Music Generation with Generative Adversarial Networks

Varun Rawal
Carnegie Mellon University
CMU, Pittsburgh, 15213
vrawal@andrew.cmu.edu

Abstract

Our work primarily delves into ways for music-piece generation through more advanced deep learning techniques like GANs and VAE-GANs, that appear to be still in nascent stages of machine learning research paradigm of content generation of any kind: images, text and music. In the current piece of work, we were also able to accomplish our objective to compare the music generation tasks using Neural Network models based on several different GAN architectures. The experiments have been carried out upon Beethoven's music pieces in MIDI formats. The deep learning approaches followed here take these MIDI melody files, followed by converting them to binary or normalized vectors and then encoding them to be as inputs to our models. The primary objective is to be able to provide these models with arbitrary notes and let them begin amending the pieces gradually until one finds them producing good pieces of music. The GAN models seem to require a lot more data to prevent overfitting to the fed datasets, as well as a lot more time to train to convergence in a generalized way.

1 Introduction

The pilot studies on deep learning-generated music tend to reveal that the generation of music still persists as a challenging task, owing to lack of global structure in the music. Our work adheres to the well-known and famous Beethoven's style of music compositions and attempts to not consider more than one variant so as to avoid any further learning complications. For simplicity, we use MIDI formats of music sequences to learn complex polyphonic structure in music.

GANs are a state-of-the-art method for generating high-quality images. To our best of knowledge, GANs have also lately been frequently used to reconstruct text sequences, and also in combination with auto-encoders, for example they have been augmented with auto-encoders that use a series of five LSTM layers [1].

The dataset samples used consist of polyphonic MIDI files with music composed by Beethoven.

Comparing and contrasting models for music generation can give more insight into one of the most fundamental questions of computer science and machine learning : "Can creativity be automated?"

2 Background and Related Works

2.1 Introduction to Generative Adversarial Networks

Generative Adversarial Networks are composed of two models:

- The first model is called a Generator and it aims to generate new data similar to the expected one. The Generator could be asimilated to a human art forger, which creates fake works of art.

- The second model is named the Discriminator. This model's goal is to recognize if an input data is 'real' — belongs to the original dataset — or if it is 'fake' — generated by a forger. In this scenario, a Discriminator is analogous to the police (or an art expert), which tries to detect artworks as truthful or fraud.

How do these models interact? It can be thought of the Generator Network as having an adversary, the Discriminator Network. The Generator (forger) needs to learn how to create data in such a way that the Discriminator isn't able to distinguish it as fake anymore. The competition between these two teams is what improves their knowledge, until the Generator succeeds in creating realistic data.

Mathematically Modeling a GAN

Though the GANs framework could be applied to any two models that perform the tasks described above, it is easier to understand when using universal approximators such as artificial neural networks.

A neural network $G(z, \theta_1)$ is used to model the Generator mentioned above. It's role is mapping input noise variables z to the desired data space x (say images). Conversely, a second neural network $D(x, \theta_2)$ models the discriminator and outputs the probability that the data came from the real dataset, in the range (0,1). In both cases, θ_i represents the weights or parameters that define each neural network.
As a result, the Discriminator is trained to correctly classify the input data as either real or fake. This means it's weights are updated as to maximize the probability that any real data input x is classified as belonging to the real dataset, while minimizing the probability that any fake image is classified as belonging to the real dataset. In more technical terms, the loss/error function used maximizes the function D(x), and it also minimizes D(G(z)).

Furthermore, the Generator is trained to fool the Discriminator by generating data as realistic as possible, which means that the Generator's weight's are optimized to maximize the probability that any fake image is classified as belonging to the real datase. Formally this means that the loss/error function used for this network maximizes D(G(z)).

In practice, the logarithm of the probability (e.g. log D(...)) is used in the loss functions instead of the raw probabilies, since using a log loss heavily penalises classifiers that are confident about an incorrect classification.

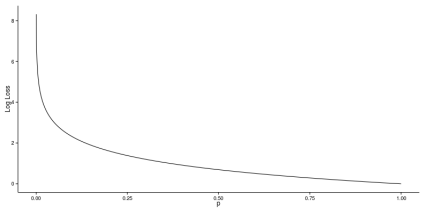

Figure 1: Log Loss Visualization: Low probability values are highly penalized

After several steps of training, if the Generator and Discriminator have enough capacity (if the networks can approximate the objective functions), they will reach a point at which both cannot improve anymore. At this point, the generator generates realistic synthetic data, and the discriminator is unable to differentiate between the two types of input. Since during training both the Discriminator and Generator are trying to optimize opposite loss functions, they can be thought of two agents playing a minimax game with value function V(G,D). In this minimax game, the generator is trying to maximize it's probability of having it's outputs recognized as real, while the discriminator is trying to minimize this same value.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Figure 2: Value Function of Minimax Game played by Generator and Discriminator

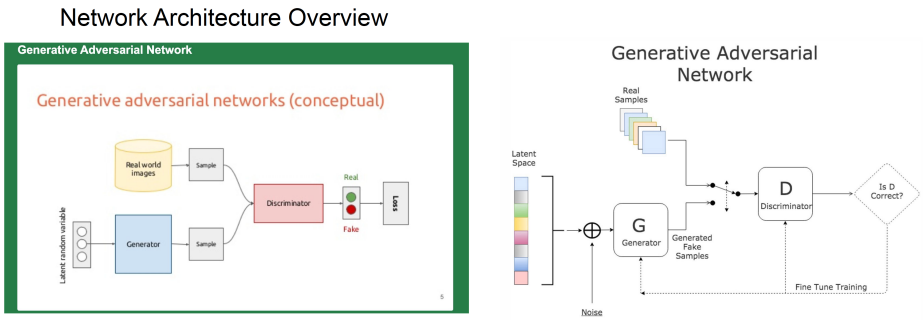

Figure 3: figure-GAN-architecture-overview

2.2 Comparison across Pre-Trained GAN Models [for Music Sample Generation Task]

2.2.1 Architecture - Type-1 : GANSynth (source : Magenta Art Models [Google AI]) [1]

GANSynth uses a Progressive GAN architecture to incrementally upsample with convolution from a single vector to the full sound. Rather than generate audio sequentially, GANSynth generates an entire sequence in parallel, synthesizing audio significantly faster than real-time on a modern GPU. Unlike the usual autoencoders that used a time-distributed latent code, GANSynth generates the entire audio clip from a single latent vector, allowing for easier disentanglement of global features such as pitch and timbre.

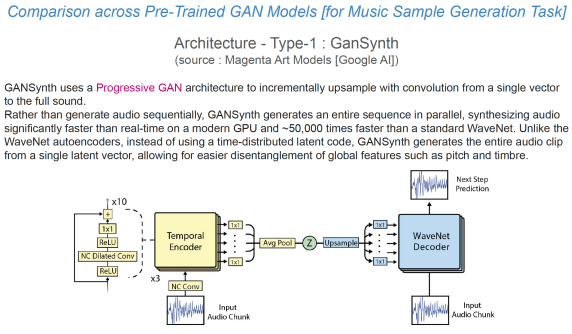

Figure 4: Architecture - Type-1 : GanSynth (source : Magenta Art Models [Google AI])

Comparison across models : Quality of Audio Waveform Visualizations

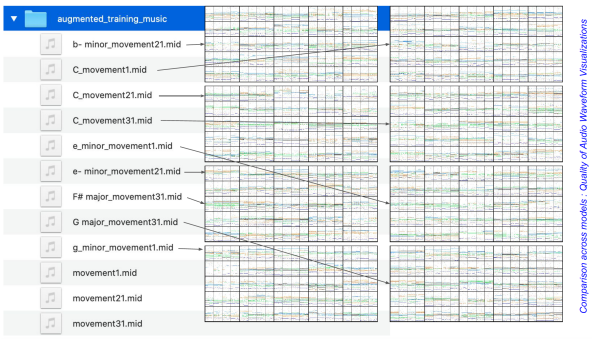

Figure 5: Audio Waveforms plotted for Musical Piece samples generated by GanSynth on different music seeds

Comparison across models : Human Listening Evaluations

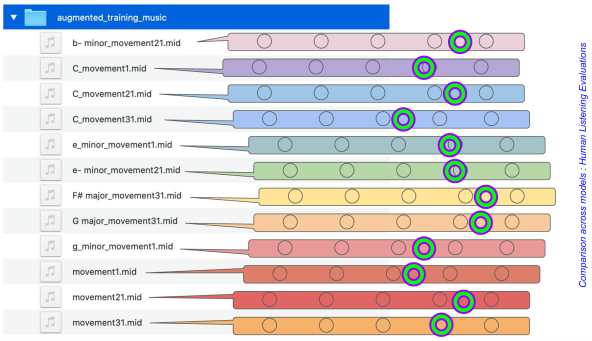

Figure 6: Human-Listening based Ratings on 1-5 Scale for Evaluating Musical Piece samples generated by GanSynth on different music seeds

2.2.2 Architecture - Type-2 : MuseGAN (source : Music and AI Lab,Research Center for IT Innovation, Academia Sinica) [2]

The MuseGAN Model presents completely different architecture when compared to the GANSynth Magenta Model.
The model consists of two parts: a multitrack model and a temporal model. The multitrack model is responsible for the multitrack interdependency, while temporal model handles the temporal dependency.

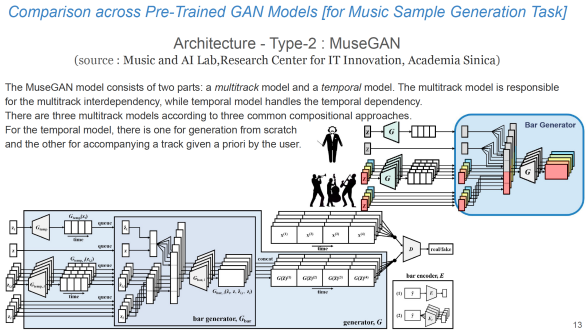

Figure 7: Architecture - Type-2 : MuseGAN (source : Music and AI Lab,Research Center for IT Innovation, Academia Sinica)

Comparison across models : Quality of Audio Waveform Visualizations

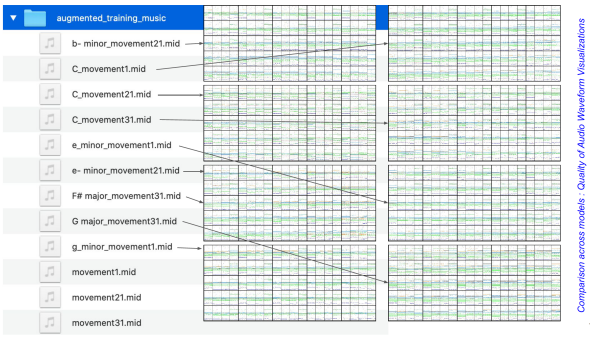

Figure 8: Audio Waveforms plotted for Musical Piece samples generated by MuseGAN on different music seeds

Comparison across models : Human Listening Evaluations

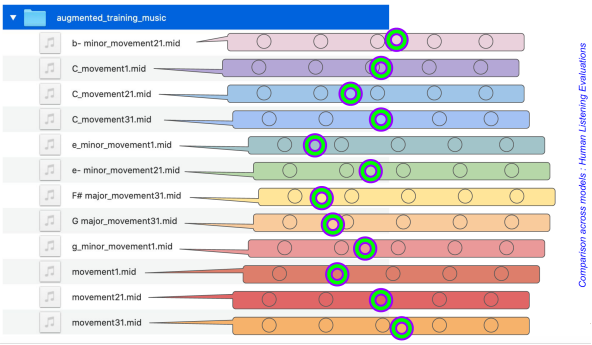

Figure 9: Human-Listening based Ratings on 1-5 Scale for Evaluating Musical Piece samples generated by MuseGAN on different music seeds

2.2.3 Architecture - Type-3 : GANs-&-Reels (source : CS Department; Memorial University of Newfoundland) [3]

The GANs&Reels Model presents a contrasting architecture when compared to the GANSynth Magenta Model as well as the MuseGAN Model.

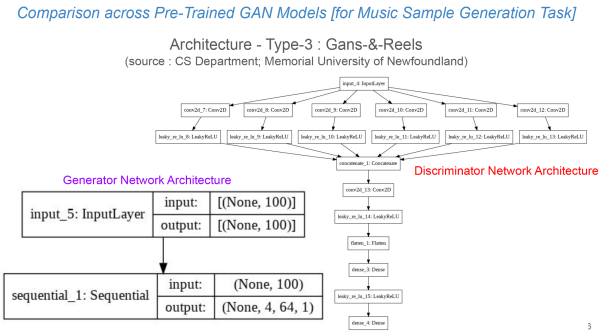

Figure 10: Architecture - Type-3 : GANs-&-Reels (source : CS Department; Memorial University of Newfoundland)

Comparison across models : Quality of Audio Waveform Visualizations

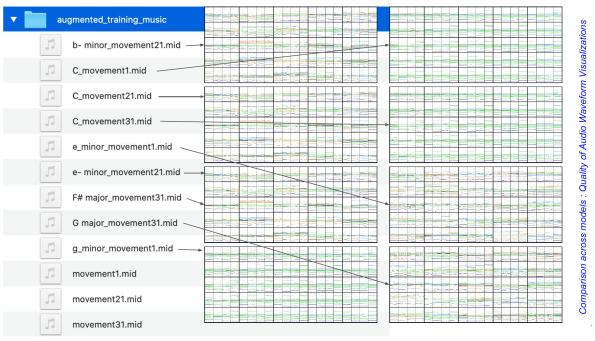

Figure 11: Audio Waveforms plotted for Musical Piece samples generated by GANs&Reels on different music seeds

Comparison across models : Human Listening Evaluations

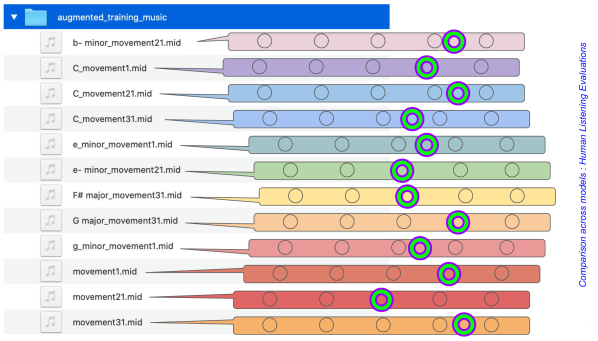

Figure 12: Human-Listening based Ratings on 1-5 Scale for Evaluating Musical Piece samples generated by GANs&Reels on different music seeds

3 Methods and Approach

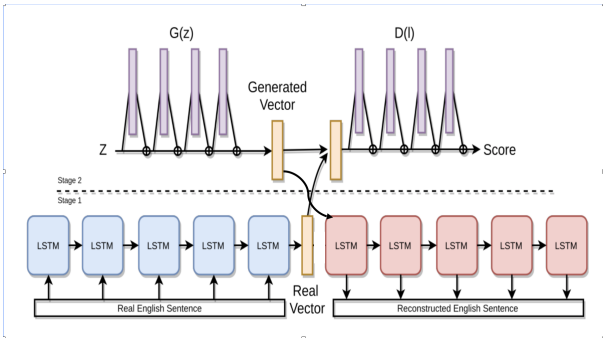

Figure 13: figure-GAN-1

The VAEs provide a convenient way to encode the music note sequences into latent patterns. Using the GANs to learn generation of this latent space allows us to create music by feeding in seed music pieces and then reconstructing the GAN output music pieces through the decoder of the VAE model. This VAE-GAN like approach has been our major intention to go for, when working on the GAN aspect of this project. The GAN training phase and hence the hyper-parameter tuning for the same takes a lot of time. Hence, we shifted our secondary focus to using the pretrained GAN models : GANSynth (Google AI Magenta model) [1], GANs & Reels [3], and MuseGAN [2] to study how changing the GAN model architectures dictate the change in generation of audio samples from a given seed of music. Such comparative studies were done in both the aspects : with respect to Audio waveform visualizations as well as human-listening based evaluations through ratings on a 1-5 scale.

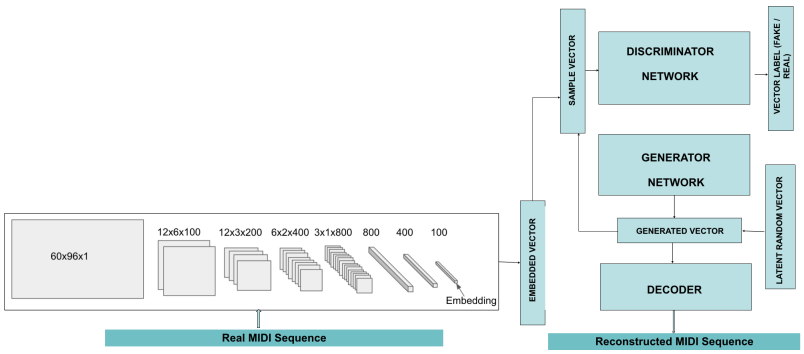

Figure 14: [figure-GAN-2] : Generative Adversarial Architecture for Music Reconstruction

The process-flow of the music-generation task in our GAN experiment arrangement could be outlined as follows, as illustrated in the Figure-GAN-2; In our MIDI data experiments, we treat a MIDI audio message as a single token, and thereby, considering each unique combination of notes across all time steps as separate tokens. As a naïve attempt towards the music-generation task, we have avoided the data augmentation step as usually followed in the other related works.

Initially, we attempted to train a rudimentary GAN model (refer Figure-GAN-3) for music generation without any additional architecture.

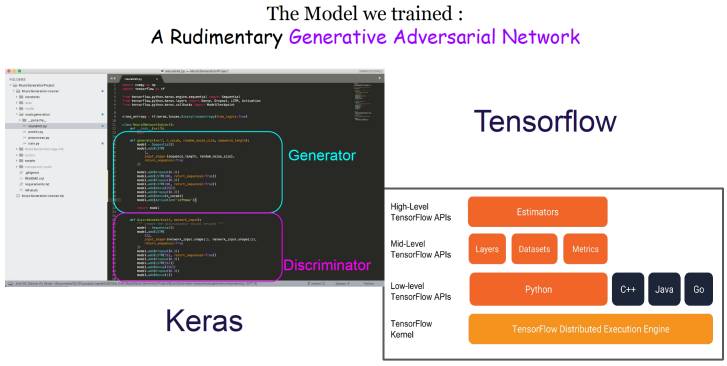

Figure 15: [figure-GAN-3] : Rudimentary GAN Model Implementation

However, the primary challenges we faced were during the evaluation phase; Figure-GAN-4; when executing the sample generation tasks on trained models : we found only one Note / Bar / Chord (Musical Entity) being generated at a time. This makes it quite challenging to evaluate and compare the sample generation quality on cross-validation datasets; Figure-GAN-4.

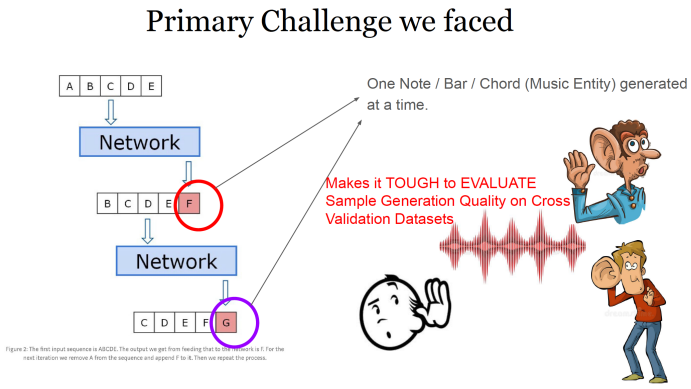

Figure 16: [figure-GAN-4] : Primary Challenge we faced with GAN Evaluation

After sufficient brainstorming and going through the earlier works, we attempted at solving this challenge by exploiting a VAE-GAN-like architecture; Figure-GAN-5.

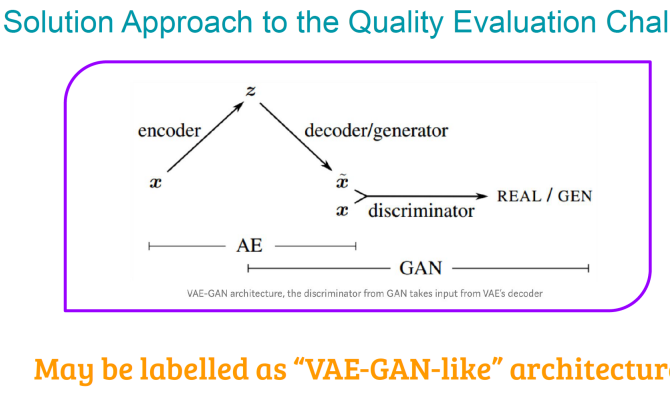

Figure 17: [figure-GAN-5] : VAE-GAN-like architecture as a solution approach to the Evaluation problem

This approach essentially revolves around the fact that adopting this approach is equivalent to asking the Generator and Discriminator networks to compete amongst themselves on the Latent space instead of trying to deal with the original musical space; This then makes it possible for us to reconstruct the generated musical pieces back to audio waveforms in MIDI / MP3 format, ready for quality evaluation tasks; Figure-GAN-6.

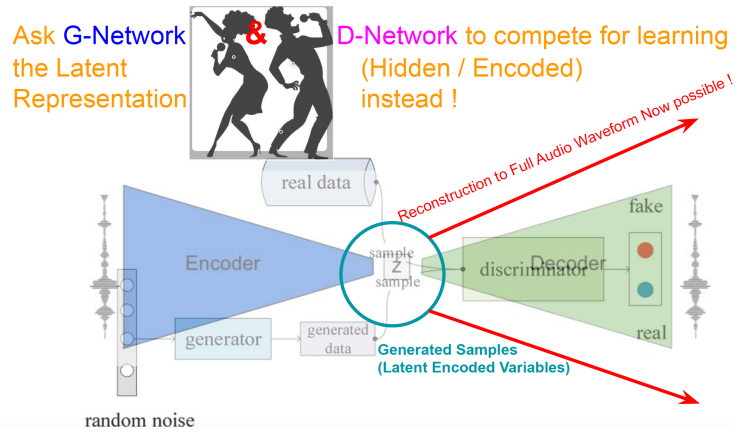

Figure 18: [figure-GAN-6] : G and D Networks compete on the Latent Embedding Space instead

This procedure of fitting a pair of encoder-decoder architecture in the middle of the pipeline would certainly allow the output produced by the Generator Network to be decodable back to MIDI sequences. This essentially asks for having available, an embedding representation for music similar to word embeddings in text-related tasks[1]. We use the concept of autoencoder: a network that consists of two parts — the encoder and the decoder. Autoencoders encode input data as vectors. Given an input, the job of the encoder is to project it to a space of (much) smaller dimensionality, and the job of the decoder is to reconstruct the input given this compressed representation. They are generally found to be quite useful in dimensionality reduction tasks; that is, the vector serving as a hidden representation compresses the raw data into a smaller number of salient dimensions.

We deploy a 2-component neural network (GAN) architecture to reconstruct the next musical note in a sequence. Autoencoder model involves a series of convolutional layers followed by fully connected layers leading to a 100-dimensional embedded feature vector, now ready to train the GAN model. We generated music by feeding short random seed sequences into our trained GAN model as 100-note sequence. This generated vector is fed into the discriminator alongside a stream of vectors taken from the actual, "Beethoven-only" MIDI dataset. The discriminator takes in both real and random sequences and returns probabilities, numbers between 0 and 1, with 1 representing a prediction of authenticity and 0 representing that of a fake feed.

4 Datasets

The experiments have been carried out upon Beethoven's music pieces in MIDI formats. The deep learning approaches followed here take these MIDI melody files, followed by converting them to binary or normalized vectors and then encoding them to be as inputs to our models. The following figures explain our approach taken for processing and dealing with the input datasets i.e. MIDI audio files used for training our models. The dataset samples used consist of polyphonic MIDI files with music composed by Beethoven.

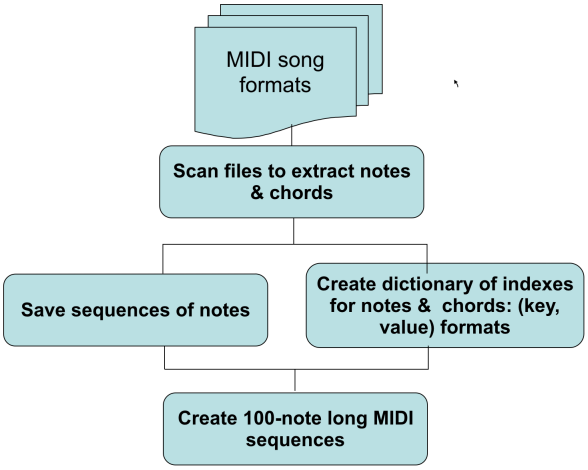

Figure 19: Dataset-generation for GAN model

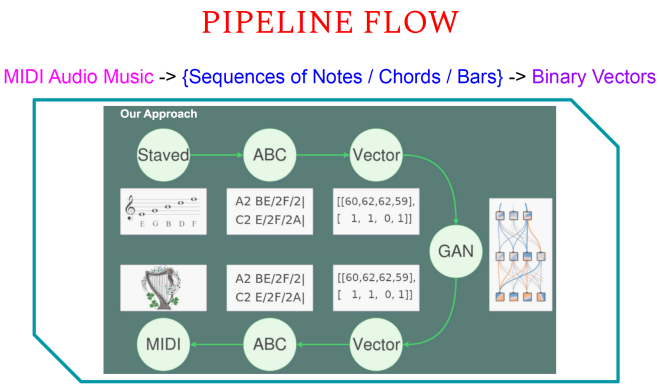

Figure 20: figure-GAN-Pipeline-Flow

5 Experiments and Results

5.1 Experimental Setup

Our architecture borrowed the hyperparameters such as number of layers, hidden unit size, sequence length and learning rate from GAN model used in [1].

Borrowing the GAN architecture introduced in [1], we also attempted to work with different novel combinations when trying to innovate and improve upon the past work, and hence used the idea of using convolutional layers instead of LSTM units (as used in [1]'s architecture) and also using convolutional layers in the autoencoder part of our architecture.

We first train our model on the "Beethoven-Only" 12 MIDI file dataset. We trained for 50 epochs per batch, where each batch consisted of music embeddings generated from 2 MIDI files per batch; hence, training was performed for 6 batches in 300 steps; referred as x-points in visualization plots (figure-GAN-7 to figure-GAN-9). It took about 3 hours per 50 epochs on a GPU. Due to time-constraints, we were forced to work with one set of data, batch-size and sequence-length and couldn't execute our pipelines on different combinations of hyperparameters so as to fine-tune.

Table 1: Hyperparameters in our GAN Model

	Part	
Name	Type / Description	# (Number / Cardinality)
Hidden	(CNN+FC) Layers (Encoder)	07
Hidden	(CNN+FC) Layers (Decoder)	07
Token Embedding Size	Dimension Length	100
Batch Size	Number of Samples per Batch	02

5.2 Results

The generator network of trained GAN model was able to produce reconstructed smoothly audible music piece in parallel. Figure-GAN-7 and Figure-GAN-8 denote the error plots during GAN model training for more than 240 epochs. After 200 epochs, the discriminator tends to show signs of being able to learn the patterns observed in the 'real' data and generator learns to generate reconstructed music, identified as real by the discriminator. The length of reconstructed music was only 4 seconds, though, irrespective of training cycle (epoch) count. This is evident through the loss function plot diminishing to zero at the output layer of discriminator after 200 epochs (figure-GAN-7), while the loss function continues to increase at the output layer of generator network (figure-GAN-8). The quality evaluation of reconstructed music is justified by the probability plot of discriminator (functioning as classifier, here) that converges almost to 1 after 200 epochs (figure-GAN-9) which was used by us to assess the quality of our work as a metric as opposed to the use of human-evaluated scores assessed on the quality of music in past literary works.

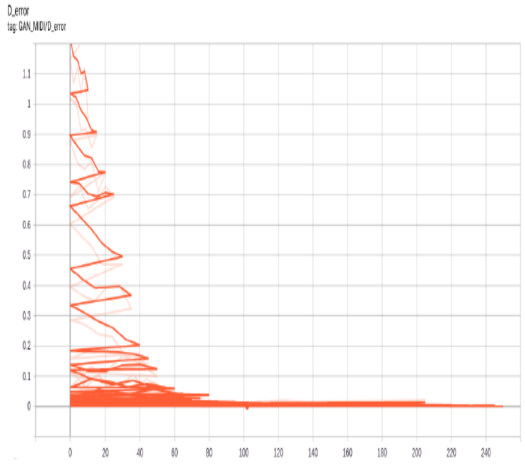

Figure 21: [figure-GAN-7] : Loss Function (error) plot for Discriminator of GAN

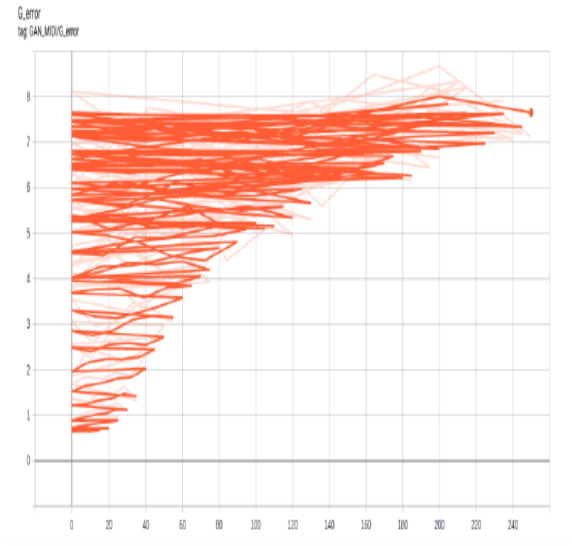

Figure 22: [figure-GAN-8] : Loss Function (error) plot for Generator of GAN

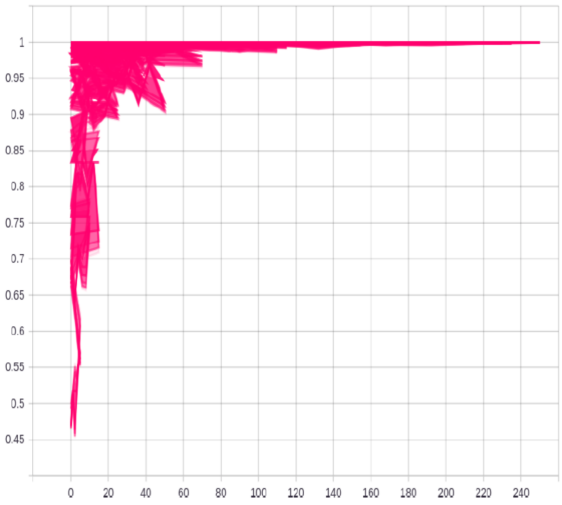

Figure 23: [figure-GAN-9] : Probability plot of Discriminator (GAN) : p(Identifying as Real (Not Fake))

However, we note that given the scenario, framework and setup of our experiment, the results we obtained certainly correspond to the case of Overfitting as is quite obvious from the high-degree of mismatch between the number of training samples used and the complexity of our GAN model architecture.

6 Conclusion and Future Work

This work has demonstrated that music-content generation with numeric encoding scheme as input is possible with the use of Generative adversarial networks (GANs). The work group also foresees to perform further experiments by network training across genres and larger corpus sizes to see how well the preliminary results generalize. After the models are trained, there exists much more scope to warm start the models using songs composed by humans. Interesting future directions include investigating the effect of modifying the architectures of

discriminator and generator networks on music generation performance. Also, LSTM and Gated Recurrent Units (GRU) networks can be chosen as the alternative design configurations of discriminator and generator components to effectively model long-term dependencies in thematic structure of musical pieces and produce compositions that sound unique and musically coherent.

Furthermore, we intend to exploit transfer learning techniques so as to share weight parameters of pre-trained models across different architectures by setting up experiments developed and evolved to adapt for cross-transfer tasks among separately trained models.

Even more advanced research could proceed in the direction of covering the domain of songs with lyrics, which would require deploying techniques and algorithms for Lyric Parsing and Speech to Text Conversion exploiting RNNs and LSTMs for sequence pattern exploration.

7 References and Citations

[1] GANSynth: Adversarial Neural Audio Synthesis, Jesse Engel and Kumar Krishna Agrawal and Shuo Chen and Ishaan Gulrajani and Chris Donahue and Adam Roberts, 2019, `https://openreview.net/pdf?id=H1xQVn09FX`

[2] MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment Hao-Wen Dong,* Wen-Yi Hsiao,* Li-Chia Yang, and Yi-Hsuan Yang (*equal contribution) in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018. `https://arxiv.org/abs/1709.06298`

[3] GANs & Reels : Creating Irish music using a Generative Adversarial Neural Network, Mitchell Billard, Robert Bishop, Moustafa Elsisy, Laura Graves, Dr. Antonina Kolokolova, Vineel Nagisetty, Zachary Northcott, Heather Patey `http://www.cs.mun.ca/~kol/GANs-n-reels/index.html`

[4] Karthik Chintapalli. Generative Adversarial Networks for Text Generation — Part 3: non-RL methods. "Becoming Human – Artificial Intelligence Magazine. 2019.

[5] Mason Bretan M., Oore S., Eck D. amd Heck L. (2017) Learning and Evaluating Musical Features with Deep Autoencoders `arxiv.org/pdf/1706.04486.pdf`