Latent Walking Techniques for Conditioning GAN-Generated Music

Logan Eisenbeiser

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfilment of the requirements for the degree of

Master of Science
In
Computer Engineering

Committee

Ryan Gerdes
Laura Freeman
Yue J. Wang

August 13, 2020
Arlington, Virginia

Keywords: Music Generation, Latent Walking, Conditional Generation, Generative Adversarial Network

Latent Walking Techniques for Conditioning GAN-Generated Music

Logan Eisenbeiser

ABSTRACT

Artificial music generation is a rapidly developing field focused on the complex task of creating neural networks that can produce realistic-sounding music. Generating music is very difficult; components like long and short term structure present time complexity, which can be difficult for neural networks to capture. Additionally, the acoustics of musical features like harmonies and chords, as well as timbre and instrumentation require complex representations for a network to accurately generate them. Various techniques for both music representation and network architecture have been used in the past decade to address these challenges in music generation.

Latent Walking Techniques for Conditioning GAN-Generated Music

Logan Eisenbeiser

GENERAL AUDIENCE ABSTRACT

Artificial music generation is a rapidly developing field focused on the complex task of creating neural networks that can produce realistic-sounding music. Beyond simply generating music lies the challenge of controlling or conditioning that generation. Conditional generation can be used to specify a tempo for the generated song, increase the density of notes, or even change the genre. Latent walking is one of the most popular techniques in conditional image generation, but its effectiveness on music-domain generation is largely unexplored, especially for generative adversarial networks (GANs). This paper focuses on latent walking techniques for conditioning the music generation network MuseGAN and examines the impact and effectiveness of this conditioning on the generated music.

Contents

1 Introduction

Artificial music generation is a rapidly developing field focused on the complex task of creating neural networks that can produce realistic-sounding music. Generating music is very difficult; components like long and short term structure present time complexity, which can be difficult for neural networks to capture. Additionally, the acoustics of musical features like harmonies and chords, as well as timbre and instrumentation require complex representations for a network to accurately generate them. Various techniques for both music representation and network architecture have been used in the past decade to address these challenges in music generation.

The focus of this thesis extends beyond generating music to the challenge of controlling and/or conditioning that generation. Conditional generation involves an additional piece or pieces of information which are input to the generator and constrain aspects of the results. Conditioning can be used to specify a tempo for the generated song, increase the density of notes, or even change the genre. Latent walking is one of the most popular techniques in conditional image generation, but its effectiveness on music-domain generation is largely unexplored. This paper focuses on latent walking techniques for conditioning the music generation network MuseGAN and examines the impact of this conditioning on the generated music.

2 Features for Music Generation

How is music created? Where does the latest pop song's melody come from? How did classical composers like Beethoven or Bach know what notes to write? By and large, the answer is a moment of inspiration and an adherence to the laws of music theory. Music creation is undoubtedly an art; however it is also formulaic. Good music, or music generally regarded as pleasing to listen to, is constrained by numerous rules and intricacies which make up the field of music theory. This is a complex topic which is largely tangent to this paper, but it is helpful to understand that there are fixed rules which can tame an assortment of notes into meaningful music. If humans can master these rules, shouldn't a computer be able to as well?

2.1 Motivation for Artificial Music Generation

Synthesis is one of the challenging, abstract tasks that distinguishes human intelligence from machine intelligence. As computers continue to improve, having already surpassed humans in terms of memory and computational power, one of their weakest aspects is the ability to create art, including music. The StyleGan2 network can already "draw" a photo-realistic human face as well as any human artist, provided it has the appropriate training data. An example of these generated faces is shown in figure 1. In contrast to these images, here is a quote from the authors of Jukebox, the most realistic music generation network

to date:

> "While Jukebox represents a step forward in musical quality, coherence, length of audio sample, and ability to condition on artist, genre, and lyrics, there is a significant gap between these generations and human-created music." [8]

There are a variety of motivations for research into AI music generation, ranging from technical to non-technical. Two motivations that will be discussed here are: the challenge of finding mathematical and algorithmic explanations for what makes music enjoyable to humans, and the curiosity at what an AI "musician" would create.

An additional motivation that deserves mentioning is the benefit to individuals and companies, particularly ones with an interest in music. OpenAI already has a suite of tools focused on artificially generated music for anyone to explore. Musicians can leverage these tools and other aspects of music generation AI for their work. As generated music becomes more enjoyable to listen to, it is not hard to imagine a film studio commissioning some music for a movie and generating the rest, reducing their cost. Artificial music is still in its very early years; even more applications, each with their own motivations, will likely arise in the coming years.

Technical Challenge as a Motivation Music generation is one of the more challenging tasks currently being tackled by state-of-the-art neural networks. In the introduction for the MuseGan paper, which introduces one of the leading audio generation models, the authors outline three reasons that music generation is a difficult task.

> "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." [11]

These challenges are evidenced by the relatively poor performance compared to tasks like image generation, object detection or speech recognition. While in these fields, neural network performance already exceeds human performance and is able to fool humans, the same can not be said for even the best performing music generators. Figure 1 shows a sample of images generated by StyleGan2.

The challenging task of generating realistic, synthetic music is compounded by the difficulty of evaluating the quality of said music. There is no widespread metric for music quality; indeed, this holds true for human-generated music as well. Classical musicians and rock musicians, for example, have differing opinions on what makes music "good". Researchers often resort to surveying a group of people on how pleasing and/or realistic their synthetic music sounds in

Figure 1: A sample of the results from StyleGan2, demonstrating the realism achievable in similar machine learning generation tasks [20]

order to measure their success. Figure 2 provides an example of the evaluation metrics used by the authors of MidiNet.

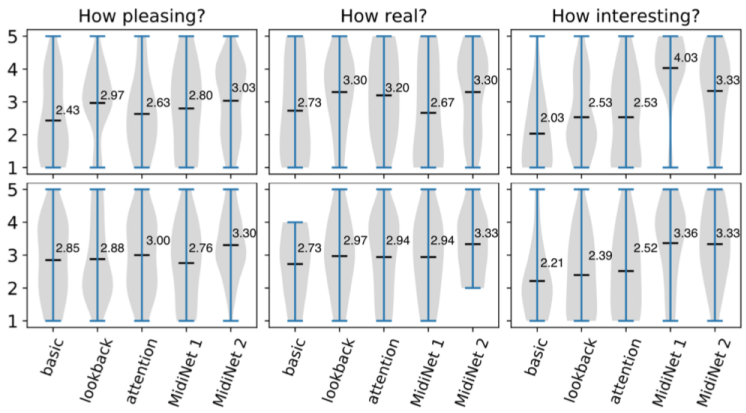


Figure 2: The evaluation metrics used by MidiNet, a survey of musicians (top) and non-musicians (bottom) on aspects of the generated music. [33]

For context, image generation is largely measured using the Fréchet Inception Distance (FID), a standard metric which measures the similarity between two sets of images where one set is real and the other is generated [19]. Standard and widespread metrics such as the FID allow for easy comparison of different networks; without any such metric, the field of music generation often requires an observer listen to samples of audio to get a sense of a network's performance.

There is still a lot of room for improvement, and the field of music generation is evolving at a breakneck pace. New techniques, models and applications are released mere months apart, constantly improving the quality of AI-generated music.

Curiosity as a Motivation Curiosity is a large motivating factor for scientists and non-scientists alike; the desire to know what a song made by a computer sounds like. AI music makes for a good headline, an example being OpenAI's Jukebox, which had over 50 articles written after its release earlier this year [8]. This could be derived from a desire to humanize artificial intelligence and hear something it creates, or an interest in the composing ability and emotional impact of songs made by an emotionless entity.

Consider this quote about a network trained to make human faces, "... it can only combine elements of the faces it has seen before to make a new face; given that it had only seen images of elderly peoples' face, it could not exercise the creativity to dream up a child's face." [17]. This leads into more philosophical questions, such as "Are human artists limited by their exposure to data?" or, "If a network was trained on all the images a human saw in their lifetime, would it have the same creative capacity as the human?" Whatever the motivation, there is substantial interest in the field of AI music generation, both inside and outside the research community.

2.2 Challenges for Music Generation

Generating music is a surprisingly complex task, due to the numerous relationships we expect to hear in any given song. Consider a simple chorale composed by the famous composer Johann Sebastian Bach in the early 18$^{\text{th}}$ century, shown in figure 3.

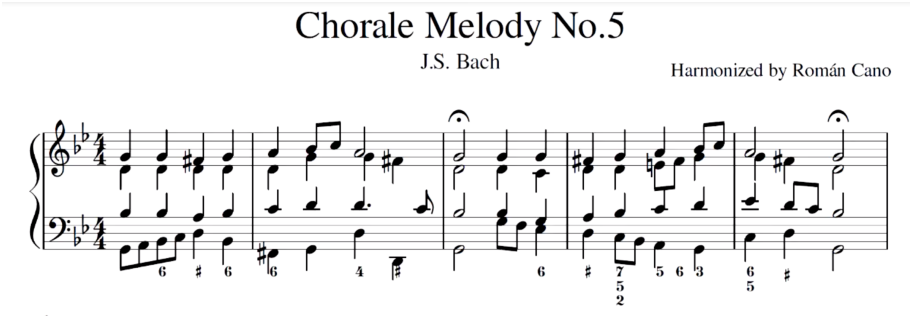


Figure 3: The first five bars of Chorale No. 5 by J.S. Bach

For those unfamiliar with reading music, the standard musical notation can be thought of as similar to a scatter plot, where the x axis represents time and the y axis represents note pitch. Two or more notes in the same x location represent multiple notes being played at once. This is a four-part harmony, so there are up to four notes being played at any given time. Note shape represents the duration of the note, with a quarter note (♩) being the "baseline" for a normal note length. If a song is played at 60 beats per minute, or bpm, the quarter note lasts exactly 1 second.

An eighth note (♪) takes up half the time of a quarter note. When pairing

two or more eighth notes together, they are linked (♫). A half note (𝅗𝅥) is the length of two quarter notes, or a note sustained for 2 seconds (at 60 bpm).

Time Signatures and Tempo Technically, the above statements for absolute note length only hold true in an X/4 time signature, which is the most common. However, the relationships are always true in that an eighth note is always twice as fast as a quarter note. Other time signatures such as X/8 or X/2 will be discussed later. Additionally, the conversion of note length to seconds will change depending on the tempo the song is played at.

2.2.1 Short-Term Structure

Looking at any note in the chorale, it is clear what time it is played, the pitch it represents and how long it is sustained. These can be considered properties of a note, and the local relationships between notes can largely be defined using these properties. "Local" in this case being used to include notes played within a short time frame: ± 1 beat.

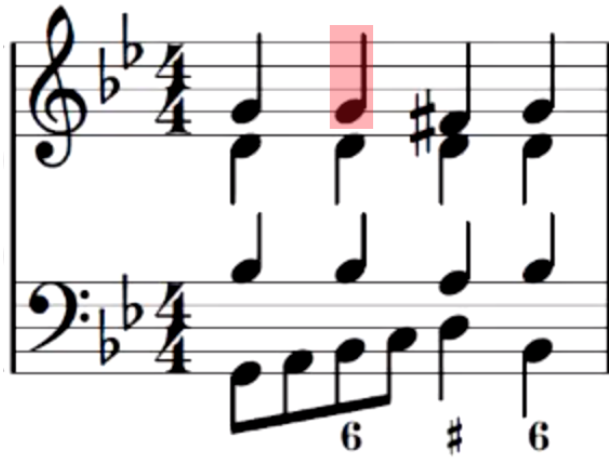


Figure 4: The first four beats of Chorale No. 5 by J.S. Bach

In figure 4, the note highlighted in red is played in the second beat of the song, has a duration of one beat, and has a frequency of 391.995 Hz (or a pitch of G_4). The local relationships for this note would be

- Difference in pitch from preceding / following note
- Difference in pitch from simultaneous notes (chord)
- Difference in length from preceding / following note
- Difference in length from simultaneous notes

Chords When more than one note is played at a time, the musical term chord is used. Chords can quickly become complex, but they are the reason some notes sound good played together, and others do not. This involves the acoustics of overlapping notes and the composite waveforms produced. Changing a single note in a chord can change the chord entirely. For instance, the red note in figure 4 starts out in a three-part chord since the bottom two notes are both B♭, but the chord changes to a four-part chord halfway through when the lowest note moves to a C.

Harmony These local relationships largely contribute to the harmony of a piece of music. Each note must be played at the proper time, pitch and duration to produce a pleasing sound. Notes spaced too close or too far, in either pitch or time, can lead to disharmonious and chaotic music.

Melody The sequence of notes as they progress through time creates the melody. This is both a short-term and a long-term structure; it is built one note at a time, with the interval between consecutive notes creating the most recognizable part of nearly any song. If you were to sing a song like "Happy Birthday" you would be singing the melody. One wrong note in a song can make a handful of notes played before and after the wrong note sound incorrect. Having a well-composed melody on the local scale is not enough to create good music; there needs to be long-term structure as well.

2.2.2 Long-Term Structure

In addition to the inter-note relationships, a key aspect of music is the long-term structure. This can range from a repeated motif or melody that is used multiple times throughout a piece to a reprise in a concerto that has similarities to an earlier movement. A well-known example is the Star-Spangled Banner, which repeats the melody of the four opening phrases. Long-term structures provide the listener with some context, and allow them to make connections with the music, often invoking more emotion and providing a better listening experience.

Repeating an entire section of music is so common that there is a notation specifically for repetition, :‖. Alternately, a composer may want to repeat parts of their music, but with modifications or additional layers added in. This provides that same context and familiarity to the listener, but also adds auditory intrigue, keeping the listener engaged and providing growth and development to the music.

Looking one more time at Chorale No. 5 by Bach, shown in figure 5, notice the upper notes in the first half of the piece are repeated exactly in the second half of the piece. This creates a familiarity between the two halves, providing structure and meaning to the music. To avoid monotony, the other three parts change the second time, to varying degrees, in both pitch and rhythm. These change the music enough to develop the piece, without feeling like a new piece of music started in the middle.

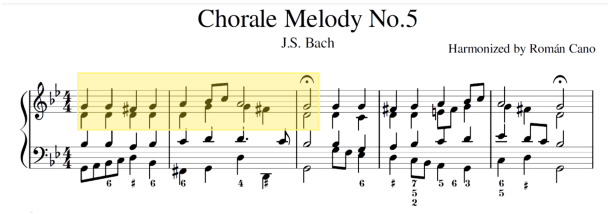


Figure 5: The first five bars of Chorale No. 5 by J.S. Bach

Long-term structure only becomes more complex when multiple lines of music are superimposed on each other as in the example piece. In addition to each of the four parts developing over time and adhering to the tone and rhythm rules to produce a meaningful melody, they must also work together to create a chord progression that works with the piece and highlights the different parts at the appropriate times, while avoiding conflicts like dissonance or chords that are in the wrong key. To further complicate matters, a wrong note can become intentional if it is used the same "wrong" way repeatedly, a tactic used by many modern composers like Eric Whitacre.

Long-term structure is one of the hardest aspects of generation for neural networks to capture, while also being one of the most crucial aspects for meaningful music.

> "...the long-term structure in the melody has posed great difficulty for designing a good model" [32]

The importance of structure, both long and short term, is the main focus of AI music generation, and has led to the development and application of many models to address these challenges.

2.3 Representations of Music

Music can be represented in many ways. For network training and generation applications, the two that are most useful are the symbolic domain and the audio domain. Both methods have pros and cons, with symbolic being simpler and cheaper, while audio is more accurate and computationally expensive.

2.3.1 Symbolic

Symbolic representation of music has been used for centuries. The traditional way of writing music, seen in figure 3, is symbolic. This means every note event is represented by a marking, with context dictating how loud, how long and what pitch to be played. However, this representation leaves a lot of room for interpretation. Two different musicians playing the same piece will almost certainly play it slightly different. This is not even accounting for the fact that this piece of music could be played on any number of instruments, or even sung

by a quartet. A symbolic representation of music relies on the interpretation of its symbols, and this limits the amount of precise audio information that it can encode.

On the other hand, symbolic representations are quite efficient for representing things like melody and harmony, which is why they have been used for so long. This is also a convenient way to provide information to a computer or neural network. Pianorolls are commonly used for symbolic music representation when training a neural network, and they can be expressed as a large binary matrix. Consider figure 6, each note event would be stored as a 1 and each empty space would be a 0. The two dimensions of the array represent pitch and time.

The biggest benefit of training a network with symbolic data is that the network does not need to encode information about how the music is actually produced, it just needs to generate a pleasing sequence of symbols. Another tool is needed to convert the pianoroll into the audio domain. While many networks have used symbolic music representations, as computational power has increased, there has been a shift towards using acoustic representations of music instead, a topic explored in the next section.

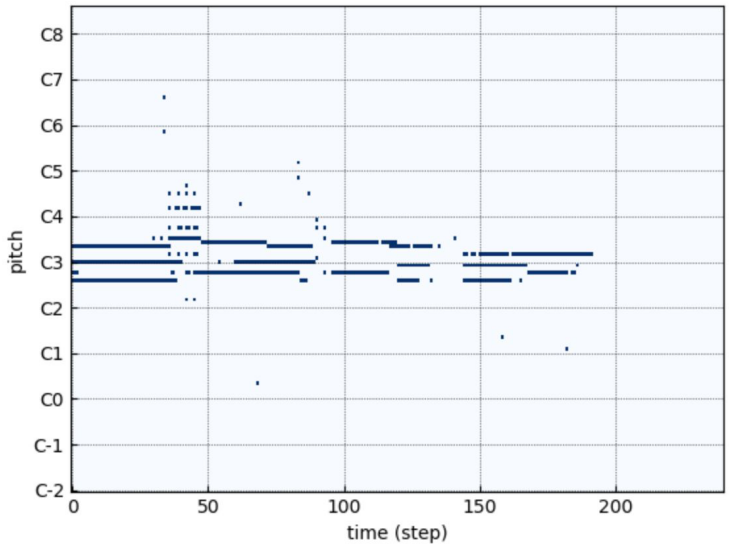
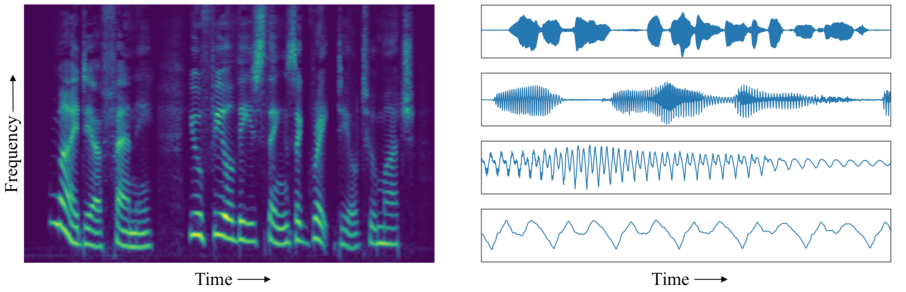


Figure 6: Symbolic representation of a piece of music using the common pianoroll format. [12]

2.3.2 Acoustic

Acoustic representation of music is a way to directly transmit audio signals, or to encode and decode the audio with minimal reconstruction loss. A perfect

example would be CDs or electronic music files like mp3. These methods record audio directly and encode them into a specific format which allows the recorded sounds to be reconstructed almost perfectly. Acoustic representations for music refer to any representation derived from the raw audio file, most commonly waveforms or spectrograms. Figure 7 shows a sample waveform and spectrogram. Both of these acoustic representations have advantages and disadvantages.



(a) Spectrogram representation (b) Waveform representation (1x, 5x, 25x, 125x magnifications)

Figure 7: An audio sample represented as both a spectrogram and a waveform. Image from MelNet: [30]

Waveforms for instance, are relatively simple to use, but they require a large quantity of data points to create even a second of sound. Waveforms are typically stored as a .WAV file, which is a series of values. These values represent amplitudes of the audio wave at each time step. Connecting these values creates a reconstruction of the waveform. With standard WAV files, 44,100 samples are needed for a single second of audio. This large amount of data can lead to issues with long term structure, where a network fails to create meaningful relationships between points that are on the order of seconds apart.

Spectrograms are a representation of audio in the frequency domain. Here is a description of the spectrogram format by the authors of MelNet.

> "[Spectrograms] are two-dimensional time-frequency representations which contain information about how the frequency content of an audio signal varies through time. Spectrograms are computed by taking the squared magnitude of the short-time Fourier transform (STFT) of a time-domain signal, i.e. $x = \|STFT(y)^2\|$." [30]

Spectrograms have not been widely used for music generation tasks, but they have been shown to be a viable format to generate and train on. Unless some breakthrough occurs, it is unlikely that spectrograms will see a surge in popularity for music generation tasks.

2.3.3 Which Representation is Better?

The main drawback of using acoustic representations of music is the increase in computational complexity. The amount of information encoded into either waveforms or spectrograms is orders of magnitude higher than the amount of information in a pianoroll. This makes it more difficult for a network to learn meaningful structures in the training data, as each network only has the capacity to model a finite amount of information. This in turn can lead to poor generation results if the network is not robust enough for audio data.

On the other hand, acoustic representations of music are able to more accurately represent the real-world samples used for training. The incorporation of information like instrument, volume or even words allows a network trained on acoustic data to produce much more realistic music than a symbolically-trained network. While papers like MuseGan had to implement complex network architecture to create coherent multi-track melodies, an acoustic representation can inherently encode the relationships between many instruments directly in the waveform [11]. As computational power increases, and networks become more advanced and are able to encode more information within their weights, networks trained on acoustic representations of music are becoming viable, often superior, options when compared to symbolic networks.

2.4 Summary

Generating music is a deceptively complex task. From the structures at varying time scales to the complex interaction of notes in a chord, there are numerous features of good music that often go unnoticed until they are not there. Many diverse approaches and techniques have been utilized to improve the quality of artificially generated music, and there is a rapidly growing body of knowledge on the topic.

3 Music Generation Techniques

For hundreds, if not thousands, of years humanity has been devising ways to generate music. A historic overview of many of these techniques is provided in great detail in the book *Algorithmic Composition: Paradigms of Automated Music Generation* by Gerhard Nierhaus [25]. Nierhaus breaks the techniques down into the following evolution:

1. Simple Algorithms
2. Markov Models
3. Generative Grammars
4. Transition Networks
5. Chaos Theory and Self-Similarity

6. Genetic Algorithms

7. Cellular Automata

8. Neural Networks / AI

This paper is skipping to number eight and focusing on Neural Network approaches to music generation, as they are largely considered the state-of-the-art and have shown the most promising results at this time. Since the publication of this book, there has been an explosion of unique neural network techniques, and many were applied successfully to music generation. The three predominant techniques for music generation are Variational Autoencoders, Recurrent Neural Networks, and Generative Adversarial Networks, which will all be discussed in-depth. These network architectures are largely independent of the music representation format used, i.e., a Variational Autoencoder could be trained on either symbolic or acoustic music [8, 27]. Wherever possible, examples of the networks will be provided.

3.1 Symbolic vs. Acoustic

WaveNet (2016) was a critical breakthrough in networks trained on acoustic audio data, and can be viewed as a turning point for music representation in neural netowrks [26]. While the authors of WaveNet mainly applied their approach to speech generation, many future networks for music generation built off WaveNet's methods. Before WaveNet, symbolic representations were used almost exclusively to represent music for neural network tasks [10]. WaveNet used acoustic information by conditioning each new sample of audio data on all the audio that preceded it, shown in equation 1.

$$p(X) = \prod_{i=0}^{T-1} p(x_{i+1}|x_1, ..., x_i) \tag{1}$$

Another critical contribution of WaveNet was generating audio data *without* a Recursive Neural Network (RNN), the prevailiaing technique at the time. Instead of using any type of memory cell like in RNNs, WaveNet used an approach called "causal convolution". This is a modification of the standard convolutional neural network (CNN) architecture which is used widely in machine learning applications. Causal convolutions restrict generation to be conditioned on information from the past, as well as current information. This allows time-dependent structure to be captured by the network. WaveNet's structure is shown in figure 8.

WaveNet was not applied directly to music generation, but it paved the way for many of the acoustic-domain and non-RNN models that will be discussed in the next sections. Presently, there is an even division between networks, with about half using symbolic representations and half using acoustic representations. The highest quality generated music is currently being produced with acoustic representations through the Jukebox network [8].

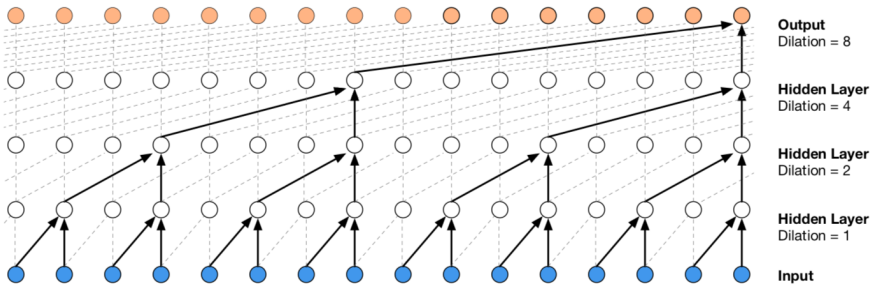


Figure 8: WaveNet architecture, showing dilated causal convolutions. Image source: Oord et. al. [26]

3.2 Recurrent Neural Networks (RNNs)

"Generic" neural networks consist of dense connections between each layer, where the output depends only on the current input. This type of memoryless network is called a feed-forward network. Recurrent neural networks differ from feed-forward networks by including some form of "hidden" information which is used to store knowledge about the past. The most common approach is to use Long Short-Term Memory (LSTM) cells [15]. This allows for the network to have memory, whereas a dense or convolutional network only generates based on the current information. The memory can be used to model time-dependent structure, like melodies present in music. The authors of one of the earliest papers to apply RNNs to music generation put it this way:

> "Lacking the ability to store any information about the past, [a feed-forward] network would be unable to keep track of where it is in a song. In principle an RNN does not suffer from this limitation. With recurrent connections it can use hidden layer activations as memory and thus is capable of exhibiting (seemingly arbitrary) temporal dynamics. In practice, however, RNNs do not perform very well at this task." [14]

RNNs train by attempting to predict the next value in a sequence, $X = \{x_1, x_2, ..., x_T\}$, see equation 2. This method of generation is tractable to the case of symbolic, single-track music. The RNN takes the notes that led to the current point and then predicts the next note. A technique called softmax can be used in the final layer to determine which note is most likely to be added to the sequence. For training, this generated note is checked against the ground truth value of the corresponding note in the training sequence. For generation, a starting note is chosen and then the RNN generates a note sequence, where each note is conditioned on the preceding notes.

$$p(X) = \prod_{i=0}^{T-1} p(x_{i+1}|x_1, ..., x_i) \tag{2}$$

For an RNN, the following equations are used, where $\mathcal{H}$ represents a memory cell like LSTM. Source: [23]

$$h_t = \mathcal{H}(h_{t-1}, x_{i=t}) \tag{3}$$

$$p(x_{i+1}|x_1, ..., x_i) = Softmax(MLP(h_t)) \tag{4}$$

3.2.1 RNN-RBM

One of the challenges for symbolic RNNs is enabling the network to produce polyphonic music, or music where more than one note is being played at a time. In 2012, Boulanger, et. al tackled this problem by augmenting an RNN structure with an energy-based model called a restricted Boltzmann machine (RBM).

> "For the case of polyphonic music, it is obvious that the occurrence of a particular note at a particular time modifies considerably the probability with which other notes may occur at the same time. In other words, notes appear together in correlated patterns, or *simultaneities*, that cannot be conveniently described by a typical RNN architecture ..." [2]

The proposed solution, RNN-RBM, used restricted Boltzmann machines to model structures like chords. One major limitation of this approach was the lack of long term structure, along with the complex interactions between the RBM and RNN structures of the network architecture, shown in figure 9.

3.2.2 C-RNN-GAN

The first RNN network to use an acoustic representation for music generation was C-RNN-GAN in 2016, where the "C" stands for continuous [24]. The structure of this network is shown in figure 10. The network connections are RNN-styled, but instead of predicting one note at a time, an adversarial approach is used. The pink LSTM nodes comprise the RNN structure; once in the generator (blue block) and once in the discriminator (yellow block). The GAN approach is discussed in section 3.4. The results from C-RNN-GAN were moderately successful; however, by the author's admission, "The generated music can not yet compare to the music in the training data, by human judgement." [24]. The sheet music from one of their generated samples is shown in figure 11.

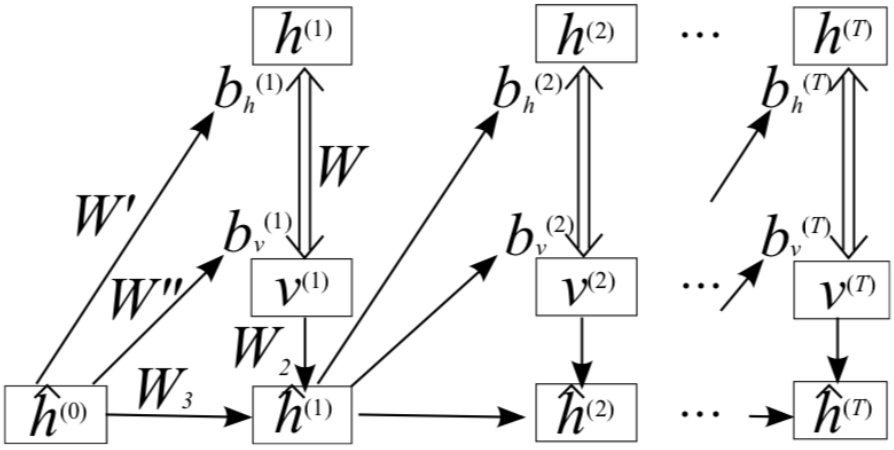


Figure 9: RNN-RBM structure. The RNN is the series of nodes along the bottom, while the upper nodes constitute the restricted Boltzmann machine. Image source: Boulanger, et. al. [2]

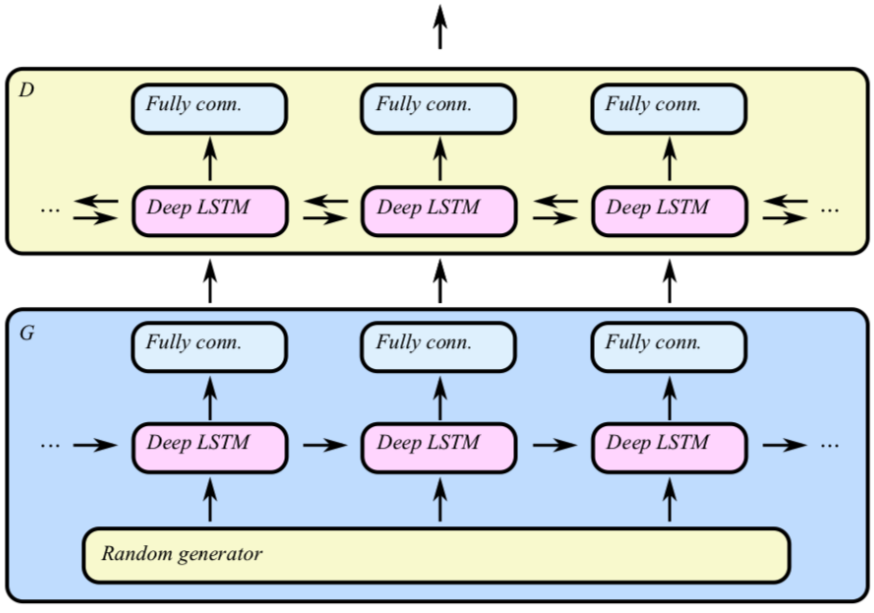


Figure 10: C-RNN-GAN network structure. Image source: Mogren [24]

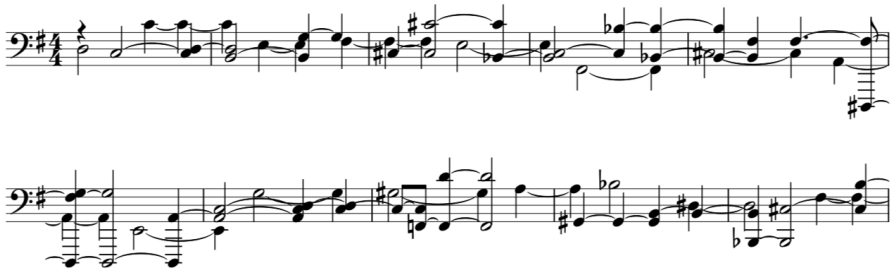


Figure 11: Sample result produced by C-RNN-GAN. Image source: Mogren [24]

3.2.3 SampleRNN

SampleRNN addressed the issue of long term structure by using a hierarchical approach to acoustic audio generation [23]. They implement RNN structures at three different time scales, capturing structures of varying length. Figure 12 shows this architecture. The authors of SampleRNN also used truncated back-propagation through time (Truncated BPTT) to train their network efficiently, addressing another known issue with RNNs [26]. The authors saw very good performance compared to other available networks, but they do not show any comparisons to real-world music samples.

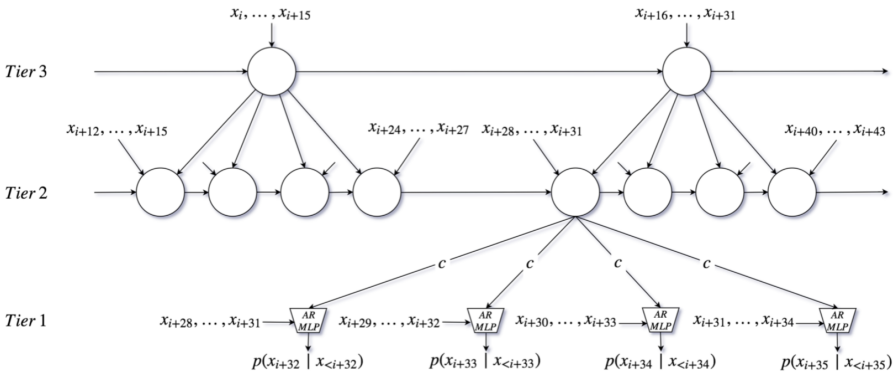


Figure 12: Slice of the hierarchical RNN structure in SampleRNN Image source: Mehri [23]

3.2.4 MelNet

MelNet applies a similar hierarchical structure of RNNs as in SampleRNN, but uses the spectrogram representation of audio data instead of the waveform representation with the goal of better capturing long-term structure.

> "The temporal axis of a spectrogram is orders of magnitude more compact than that of a waveform, meaning dependencies that span tens of thousands of timesteps in waveforms only span hundreds of timesteps in spectrograms." [30]

The hierarchical approach implemented in MelNet is relatively complicated; a brief summary is provided in figure 13. The result is a network that effectively models high-level structure, at the loss of fine details in audio quality. The authors suggest that a combination of MelNet and another approach like WaveNet could be used together to generate better audio samples. Importantly, MelNet showed that the frequency domain can be used for audio data to aid in modeling long-term structures.

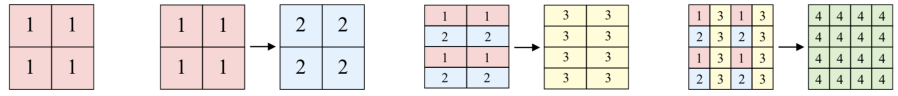


Figure 6. Schematic showing how tiers of the multiscale model are interleaved and used to condition the distribution for the subsequent tier. a) The initial tier is generated unconditionally. b) The second tier is generated conditionally given the the initial tier. c) The outputs of tiers 1 and 2 are interleaved along the frequency axis and used to condition the generation of tier 3. d) Tier 3 is interleaved along the time axis with all preceding tiers and used to condition the generation of tier 4.

Figure 13: MelNet hierarchy breakdown. Source: Vasquez, et. al [30]

3.2.5 HRNN

The most recent development in RNN music generation surprisingly switched back to modeling in the symbolic domain [32]. Their network, Hierarchical RNN (HRNN) is extremely similar to SampleRNN, but learns from symbolic data instead of acoustic data. The architecture is shown in figure 14. It has the same three-tiered structure, with note level, beat level and bar level RNNs. While there were no groundbreaking developments with HRNN, they provided interesting performance results. Of the roughly 2000 samples generated, with 659 people responding, 33.69% of the pieces were thought to be human-generated (real) [32]. HRNN also provides a method of conditioning the generation, which will be discussed in section 4. A sample of the output from their network is provided in figure 15.

In figure 15 the labels correspond to the hierarchical layers of the HRNN implementation. HRNN-1L has only note level RNNs. HRNN-2L has note and beat level RNNs, and HRNN-3L has note, beat and bar level RNNs. The lack of long term structure for HRNN-1L is pointed out in the figure. HRNN-3L has the most coherent long term structure with the rhythm of the first four bars being repeated exactly in the second four bars. This is the same scale of repetition seen in the real-world example used earlier in this paper: Chorale no. 5 by JS Bach. The chorale is provided in figure 3.

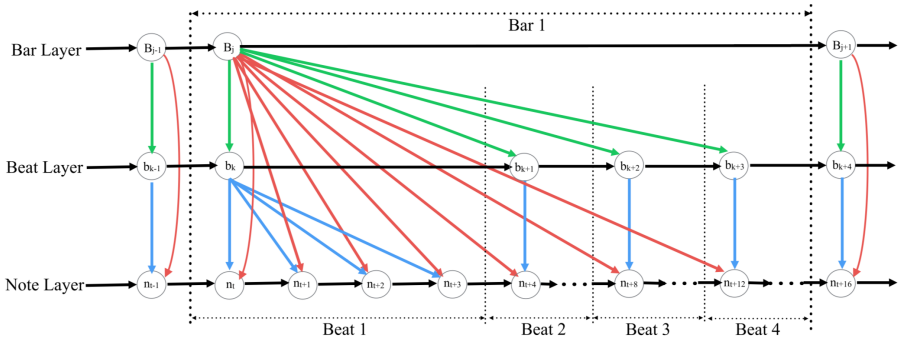


Figure 14: Hierarchical RNN structure Image source: Wu, et. al [32]

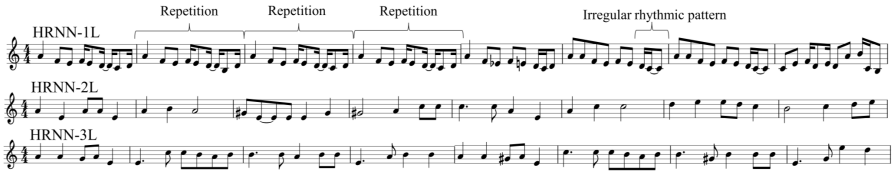


Figure 15: Sample HRNN melodies Image source: Wu, et. al [32]

3.3 Variational Autoencoders (VAEs)

The defining characteristic of an autoencoder is the "butterfly" shape, with an encoder and decoder, see figure 16. This structure compresses inputs like images or audio down to a low-dimensional latent space before reconstructing them. Variational autoencoders were introduced in 2013 by Kingma, et. al, and provide improved performance with variational bayes [21].

While VAEs were initially constructed with fully connected layers in the encoder and decoder, other network architectures have also proven useful. Among these are convolutional layers (CNNs) and even recurrent layers (RNNs). This allows a wide range of diversity in the implementation of VAEs for different applications.

One of the useful features of VAEs is the latent space where inputs and outputs are encoded. This latent space is encouraged to group similar inputs together through the loss function used to train the network, see equation 5. The groupings and structures in the latent space can be used to control aspects of generated music as shown in [27].

$$\mathcal{L}_{VAE} = \mathbb{E}_{q_\theta(z|x)}[\log p_\phi(x|z)] - \beta D_{KL}[(q_\theta(z|x)\|p(z)] \tag{5}$$

This equation shows the loss function that is used to train a VAE. The two components are the reconstruction loss and the KL divergence, which are balanced with the β parameter. The reconstruction loss encourages minimal

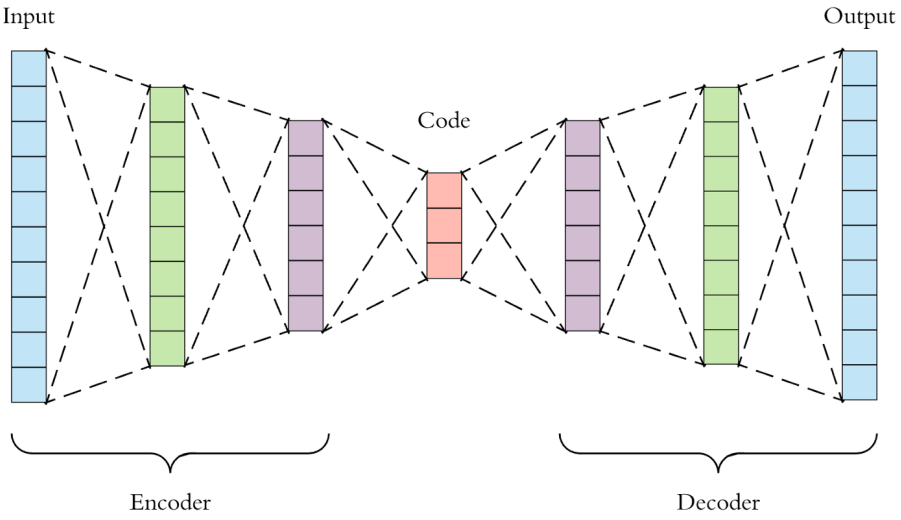


Figure 16: A simple autoencoder structure. Image source: Dertat [6]

change from a sample that is pushed from the encoder through the latent space and out the decoder, while the KL divergence is used to encourage the latent space to be compact and structured.

There has also been promising work with autoencoders that do not fall under the VAE classification, such as the ArgMax Autoencoder (AMAE) proposed by Dieleman, et. al in 2018 [10]. To keep the information concise, these networks will not be examined further in this paper.

3.3.1 Midi-VAE

Midi-VAE trains a VAE on symbolic music data, specifically midi files, which are similar to the pianoroll shown in figure 6 [3]. They use recurrent connections in their encoder and decoder, as shown in figure 17. While relatively simple, Midi-VAE was able to use the latent space to perform style transfer on midi files, converting given midi files to a specific genre.

3.3.2 MusicVAE

MusicVAE is a symbolic domain, hierarchical VAE model with recurrent structure [27]. The authors leverage the latent space to perform averaging between two pieces of music, among other things. MusicVAE is unique in that there is a hierarchical structure only in the decoder of the VAE, while the encoder is a single layer RNN, see figure 18.

MusicVAE also demonstrates the ability to determine feature directions in the latent space and use those to control attributes of generated music, similar to the approach used in InterfaceGan [28].

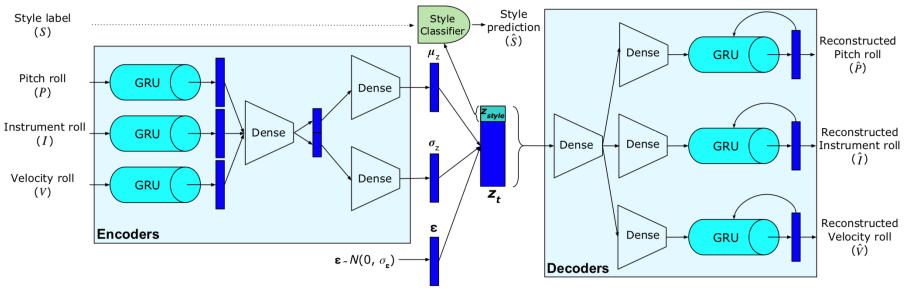


Figure 17: Midi-VAE structure. Image source: Brunner, et. al [3]

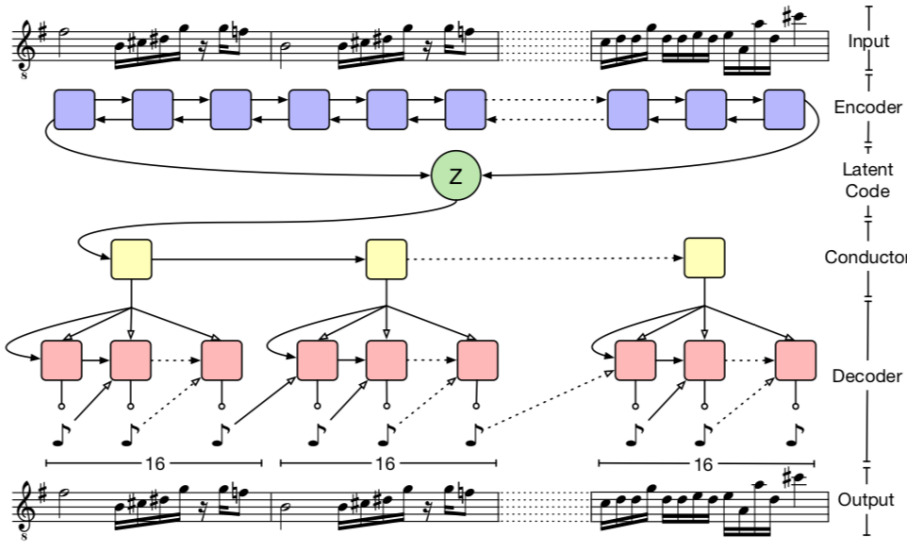


Figure 18: Music-VAE structure. Image source: Roberts, et. al [27]

3.3.3 Jukebox

Jukebox is an acoustic domain, VQ-VAE model that not only generates music, but also can generate singing. Instead of using a direct hierarchical approach to address the issue of differing time scale structures, Jukebox uses a special implementation of a VAE called a Vector-Quantized VAE (VQ-VAE), explained by the authors here: [29]. The network structure for Jukebox is relatively complex and will not be discussed here further. For details, see the paper by the Jukebox authors [8].

Some of the highlights of Jukebox are the ability to generate music in a specific genre, or by a specific artist. See their website for sample audio files [8]. One of the weaknesses of Jukebox is a lack of long term structure, "we do not hear familiar larger musical structures such as choruses that repeat." [8]. Another weakness is a prohibitively slow sample time; "It takes approximately 9 hours to fully render one minute of audio through our models" [8].

3.4 Generative Adversarial Networks (GANs)

GAN networks are a relatively new network structure, introduced in 2014 by Ian Goodfellow [16]. The GAN architecture leverages two competing networks, one called a generator and one called a discriminator, see equation 6. The goal of the generator is to produce samples that look like they are real, while the discriminator looks at some samples from the generator and some samples from the training data and tries to tell which ones were generated. See figure 19 for a visual representation of the GAN structure. At the end of training, the generator can be "disconnected" from the rest of the training framework and used to generate realistic looking samples.

$$\min_G \max_D V(D,G) = E_x[log(D(x))] + E_z[log(1 - D(G(z)))] \tag{6}$$

This equation is the objective function of a GAN architecture, where G is the generator and D is the discriminator.

3.4.1 MidiNet

MidiNet is a network in the symbolic domain that contains only convolutional or dense layers in the generator and discriminator [33]. Additionally, a conditioner CNN network is trained that allows a degree of control over the generation. See figure 20 for the network structure. While MidiNet did not introduce any revolutionary developments to the field of music generation, it did demonstrate the capability for memory-less architecture. This resulted in a lack of long-term structure, as evidenced by the results shown in their paper.

3.4.2 MuseGan

MuseGan represents the state-of-the-art in GAN generated music [13]. The network architecture is moderately complex, and is shown in figure 21. MuseGan

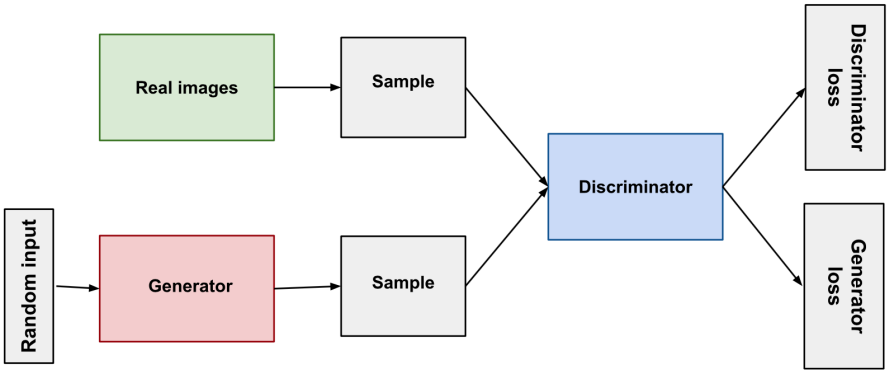


Figure 19: Generative Adversarial Network structure. Image source: Google [7]

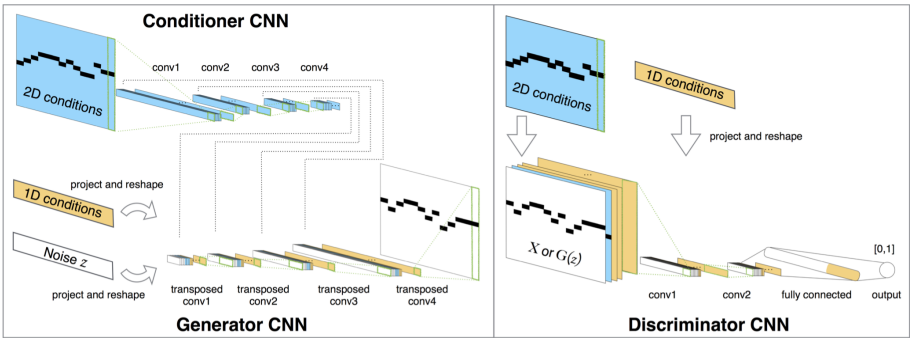


Figure 20: MidiNet Architecture. Image source: Yang, et. al [33]

employs individual track generation as well as a conditioning input across all tracks to maintain cohesiveness. They also use a time dependent architecture for generating time-conditioned latent vectors which are used to create long-term structure at both the individual and global level.

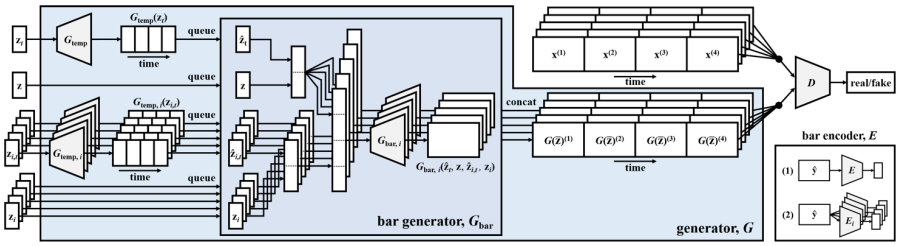


Figure 21: MuseGan Architecture. Image source: Dong, et. al [11]

3.5 Additional Network Structures

The above summary of networks for music generation is not exhaustive; this field is still rapidly evolving and it would be intractable to attempt to examine every network that has addressed this challenge over the past decade. The models chosen provide a good overview of the main approaches and implementations for music generation within the past decade.

There are two other topics that deserves a brief mention, the first is the "transformer" architecture, which is used in a handful of papers for music generation, including the recent Wave2Midi2Wave network [18]. The transformer uses a "self-attention" technique to maintain long-term coherence. The other topic is composite networks, for instance VAE+GAN which uses both network types to create a single model with more capability than standalone VAEs or GANs by using a shared latent space between the two components [22].

3.6 Summary

In the field of neural network generated music, there are two main representations for audio: the symbolic domain and the acoustic domain. Symbolic domain networks are generally easier and faster to train, at the expense of reliance on an additional tool to produce the music sound waves and a lack of audio features like instrument type. Acoustic domain networks are rising in popularity as neural networks become more powerful, and they are able to generate increasingly realistic music.

The three major network architectures for music generation are RNNs, VAEs and GANs.

RNNs are characterized by the inclusion of memory cells, often LSTM cells. These memory cells allow for time dependencies in the network which can improve the long-term structure of generated music. RNNs train by attempting

to predict the next note in a sequence, and they generate in the same fashion, which can lead to slower sample times.

VAEs are characterized by their "butterfly" shape, with an encoder and decoder that meet in a latent space. VAEs train to reconstruct real-world inputs and learn a meaningful embedding between the encoder and decoder. Generation is performed by sampling points in the latent space.

GANs are comprised of a generator and a discriminator which compete throughout the training phase. GAN generation is performed by passing a noise vector to the generator. Both GANs and VAEs are flexible in their implementations, allowing different connection types like CNN, RNN or Dense to be used.

4 Formal Definitions

With the context of the various neural network approaches that are currently being used to produce music, conditional generation can be examined. First, a formal definition for many of the concepts related to conditioning will be defined here, followed by their implementation and uses in section 5. These definitions are largely based on the work in InterfaceGAN, and additional properties and definitions can be found there [28]. The network type chosen for this paper is a GAN, so the definitions presented will be framed using the structure of a GAN network, although many of the same concepts apply to VAEs as well.

4.1 Latent Space

Adopting InterfaceGan's initial framework:

> Given a well-trained GAN model, the generator can be formulated as a deterministic function $g : Z \mapsto X$. Here, $Z \subseteq \mathbb{R}^d$ denotes the d-dimensional latent space, for which Gausian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ is commonly used. X stands for the image space, where each sample $\mathbf{x}$ possesses certain semantic information [28].

The only adjustment made here is that X will stand for the music sample space, instead of the image space. For MuseGan in particular, the latent dimension d is 128 and the distribution used is $\mathcal{N}(\mathbf{-2}, \mathbf{2})$

Semantic Information Every sample of music has semantic information which represents qualitative and quantitative features of a music sample such as tempo, polyphonicity or genre. The dimension of this space is arbitrarily decided by the number of features analyzed for a given music sample. Each sample in the music space can be evaluated manually or with a feature extractor to determine values for the semantic features of interest. This evaluation can be written as $f : X \mapsto S$ where $S \subseteq \mathbb{R}^m$ is the semantic space with m features [28]. A sample in latent space can be mapped to semantic values by the

composite function $\mathbf{s} = f(g(\mathbf{z}))$ where $\mathbf{s}$ and $\mathbf{z}$ represent the semantic features and latent code respectively.

4.2 Vector Operations

Assuming a smooth latent space, basic linear algebra operations can be used on latent space samples to produce predictable results in the semantic space for the resultant music samples. If the latent space is not smooth, the results of latent space manipulations can be unpredictable or random. This is one of the reasons VAEs are preferred for latent space manipulation: they have a component in their objective function that encourages a smooth latent space. However, GANs have also exhibited smooth latent spaces which allow for predictable manipulation, as showcased in [5, 20, 28].

Averaging One of the simplest latent space operations is to average two or more latent samples $\mathbf{z}_{avg} = \frac{1}{k}\sum \mathbf{z}_k, k \geq 2$. If the latent space is well-behaved, this operation should result in the semantic space values also being averaged, $\mathbf{s}_{avg} = \frac{1}{k}\sum \mathbf{s}_k$.

Averaging a large number of latent codes with a similar semantic value can produce a useful centroid in the latent space which, in theory, produces a similar centroid in the semantic space. This is accomplished by selecting a subset of Z, denoted Z_x, where

$$\{f(\mathbf{z}_x) = \beta \pm \delta | \forall \mathbf{z}_x \in Z_x\}$$

where β is the desired value of feature $\mathbf{s}$ and δ is the acceptable variance. Averaging can be used on Z_x to find a centroid in the latent space. Latent space samples taken near this centroid are likely to have the target value β for semantic feature $\mathbf{s}$.

This technique is used in MusicVAE to find the average latent code for high and low values of five semantic features [27]. They leverage these centroids to control the corresponding semantic features in their generation.

Interpolating A linear interpolation can be found between any two latent codes, $\mathbf{z}_a$ and $\mathbf{z}_b$, by subtracting one from the other. With a smooth latent space, this can allow for smooth interpolation in the semantic space between the semantic values of each sample, $\mathbf{s}_a$ and $\mathbf{s}_b$.

When $\mathbf{z}_b$ is a known latent vector with a desirable semantic quality and $\mathbf{z}_a$ is a randomly chosen vector, this interpolation is sometimes referred to as the "truncation trick" as seen in work like StyleGan [20]. This technique can be used to improve the quality of random generation if $\mathbf{z}_b$ represents a high-quality generated sample.

4.3 Feature Vectors

Using the interpolation notation from above, if $\mathbf{z}_a$ and $\mathbf{z}_b$ vary primarily in only one semantic feature, the difference vector between them should primarily

encode that semantic feature, $\overrightarrow{\lambda}_{feature} = \mathbf{z}_b - \mathbf{z}_a$. The most common approach to find a robust feature vector is to average a large number of latent codes with a similar semantic value for $\mathbf{z}_a$ and a similarly large set of latent codes with an opposite semantic value for $\mathbf{z}_b$ [27]. Another, "low-shot" approach is to find two samples $\mathbf{z}_a$ and $\mathbf{z}_b$ which are identical in every semantic feature except the target feature and take their difference. Identifying suitable latent vectors often proves to be a difficult task, so the averaging approach is typically used.

Once a feature vector is found, it can simply be added to any randomly selected latent code, $\mathbf{z} + \beta(\mathbf{z}_b - \mathbf{z}_a)$. This imparts the target feature to the random generation, and the parameter β controls the strength of the feature. A negative value for β can be used to remove the feature from the resultant music sample. As β gets large, the generation results often degrade into lower quality samples.

4.3.1 Limitations

Due to the often non-linear structure of neural network latent spaces, the same feature vector can produce different semantic changes on two different latent codes. Depending on the random starting point for generation, unwanted artifacts can appear as a result of latent walking. Consider a feature vector for the starting pitch of a song. Using this feature vector to raise the starting pitch may work well for a majority of songs, but if the random generation already starts on the highest possible note there may be issues with latent walking. Verification techniques post-latent walking can be used to minimize this limitation until a better understanding of latent spaces can be developed.

4.4 Hyperplane Boundaries

Another approach to find feature vectors is to use a linear support vector machine (SVM). This is a classifier that uses machine learning to find the best hyperplane boundary between two sets of points. If the semantic values for a single feature are used to label a representative sample of the latent space vectors, an SVM can be used to divide the latent space so that samples with different semantic values are separated. This hyperplane boundary is represented by a normal vector and a point located on the boundary, where the normal vector can be used as a feature vector as showcased in [28]. Walking along the normal vector can move a sample in the latent space across this boundary, typically changing the semantic value for the generated music sample.

The following equation can be used to tell how far any given point in latent space is from the hyperplane boundary, as well as which side of the boundary it is on. α is the point that defines the hyperplane with normal vector $\mathbf{n}$.

$$d = \mathbf{z} \bullet \mathbf{n}^T + \alpha \tag{7}$$

The magnitude of the distance d is the distance from the latent vector to the plane and the sign of d is the side of the boundary (the hyperplane is oriented w.r.t its normal vector). This allows for approximation of not only the semantic

value for the feature in question, but also the strength of that semantic value. This is all done pre-generation and requires no feature extraction or evaluation on the generated sample, making it a very efficient method to evaluate which latent space vector will produce a generation sample with desired characteristics.

4.4.1 Latent Walking

Given a latent code, $\mathbf{z}$, and a feature vector, $\overrightarrow{\lambda}$, the equation for latent walking is simply:

$$\mathbf{z}_{new} = \mathbf{z} + \beta \frac{\overrightarrow{\lambda}}{\| \overrightarrow{\lambda} \|} \tag{8}$$

The sign of the β parameter determines the direction for walking, and the magnitude represents the number of "steps" to walk in the latent space. The euclidean distance travelled with each step is one.

4.5 Orthogonalization and Entanglement

The similarity of any two vectors in latent space can be computed with the dot product of their normalized directions. The dot product is the inner product, and is denoted by $\bullet$.

$$similarity = \frac{\mathbf{z}_a}{\| \mathbf{z}_a \|} \bullet \left(\frac{\mathbf{z}_b}{\| \mathbf{z}_b \|} \right)^T \tag{9}$$

If this similarity is 0, the vectors are orthogonal. Given two latent space vectors, an orthogonal pair can be found using the equation below. One of the given vectors is selected as the basis and the other, called the primal, is orthogonalized w.r.t the basis.

$$\mathbf{z}_\perp = \mathbf{z}_{primal} - \frac{\mathbf{z}_{primal}}{\| \mathbf{z}_{primal} \|} \bullet \left(\frac{\mathbf{z}_{basis}}{\| \mathbf{z}_{basis} \|} \right)^T \times \mathbf{z}_{basis} \tag{10}$$

The primary interest with orthogonalization in the latent space is for disentangling feature vectors. Once a set of feature vectors are found, they will often have nonzero similarities; this is referred to as entanglement. For instance, moving along the feature vector for tempo might have the unintended effect of changing the genre of a sample of music. In this case, orthogonalizing the tempo feature vector w.r.t. the genre feature vector would result in a direction that can be latent walked which will change the tempo without changing the genre. This process is sometimes called disentanlging the latent vectors or latent space. When orthogonalizing / disentangling, the norm of $\mathbf{z}_\perp$ is often used to represent the orthogonalized direction without any magnitude information.

The Gram-Schmidt process can be used to orthogonalize a large set of n feature vectors, as long as $n \leq \dim(Z)$. In practice, only a handful of feature vectors can be orthogonalized before each subsequent vector encodes more noise than true feature information.

5 Conditioning Techniques

Various networks have proven useful for generating synthetic music, often with the ability to create a diverse range of music types. To demonstrate this range of generation capability, some networks classify their music types by genre, or by artist, while other networks divide their music by attributes such as key or rhythm [27, 32].

The ability to control certain features of generated music by applying changes to the input is called conditioning. Most commonly this is done by providing additional information to the network when sampling from it, or by doing a preprocessing step on one of the network inputs. There is also some work focused on conditioning post-generation, with techniques like style transfer or running generated music through an additional style conversion network, but these techniques often require significantly more processing than conditioning pre-generation [4].

5.1 Levels of Conditioning

There are different methods of conditioning and different amounts of information that can be fed to a network to control its generation. Less information provided typically provides less control, but increases the scope and diversity of the generated music. Trained networks have a limited scope for their generative capabilities, due to the finite nature of their training data. This means there are a fixed number of distinguishable songs any given network can produce. Constraining their generation with conditioning reduces the number of songs that can be produced. The tighter the constraint, the more limited the generation.

Common conditioning techniques for music generation include conditioning on a genre, composer or style of music, conditioning on a symbolic representation of the target music, and conditioning on an acoustic sample of music similar to the desired output. Sander Dieleman, one of the authors of the AMAE network, presented a helpful graphic for the various levels of conditioning, see figure 22 [10, 9].

5.2 Priming

The most common method of conditioning in the field of artificial audio generation is priming. Priming involves feeding a sequence of information to the generation network each time it runs. This information can vary in length from a single note to an entire melody. The priming sequence is used by the network to generate additional notes, in a well trained network these generated notes should sound similar to the priming sequence. Priming has been shown to be an effective conditioning technique for regressive networks which predict one note at a time [26, 33]. See equation 10 for an example of priming used in WaveNet.

Unfortunately, using priming for conditional generation is a restrictive technique. Priming sequences impose a handful of limitations on conditional genera-

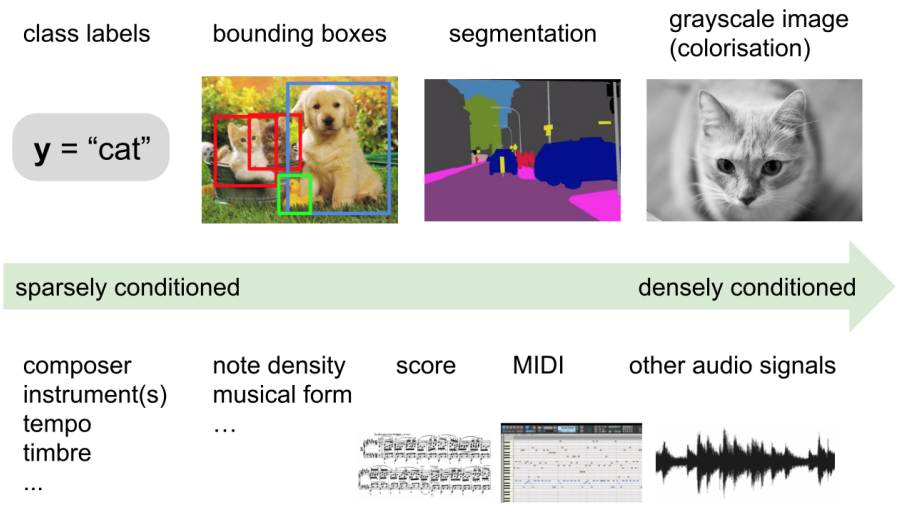


Figure 22: Levels of generative conditioning for the image domain (top) and the audio domain (bottom). Image source: Dieleman [9]

tion which can be avoided with other forms of conditioning. The first limitation is needing a sample of the desired output to use for priming. If the goal of a generation sample is to produce a blues song, a sample blues song must be provided to the generator, preferably with the desired tempo, tone, and as many other key features as possible. Finding the right priming sequence with all the desired characteristics can be difficult. If the goal of generation is to create unique music, an appropriate priming sequence may not even exist yet. This creates a sort of catch-22; the target sequence is needed to create the target sequence.

Additionally, priming sequences alone do not allow for fine control over how the network will use the provided sequence. This creates limitations like being unable to use multiple priming sequences or control the strength of the impact from a priming sequence. These limitations motivate a different conditioning technique with more flexibility; one option is latent walking.

5.3 Latent Walking

A popular approach to conditioning for image domain generation is Latent Walking. This technique involves learning a meaningful direction in the latent space, and moving a sample in that direction to effect a desired change in the semantic features of the output, see equation 11. Often these directions are feature vectors, as described in section 4. There are various techniques to discover these directions, from simple linear algebra operations to training auxiliary networks like SVMs [28, 27, 31].

One example of latent walking is provided in MusicVAE. The authors were

able to control five features of their music with high consistency. They do not discuss the impact of each of these features in-depth, aside from saying that some of them appear entangled. The results of their latent walking are shown in figure 23.

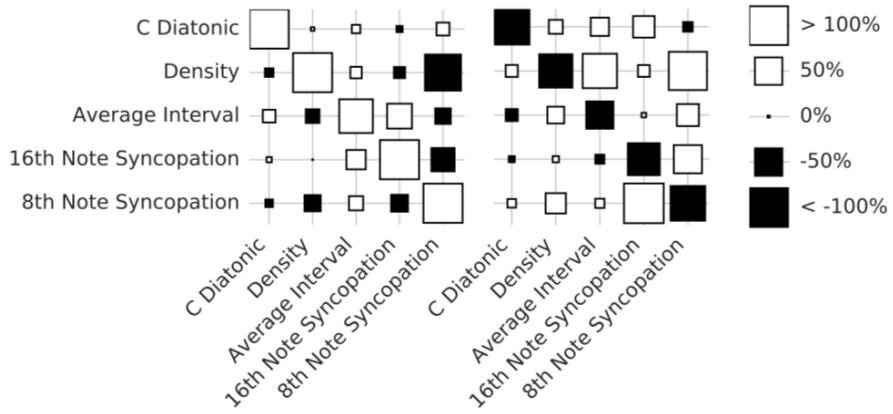


Figure 23: Results of latent walking in MusicVAE with a linear algebra approach. Y-axis is feature that was changed, and X-axis is the feature measured.

Latent walking provides a great deal of flexibility for which features are conditioned and the strength of the conditioning. It is also possible to isolate features and control them separately, or condition on multiple features in a single generation [28].

The largest drawback with latent walking is the additional computation required to determine the latent vectors or latent directions. This can become a complex task depending on the number of features and the difficulty of extracting the features from the music. Qualities like tempo and range are relatively simple to utilize for latent walking, while complex concepts like mood or genre are more difficult. Midi-VAE demonstrated the ability to latent walk genre, but required training an additional network for each pair of genres [3].

5.4 Conditioning Examples

Different network architectures respond better to different conditioning techniques. For instance RNNs work well with a priming sequence of notes, since their generation is performed by predicting the next note in a sequence [23]. The produced music can be heavily influenced by changing the initial sequence of notes provided, however the generation can be difficult to control precisely. Wavenet, the first major architecture to use acoustic audio representations, provided the ability to condition using an additional input, $\mathbf{h}$, see equation 10.

> "By conditioning the model on other input variables, we can guide WaveNet's generation to produce audio with the required charac-

teristics. For example, in a multi-speaker setting we can choose the speaker by feeding the speaker identity to the model as an extra input. Similarly, for TTS [text-to-speech] we need to feed information about the text as an extra input." [26]

$$p(X|\mathbf{h}) = \prod_{i=0}^{T-1} p(x_{i+1}|x_1, ..., x_i, \mathbf{h}) \tag{11}$$

While priming sequences are the most common technique for conditioning audio generation, in the image-generation domain latent walking and feature vector manipulation are the preferred method, as seen in [28, 5, 20]. The equation below shows how conditioning with a feature vector, $\overrightarrow{\lambda}$, on a random sample, $\mathbf{z}$, is implemented. g denotes the generation function and f represents semantic feature extraction.

$$\mathbf{s} = f\left(g\left(\mathbf{z} + \overrightarrow{\lambda}\right)\right) \tag{12}$$

Latent walking is a more flexible conditioning approach than priming and is seeing some experimental research with applications to the music domain. The research in this paper is focused on latent walking and its effectiveness on a music generation GAN, MuseGAN.

5.5 Summary

Conditional generation adds a layer of complexity to the already difficulty task of music generation. Various approaches exist to control features of generated music, including priming melodies and latent walking. Latent walking is an extremely flexible technique that allows for fine-grained control over many aspects of generated music, but it is largely unexplored for music generation, specifically GAN networks. This brings up the core question of this paper; can latent walking be used to condition GAN generated music? And if so, how can it be implemented?

6 Can Latent Walking Techniques be Applied to Effectively Condition Music-Domain GANs?

The primary research objective of this work is to examine the effectiveness of latent walking with MuseGan, a symbolic audio generation GAN. It is well-known that latent walking is effective on VAE networks, and has even been used in work like MusicVAE to condition music generation [27]. While VAEs leverage their encoder to store information in the latent space, GANs do not have any structure in their latent space; all the generation information is stored in the weights of the generator. While the structures encoded in the generator of a GAN will often not be as well-defined as the latent structures in a VAE, they can still be leveraged for latent walking.

GAN Latent Walking Despite having an unconditioned and unstructured latent space, latent walking has shown promising results in GAN image generation with InterFaceGAN, applied to the StyleGan2 network [28, 20]. Well-trained GAN networks have a "smoothness" to their output where two very similar latent codes will produce correspondingly similar samples. For example, consider a semantic feature **s**. Two generated music samples with a small distance between their latent codes will likely have a similar value for **s**. Assuming the training data used has samples with a wide range of values for **s**, for a GAN to efficiently encode the entire range smoothly, it places the latent codes which produce samples exhibiting very high **s** as far away as possible from the latent codes which produce samples exhibiting very low values for **s**. For example, consider the semantic feature of note count; a sample with a low number of notes will have a latent vector that is a large distance away from any of the samples with a high note count. This type of structure is sufficient to provide a basis for latent walking; moving towards the samples with a high number of notes should increase the note count of any given sample.

Any feature which was present in the training data in a continuous manner has the potential to be encoded in this way in the weights of the GAN generator and provide a latent-walkable feature. Notably, the existence of a feature in the training data is not sufficient to guarantee it will be encoded smoothly in the latent space, even though this is often the case. The smoothness present in the generator output is the basis of my hypothesis that latent walking will be effective in conditioning music-domain GAN generation.

6.1 Metrics

Before discussing the implementation used to explore the research question, it is helpful to understand the target metrics. With the overarching goal of improving the quality of the generated music, I selected two metrics which have strong relationships to the audio appeal of a song, qualified note rate and polyphonicity. I chose these two metrics specifically based on their use in papers like MuseGan, MusicVAE, and others [11, 27]. Qualified note rate (QNR) and polyphonicity are also quantitative metrics which correlate to the qualitative metric of music quality and allow for a consistent and repeatable method of evaluation. It is important to note that while realistic music will often have high QNR and polyphonicity, a high QNR and high polyphonicity do not guarantee realistic-sounding or high quality music.

A third metric was chosen which does not have a direct correlation to quality: note count. This metric measures the number of notes present in a piece of music and is an easy metric to evaluate and visualize. One of the common flaws in symbolic, generated music is an over-abundance of short notes. Latent walking note count could provide a method to counteract this flaw and improve the generation quality. If latent walking can be implemented on these features, conditional generation could be used to control aspects of the music generation, including potential improvements to the average quality of the generated music.

6.1.1 Qualified Note Rate (QNR)

QNR is a measure of the percentage of notes in a song that are qualified, or meet a specific condition. In this case, qualified notes are notes that are longer than two time steps, where there are 12 time steps per beat. See figure 24 for a visual representation. A higher QNR typically corresponds to more cohesive music with note lengths comparable to human-composed music, while a low QNR shows that the music consists largely of very short notes, resulting in deterioration of musical quality.

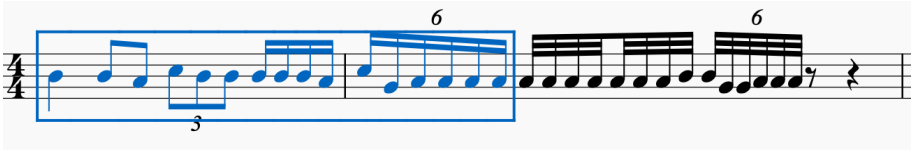


Figure 24: Different divisions of notes. The blue notes are considered qualified, or longer than 1/6th of a time step. Any smaller subdivision would not be qualified.

6.1.2 Polyphonicity

Polyphonicity is a measure of the percentage of the song where more than one note is being played at a time. An example of polyphonicity in standard musical notation is provided in figure 25. Similarly to QNR, a higher polyphonicity corresponds to more realistic music. A low polyphonicity is indicative of music that consists mainly of one note at a time and as such cannot produce harmony. Since human-composed music typically relies on harmony, and harmony tends to make music more complex and acoustically satisfying, a higher polyphonicity is desired in generated music.

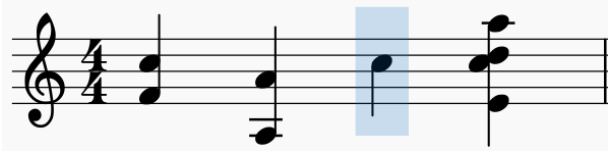


Figure 25: One bar of music with a polyphonicity of 0.75. Only the blue note is non polyphonic, and represents 1/4 of the duration of the total bar.

6.1.3 Application

One of the highlights of MuseGan is the ability to generate multi-track music. The number of tracks generated for these experiments was five, with the first track being a percussion track and the next four tracks corresponding to non-percussion instruments. The QNR and polyphonicity of a given sample were

taken by averaging the QNR and polyphonicity across the four non-percussion tracks. Percussion tracks were left out since they consist of single time step events, with only one event at a time, i.e. the QNR and polyphonicity of a percussion track are always zero.

6.2 Approach

The approach used was based off the work in InterfaceGan [28]. A batch of 30-second samples was generated, an auxiliary feature detector was used to label the QNR and polyphonicity of each sample, and the labeled data was fed to a linear support vector machine (SVM). The SVM creates a divided latent space with respect to the labeled feature. Once the division in latent space is known, it is straightforward to move samples along a path that is perpendicular to the division boundary. This allows control over which half of the latent space any random generation point falls in, which allows for conditional generation.

6.3 Implementation

6.3.1 Generation

The first step was generating a sufficient number of latent codes for the network. The MuseGan network was downloaded from the official GitHub [12]. All default configurations and parameters were kept. 10,000 seed numbers were used, ranging from 0 to 9,999, and the python package scipy was used to generate 25 random vectors of length 128 from each seed number. In other words, for each seed number provided to the scipy random number generation, a 128×25 length $\mathbf{z}$ vector was generated, with random values ranging from -2 to +2, and this was divided into 25 latent vectors.

The result was 25,000 latent vectors $\mathbf{Z}$ where $\mathbf{z} \in \mathbf{Z}$ represents a single vector with 128 random elements in the range [-2, 2].

The MuseGan code was modified to allow a $\mathbf{z}$ vector to be provided to the inference generation. The pretrained model was downloaded, and a loop was set up to generate the audio samples, $\mathbf{X}$, for all 25,000 latent vectors, $\mathbf{Z}$.

6.3.2 Labeling

With 25,000 ($\mathbf{z}$, $\mathbf{x}$) pairs, the next step was to assign semantic labels to the feature of interest. For image domain GANs, this is often done by training an auxiliary network to extract the desired feature, e.g. hair color or age. For lower level metrics, such as color range or blurriness, often the image can be analyzed directly. The first audio metric that was examined was Qualified Note Rate (QNR).

QNR A custom method for finding the QNR of a pianoroll was used, and is detailed here. Pianoroll tracks are comprised of 0s and 1s where a 1 represents a note being played and a 0 is silence. A note will remain on for as long as there

are 1s at the appropriate pitch. See section 2.3.1 for a further breakdown of the pianoroll.

Using the python library numpy, the note starts and stops were determined. Any time a pitch transitioned from 0 to 1 was considered a note-start, and any time a pitch transitioned from 1 to 0 was considered a note-end. The locations of every note start and end were kept, and then the distance between each pair was found. Due to the polyphonic nature of the music, it was imperative to calculate the note lengths for one pitch at a time, so that the appropriate note-start and note-end events were paired. The note lengths were tallied for every pitch and all four non-percussion tracks of a given sample, $\mathbf{x}$.

Finally, the ratio of lengths greater than two to the total number of notes were found for each of the four non-percussion tracks and averaged to find the QNR for the multitrack sample. This process was repeated for all 25,000 samples. A distribution of the QNR is shown in figure 26.

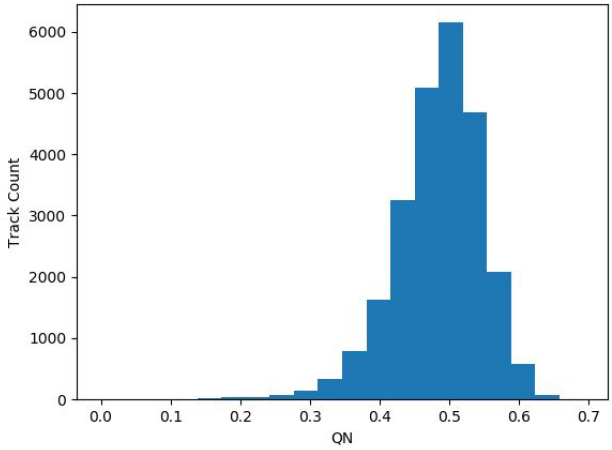


Figure 26: Distribution of Qualified Note Rate across 25,000 samples generated with MuseGan

The highest QNR observed was 0.6926 and the lowest was 0.0. The average QNR across the samples was 0.4845.

Polyphonicity Polyphonicity was measured using the *polyphonic rate* method of the pypianoroll.metrics class. More information on this python package can be found here:[1]. The polyphonicity was computed with a threshold of two, meaning the polyphonic rate is equivalent to the percentage of the song that two or more notes are being played simultaneously. The polyphonic rate was calculated for each track independently, such that if all tracks are each playing only one note at the same time, it does not count as polyphonic. If any track is playing two or more notes at once, it counts as polyphonic, regardless of what

the other tracks are playing. The polyphonicity for each of the four instrumental tracks was averaged to find the average polyphonicity for each multitrack song. A distribution of the polyphonicity for all 25,000 samples is shown in figure 27.

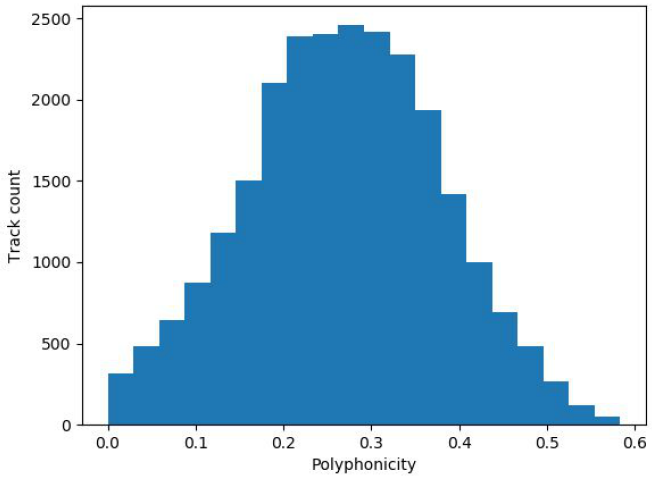


Figure 27: Distribution of Polyphonicity across 25,000 samples generated with MuseGan

The highest polyphonicity observed was 0.5833 and the lowest was 0.0. The average polyphonicity across the samples was 0.2681.

Note Count Note count was the last metric measured, and is simply the number of notes played in a single pianoroll track. The same approach from the QNR was used, and all note-start events were counted. As with all the metrics, this was averaged across the four tracks to produce a note count value for each multitrack pianoroll. A distribution of the note count for all 25,000 samples is shown in figure 28.

The highest note count observed was 192.0 and the lowest was 12.25. The average note count across the samples was 61.5882.

6.3.3 Classifying

The linear SVM was adopted from InterFaceGan [28] and fed the pairs of (latent vector, QNR value) for the 25,000 samples. The SVM trains by taking the highest and lowest 3% of the data and constructing a boundary. After training, the SVM typically converged to 70-75% accuracy across multiple runs.

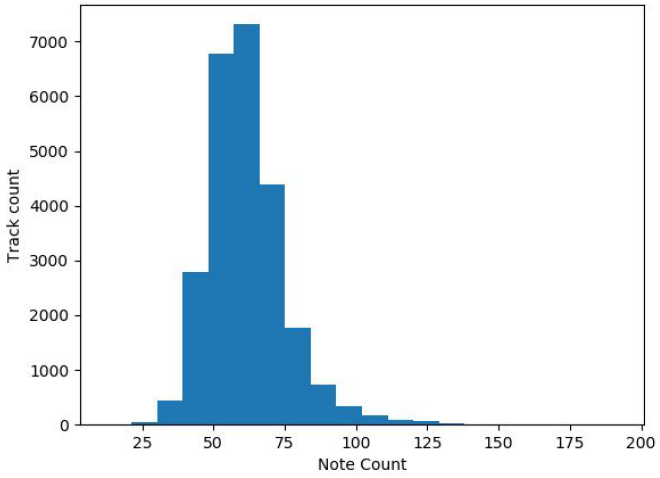


Figure 28: Distribution of Note Count across 25,000 samples generated with MuseGan

6.3.4 Latent Walking

Latent walking was implemented by moving any given sample in the latent space $\mathbf{z}$ a step of one in the direction perpendicular to the boundary found with the linear SVM. The distance for each step is calculated as a euclidean distance, and is oriented the same direction as the normal of the boundary. The relative location of a point to the boundary can be computed using the dot product of the normal, the point in space, and a point on the hyperplane boundary, as outlined in section 4.4. The equation for latent walking is repeated below for convenience.

$$\mathbf{z}_{new} = \mathbf{z} + \beta \frac{\overrightarrow{\lambda}}{\|\overrightarrow{\lambda}\|} \tag{13}$$

Unlike a VAE, the MuseGAN network does not have any explicit structure in the latent space; a sample latent code will have random values for its 128 elements. The "latent structure" that can be explored and leveraged is instead information encoded in the weights of the generator. This can make direct analysis of these "latent structures" difficult, as the latent space cannot be meaningfully analyzed without the use of the generator. The SVM approach described above is one such way to determine "latent structure" by using pairs of latent code and their corresponding generation result.

6.4 Initial Results

1,000 samples were randomly chosen from the batch of 25,000 and were walked nine steps in either direction using the β parameter from equation 13. This was repeated for all three hyperplanes: QNR, polyphonicity and note count. After walking, the average value for every metric was recomputed and are plotted on the composite graphs, shown in figures 29, 32 and 33.

Walking each of the three features imparted changes on the other two metrics, to varying degrees. This effect is called entanglement; see section 4.5 and 6.8 for more information. Plots showing the distribution for the evaluated metrics are in appendix A. Samples of the generated music are in appendix B.

6.4.1 QNR

Figure 29 shows how each of the three metrics change as the feature vector for QNR is walked. In a well-formed and linear latent space, the QNR line would be monotonically increasing or decreasing, however it is clear from the graph that this is not the case.

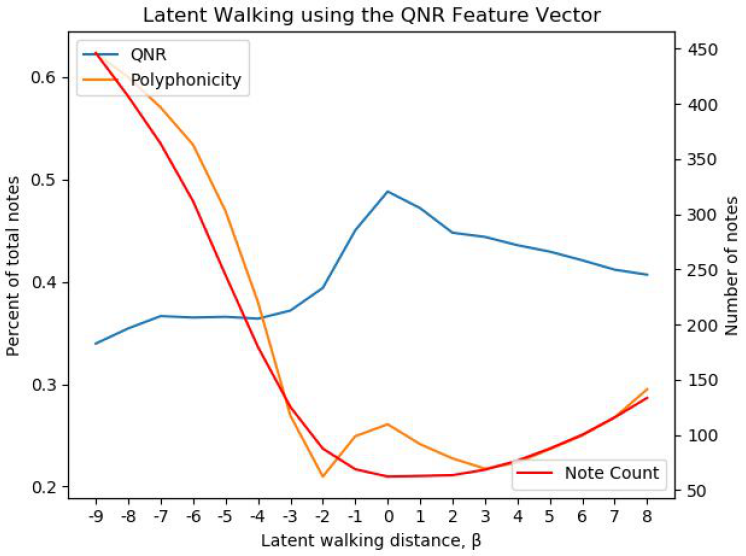


Figure 29: Distribution of metrics across 1,000 samples after walking the QNR boundary

A sample pianoroll is shown after walking a distance of -1, -3, -5 and -9 in figure 30. The degradation in music quality is apparent, along with the increased number of notes and increased ratio of short notes to total notes. The pianoroll shown is track 2/5, or the first non-percussion track in the multitrack sample.

However, a sample pianoroll of walking in the opposite direction showed similar results, when the expected result should have been the opposite, see the

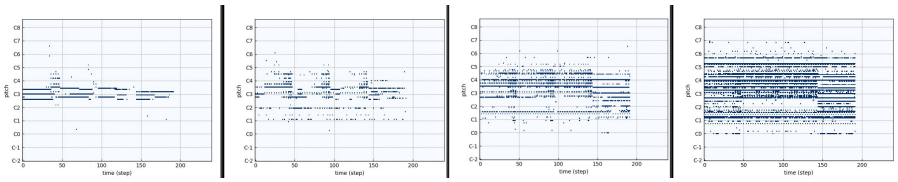


Figure 30: Pianoroll corresponding to latent walking w.r.t the QNR boundary. Distances walked from left to right are -1, -3, -5 and -9

results in figure 31.

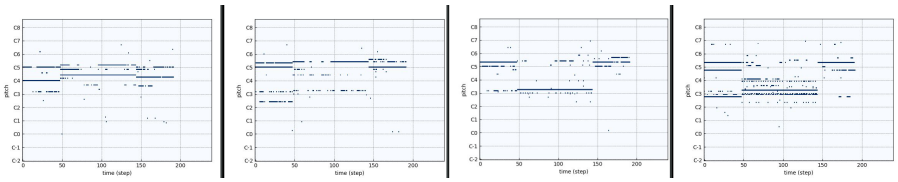


Figure 31: Pianoroll corresponding to latent walking w.r.t the QNR boundary. Distances walked from left to right are +1, +3, +5 and +9

The indication from this experiment is that this approach for latent walking fails for QNR because the direction walked appears to provide very little control over the resultant QNR. This is explored further in section 6.6.

6.4.2 Polyphonicity

Figure 32 shows how each of the three metrics change as the feature vector for Polyphonicity is walked. In a well-formed and linear latent space, the polyphonicity line would be monotonically increasing or decreasing, however it exhibits the behavior of a polynomial function rather than a linear one.

While QNR decreased in every direction walked when starting at the origin, polyphonicity exhibited strictly increasing behavior within the range of [-2:1]. This means there is latent walking capability using this technique for polyphonicity, but it is limited to small changes.

6.4.3 Note Count

Figure 33 shows how each of the three metrics change as the feature vector for Note Count is walked. This metric seems to be the best suited of the three for latent walking with this approach. The plot is monotonically increasing, meaning that latent walking produces the expected result of increasing or decreasing the number of notes present. However, the curve is clearly not linear, but appears to be exponential. This means there are significant limitations to latent walking the note count.

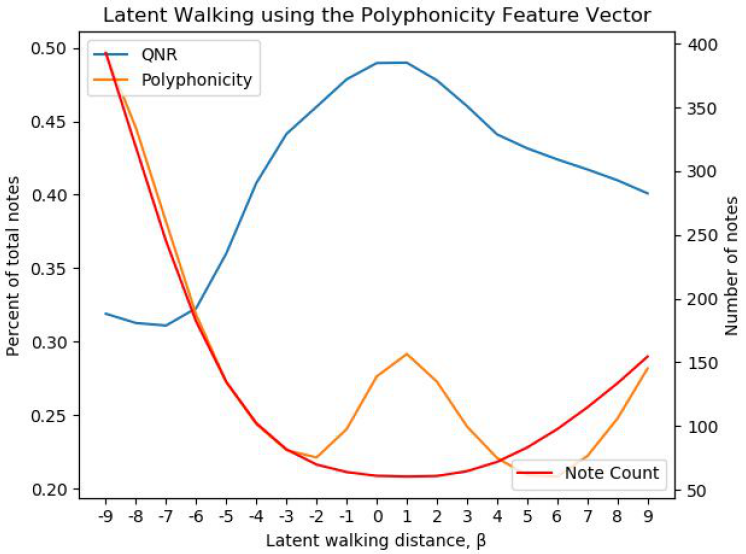


Figure 32: Distribution of metrics across 1,000 samples after walking the Polyphonicity boundary

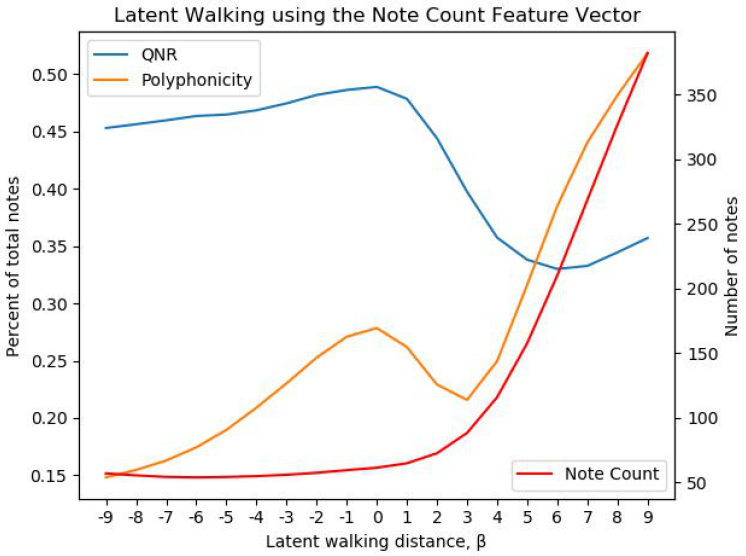


Figure 33: Distribution of metrics across 1,000 samples after walking the Note Count boundary

6.4.4 Evaluation

The three features examined all performed differently in the latent walking experiment. There is a strong indication that the range for the β parameter was too large, and a separate analysis of this is provided in section 6.5. Additional figures on the distributions of the metrics are provided in appendix A. A sample of the generated music when walking the note count boundary is shown in figure 34. The results from the three boundaries are examined in-depth in the following paragraphs.

Note Count was the most successful, and provided reliable results in increasing the number of notes. However, the feature vector used was unable to reduce the number of notes. This could be due to a few limitations. First, there could be an under-representation of low-note-count songs in the training data used to train the MuseGAN model. If the network never learned about songs with fewer than 50 notes, it would be very difficult for it to produce songs with note counts in that range. This would be one explanation for the horizontal asymptote in figure 33. Another explanation could be a failure of the feature vector to accurately encode the direction of these extremely low-note-count songs in the latent space. This is a limitation of the currently available techniques to find feature vectors. Until better techniques are developed to fully understand the latent space, it will be difficult to say if these low-note-count samples even exist in the latent space, let alone find the corresponding feature vector which creates them.

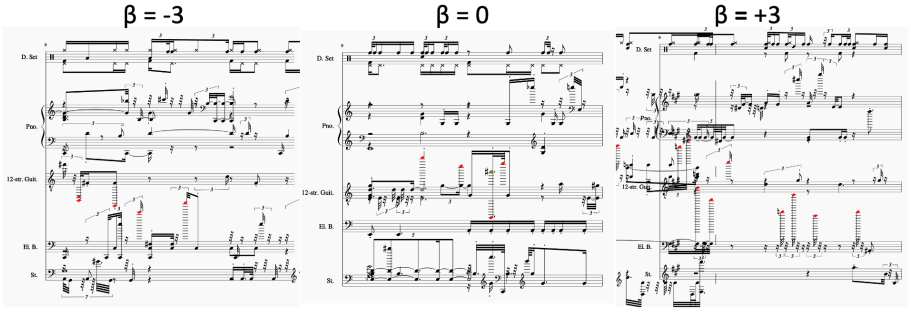


Figure 34: Samples of a single bar of generated music at three values for β along the note count boundary

Polyphonicity was quite unpredictable when latent walking. The feature vector for this metric worked for a small range, but then quickly became unpredictable, particularly for increasing the polyphonicity. Compared to the note count feature, there is evidence that songs with a high polyphonicity exist in the latent space, as shown on the left-hand side of figure 32. This means that the most likely shortcoming of the latent walking approach is the linear nature

of the SVM used to find the feature vector. If a polynomial feature vector were found instead of a linear one, I believe that latent walking for polyphonicity would be more effective, even at large β values.

QNR was the least controllable of the three metrics. Even at tiny values for β, the QNR decreased in both directions, meaning the feature vector failed to encode a meaningful direction for QNR. This is particularly intriguing because it contradicts the performance of the SVM used to find the feature vector, which was able to learn a mapping from low to high QNR with relatively high accuracy (74%). The disconnect between the boundary existing and it not being usable imply that the data used to create the boundary might not have been representative of the entire latent space. This is explored in section 6.6.

6.5 β Parameter Limitation

The critical assumption for using a linear SVM to find the feature vectors is that the latent space is largely linear. If the structure of the latent space is non-linear then the feature vectors found will be local approximations and will only work for small values of β. The plots from the previous section are shown again below, with the bounds for β limited to show the local behavior of the feature vector, figures 35, 36, 37

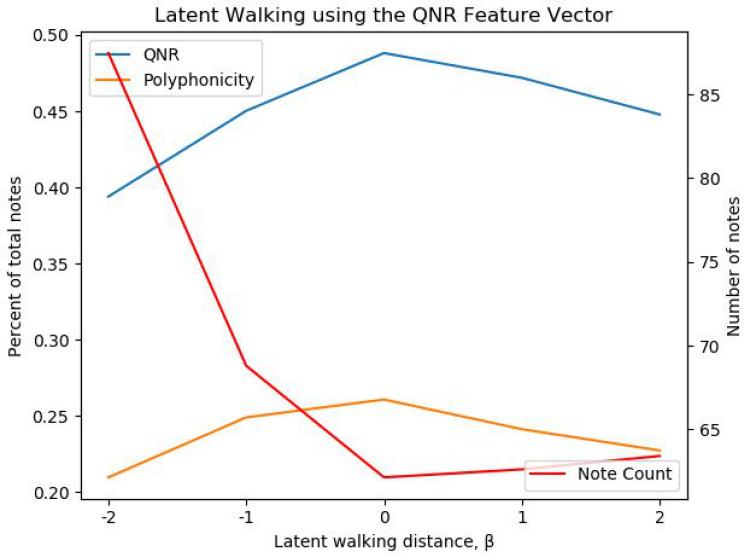


Figure 35: Local distribution of metrics across 1,000 samples after walking the QNR boundary

The behavior of the note count and polyphonicity feature vectors is much more predictable and usable for conditional generation with the restrictions on

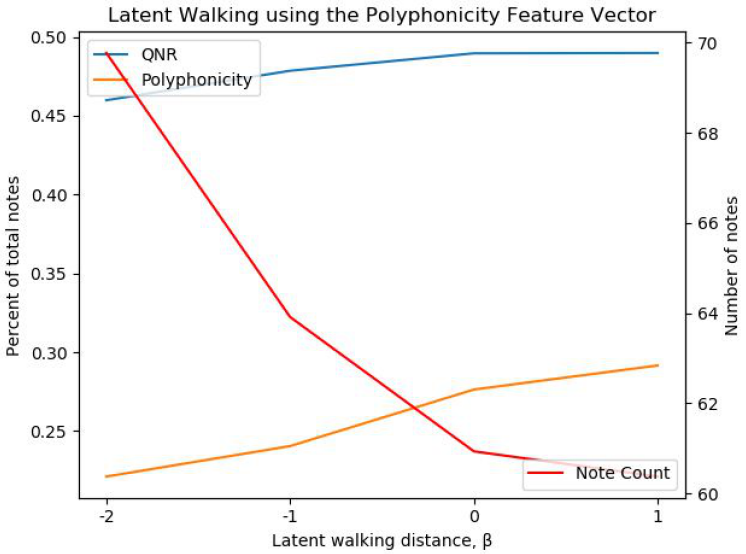


Figure 36: Local distribution of metrics across 1,000 samples after walking the polyphonicity boundary

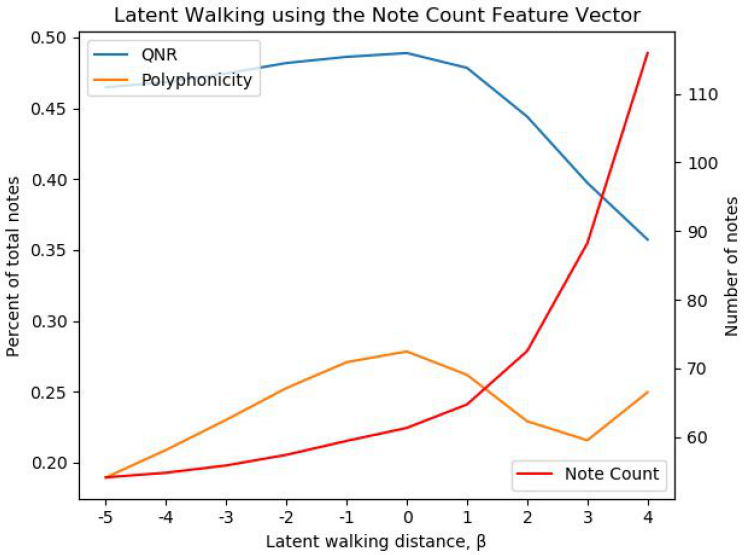


Figure 37: Local distribution of metrics across 1,000 samples after walking the Note Count boundary

β. The QNR feature vector still lacked control over the target metric, motivating a further look into why this failure would arise.

6.6 Dataset Truncation

While the linear SVM was able to learn a hyperplane boundary that correctly classified 74% of the latent samples analyzed, the feature vector produced from that hyperplane was unable to increase the average QNR of the 1,000 samples that were used for latent walking. For comparison, the SVMs for polyphonicity and note count had classification accuracies of 71% and 70% respectively, and both saw better latent walking performance than QNR.

The average value for QNR was 0.4845 and the highest value was 0.6926 in the original 25,000 songs generated. This shows there is room for the average QNR to increase, so the failure of latent walking cannot be blamed solely on under-representation in the latent space.

A possible explanation would be that the linear SVM trained on extreme-valued samples which were not representative of the latent space as a whole. A solution to this is to truncate the dataset to remove the samples with the top and bottom QNR ratings before training the SVM. I decided to drop out 20% of the data from each end, removing 10,000 samples and training on the median 15,000 samples. Unfortunately the SVM accuracy decreased to 60%, meaning that removing the extreme values resulted in a latent vector with even weaker correlation to the overall dataset than the original.

One last attempt was made by binarizing this data before running it through the SVM. The 15,000 median samples were used, and any sample with a value above the mean, 0.4845, was set to 1 while the remaining values were set to 0. This was intended to give the linear SVM an advantage in creating a clear disctinction between high and low QNR, hopefully increasing the classification accuracy. The resultant boundary had the lowest accuracy yet: 52%, which amounts to little more than random choice.

The plot for all three metrics is shown below for latent walking using the truncated QNR data (15,000 samples, non-binarized), figure 38. While the QNR curve is smoother and has less impact on the other metrics, i.e. it is less entangled, it still does not perform in the desired manner. The feature vector, on average, is unable to increase QNR from the initial value.

This motivated one last look at the QNR feature vector, the non-SVM approach to finding a latent direction, as presented in MusicVAE [27].

6.7 Non-SVM Approach

This approach to finding latent feature vectors relies solely on linear algebra. It still assumes a linear latent space, but does not rely on a SVM. The process to find the feature vector involves sorting all 25,000 latent samples by their QNR, then dividing the dataset into five partitions. A centroid is found for each partition, in theory pointing to a "hotspot" for the average QNR value of that partition. A linear interpolation is found from the centroid for the lowest

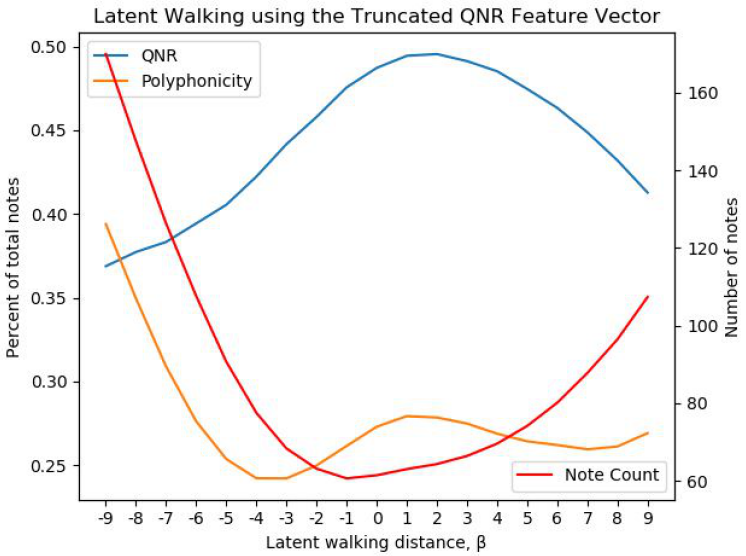


Figure 38: Distribution of metrics across 1,000 samples after walking the Truncated QNR boundary

QNR to the centroid for the highest QNR and this vector is used as the feature vector for latent walking.

For my dataset, I partitioned the 25,000 samples into groups of 5,000, and computed the mean and variance for the highest and lowest QNR groups, shown in figure 39.

	Average QNR	QNR Variance	Location Variance
Group 1	0.3916	0.0023	0.7800
Group 2	0.4583	0.0001	0.7737
Group 3	0.4898	0.0001	0.7703
Group 4	0.5187	0.0001	0.7709
Group 5	0.5615	0.0005	0.7708
Full Dataset	0.4839	0.0039	0.7737

Figure 39: Mean and Variance of the 25,000 QNR samples. Location refers to the 128-dimensional latent code.

The variance in location was quite high and also very consistent across all partitions, as well as matching the variance of the unpartitioned data. The distance between each of these centroids was computed, and is shown in figure 40. The distances were minuscule, well within the variance of each group, showing no clustering in the latent space by QNR value.

	Distance between centroids					
	Group 1	Group 2	Group 3	Group 4	Group 5	Full Dataset
Group 1	0	0.006	0.0069	0.0065	0.0066	0.0052
Group 2	0.006	0	0.0008	0.0004	0.0005	0.0009
Group 3	0.0069	0.0008	0	0.0004	0.0003	0.0017
Group 4	0.0065	0.0004	0.0004	0	0.0001	0.0013
Group 5	0.0066	0.0005	0.0003	0.0001	0	0.0014
Full Dataset	0.0052	0.0009	0.0017	0.0013	0.0014	0

Figure 40: Euclidean distance between the centroids of each group, partitioned by QNR value

The feature vectors determined using this method were not used for latent walking. The variance and close proximity of the centroids for all five partitions strongly suggests there is no structure in the latent space of MuseGAN for QNR. This is likely a limitation of the network; with a latent space dimensionality of 128, there is a limited amount of information the model can encode. An improved architecture may be able to include information about QNR.

After completing this analysis, I believe that it will be impossible to control QNR in MuseGAN using any conventional latent walking techniques, due to the lack of latent features corresponding to QNR.

6.8 Entanglement

As can be seen from the figures in section 6.4, e.g. figure 33, latent walking the target feature also changes other aspects of the generated music, including the other measured metrics. This correlation is called entanglement and the theory behind it is explained in detail in section 4.5.

The hyperplanes found using the SVM were relatively orthogonal; the values for entanglement are shown in figure 41. A value of 1 represents two planes facing the same direction while a value of 0 is two orthogonal planes. The sign of the entanglement tells if the normal vectors to the hyperplanes are oriented in the same (+) or in opposite (-) directions.

The labels with an "_**T**" represent the truncated boundaries. The boxes with a dark border show the similarity between a boundary and its truncated counterpart. Interestingly, despite having a low correlation, the truncated and original boundary for note count both worked well to produce a usable feature vector, showing that there are multiple directions that can be latent walked to achieve the same conditioning effect. See appendix A for plots involving the truncated note count boundary.

	QN	PP	NC	QN_T	PP_T	NC_T
QN	1	0.35951704	-0.34886378	0.18390617	0.23267594	-0.20508622
PP	0.35951704	1	-0.16619876	0.080380216	0.23816673	-0.1349155
NC	-0.34886378	-0.16619876	1	0.17678372	-0.06803946	0.29184407
QN _T	0.18390617	0.080380216	0.17678372	1	0.065104894	0.11454768
PP_T	0.23267594	0.23816673	-0.06803946	0.065104894	1	0.11454768
NC_T	-0.20508622	-0.1349155	0.29184407	0.11454768	0.11454768	1

Figure 41: The entanglement of each of the hyperplanes found using the SVM approach

7 Conclusion

Artificial music generation is a rapidly developing field with many unanswered questions and challenging problems. As the networks improve and the music becomes more realistic, there is a motivation to develop tools to shape the resultant music; one such tool is latent walking. After conducting my research on one of the best-performing GAN models, MuseGAN, I have shown that latent walking is possible to condition select features of the generated music. Despite not having a conditioned latent space, MuseGAN's generator provided sufficient structure to manipulate the generation samples in a predictable manner with latent walking. While one of the three metrics failed to provide meaningful conditioning, I believe this is due to a limitation of the network and the amount of information encoded in the generator. The success of the other two metrics shows a "smooth" structure in the semantic features of the generated samples which was sufficient to enable latent-walking conditional generation. This latent walking approach is better than the comparable technique of priming because it does not require a target sample and allows more control over the features imparted to the music. There is also greater flexibility with latent walking in conditioning multiple features, as well as capability to control more abstract features like quality or genre.

Latent walking is not without its downsides; it requires a significant amount of computation to generate a sufficient dataset as well as a method of feature extracting the target metric. Even with all of these, there is not a guarantee that the relevant feature will be encoded smoothly by the generator, as seen with QNR in the examples above. Additionally, there is still a lack of understanding about how features in the latent space of a GAN are formed and what feature vectors would best approximate feature directions. These downsides can hinder the ability to latent walk features by large amounts, as seen in the results from these experiments. Overall, latent walking is an effective means to condition GAN generated music. Within appropriate bounds, the results are controllable and predictable, as long as the feature in question exhibits linear behavior in the latent space.

7.1 Future Work

There are many opportunities for future work in the area of music-domain latent walking, as this is a largely unexplored topic. Some of the extensions of this work would be exploring additional GAN networks, or even additional versions of the MuseGAN network to determine how consistent the results shown here are between distinct training runs. In particular, I believe that increasing the latent dimension from 128 to a larger number may provide increased latent walking ability for features like QNR which were not captured in the latent space of the network used for this paper. Another extension of this work is the addition of more metrics, either related to music quality or not, to further understand the capabilities of latent walking to control multiple facets of music generation. There is also the ability to disentangle the learned directions to control features independently. This was not explored in-depth due to a large amount of both entanglement and noise in the learned directions, but disentanglement may be useful if a more accurate method is used to learn the feature vectors. One such method is to learn a non-linear representation of the latent features, which may provide improved latent walking capabilities. Finally, a comparison could be made using the presented latent walking techniques on an acoustic-domain GAN (versus the symbolic-domain MuseGAN used here) to examine the impact of music representation on the ability to latent walk.

8 References

References

[1] Pypianoroll documentation: https://salu133445.github.io/pypianoroll/, 2020.

[2] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

[3] Gino Brunner, Andres Konrad, Yuyi Wang, and Roger Wattenhofer. Midi-vae: Modeling dynamics and instrumentation of music with applications to style transfer. *arXiv preprint arXiv:1809.07600*, 2018.

[4] Gino Brunner, Yuyi Wang, Roger Wattenhofer, and Sumu Zhao. Symbolic music genre transfer with cyclegan. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 786–793. IEEE, 2018.

[5] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[6] Arden Dertat. Applied deep learning - part 3: Autoencoders, Oct 2017.

[7] Google Developers. Overview of gan structure — generative adversarial networks, 2020.

[8] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music.

[9] Sander Dieleman. Generating music in the waveform domain, Mar 2020.

[10] Sander Dieleman, Aaron van den Oord, and Karen Simonyan. The challenge of realistic music generation: modelling raw audio at scale. In *Advances in Neural Information Processing Systems*, pages 7989–7999, 2018.

[11] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Demonstration of a convolutional gan based model for generating multi-track piano-rolls. *ISMIR Late Breaking/Demos*, 2017.

[12] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[13] Hao-Wen Dong and Yi-Hsuan Yang. Convolutional generative adversarial networks with binary neurons for polyphonic music generation. *arXiv preprint arXiv:1804.09399*, 2018.

[14] Douglas Eck and Juergen Schmidhuber. Finding temporal structure in music: Blues improvisation with lstm recurrent networks. In *Proceedings of the 12th IEEE workshop on neural networks for signal processing*, pages 747–756. IEEE, 2002.

[15] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[16] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[17] Jack Grundy. Are neural nets close to producing real art?, Apr 2019.

[18] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the maestro dataset. *arXiv preprint arXiv:1810.12247*, 2018.

[19] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017.

[20] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. *arXiv preprint arXiv:1912.04958*, 2019.

[21] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[22] Wonkwang Lee, Donggyun Kim, Seunghoon Hong, and Honglak Lee. High-fidelity synthesis with disentangled representation. *arXiv preprint arXiv:2001.04296*, 2020.

[23] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. Samplernn: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*, 2016.

[24] Olof Mogren. C-rnn-gan: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*, 2016.

[25] Gerhard Nierhaus. *Algorithmic composition: paradigms of automated music generation*. Springer Science & Business Media, 2009.

[26] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[27] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. *arXiv preprint arXiv:1803.05428*, 2018.

[28] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9243–9252, 2020.

[29] Aaron van den Oord, Oriol Vinyals, et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6306–6315, 2017.

[30] Sean Vasquez and Mike Lewis. Melnet: A generative model for audio in the frequency domain. *arXiv preprint arXiv:1906.01083*, 2019.

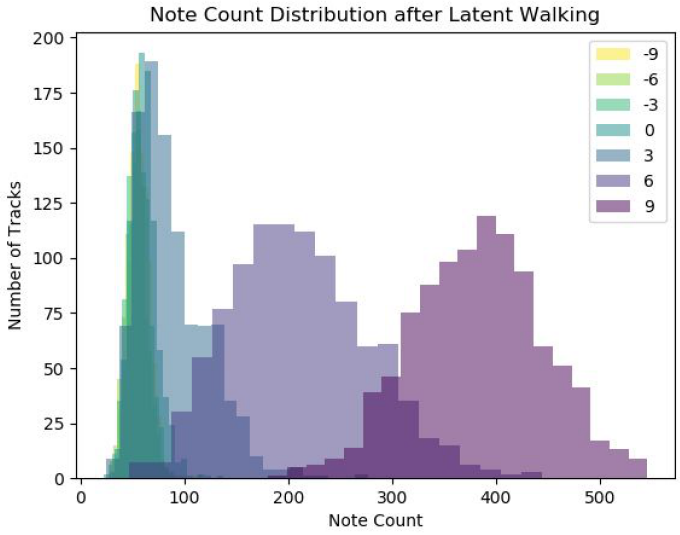
[31] Andrey Voynov and Artem Babenko. Unsupervised discovery of interpretable directions in the gan latent space. *arXiv preprint arXiv:2002.03754*, 2020.

[32] Jian Wu, Changran Hu, Yulong Wang, Xiaolin Hu, and Jun Zhu. A hierarchical recurrent neural network for symbolic melody generation. *IEEE Transactions on Cybernetics*, 50(6):2749–2757, 2019.

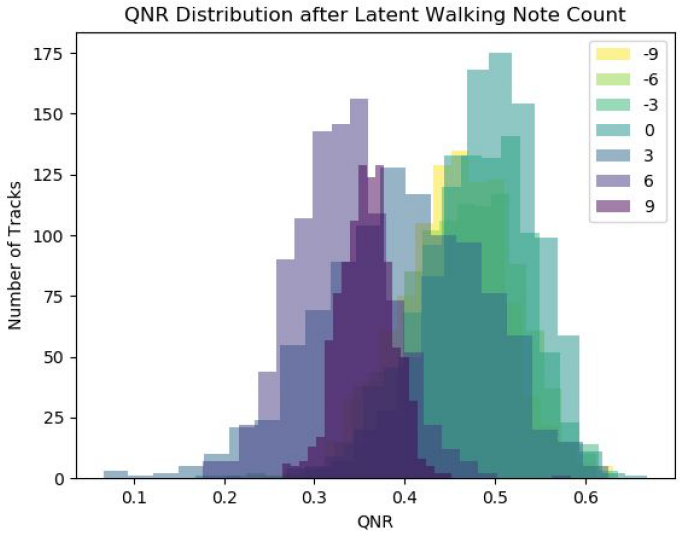
[33] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint arXiv:1703.10847*, 2017.

A Supplementary Figures

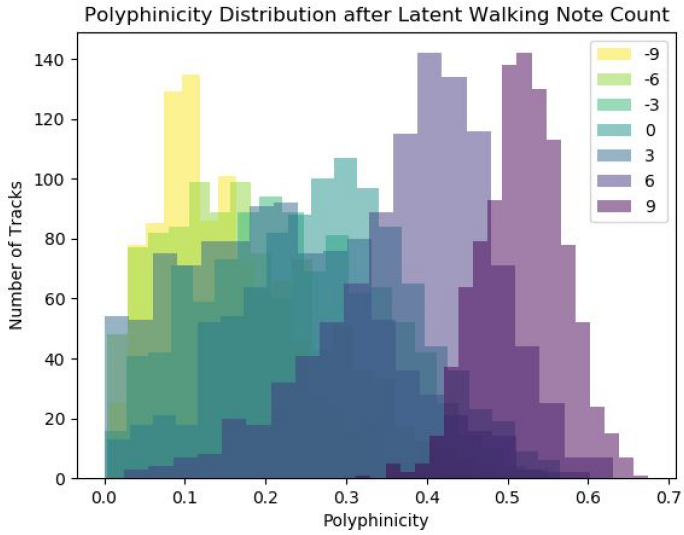
A.1 Note Count Boundary



The distribution of note counts at select β values along the note count boundary



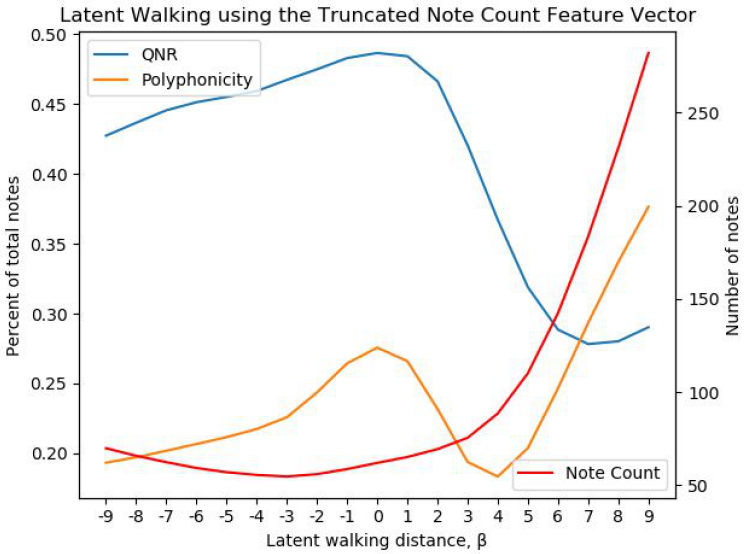
The distribution of QNR at select β values along the note count boundary



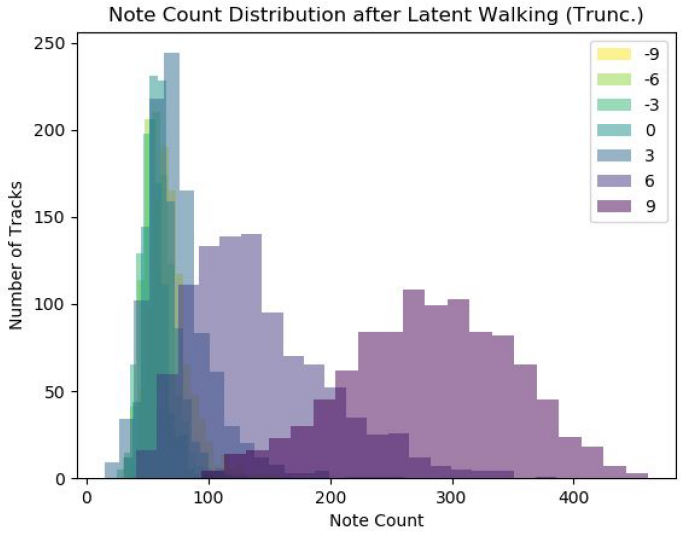
The distribution of polyphonicity at select β values along the note count boundary

A.1.1 Truncated Note Count Boundary

*Following the truncation technique described in section 6.6

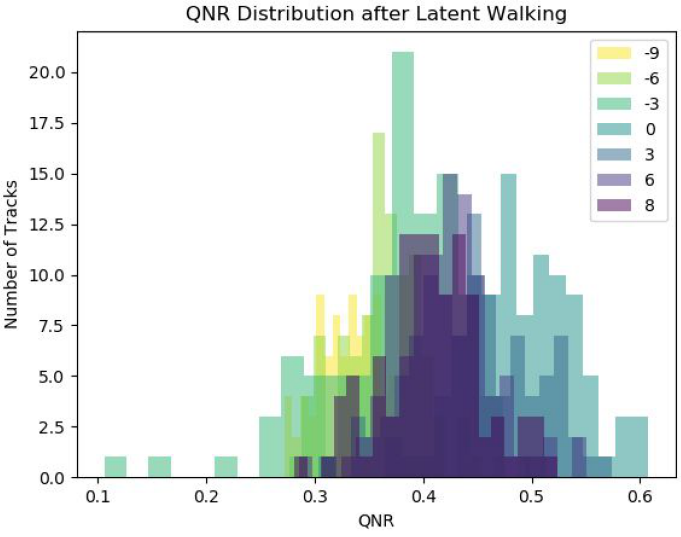


The average value of each of the metrics at select β values along the truncated note count boundary

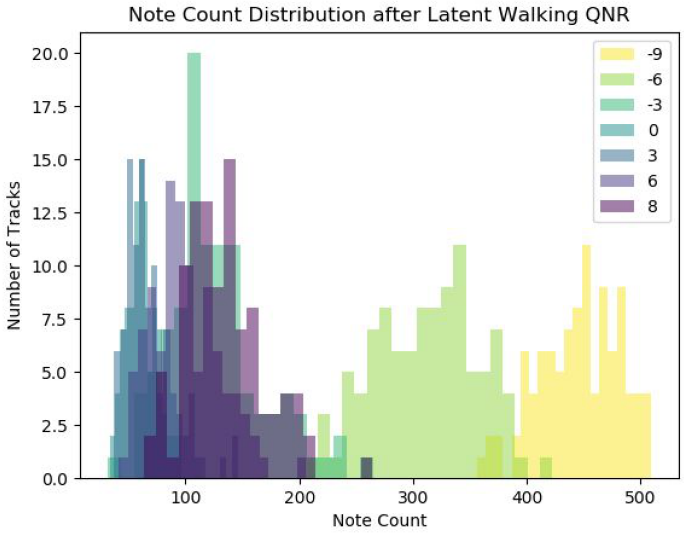


The distribution of note counts at select β values along the truncated note count boundary

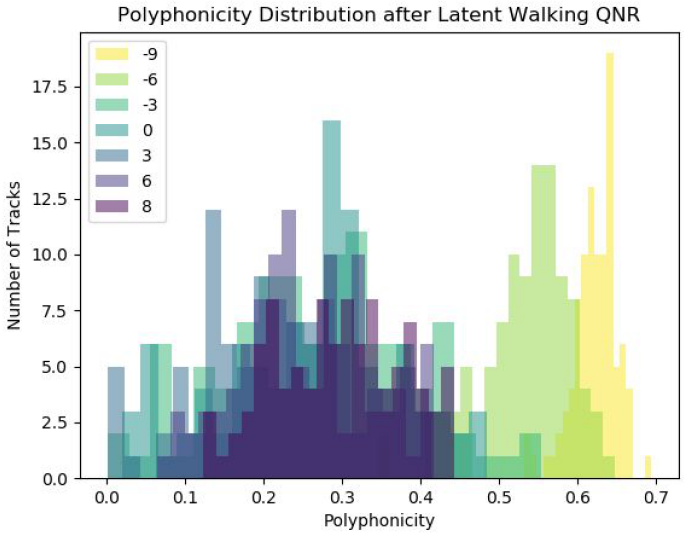
A.2 QNR Boundary



The distribution of QNR at select β values along the QNR boundary

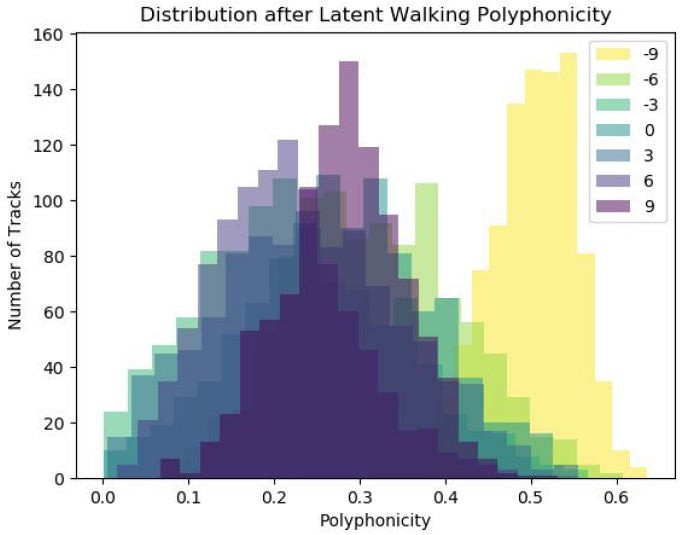


The distribution of note count at select β values along the QNR boundary

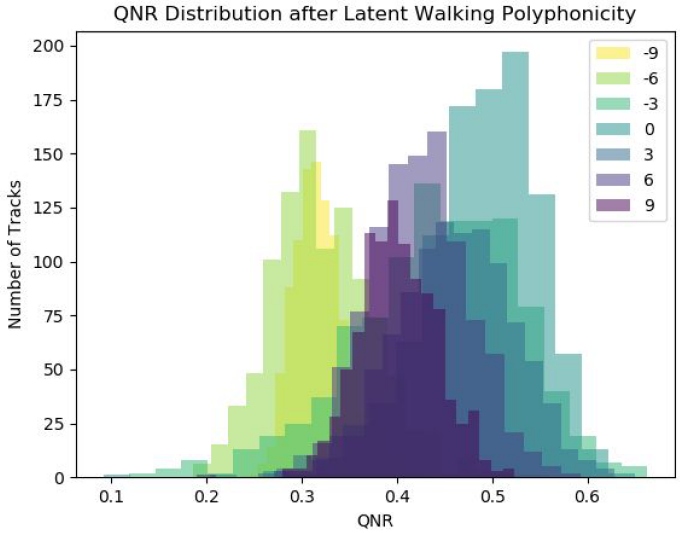


The distribution of polyphonicity at select β values along the QNR boundary

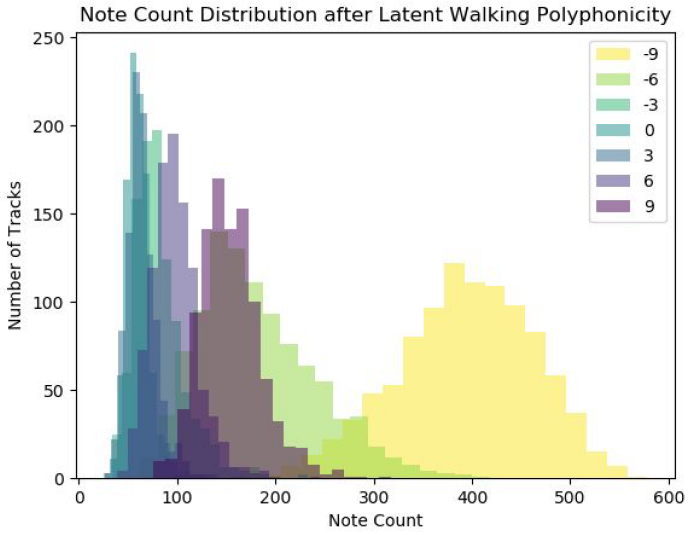
A.3 Polyphonicity Boundary



The distribution of polyphonicity at select β values along the polyphonicity boundary



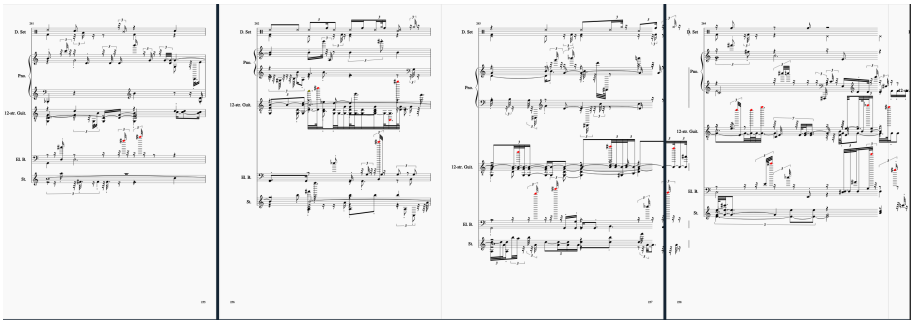
The distribution of QNR at select β values along the polyphonicity boundary



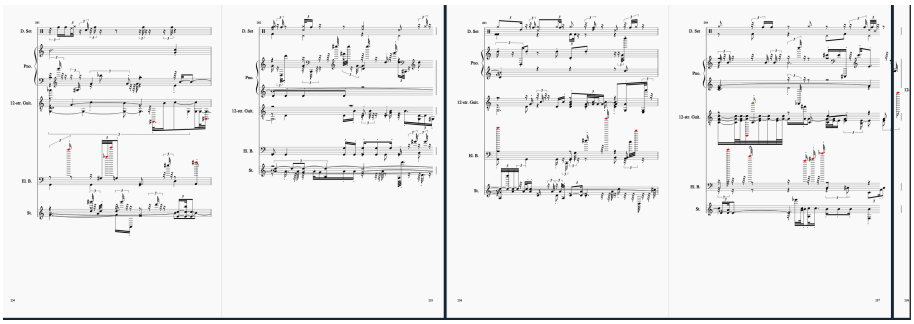
The distribution of note count at select β values along the polyphonicity boundary

B Sample Sheet Music

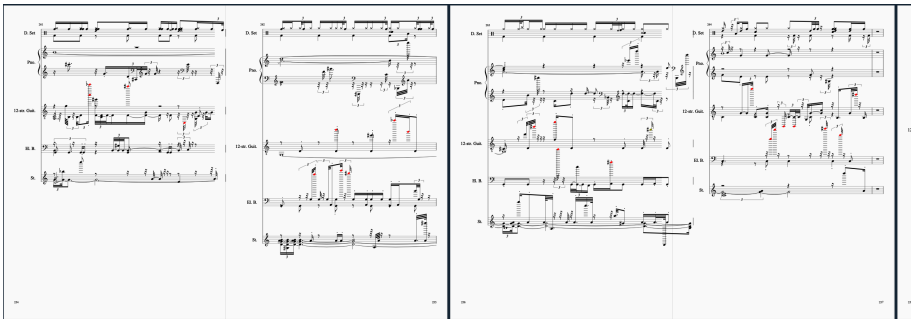
A random bar of music was selected from the note count latent walking experiment. It was converted from pianoroll to midi file by pypianoroll and then to sheet music by the application MuseScore3.



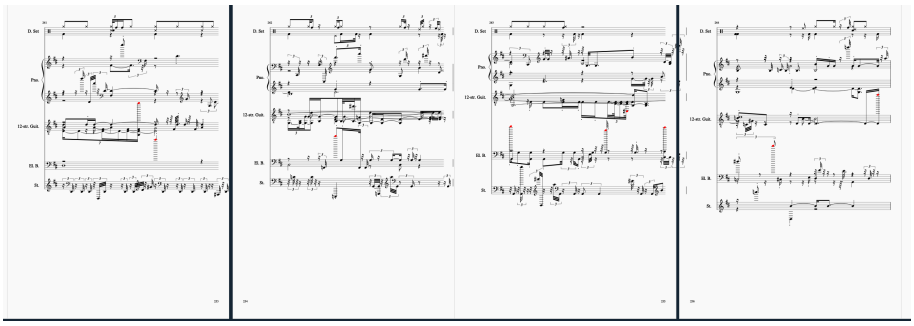
Random sample of generated music without latent walking, $\beta = 0$.



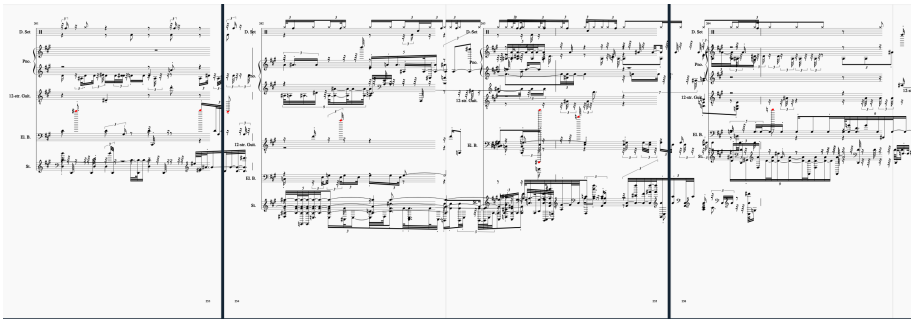
Random sample of generated music latent walked along the note count boundary, β = -1.



Random sample of generated music latent walked along the note count boundary, β = -3.



Random sample of generated music latent walked along the note count boundary, $\beta = +1$.



Random sample of generated music latent walked along the note count boundary, $\beta = +3$.