How can state of the art generating algorithms be used to create an interactive Artificial Intelligence that generates electronic music

Dennis Netzer

University of Applied Science Darmstadt - Faculty Mathematics and Natural Science

Summary

This thesis deals with the use of state-of-the-art deep learning algorithms in the creative field of generating new content. The aim is to develop a deep learning model that can generate a random song in the electronic music genre without vocals. This is motivated by the objective to expand the existing customer base of the partner company of this thesis by presenting AI expertise on a fair or conference. The resulting model can serve as an exhibit.

There are three mayor ways to present audio data to a computer that are used for generative tasks. The physical analog waveform can be digitized by measuring the change in air pressure over time for a certain location, and then by taking a certain number of samples from these values each second. This resulting vector of values is the raw audio data. Every waveform is a superstition of several waveforms with a consistent frequency. These base frequencies can be extracted by the Fourier Transform and presented over time in a spectrogram. The last format called Midi incorporates musical knowledge into events. An event indicates which musical note from which instrument should be played in what way. Different authors tackled the task of generating music in various combinations of audio formats and deep learning architectures. The most intriguing is the WaveGAN [1] that uses a Generative Adversarial Network (GAN) to model raw audio data. It can only generate audio of one second. In 2018 an architecture called progressive growing of GANs [2] showed that GANs can be used to generate over one million pixels in a coherent image when trained iteratively. This magnitude would result in audio of over a minute. This thesis implements this algorithm for raw audio data.

Requirements of the final AI

After analyzing architectures of other authors and discussing their approaches and results, some requirements for the final AI product were defined.

Requirement 1: The model has to produce a song of at least ten seconds.

Requirement 2: The model has to model long-term dependencies. The underlying beat must not alternate too much, resulting in a coherent song.

Requirement 3: The user should be able to influence the produced song in some way.

Requirement 4: The song has to be free of noise, and it should be pleasant to listen to it.

Growing GAN Algorihm

Originally, the growing GAN Algorithm was used by Karras and his colleagues to generate images of human faces of 1024×1024 pixels. Instead of training a GAN directly on this high resolution, they scaled the images down to a 4×4 representation and started to train a GAN. After training the GAN network for a long enough time, they increased the size of the network such that the resolution doubles. That means adding a new output block to the generator and a new input block to the discriminator, while keeping all parameters of the rest of the network. They repeated this procedure eight times until they reached their desired image size. The generator and the discriminator both consist of convolutional neural networks.

The idea of this iterative approach is that the network can learn the mapping from an initial random vector to a complex image structure in steps rather than directly. The downscaled images only contain some global structure of the image that is learned first by the network. With each iteration, the images contain more and more details that the network can focus on over time. To not shock the network with the increased image resolution, for each iteration the new size is introduced slowly to the network by smoothly fading in the new network blocks. This strategy is depicted in Figure 1.

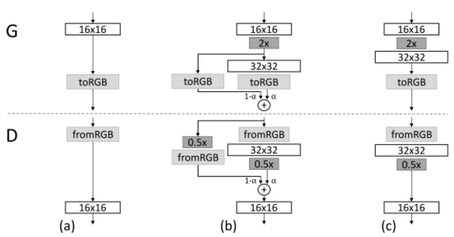

Figure 1: Strategy of fading in new convolutional blocks in the growing GAN architecture

The fields “toRGB” and “fromRGB” stand for the operation of transforming data to an RGB image or the inverse. In (a), the generator outputs an 16×16 image that is sent to the discriminator. (b) represents the fade in phase where a new higher resolution is introduced. The 16×16 image is upsampled by a factor of two and sent through two different routes. In one path, the upsampled version is directly turned into an RGB image and in the other path it is first passed through another convolutional block represented by the 32×32 field. The circle with the plus represents a weighted sum layer where the two inputs are scaled with $\boldsymbol{\alpha}$ and 1-$\boldsymbol{\alpha}$ respectively and then added up. $\boldsymbol{\alpha}$ is slowly increased from zero to one, increasing the weight of the new convolutional block until the state (c) is reached. The reason for the smooth fade in is that the network should not unlearn the already learned mapping by changing the parameters too much when presented with an entirely new representation of the data. By smoothing in the higher resolution, the network has time to slowly adapt.

How does Growing GAN help meeting the requirements

Requirement 1: The length of the song depends on two factors. The first factor is the number of samples generated by the generator network in its output layer. The second factor is the sample rate that is used. The higher the sample rate, the more samples are needed to produce one second of audio, increasing the quality of the audio. Requirement 1 can therefore be met by having a generator network with the required output size defined by the sample rate. The output size and the sample rate have to be selected deliberately.

Requirement 2: Learning the mapping from a random vector to a complex data structure directly is a complex task. The growing GAN algorithm decomposes this complex task into smaller, more feasible ones. It starts with learning a lower representation of the data. For audio data, that corresponds to a lower sampling rate. If, for example, a sampling rate of 100 Hz is used, the network has to predict only 100 samples for one second of audio. This way, it is easier to learn some global structure of the songs. According to the Shannon Nyquist sampling theorem, all frequencies above half the sampling rate are lost. This lower representation therefore contains only the low frequencies of the signal that are first learned by the model. For every growing iteration of the network, the sampling rate is increased simultaneously with the network size, such that the length of the song stays the same for every iteration. The model is thus introduced to higher frequencies every iteration. The model can concentrate on lower frequencies and global structure at the beginning, and then on the high-fidelity features in later iterations.

Requirement 3: To manipulate a song, a lower representation of the song is needed, or the model has to be conditioned on some predefined features. The GAN architecture takes a latent vector as input that it maps to a song. Even if this latent vector is just sampled from a probability distribution, the network learns to map these number to the complex data structure. When giving the network the same input vector twice, the output is both times the same. The input vector therefore determines the song and can be manipulated. When the network is trained well, latent vectors that are close in the hyperspace should result in similar songs.

Requirement 4: The main aspect of good sound quality is the sampling rate used to process the audio signal. If it is too low, higher frequencies cannot be captured. The growing GAN architecture helps to further improve sound quality for longer sequences, with the same principle explained for requirement 2. The network does not have to learn too many relations at once, which improves the quality of the outcome. Additionally, the smooth fade in facilitates to stabilize the growing iteration. If the outcome is still noisy, an anti-aliasing filter can further be used to avoid artifacts when using a low sampling rate and therefore to improve the sound quality.

Implementation of Growing GAN for raw audio data

Generator	Act.	Norm.	Output shape	Params
Latent vector	-	-	256 × 1	-
Dense	-	-	4096 × 1	1,052,672
Reshape	-	-	1024 × 4	-
Conv1D (25)	LeakyReLu	PixNorm	1024 × 256	25,856
Conv1D (25)	LeakyReLu	PixNorm	1024 × 256	1,638,656
Upsampling	-	-	2048 × 256	-
Conv1D (25)	LeakyReLu	PixNorm	2048 × 256	1,638,656
Conv1D (25)	LeakyReLu	PixNorm	2048 × 256	1,638,656
Upsampling	-	-	4096 × 256	-
Conv1D (25)	LeakyReLu	PixNorm	4096 × 128	819,328
Conv1D (25)	LeakyReLu	PixNorm	4096 × 128	409,728
Upsampling	-	-	8192 × 128	-
Conv1D(25)	LeakyReLu	PixNorm	8192 × 64	204,864
Conv1D (25)	LeakyReLu	PixNorm	8192 × 64	102,464
Upsampling	-	-	16384 × 64	-
Conv1D (25)	LeakyReLu	PixNorm	16384 × 64	102,464
Conv1D (25)	LeakyReLu	PixNorm	16384 × 64	102,464
Upsampling	-	-	32768 × 64	-
Conv1D (25)	LeakyReLu	PixNorm	32768 × 32	51,232
Conv1D (25)	LeakyReLu	PixNorm	32768 × 32	25,632
Upsampling	-	-	65536 × 32	-
Conv1D (25)	LeakyReLu	PixNorm	65536 × 32	25,632
Conv1D (25)	LeakyReLu	PixNorm	65536 × 32	25,632
Upsampling	-	-	131072 × 32	-
Conv1D (25)	LeakyReLu	PixNorm	131072 × 16	12,816
Conv1D (25)	LeakyReLu	PixNorm	131072 × 16	6,416
Upsampling	-	-	262144 × 16	-
Conv1D (25)	LeakyReLu	PixNorm	262144 × 16	12,816
Conv1D (25)	LeakyReLu	PixNorm	262144 × 16	6,416
Conv1D (1)	tanh	-	262144 × 1	17
Total trainable parameters				**7,896,017**

Figure 2: Structure of the final generator

Figure 2 shows the final generator for the GAN network. Each box represents one block of the generator. The input block consists of the input vector that is rescaled to the format of the lowest resolution. The latent vector is a random vector sampled from a uniform distribution between negative one and one. Each following convolutional block consists of two convolutional 1-D layers with a kernel size of 25, a LeakyRelu activation function and a PixNorm normalization layer. PixNorm is a self-defined layer invented by Karras that normalizes a feature vector to unit length. The output layer is a convolutional 1-D layer with one kernel and a tanh activation function that outputs values between negative one and one, which is the interval of audio data. The first convolutional block generates 1024 samples. For the chosen sampling rate of 86.1328125 this results in a song of around eleven seconds. The training happens the same way as in the original growing GAN algorithm. The network is trained for this lower representation of the data. Subsequently, new convolutional blocks are added, introducing higher frequencies to the network. The process is also implemented as a smooth fade in. Therefore, the output signal of one convolutional block is split into two paths. One path upsamples the signal and sends it directly to the output layer. The other path upsamples the signal and sends it through the new convolutional block before also processing it with the output layer. Both outputs are then added as a weighted sum, where the weight is slowly increased such that the path with the new convolutional block gains importance over time until the signal also takes this path. This process is repeated eight times until the network outputs 262,144 which results in a song of about eleven seconds sampled with 22050 Hz.

Data Collection and Preperation

To train the GAN network, royalty free songs were downloaded from https://freemusicarchive.org/ in the MP3 format. With the help of the python library Librosa the waveform data of the songs were extracted. This was done for all sampling rates used in the iterative training process. The first five seconds of every song were skipped to avoid showing the network the build up phase of the songs. In that way, 252 songs were downloaded and converted into 7288 objects of waveform data with enough samples for eleven seconds of music depending on the used sampling rate.

Results

The training of the network could not be finished because the needed capacity of the vRAM exceeded the available vRAM of the GPU. The training was conducted in the Google Cloud, and a larger GPU was not available for the intended period. Figure 3 shows the loss history of level seven for the generator and the discriminator in epoch four. The generator loss in orange has a high volatility. It is defined by the critic score of the discriminator for the generated samples of the generator. Fluctuating values indicate that the discriminator has not found a good rule to differentiate between real and fake samples.

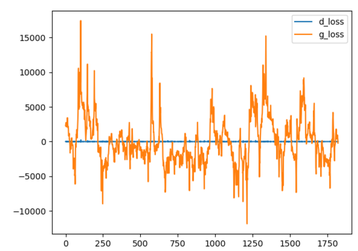

Figure 3: Loss values of generator and discriminator for iteration seven in epoch four

The loss values of the model only give a notion of how well the model is doing. To get certainty the songs have to be listened to. The model successfully outputs a song of eleven seconds and fulfills requirement 1. To generate a random song, the model needs less than one second. The user has also the possibility to alter the soundwave of the song. The latent space has to be further explored to recognize possible dependencies between the values of the latent vector and the resulting song. Hence, requirement 3 is fulfilled. The songs still contain a lot of noise, and the beat changes over the course of the song. Nevertheless, some structure can be identified, and the outcome is a step into the right direction. The model has to be further improved, and the last level has to be trained to also fulfill requirements 2 and 4.

Summary and Outlook

This thesis laid a good foundation for the developing of music generating systems. To use this model in the future, the audio quality has to be increased. This can be done by collecting more training data and by training the model for the final level and for a loner time with a better GPU with more vRAM. The program has an interactive interface for the generation process.

Quellen

[1] C. Donahue, J. McAuley, and M. Puckette, “Adversarial Audio Synthesis,” pp. 1–16, 2019.

[2] T. Karras, T. Aila, S. Laine, and J. Lehtinen, “Progressive Growing Of GANs For Improved Quality, Stability, And Variation,” pp. 1–26, 2018.