems with the approach that is causing this.

1. Though this model is able to generate outputs simultaneously instead of sequentially, the lack of probabilistic

sampling of the notes removes all randomness and causes the transformer to generate loops of the same note patterns.

2. Absolute positional encoding and Self Attention are not very helpful in Music Generation. Relative embeddings or Relative Self-Attention is much more effective for capturing structures in Music as demonstrated by Magenta and in TransformerXL.

Due to computational constraints, we addressed these problems by using LSTMs instead of TransformerXL as follows in the next section. [1]

2.4. Variational Autoencoders with LSTMs

This VAE model was built with an Encoder made up of a 3 layer LSTM and a Decoder made up of another 3 layer LSTM. This decoder generates one Output at a time like traditional models, but it is considerably slower in the process. The final hidden state is used to generate 20 dimensional latent vectors for mean and variance of z.

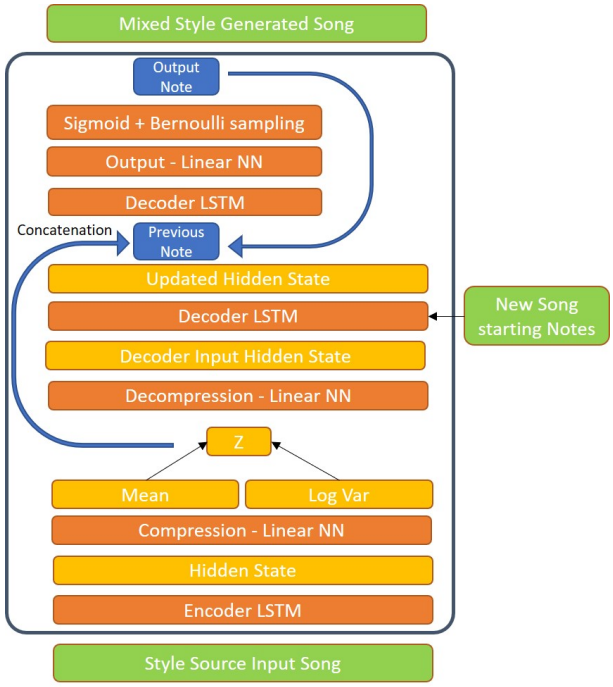

Figure 2. LSTM VAE Architecture

This model incorporates fixes for the two problems faced above. The sigmoid over model outputs are considered as Bernoulli probabilities for turning the Note On/Off. Sampling Bernoulli probabilities from inbuilt PyTorch functions also maintains the computational flow graph for AutoGrad to function. The randomness introduced by this sampling procedure ensures that the decoder does not produce continuous loops of the same notes. The LSTM does not have the same problem as the Transformer had with positional encoding as the more recent notes generally have a greater effect on the current output, compared to transformers where this is purely determined by self attention without any account of relative positioning.

The sampled z vector is used to generate the hidden state for the decoder. This z vector is also concatenated with every input for the decoder. A sigmoid function was applied on decoder outputs to convert them into probabilities and BCE loss was used along with KL divergence to train the model. One of the main challenges during training was posterior mode collapse due to vanishing KL divergence term. We used linearly increasing KLD weight with Beta VAE to prevent this.

We also tried using MSE loss, but since the outputs were 0/1, most of the outputs generated had 0 as the value with occasional 0.5 values in the middle octave which more or less represents the average of notes over time periods. BCE loss enabled us to generate output probabilities between 0 and 1, which can directly be used for Bernoulli sampling to get the output notes, and it also penalizes misclassification of 0s and 1s on a higher scale. Empirically, BCE loss helped smoother training, lower reconstruction losses and better quality outputs.

This VAE model was able to produce very good outputs (subjectively determined) and were able to transition between pop, classical styles smoothly incorporating the styles of both the songs given to the model. Some sample outputs are included in the zip.

2.5. Generative Adversarial Networks

A generative adversarial network (GAN) has two parts: The generator learns to generate plausible data. The generated instances become negative training examples for the discriminator. The discriminator learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.

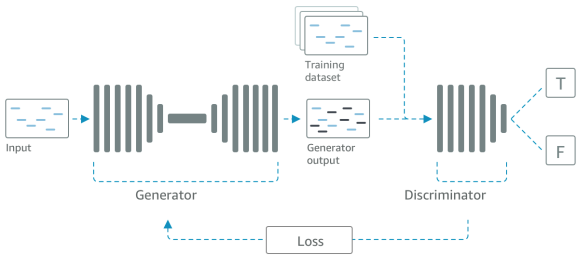

Figure 3. GAN Architecture

Generating music has a few notable differences from generating images and videos. First, music is an art of time, necessitating a temporal model. Second, music is usually composed of multiple instruments/tracks with their own temporal dynamics, but collectively they unfold over time interdependently. Lastly, musical notes are often grouped into chords, arpeggios or melodies in polyphonic music, and

thereby introducing a chronological ordering of notes is not naturally suitable. [6]

When training begins, the generator produces fake data, and the discriminator quickly learns to tell that it's fake. As training progresses, the generator gets closer to producing output that can fool the discriminator. Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases. Both the generator and the discriminator are neural networks. The generator output is connected directly to the discriminator input. Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights.

The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data created by the generator. It could use any network architecture appropriate to the type of data it's classifying. The discriminator connects to two loss functions. During discriminator training, the discriminator ignores the generator loss and just uses the discriminator loss. We use the generator loss during generator training, as described in the next section. During the training, the discriminator classifies both real data and fake data from the generator. The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real. The discriminator then updates its weights through backpropagation from the discriminator loss through the discriminator network.

While training, as the generator improves with training, the discriminator performance gets worse because the discriminator can't easily tell the difference between real and fake. If the generator succeeds perfectly, then the discriminator has a 50% accuracy. In effect, the discriminator flips a coin to make its prediction. If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback, and its own quality may collapse. [7] [3]

There is a plethora of information available and generative models are one of the most promising approaches towards analysing and understanding this treasure trove of data. Generative models have many short-term applications. But in the long run, they hold the potential to automatically learn the natural features of a dataset, whether categories or dimensions or something else entirely.

For GANs based models, our approach was inspired by the DCGAN network but we modified the network architecture for its use with a MIDI file. A colored image is generally structured as an RGB tensor with continuous values which is altogether different as compared to a MIDI file. A MIDI file is a binary matrix where a note is played or not. The main idea here is to train a GAN by taking 128*128 snippets of the various songs and then come up with new song sequences of the same size. [11] [5]

We first created discriminator with 6 2-D CNN layers and passing the output of the last layer to a Sigmoid layer to detect whether the input song sequence is from a real song or a fake (generated by generator). [10] Additionally, we used the batch normalization throughout the network in order to keep the network stable. A latent vector of size 1*128 was chosen randomly to initiate the generation process of new song sequences. A generator was built using a 6 Transpose Convolution layers. The input is provided as a randomly generated latent vector which is then amplified through the generator network to give a 128*128 sequence.

While training the discriminator-generator game we used a balanced approach by passing an equal number of real and generated sequences. We used BCE loss in order to train the model but found out that aggregating the batch loss by sum was more helpful as compared to taking the mean. The rationale behind choosing sum over mean was that the target matrix is of binary nature and taking the mean of the sequence-loss might would lead the discriminator into giving suboptimal results. The same was experimentally verified and carried over to the generator network. While first pass on the training, we faced a convergence issue where the generator and discriminator losses were not stabilizing. Decreasing the learning rate helped in this issue. Post first pass training, the model generated white noise. We then introduced L2 regularization in order to prevent overfit, specifically for discriminator resulting in better sounding outputs.

3. Experiments and Results

We have generated several output samples to examine from our VAE model with LSTMs which produced the best outputs. The samples have been included in the zip file along with the code. The output files are in pairs of composer_in.mid, composer_out.mid pairs. The files having a single composer name only are outputs which try to continue the same song in the style of the same composer. The files having multiple composer names are outputs which try to transfer styles slowly and smoothly from one composer to another.

Although all of our outputs are worth checking out, we picked some of our favourite output samples for discussion of our results. The two style transfer outputs generated for Nirvana-Chopin pair starts off with the fast pace and low pitched drum beats initially and smoothly transitions into a more classical style.

We wanted to test the performance of our models on new artists whose work was not included in the training data. We still use the source style songs from the artists in training data set. We tried to continue the song 'Thunderstruck' from 'AC/DC' which was not present in the training data and continued it with randomly sampled z vectors from Beethoven and Chopin. The model still produced very good quality outputs which retain the fast paced rhythm of Thun-

derstruck with the drum beats at lower pitches with some classical style melodies playing at higher pitches.

VAE

Qualitatively, we could clearly differentiate between the apparent random noises produced by our earlier models like Regularized Gradient Ascent and Transformer VAEs and the musicality and melodiousness of LSTM VAEs. Quantitatively, the BCE reconstruction loss obtained from the LSTM VAE was 50-55% lower than the reconstruction loss obtained from the Transformer VAE.

The outputs generated are meant for multiple instruments but were rendered with the same instrument for simplicity. Examining the Notes and Chords of the generated stream, we can clearly see the quality of the outputs generated and we think that the outputs might be slightly better than how they sound on our single instrument rendering, for example using drums as instruments for very low pitches instead of piano should make the output much clearer. According to our subjective opinions, the generated music is of good quality, but not comparable to original composed music, though for some of the generated pieces, we found it hard to distinguish the VAE generated pieces from the original ones. We consider this project a success as the VAE was successful in fooling our team members who acted as the Original vs Fake discriminators.

GAN

The GAN model is able to generate outputs simultaneously due to the very nature of the CNNs being used from a 128-dimension probability distribution. The model mainly generated loops from the original songs but loops quickly shift their octaves according to the data provided. We found that when the training data includes classical composers like Mozart and Bach the octave shifts pretty frequently as these composers are notorious for using the whole piano board but when the data is void of classical composers the generated songs remain within 2 octaves.

References

[1] Colin Raffel Curtis Hawthorne Douglas Eck Adam Roberts, Jesse Engel. A hierarchical latent vector model for learning long-term structure in music, 2019. Google Brain.

[2] Jakob Uszkoreit Noam Shazeer-Ian Simon Curtis Hawthorne Andrew M. Dai Cheng-Zhi Anna Huang, Ashish Vaswani. Music transformer: Generating music with long-term structure, 2019. Google Brain.

[3] Julian McAuley Chris Donahue, Huanru Henry Mao. The nes music database: A multi-instrumental dataset with expressive performance attributes, 2018.

[4] Miller Puckette Chris Donahue, Julian McAuley. Adversarial audio synthesis, 2019. UC San Diego.

[5] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis, 2019.

[6] Li-Chia Yang Yi-Hsuan Yang Hao-Wen Dong, Wen-Yi Hsiao. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment, 2017.

[7] Yi-Hsuan Yang Hao-Wen Dong. Convolutional generative adversarial networks with binary neurons for polyphonic music generation, 2018.

[8] Ian Simon Monica Dinculescu Jesse Engel Kristy Choi, Curtis Hawthorne. Encoding musical style with transformer autoencoders, 2020. Stanford, Google Brain.

[9] Rafal Jozefowicz & Samy Bengio Oriol Vinyals, Andrew M. Dai. Generating sentences from a continuous space, 2015. Google Brain.

[10] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[11] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation, 2017.

4. Appendices

Experimental Details

The architecture of all our models, both the successful and failed ones are provided in the approach. Only one of our models, the VAE-RNN provided results which very closely resemble original music. Our output results were subjectively evaluated for the quality of music.

Source Code

Source Code is provided as Jupyter Notebooks in the zip file. The folder named 'GAN Code Notebooks' contains CNN and GAN based generative methods for simultaneous Music generation in parallel. The folder 'VAE Code Notebooks' contains all VAE based methods and a few other experiments. The notebook 'PreReq_Functions' contains all preliminary functions for converting MIDI files to numpy arrays and combining the steams from different instruments. The notebook 'Encoder Train' examines if we can use a classification model for generative purposes. Our final model is 'VAE-RNN' which implements a VAE with an LSTM encoder and decoder.

Computing Infrastructure

VAEs were trained on a GTX 1660 Ti with 6 GB VRAM. The GANs were trained on Google Cloud with the computing credits provided.

Datasets

The dataset was put together manually from various sources. It has 20 folders of MIDI files, each with a different composer/artist. The composers included in order are 0-bach,1-backstreetboys,2-beatles,3-beethoven,4-brahms,5-britneyspears, 6-chopin,7-coldplay, 8-debussy, 9-haydn, 10-liszt, 11-mendelssohn, 12-mozart, 13-nirvana, 14-paganini, 15-queen, 16-rachmaninow, 17-schubert, 18-schumann,19-tchaikovsky. All midi files were converted to .npz format using the functions in 'PreReq_Functions' notebook. The npz zip can be downloaded from 'https://github.com/Arunachalam-M/Music-Generation-VAE/blob/master/Dataset_npz.zip'

Results

The folder 'Results - Output MIDI files' contains all our generated output samples along with the corresponding inputs. The folder is uploaded both to Gradescope and in the Github repository. 'https://github.com/Arunachalam-M/Music-Generation-VAE'