Simple Adversarial Music Generation

Joakim Berntsson, Adam Tonderski

***Abstract*—Artificial expression of creativity has been a relevant topic for many decades, and in the recent years data and computational power have allowed us to actually run methods to accomplish this. Generating music or sounds has been done in many different ways, using a wide range of techniques and architectures in deep machine learning. This paper proposes a simple approach using a generative adversarial network with convolutional layers, inspired by MidiNet and MuseGAN. This simplified model produces samples that are visually similar to the training data. However, when converted to audio the sound is often chaotic and fragmented. Because of this, the samples are on average subjectively worse than MidiNet's or MuseGAN's. This is not surprising considering the purpose of this project is to simplify the process and model. The source code is openly available in a public repository[1] and there are also some uploaded samples[2].**

***Index Terms*—GAN, CNN, music, generation, midi, generative, convolutional**

I. Introduction

Artificial music generation is a topic that has been relevant for over 50 years [3], with a wide range of approaches from Markov Chains to neural networks. The recent advances in deep learning have also spread to this area, with a noteworthy example being SampleRNN [4] which uses a combination of recurrent neural networks (RNNs) and autoregressive multilayer perceptrons to generate audio.

A related field is artificial image generation where generative adversarial networks (GANs) have shown very impressive results [5]. The general idea is to train multiple separate networks by making them compete with each other. For example, one could generate a fake image while the other guesses whether it is real or not. GANs have also been used in music generation, for example by MidiNet [6] which uses CNN models and C-RNN-GAN which uses RNNs [7].

In this paper we attempt to generate music through the use of GANs and CNNs. This focus was chosen due to it being relatively unexplored, and GANs with CNNs are generally trained much faster than GANs with RNNs. We will primarily develop our own models and architectures with some inspiration from existing projects. The purpose is to develop a model of much lower complexity than for example MidiNet or MuseGAN, while still generating music from scratch. Finally, for comparison, we train the state of the art Progressive Growing of GANs (PGGAN) architecture [5] on the same data.

II. Related Work

There are a lot of different models and architectural patterns for music generation: recurrent neural network, feed forward network, convolutional neural network, restricted Boltzmann machine, generative adversarial network, and reinforcement learning [8]. There are also compound architectures where combinations of the mentioned techniques are used together. The majority of models utilize some technique for temporal information, for example using RNNs or conditioning.

One of the first solutions for generating sounds using CNN was the Google project WaveNet [9]. They used raw audio waveforms as training and the networks could be used for: multi-speaker speech generation, text-to-speech, music generation, and speech recognition. However, as opposed to the other mentioned works, WaveNet did not use GAN, but rather an autoregressive model.

Following the work behind WaveNet came an approach using GAN and CNN, MidiNet [6]. This project used CNNs for the discriminator, generator, and conditioner. The conditioner is meant to condition the input with prior knowledge to help the generator. The conditioner helps maintain coherence with previous samples. The degree to which the generation is based on previous knowledge is also controllable via a parameter, which can be seen as a weighing between coherence and creativity.

The authors of MidiNet also made a follow-up model called MuseGAN [10]. This was meant to improve the current temporal models, and also add multi-track support for multiple instruments. This was achieved by introducing three distinct collaborative models: jamming, composer, and hybrid. Instruments generated output through the jamming, and the composer and hybrid helped stitch together the instruments to a coherent piece in multiple tracks.

III. Method

In this section we introduce a model for generating symbolic music, with influence from MidiNet and widely used GAN architectures. The final model is also compared to Nvidia's PGGAN model.

A. Dataset

The various datasets consisted of midi files with a low number of piano tracks. During development of the model multiple genres were tested: jazz, classical, pop, and rock. The final model and training was performed on a collection of The Legend of Zelda songs that were downloaded from an online collection[11].

B. Data representation

The first step in any machine learning problem is to collect and process the data. In this case we used a collection of piano songs in a midi format. A midi file can be considered

Joakim Berntsson, at Chalmers University of Technology, Gothenburg Sweden (e-mail: joabern@student.chalmers.se).

Adam Tonderski, at Chalmers University of Technology, Gothenburg Sweden (e-mail: tadam@student.chalmers.se).

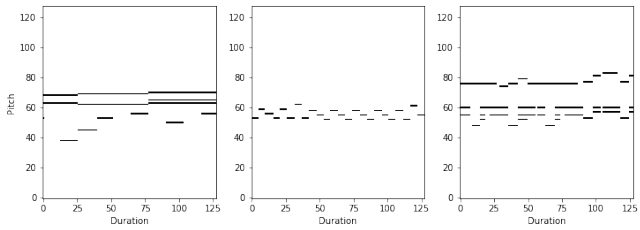

Fig. 1. Piano roll samples from the Zelda dataset.

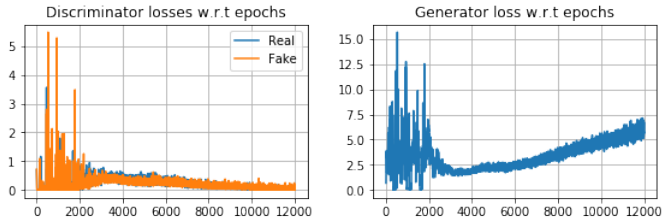

Fig. 2. Losses for the generator and discriminator during training.

as a collection of music sheets which are played in different "programs", for example piano or guitar. These files can be converted into matrices, called piano rolls, by sampling at a fixed interval. Piano rolls have a dimension of LxP, where L is the length of the roll, and P is the maximum pitch level, often set to $P = 128$. An example of a piano roll can be seen in figure 1.

Finally, the rolls are binarized to ignore volume and cut into a desired length, where we chose $L = 128$. Once this is done for all the songs we have a large set of matrices that can be used for training. Converting an image to a midi file is as easy as running the process above in reverse.

C. Model definition

In order to keep things as simple as possible the model was mostly based on a "vanilla" GAN architecture [12]. The generator uses $n = 100$ Gaussian noise values as inputs and outputs an image, whereas the discriminator takes an image as input and performs a binary classification on whether it is fake or not. The training was performed in two stages: first the discriminator is trained by feeding it a mix of real and fake images; then the generator is trained by propagating gradients through the discriminator with frozen weights. In both cases binary cross entropy was used as the loss function.

With the approach above, we could try different models for the discriminator and generator as long as the image dimensions were correct. Our initial approach was to handle this as a traditional image generation problem and so we used convolutional layers for both the generator and discriminator. The generator needs to increase the size of the image, either by up-sampling or strided deconvolutions. Symmetrically, the discriminator needs to decrease the size of the image into a binary choice, either by pooling or strided convolutions.

We also tried some more unconventional approaches. For example, inspired by MidiNet, we used very elongated kernels in the pitch dimension. This seemed to work well for the discriminator but not so much for the generator.

The final proposed model uses a generator with an initial fully connected layer of $64 \cdot 16 \cdot 16$ nodes, followed by four convolutional layers using kernel size 3, batch normalization with momentum 0.8, and leaky ReLU as activation function. The number of filters decreased in the order of 64, 48, 32, and finally 1 as the number of channels or instruments used.

The first discriminator layer uses 14 filters that summarize the whole pitch dimension by using kernel size 2×128 and a stride of 2. This is followed by a 77 filter convolutional layer with kernel size 4×1 and a stride 2. Finally, there is a fully connected layer with 512 nodes. The activation function is ReLU with the exception of the output which uses sigmoid for binary classification.

D. Experiments

For comparison with the proposed model, some experiments was performed using PGGAN which is famous for generating faces of extremely high quality. The source code is hosted in a public repository so it was straightforward to train with custom piano roll images instead.

There were also some attempts at getting the MidiNet source code to run, but their code base is quite old with some very outdated dependencies, so getting it to run was outside the time requirement for this project. The successor MuseGAN was also troublesome to get up and running with our data since they required complicated preprocessing.

E. Evaluation

To evaluate the generated results, we used two metrics from MuseGAN: number of used pitches UPC, and ratio of "qualified" notes QN. UPC is a number between 0 and 128, which corresponds to how many of the available pitch levels are used by the samples. QN is the ratio of long notes (≥ 2) in the samples, basically measuring the sample fragmentation.

These metrics may not be very intuitive, however they measure objective quantities which affect the sound of the generated samples. Some of these characteristics could be problematic, such as too wide pitch range or too many short notes. It is important to note that these values can not be evaluated independently, rather they must be compared to the corresponding values for the training data. The reason for this is that the metrics can vary immensely among music styles.

IV. Results

Figure 2 displays the losses for the generator and discriminator with respect to training epochs. In the beginning they are both fluctuating a lot, and then they become more stable. In later iterations we can see that the spikes in generator and discriminator losses are roughly synchronized.

The samples produced by the network are presented in Figure 3. These are extracted during training of 12000 epochs with row-first ordering. The final sample in the bottom right corner is the generated sample of the final epoch. As can be seen in the first epoch the sample is just noise since it has not been able to learn anything yet. Immediately in the next

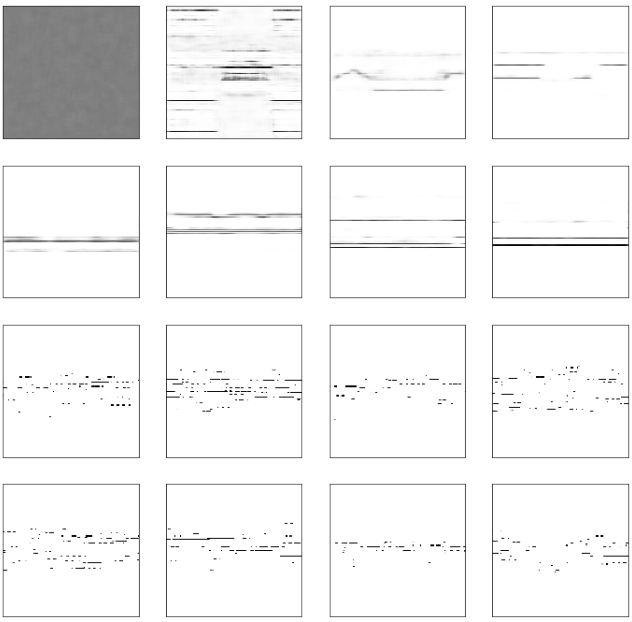

Fig. 3. Snapshots of generated samples during training. The samples are chronological from left to right and top to bottom.

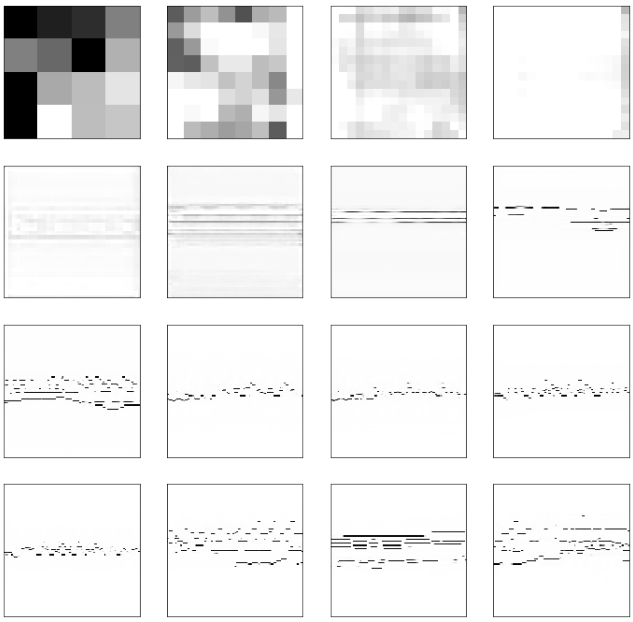

Fig. 5. Snapshots of generated samples during training with PGGAN. The samples are chronological from left to right and top to bottom.

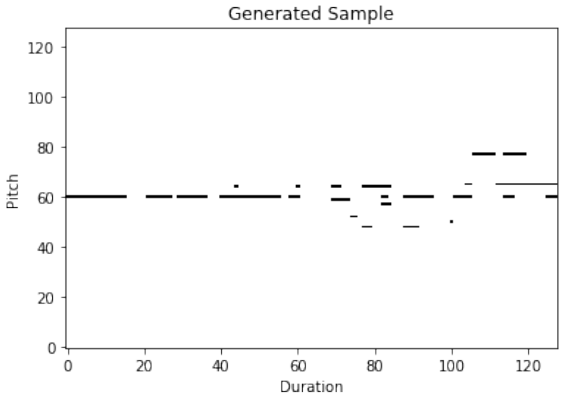

Fig. 4. Generated sample at the end of training for 12000 epochs.

few iterations we can see some structure emerging, mostly in the middle area, and with some blurring. In the final epochs more complex shapes are created, with generally shorter note lengths. A single generated sample can be seen in Figure 4.

Finally for the metrics we have that the $QN = 0.91$ and $UPC = 17.24$ for the training data, and $QN = 0.71$ and $UPC = 20.70$ for the generated samples. Note that these numbers are averaged over the same number of training and generated samples.

Analogously to Figure 3, the training progress for PGGAN is shown in Figure 5. One apparent difference is that the initial samples use a very low resolution which is then increased as training progresses until it reaches its full resolution. This is the progressive growing that gave name to the model. The last sample was generated after training for 12 hours on a Nvidia 1080Ti GPU. This can be compared to the 1 hour of training our GAN model on a K80 GPU.

Both models were prone to overfitting when using a low number of training samples. This led to exact copying of training sequences, and in the case of PGGAN copying an entire song.

V. Conclusions

Our model was successful in creating piano rolls with clear structure and some visual similarities to real tracks. However, when playing the actual sounds, most samples subjectively sound bad with a lot of random short notes. This fragmentation is indicated by the significantly lower QN value compared to the training data. The pitch range sounded reasonable which is confirmed by the close UPC values.

The PGGAN model had visually similar results, however the sound seemed slightly superior to our model. This could be the case of more parameters in this complex model and a more developed GAN architecture.

As expected, it is difficult to capture transitions and relationships between notes using CNNs. This could be the reason why MidiNet, MuseGAN, and WaveNet use additional methods, such as conditioning and dilated convolutions.

A. Future Work

GANs are typically difficult to train and prone to spiral out of control. There are many proposed solutions to this problem, such as Wasserstein GAN with gradient penalty [13] which introduces a more sophisticated loss function. Another approach to improve training stability is to keep a balance between the capabilities of the generator and discriminator, for example by crippling the final layers of the discriminator. We tried these methods with varying degrees of success but this is definitely an area with a lot of room for improvement.

Another rather time consuming task would be to more carefully choose and preprocess the results. During training of

the network it became obvious that preprocessing the data was not as easy as initially thought. This step also ties into the fact that we only generated single instrument samples. In theory it should be similar to going from grayscale to colour, simply add additional channels to the image. However, in practice there was no time to construct such a multichannel dataset.

Adding conditioning could be beneficial to the network. First of all, it would allow us to create multiple samples which fit together. Secondly, it could improve the coherence within the sample, both by adding additional prior knowledge, and putting constraints on the allowed outputs.

References

[1] J. Berntsson and A. Tonderski, *Simple adversarial music generation*, https://github.com/jcberntsson/music-generation-project, 2018.

[2] ——, *Adversarial music generation*, https://soundcloud.com/gan-music/sets, 2018.

[3] L. Hiller and L. M. Isaacson, "Experimental Music: Composition With an Electronic Computer," *New York, McGraw-Hill*, 1959.

[4] S. Mehri, K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. Courville, and Y. Bengio, "SampleRNN: An Unconditional End-to-End Neural Audio Generation Model," *ArXiv e-prints*, Dec. 2016. arXiv: 1612.07837 [cs.SD].

[5] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive Growing of GANs for Improved Quality, Stability, and Variation," *ArXiv e-prints*, Oct. 2017. arXiv: 1710.10196.

[6] L. Yang, S. Chou, and Y. Yang, "Midinet: A convolutional generative adversarial network for symbolic-domain music generation using 1d and 2d conditions," *CoRR*, vol. abs/1703.10847, 2017. arXiv: 1703.10847. [Online]. Available: http://arxiv.org/abs/1703.10847.

[7] O. Mogren, "C-RNN-GAN: Continuous recurrent neural networks with adversarial training," *ArXiv e-prints*, Nov. 2016. arXiv: 1611.09904 [cs.AI].

[8] J. Briot, G. Hadjeres, and F. Pachet, "Deep learning techniques for music generation - A survey," *CoRR*, vol. abs/1709.01620, 2017. arXiv: 1709.01620. [Online]. Available: http://arxiv.org/abs/1709.01620.

[9] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," *ArXiv e-prints*, Sep. 2016. arXiv: 1609.03499 [cs.SD].

[10] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment," *ArXiv e-prints*, Sep. 2017. arXiv: 1709.06298.

[11] vgmusic.com, *Zelda dataset*, https://www.vgmusic.com/music/other/miscellaneous/piano/, 2018.

[12] E. L. Noren, *Keras implementations of generative adversarial networks*, https://github.com/eriklindernoren/Keras-GAN, 2018.

[13] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," *ArXiv e-prints*, Mar. 2017. arXiv: 1704.00028.