Heinz 95-845: Music Transformation using Neural Networks to Produce Vocal Sounds

Jiayong Hu jiayongh/jiayoungh@andrew.cmu.edu
Department of Civil and Environmental Engineering
Carnegie Mellon University
Pittsburgh, PA, United States

Hao Wu haowu2/haowu2@andrew.cmu.edu
Department of Civil and Environmental Engineering
Carnegie Mellon University
Pittsburgh, PA, United States

Nicholas Wells nwells/nwells@cmu.edu
Heinz College of Information Systems and Public Policy
Carnegie Mellon University
Pittsburgh, PA, United States

Abstract

The creation of music using neural networks has seen advancements in recent years, not only has music been generated from midi files but also from raw signals. Though there have been advancements, there seems to be little work done in relation to the production of vocal music. This may be because it combines two difficult problems that are not easily solved, human voice generation and music generation. The generation of human voice has also made advancements in recent years and we build on the knowledge gained from that work. Using neural networks we explore ways in which vocal music could be produced using raw music files. Ultimately, our attempts proved unsuccessful. Much greater understanding of signals, and higher computation power is needed.

1. Introduction

Neural networks have been used to do all sorts of things: imitating the human voice, and generating instrumental music. Our interest is in seeing if vocal music can be produced from already provided instrumental music. By doing this, we would be able to provide a tool to assist a composer in the composition of music. Background music could be provided, and some generated vocal music could be placed over the background music.

This problem is a combination of two problems: the generation of the human voice, and the production of music. Each of these problems are being tackled by others, one example being Mao et al. (2018). WaveNet, a product of Google's DeepMind, is leading in both of these problems, see van den Oord et al. (2016a). This is a tool that has generated both music and human voice, but, as far as we can tell, it has not been used to create both together.

Unfortunately, this is propriety and not publicly available for us to work with, otherwise we would have used WaveNet to try generating lyrical music.

Some work has been done in relation to vocal music as seen here by Koretzky (2019), but this is opposite to the problem we are focused on. This aims to remove vocals from music, where we are trying to generate it. The results achieved from this method are very impressive, and give hope to our potential in generating music.

A deeper understanding of how music and neural networks have done together is laid out in these two articles by Pons (2018) and McDonald (2017). There certainly has been a rich history in working with the two. We will not expound on that history here.

We will begin our paper by explaining what we intend to accomplish with generating vocal music, and how we plan to do it. After that we will explain the various models we use to achieve our goal. The methods used to perform the experiments are laid out and the results are provided. We end with a brief discussion about our results and potential ways to build upon what we have learned.

2. Background

Our aim is produce lyrical music from raw signals. Music, or more generally sound, is made up of sound waves which do not easily translate into a digitally encoded form. To encode them, the wave is sampled many times per second; the more frequent the sample, the more accurate representation of the actual wave. This project sampled music at a rate of 22,050 times per second. Here is an example of what a sound wave looks like when sampled this way:

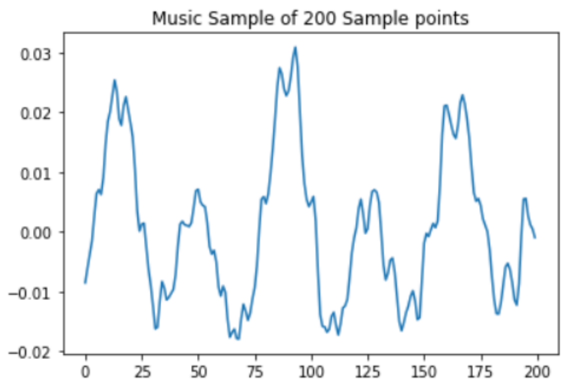


Figure 1: Music Sample

Compare this with regular sound waves we see in a textbook and we begin to realize that this is highly complex. This is only 200 samples, or just under one percent of a second.
To produce lyrical music from raw signals, we have decided the best way to transform music that does not have lyrics into music that does have lyrics. This problem is much more simple

than trying to create music signals from nothing. To illustrate this, consider the two sound waves below:

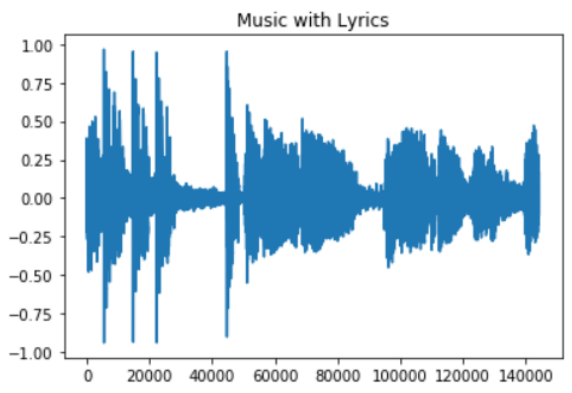


Figure 2: Music Sample with Lyrics

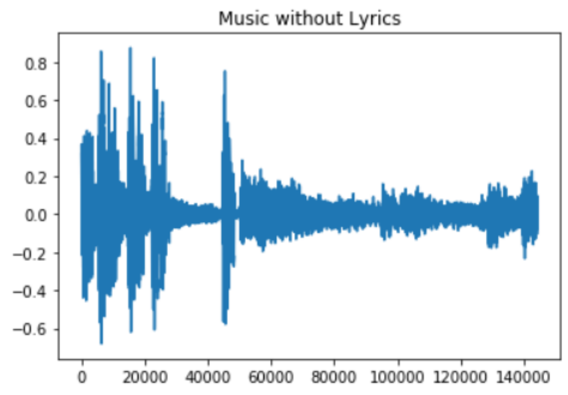


Figure 3: Music Sample without Lyrics

Figure 2 and Figure 3 show the same moment in time on a song, the first with lyrics, and the second without lyrics. From these images it can be seen that the sound waves are capturing the lyrical data. Since these sound waves are digitally encoded, there is a numerical difference between these two waves. By using neural networks, we want to see if our models can discover this difference and learn to replicate it on its own.

3. Method: RNN, PixelCNN, GAN

We are using 3 different models. Simple RNN, PixelCNN, and a Generative Adversarial Network. The simple RNN will act as a baseline model for the music we are able to produce. Code for each of these models is available at https://github.com/Nyakult/S19-aamlp.

3.1 RNN

We have chosen to use Recurrent neural networks (RNNs) because they can handle sequences of data. In order for a neural network to accurately understand what is happening within music, especially from a raw signal, it needs to learn what has come before it. Music is highly structured, with rhythm and pitches. Without knowing what the current rhythm pattern is, and knowing what pitches were just played, there is no way to create music that would sound pleasing to a human.

The Simple RNN is acting as our baseline since it is the least complicated of the models we are using. Our simple RNN is defined by only having three layers: the RNN input layer, one dense with 128 units, and a time distributed dense layer as the output. The model is run in batches of 128.

3.2 PixelCNN

Pixel Convolution Neural Network(PixelCNN) van den Oord et al. (2016b) is a popular auto regressive models that can generating images by modeling the joint probability distribution of the data we feed in.

To model the distribution of images, PixelCNN make the following assumption about pixel intensities: the intensity value of a pixel is dependent on all pixels traversed before it. The image is traversed left-to-right and top-to-bottom along the image.

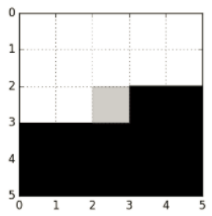


Figure 4: masked convolutions

So the joint probability of image x can be represented as the product of probability $p(x_i|x_1, x_2, ...x_{i-1})$
.

$$p(x) = \prod_{i=1}^{n^2} p(x_i|x_1, x_2, ...x_{i-1})$$

We can make similar assumption about our audio signal: the intensity value at certain point is dependent on all points traversed before it. The audio is traversed according to the time(left to right).

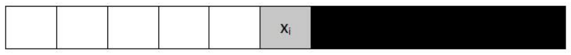


Figure 5: audio

Then, like pixelCNN, we build our model described in Table 1 and Figure 6.

Table 1: model detail

PixelCNN
1*63 conv mask A
1*1 conv mask B
ReLU followed by 1*1 conv mask B
1*1 conv mask B
output, loss = MSE

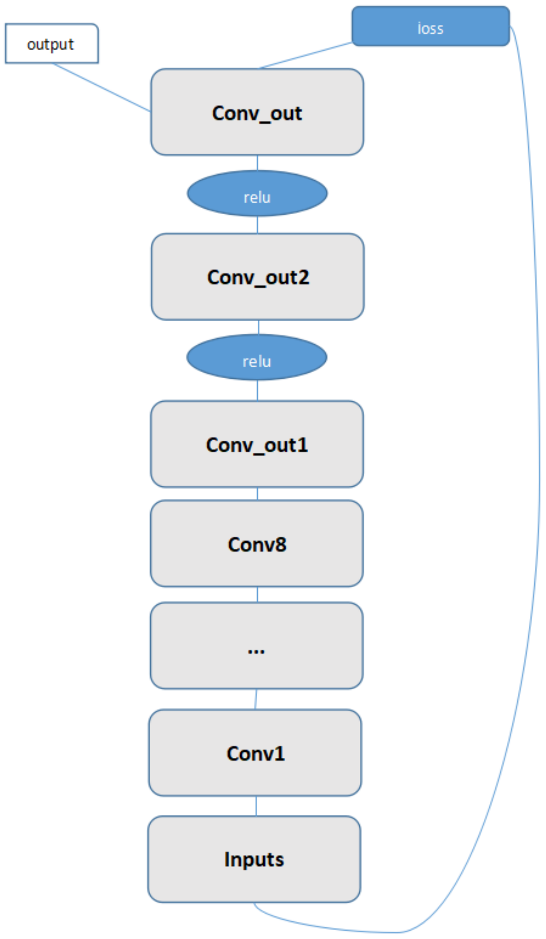


Figure 6: pixel CNN

3.3 GAN

A generative adversarial network (GAN) is a machine learning system with a generative network generating candidates from a latent space of data distributions and a discriminative network trying to distinguish between candidates and true data distribution. The generators effort to fool the discriminator which becomes better at telling candidates from true data at the same time is based on the idea of zero-sum game, according to Goodfellow et al. (2014) and Radford et al. (2016)

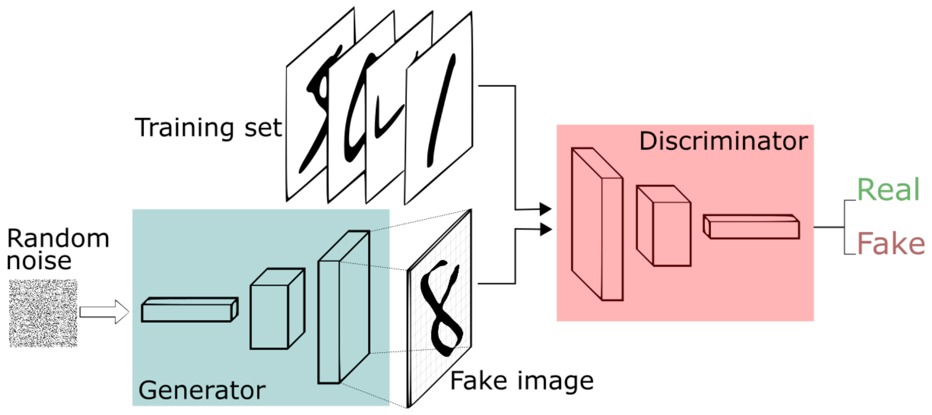
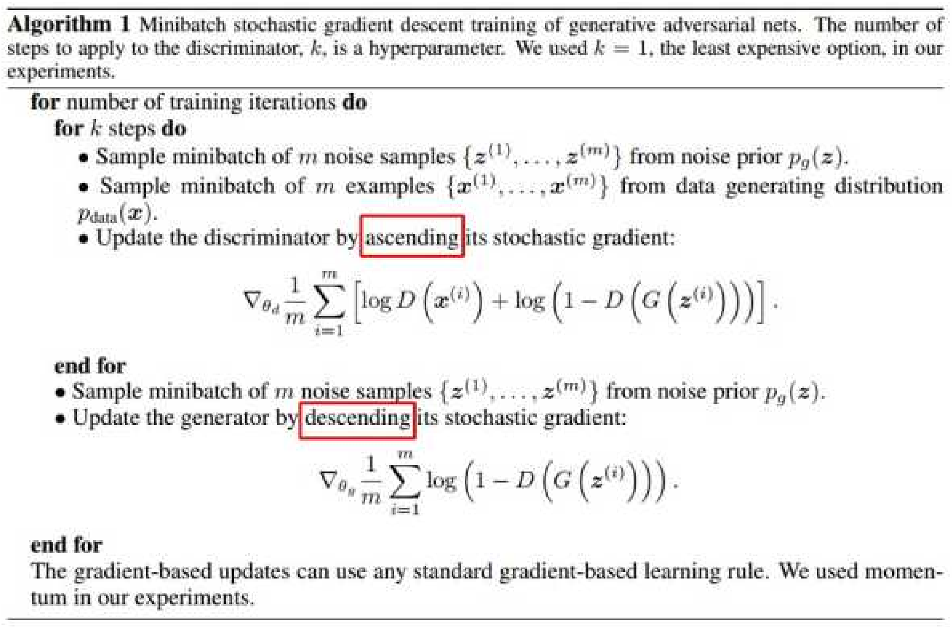


Figure 7: GAN



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
  for $k$ steps do
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Sample minibatch of $m$ examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
    • Update the discriminator by $\boxed{\text{ascending}}$ its stochastic gradient:
        $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$
  end for
  • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
  • Update the generator by $\boxed{\text{descending}}$ its stochastic gradient:
      $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$
end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
```

Figure 8: Algorithm for GAN

Although GANs have achieved great accomplishment in image, there are way much fewer researches and applications in audio. One reason is that RNNs beats GANs in generation quality, the other is that GANs are extremely hard to train. In the process of GANs construction, two problems emerged:

- Few GANs were designed to process audio files. Instead of using MP3 or WAV, almost every GAN uses MIDI files to generate music.
- Most GANs including C-RNN-GAN run on Tensorflow, nevertheless I run my models on PyTorch.

I have no choice but to build my own GANs based on others' models. Code is available at: https://github.com/BigT0e/GANs-for-generating-songs
Due to the fact that GANs are hard to train, I summarized my own pipeline for GANs construction:

- Design the basic structure for GAN.
- Choose the appropriate optimizer.
- Determine whether generator is stronger than discriminator or other way around and adjust the number of iterations for each of them.
- Update the structures of networks.
- Test and decide the number of epochs.

The experiment design for the construction of GANs include three models: simple GAN, RNN GAN, and CNN GAN.

The song tested for the purpose of model construction is a 10 seconds audio file cut from a Japanese song called Unreachable Love. The input for discriminator is the song with human voice and the input for generator is the song of pure background music.

After the construction of GANs, I chose the model with the best performance for the scope our project.

3.3.1 SIMPLE GAN

My simple GAN is designed based on a GAN which trains two nets to battle on a shifted/scaled Gaussian distribution. The optimizer chosen is Adam. Both generator and discriminator have only two hidden layers of 100 units. The input size and the output size are 220500, which seems extremely large compared to the number of units in each layer.

After days of hyper-parameter tuning, the results are astonishing. The generator successfully replicated the true song after 500 epochs even though the output is very noisy. $D(fake)$ is close to zero while $D(real)$ is around 0.60-0.65.

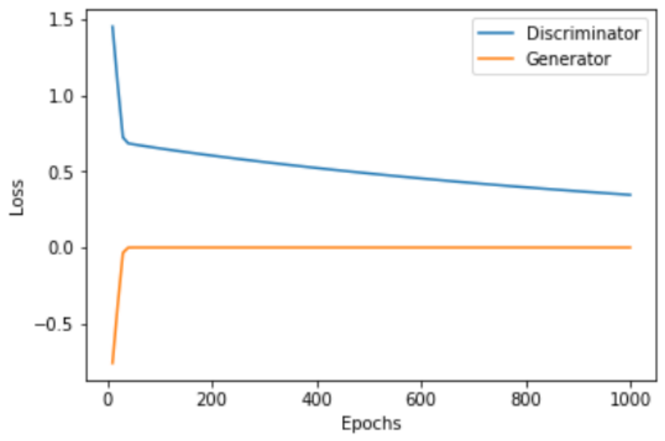


Figure 9: Loss function of simple GAN

The loss functions for both discriminator and generator change rapidly before 50 epochs. With the growth of epochs, both loss functions smoothly converge to 0.

However, there are several problems with simple GAN. First, the discriminator outperforms the generator. Second, the input size and the output size for a 10 seconds song is so large that the file size of the trained generator is 216 MB. In order to reduce the noise in the output, I had to add more layers and units, which crashed the memory in Google Colaboratory and failed to achieve better performance. It seems more reasonable to process time series data by incorporating RNN instead. Finally, the generator trained is such a strong imitator that anything runs through it become the noisy version of the true song.

3.3.2 RNN GAN

C-RNN-GAN is a generative adversarial model that works on continuous sequential data and is trained on a collection of classical music, based on the work of Mogren (2016). By using a similar structure and learning the models from Theis and Bethge (2015), I built my own RNN GAN. Both generator and discriminator use two layers of LSTMs and two fully connected hidden layer. The activation functions for generator are leaky ReLU and Tanh while the activation functions for discriminator are leaky ReLU and Sigmoid. RMSprop is chosen as optimizer and gradient clipping are implemented in order to prevent gradient vanishing and gradient explosion.

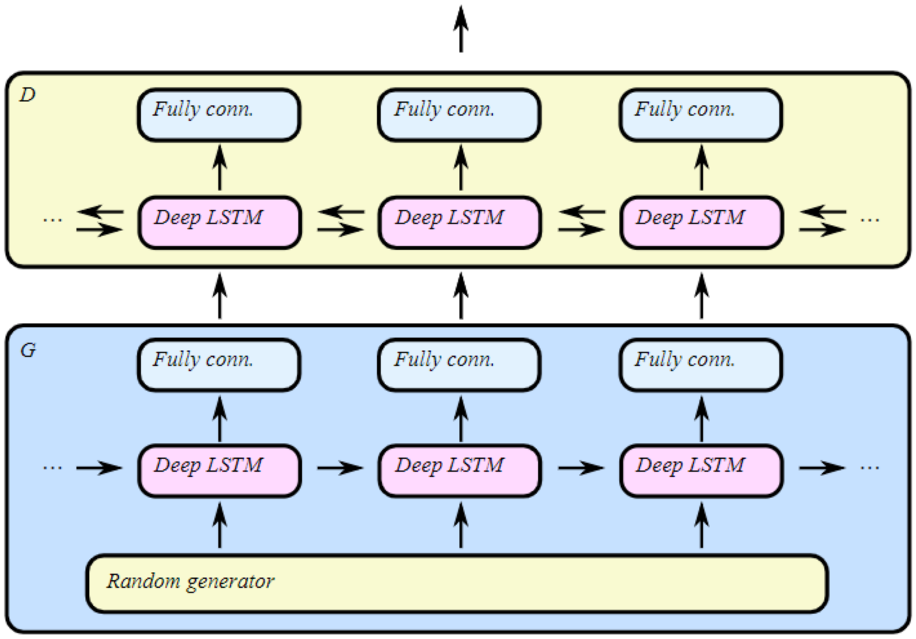


Figure 10: C-RNN-GAN

What prevents models from producing good result is the different learning capacity between the discriminator and generator. In each epoch, the discriminator iterates 10 times while the generator iterates 1 time. After 100 epochs, a music with vague human voice and lots of noise was produced, proving the feasibility of my construction of RNN GAN. $D(real)$ is around 0.40-0.50 while $D(fake)$ is around 0.35-0.40.

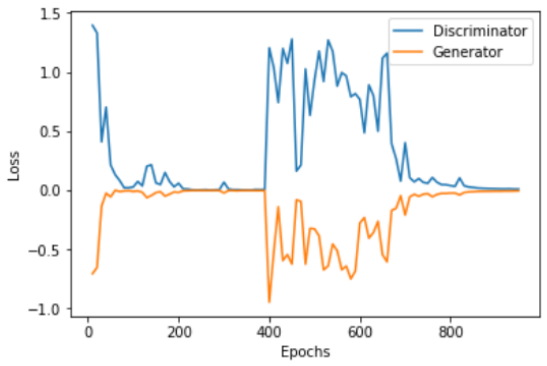


Figure 11: Loss function of RNN GAN

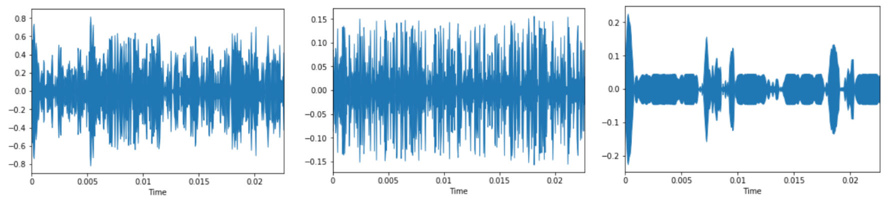
However, the discriminator and the generator haven't been properly balanced since the output songs will be noisier and indistinguishable if trained with more epochs. Moreover, the sudden changes in loss functions at 400 epochs indicate that the models are unstable and there is no guarantee for better performance with more epochs.

3.3.3 CNN GAN

There is no doubt that CNN can generalize better than simple GAN. Moreover, the core idea of WaveNet is using modified CNN to process time series data, which makes the CNN GAN promising. Unfortunately, the duration of the project is only one month and resources we have such as computation power are limited, so that it is impossible for me to implement CNN GAN in this project. Further research will be conducted to implement the construction and testing of CNN GAN in the future.

3.3.4 COMPARISON OF PERFORMANCE

The wave plots for part of the inputs for discriminator and the outputs for different kinds of GANs are shown below.



(a) Sample Song (b) Simple GAN (c) RNN GAN

Figure 12: The wave plots for sample song, simple GAN and RNN GAN

Simple GAN is a strong imitator that basically produce the noisy version of the input of discriminator. RNN GAN is a weak imitator but an excellent generalizer that captures the trend of the sample song in a certain way. While RNN GAN doesn't have a significantly better performance than simple GAN, it saves a lot of memory and properly handles time series data. In conclusion, RNN GAN was chosen to run evaluation.

4. Experimental Setup

4.1 Cohort Selection

We chose to use songs produced in the same genre, under the same artist. Since this a first attempt we want to make it as simple as possible for the algorithms. By using the same artist, the neural networks will be able to better learn the nuances of an individual singer, without the confusion of adding more vocalists in. The artist we chose is Jay Chou. We will train and test our models using these 10 songs:

- Thousands of miles away
- Listen to my mother's words
- Chapter 7 of the Night
- Heart Rain
- Compendium of Materia Medica

- White Windmill
- Red Imitation
- Chrysanthemum Taiwan (”The Golden Flower of the City” movie endings)
- Rosemary
- Back

The beginning and ending of each song are dropped because these parts of a song do not typically have vocal music. This will help the neural nets by feeding in less music that does not have vocal sound.

4.2 Data Extraction

Using audio processing library librosa, we can extract audio data from mp3 file with the original sampling rate and represent it as an array. This method allows us to encode the raw signal in a numerical way that can be understood by neural networks.

4.3 Feature Choices

Raw sound waves do have features that could potentially be used to solve this problem. We tried to use audio features like Chroma Frequencies, Spectral Centroid, Spectral Bandwidth, Mel-frequency cepstral coefficients, etc., to represent the music and generate features from these features, but this method did not work as well as generating music from the raw BGM wave. Ultimately, we decide not to do the feature engineering for this project.

4.4 Comparison Methods

As explained above, this is a comparison of three models: Simple RNN, PixelCNN, and RNN GANs. Each will be trained on the same eight songs, and tested on the last two. Having the same input and the same output will enable us to evaluate each of these models on a similar scale. If one model out-preforms another it will be because it is better, not because of the data used.

4.5 Evaluation Criteria

We compare the performance of the models by evaluating the music they generated using mean opinion scores(MOS).

After listening to the result, our team members are asked to rate the change in pitch, change in rhythm, change in volume, change in way the static sounds (timbre), and the presence of human voice in a five-point Likert scale score(0: Bad, 1: Poor, 2: Fair, 3: Good, 4: Excellent).

Because the generated music is very similar, we have chosen to only evaluate the first 90 seconds of output music generated by each model. This still produces a fair understanding of how the model preforms overall.

The average of the scores were then taken for each category (pitch, volume, etc.), as well as an average for the whole segment. These scores provide a mechanism for comparison between the three models.

5. Results

The results for our evaluation are shown in table below. Also, listen to the result at https://github.com/Nyakult/S19-aamlp.

Category	RNN	PixelCNN	GAN
Change in Pitch	2	2.22	1.89
Change in Rhythm	2.56	2.89	1.89
Change in Volume	2.89	2.44	1.33
Change in way the static sounds (timbre)	0.89	1.33	0.56
Sounds like human voice	0.89	0.89	0.33
Total avg	1.846	1.954	1.2

Table 2: Outcome by method used.

Every score here is low. Not a single model was able to produce music that was considered good (3 or above). There are some categories that did better than others, as well as some models that did better than the others.

This table shows that rhythm and volume outscore almost every other category for all three models. Changes in pitch preform just slightly worse across the models. The more difficult categories, timbre and sounding like a human, did not preform well at all. This is not surprising, considering that these are encoded in music through combinations of lots of sound waves, whereas the other categories are defined more by the characteristics of a single wave.

6. Discussion and Related Work

The Simple RNN model turned out to be too simple. The model was not able to distinguish the difference between vocal and non-vocal music. By increasing the layers, and iterating over the data in more epochs, the model would have better capability and opportunity to learn. By having a greater understanding of sound waves, the model could be constructed in a way that makes sense for sound waves.

Even though PixelCNN does a great a job in processing MIDI files, it is unable to produce satisfying results by simply feeding mp3 files without further feature engineering.

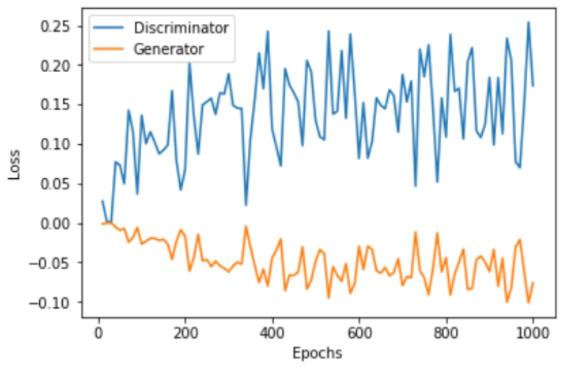


Figure 13: Loss function of RNN GAN

Although RNN GAN can achieve good performance on small sample, its tendency to fluctuate in loss functions prevents it from converging on multiple songs. In fact, there is no guarantee for music-ish results even with 10 secs sample. RNN GAN is so unstable that it can only produce 1 good output in 4 trials. In order to produce better results, more advanced model structures will be implemented and tested in the future study, such as SeqGAN by Yu et al. (2016) and Stacked GAN by Huang et al. (2016)

Our methods did not preform well and the problem of generating lyrical music is much more complex than anticipated. Additional computation power is also needed for this problem. Due to limited time and resources we were not able to run our machines for hours on end. We know that computation power is needed for the current state of the art music generators; it is speculated that WaveNet spends 90 minutes to generate one second of audio. This and more intuitive models constructed for raw sound waves could potentially bring significantly better results.

7. Conclusion

In general, our models learn some of the distribution of the music data like pitch, rhythm and volume but the results of all our models are still noisy and not musical. Comparing 3 different models, PixelCNN proves to be the best. Simple RNN is easy to use but hard to improve. GAN shows certain promise but is extremely unstable and hard to train. New loss function and more work on the feature engineering are necessary to generate better music.

References

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014. URL http://arxiv.org/abs/1406.2661.

Xun Huang, Yixuan Li, Omid Poursaeed, John E. Hopcroft, and Serge J. Belongie. Stacked generative adversarial networks. *CoRR*, abs/1612.04357, 2016. URL http://arxiv.org/abs/1612.04357.

Ale Koretzky. Audio ai: isolating vocals from stereo music using convolutional neural networks, 2019. URL https://towardsdatascience.com/audio-ai-isolating-vocals-from-stereo-music-using-convolutional-neural-networks-210532383785.

Huanru Henry Mao, Taylor Shin, and Garrison W. Cottrell. Deepj: Style-specific music generation. *CoRR*, abs/1801.00887, 2018. URL http://arxiv.org/abs/1801.00887.

Kyle McDonald. Neural nets for generating music, 2017. URL https://medium.com/artists-and-machine-intelligence/neural-nets-for-generating-music-f46dffac21c0.

Olof Mogren. C-RNN-GAN: continuous recurrent neural networks with adversarial training. *CoRR*, abs/1611.09904, 2016. URL http://arxiv.org/abs/1611.09904.

Jordi Pons. Neural networks for music: A journey through its history, 2018. URL https://blog.exxactcorp.com/neural-networks-for-music/.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.06434.

Lucas Theis and Matthias Bethge. Generative image modeling using spatial lstms. *CoRR*, abs/1506.03478, 2015. URL http://arxiv.org/abs/1506.03478.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016a. URL http://arxiv.org/abs/1609.03499.

Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1747–1756, 2016b. URL http://jmlr.org/proceedings/papers/v48/oord16.html.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *CoRR*, abs/1609.05473, 2016. URL http://arxiv.org/abs/1609.05473.