Artificial Composer

Intelligent Information Systems

Paulina Szwed

February 9, 2021

1 Introduction

The objective of the project was to create an intelligent information system that is able to generate music on its own. Although content generation is not one of the main usecases for the information systems, this subject is progressing very quickly, mainly because of the possibilities created by deep neural networks.

One of the techniques to build such systems that often gives very promising results is the use of generative adversarial networks (GAN). GAN is a compound system made of two neural networks - the generator and the discriminator. The discriminator is the component that determines whether the content shown to it is real or generated artificially (fake). The generator tries to deceive the discriminator by creating more and more realistic content. During my experiments I used three variants of GAN architecture in order to create a system generating the most realistic music.

2 Experiments

Experiments were conducted on different types of GAN but every one of them used the same training loop. Two of them were trained on songs from Maestro dataset [2] from 2018, the last experiments used songs by Queen from [1]. All experiments used files in MIDI format, but converted into different representations.

2.1 Training loop

The training loop was divided into two stages. First, the discriminator was trained on a whole epoch of real samples (with expected output of ones) and a whole epoch of fake samples (with expected output of zeros). Then, the generator was trained by exposing the combined system with frozen discriminator weights to an epoch of fake samples, but with expected output of ones.

Before the actual training loop I introduced a pretraing stage, to initially enhance the performance of the generator - the pretraining stage was only training the combined system like in the second stage of the main training loop.

In every case I trained the models on batches of 8 samples each and I performed 5 epochs of pretraining.

2.2 Models

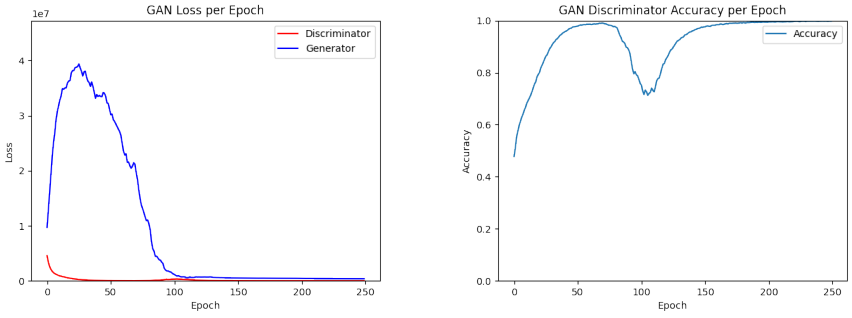
During my reserach I tried out different models for generators and discriminators. I also used different methods for representing information from the training set.

2.2.1 One Hot Encoding DCGAN

At first I used a DCGAN (deep convolutional generative adversarial networks [3]). In this model the discriminator processed genetrated content as images, using convolutional layers and the generator uses deconvolution to create an image from vector of random numbers. Table 1 shows detailed description of the model that I trained.

The representation for the music I used in the first experiment was a matrix created from notes in MIDI file. Each note was a integer value between 0 and 127, so I decided to build an matrix of size 127 on 127 pixels in which every row represented one note and it contained 127 notes long fragment of a song. For representing notes I used one-hot encoding, which means that I converted the value to an index of a element in the matrix row, filled it with value 1.0 and the rest of them with zeros.

The training process is shown in figures 1a and 1b.



(a) Loss during training

(b) Discriminator accuracy during training

Figure 1: Loss and accuracy during training of simple one-hot encoding DCGAN

The results shown that such architecture can be a mechanism to create some kind of music, but it's possibilities were limited to simple sounds - one-hot

Index	Layer type	Description
Generator		
1	Dense	8192 neurons
2	Leaky ReLU	alpha 0.2
3	Reshape	Reshaping into vector 8 x 8 x 128
4	Deconvolution	64 filters 4x4
5	Leaky ReLU	alpha 0.2
6	Deconvolution	64 filters 4x4
7	Leaky ReLU	alpha 0.2
8	Deconvolution	64 filters 4x4
9	Leaky ReLU	alpha 0.2
10	Convolution	1 filter 7x7, sigmoid activation function
Discriminator		
1	Convolution	128 filters 5x5
2	Leaky ReLU	alpha 0.2
3	Dropout	dropout factor 0.2
4	Convolution	64 filters 3x3
5	Leaky ReLU	alpha 0.2
6	Dropout	dropout factor 0.2
7	Convolution	32 filters 3x3
8	Leaky ReLU	alpha 0.2
9	Dropout	dropout factor 0.2
10	Flatten	
11	Dense	1 neuron, sigmoid activation function

Table 1: Simple one-hot encoding DC GAN structure

encoding does not allow to use chords. There is also only one instrument in those generated songs.

2.2.2 LSTM-based GAN

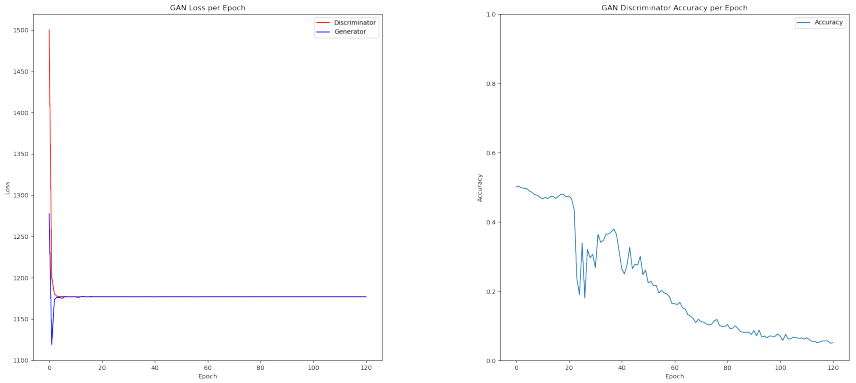
Next I used a generator made of densily connected layers and a discriminator that contained 2 layers of LSTM cells. I was hoping that LSTM's "memory" will improve performance by better perception of sequential patterns in songs. Music was represented as a sequence of numerical values indicating notes of the song. The detailed description of the layers is provided in table 2.

Figures 2a and 2b shows the training process. As we can see, the LSTM-based architecture had troubles with short learning span. It's noticeable that this system quickly degraded and music generated by it becomes a sequence of minimum and maximum values. Those troubles were probably caused by exploding gradients problem.

LSTM-based GAN can be a system for music generation, but it has the same limitations as DCGAN with one-hot encoding. The exploading gradients problem made it impossible to aquire more realistic results, but it still did

Index	Layer type	Description
Generator		
1	Dense	256 neurons
2	Leaky ReLU	alpha 0.2
3	Batch Normalization	momentum 0.8
4	Dense	256 neurons
5	Leaky ReLU	alpha 0.2
6	Batch Normalization	momentum 0.8
7	Dense	512 neurons
8	Leaky ReLU	alpha 0.2
9	Batch Normalization	momentum 0.8
10	Dense	16384 neurons
11	Reshape	reshaping into matrix 128 x 128
Discriminator		
1	LSTM	128 cells
2	Bidirectional LSTM	512 cells
3	Dense	256 neurons
4	Leaky ReLU	alpha 0.2
5	Batch Normalization	momentum 0.8
6	Dense	128 neurons
7	Leaky ReLU	alpha 0.2
8	Batch Normalization	momentum 0.8
9	Dense	64 neurons
10	Leaky ReLU	alpha 0.2
11	Batch Normalization	momentum 0.8
12	Dense	1 neuron, sigmoid activation function

Table 2: LSTM-based GAN structure



(a) Loss during training

(b) Discriminator accuracy during training

Figure 2: Loss and accuracy during training of LSTM-based GAN

generate some interesting samples in which we can hear some repeating patterns.

2.2.3 Pianoroll-based DCGAN

At the next stage of the experiments I went back to the DCGAN architecture but with different approach to representation of music. I used a pianoroll format and converted it into a matrix that had 3 channels (depth equal to 3) so that I could place there an information on chords played by different instruments and even drums. Table 3 shows detailed description of layers in the system.

Training process is shown in figures 3a and 3b. As we can see it was longest training process and it did not lead to exploading or vanishing gradients.

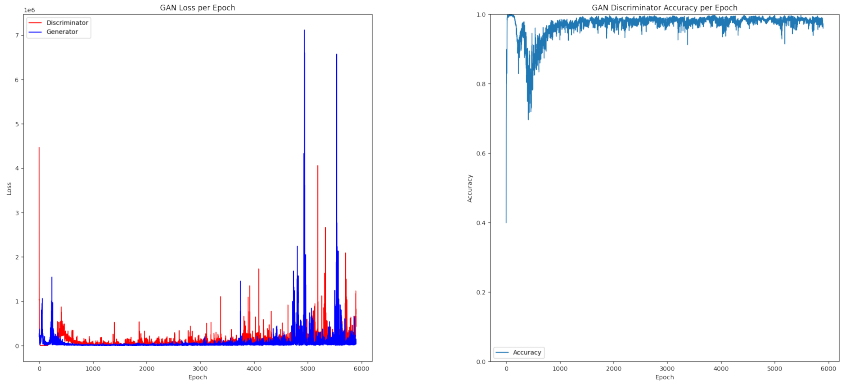
The results from this process were very satysfying. You can see and hear rythm in drums (red channel) and also repeating patterns and chords in instrumental channels (see figure 4)

2.2.4 Others

During my experiments I also tried to implement a system that would process sound in a form of spectrogram created from WAV file by short-time Fourier transform and converting it back to a WAV file, but the images to be processed were very large. The neural networks capable of processing such images could not be handled by my machine.

Index	Layer type	Description
Generator		
1	Dense	8912 neurons
2	Leaky ReLU	alpha 0.2
3	Reshape	reshaping to matrix 8 x 8 x 128
4	Deconvolution	64 filters 4x4
5	Batch Normalization	momentum 0.8
6	Leaky ReLU	alpha 0.2
7	Deconvolution	64 filters 4x4
8	Batch Normalization	momentum 0.8
9	Leaky ReLU	alpha 0.2
10	Deconvolution	64 filters 4x4
11	Batch Normalization	momentum 0.8
12	Leaky ReLU	alpha 0.2
13	Convolution	3 filters 7x7
Discriminator		
1	Convolution	128 filters 5x5
2	Leaky ReLU	alpha 0.2
3	Convolution	128 filters 5x5
4	Leaky ReLU	alpha 0.2
5	Convolution	64 filters 3x3
6	Leaky ReLU	alpha 0.2
7	Convolution	32 filters 3x3
8	Leaky ReLU	alpha 0.2
9	Flatten	
10	Dense	1 neuron, sigmoid activation function

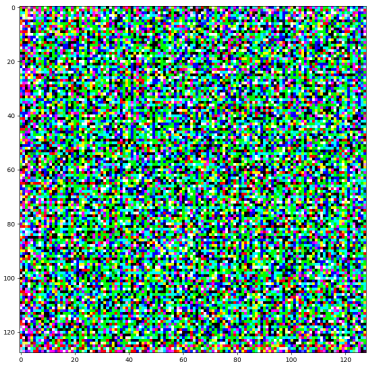
Table 3: Pianoroll DCGAN structure



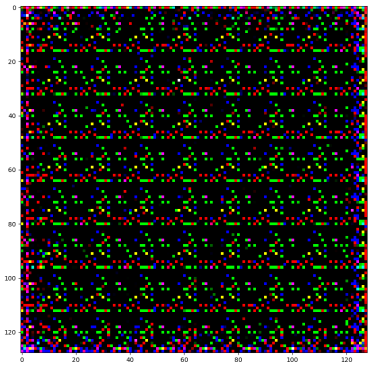
(a) Loss during training

(b) Discriminator accuracy during training

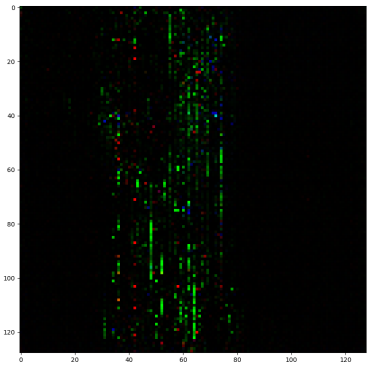
Figure 3: Loss and accuracy during training of LSTM-based GAN



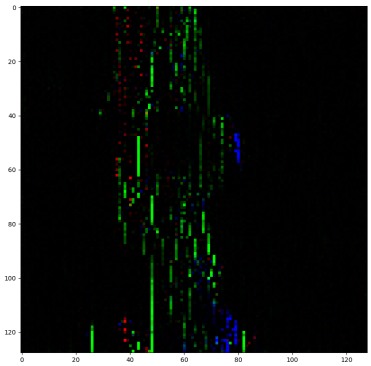
(a) Result after 0 epochs



(b) Result after 100 epochs



(c) Result after 3600 epochs



(d) Result after 28800 epochs

Figure 4: Loss and accuracy during training of LSTM-based GAN

3 Implemented software

In order to prepare training and generation scripts I used the following tools:

- Python 3.8
- Tensorflow and CUDA as training tool
- Keras as a library for building neural networks
- mido and pypianoroll libraries for processing MIDI files
- matplotlib library for visualization
- argparse library for handling command line interface.

Instructions for both training and generation scripts can be found in readme.md file in the repository.

4 Summary

As a result of the project I managed to deliver a intelligent information system that is capable of generating music. I did not expect spectacular results from a rather small system trained on my PC, since it requires a lot of computations, but what I managed to deliver is a satysfying outcome despite its limitations.

References

[1] Queen - royal legend - audio archive: Queen midi, 2021.

[2] 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. In *International Conference on Learning Representations*, 2019.

[3] Nathan Inkawhich. Dcgan tutorial, 2021.