Generating classical piano pieces using Generative Adversarial Network

Jonny Fredriksson,* Jamie Santos,† Jules Sintes,‡ and Niranjan Suresh§
Chalmers University of Technology
Advanced Machine Learning with Neural Networks
TIF360 / FYM360
(Dated: May 16, 2023)

The creation and enjoyment of music is a hallmark of the human experience and one shared across all cultures and societies. Machine learning provides exciting new methods of producing music and the potential to explore innovative combinations of notes, rhythm, pitch, and so on. Generative adversarial networks (GANs) have historically been used for generating images, and more recently, music [19]. The aim of this project was to generate novel segments of music based on classical music samples using a GAN machine learning architecture. More specifically, the question addressed was: how "musical" does output from a GAN model sound when trained with classical piano music in the form of MIDI files? To address this question, a GAN using an LSTM-based generator and a dense network-based discriminator was used. The results obtained were promising; while the output from the trained GAN originally sounded discordant, longer training produced more melodic output. Furthermore, the output produced is original, and could be adjusted in music-editing software as inspiration for new melodies.

I. INTRODUCTION

The realization that artificial intelligence one day, if not already, will outperform humans both physically and intellectually, is as old as the concept itself. There are at least two areas, however, where AI is still oceans behind humans; imagination and creativity, arguably the features that makes us human. A fundamental representation of these features is the art of music. Can modern AI generate music indistinguishable from music generated by a human?

The use of deep learning to generate music has been researched extensively, and is still a highly active field. One popular approach is the *next note prediction* used by MuseNet [4], where a large scale transformer model is used to iteratively predict the next note, until a melody is generated. The dataset consists of MIDI music which has been tokenized to represent all elements of the music, e.g. pitch, duration, velocity, etc. Another common approach is the *raw music representation* used by WaveNet [2], where the music is represented as audio waveforms, i.e. as images. The model architecture is a convolution neural network, as is common when treating images.

When it comes to image generation, however, the state of the art architecture is that of the generative adversarial network, or GAN for short, first introduced in the paper *Generative adversarial nets* by Ian J. Goodfellow et. al. [11]. One such example is Style , used for the application *This Person Does Not Exist* [9]. Trained on a dataset consisting of real person portraits, the model outputs a portrait of someone indistinguishable from a real person, even though that person does not exist. Using this same idea, one should be able to train a GAN to generate music, indistinguishable from human made music. One recent example of music generation using GANs is MuseGAN [6] [7].

There are several architectures commonly used for GANs, with the most frequently used being the convolutional/deconvolutional framework, as it works well with image data. In this project a sequential representation of the music, rather than an image representation, was used [8]. Therefor a long short-term memory (LSTM)-based GAN were implemented, as LSTMs are well suited to work with sequential data. The format of the music is MIDI, encoded and decoded using a Python MIDI tokenizer called MidiTok.

II. METHODS

This deep learning project is implemented with Python using the TensorFlow and Keras libraries for the neural networks. The MIDI files are read and tokenized using the Midi Tok library [16]. Since GAN training are known to require heavy computational power, we choose to use the Kaggle environment which gives access to GPU accelerator.

A. Midi dataset and tokenization

Classical piano dataset
In order to build a consistent dataset, we choose to use midi transcription from classical piano pieces. Since these pieces are copyright free it was quite easy to find all the midi files from a single website [13]. We choose

* jonnyfr@student.chalmers.se
† jamies@student.chalmers.se
‡ sintes@student.chalmers.se
§ sureshni@student.chalmers.se

to include piano pieces from Mozart, Beethoven and Chopin. All these classical composers have different styles but nonetheless composed classical music for piano solo.

In occidental music theory, music transcriptions are written with time segments called bars corresponding to a specific number of beats. This division gives reference points in the music. Especially, one bar can be assimilated to one musical idea. Usually, melodies or phrases in a musical piece or organized as groups of 2^n bars and often as 8-bars sections. Thus, to keep the deep learning problem simple enough to expect good results, we choose to split the pieces in 8-bars sequences. Using 132 pieces classical piano pieces from various composers and by splitting them into 8-bars sequences.

Tokenization method
Midi files contain sequential structured information readable by specific hardware/software for music purpose. However, in order to process midi with neural networks, we need to convert these files into sequences of tokens, e.g. integers ready to be fed to neural networks and especially Transformers or RNN.
The MidiTok library was designed for this purpose and features several different tokenization strategies from very simple to very advanced. Considering our data and benchmarking the tokenization strategy, we considered different tokenization methods, especially REMI method already successfully used with RNN networks [24]. However, we finally chose to work with Octuple tokenization [15] which as the advantage to be beat based and encodes each note separately. Moreover, it has already been implemented in automatic composition project.

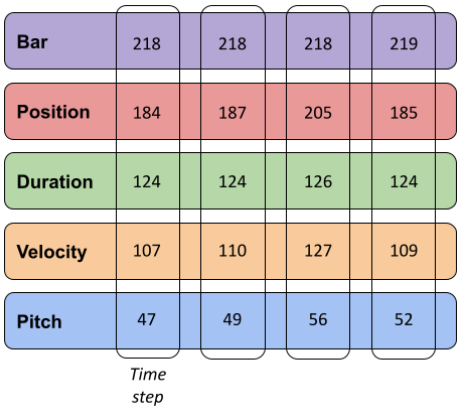

FIG. 1: Sample of a tokenized midi sequence using octuple strategy

Using this tokenization method on our dataset, it leads to 1120 sequences with various length from 15 to 227 tokens were each token is a 5-integers vector as in fig.1. In order to standardize the sequence length of our data, we choose to extend the shorter sequences by replicating every sequences up to the maximum length.

B. Generative Adversarial Network

The GAN consists of two neural networks; the generator and the discriminator. The task of the discriminator is to evaluate the generator's output and distinguish it from the original data. The generator's task, given a random noise input, is to trick the discriminator by generating synthetic data which is indistinguishable from the original data. The architecture of the generator/discriminator is based on the problem at hand, i.e. the format of the data.

C. LSTM and Dense-based GAN

The overarching architecture of the machine learning model, as briefly described above, is a generative adversarial network. Since, music is inherently time dependant, it was quite natural to consider using RNN based network, at least for the generator[20]. GANs architecture implemented specifically for time series generation have already been implemented successfully [23] and already use LSTM based networks. We considered LSTM on the one hand and transformers layers on the other hand. However, we finally decided to use only LSTM layers which was easier to implement and lead to better results.

Generator The generator is defined by three LSTM layers. The output is then a simple dense layer with a linear activation function. Since, GANs are known to be very unstable and hard to tune, we wanted to keep the architecture simple enough to be tuned easily without having too heavy computation[1].
Discriminator The discriminator is based on 3 dense layers followed by dropout layers to avoid overfitting of the discriminator and also reduced the speed of learning of the network while training the GAN.

The global architecture of the network is described in 2. A lot of experiments with various parameters and various architecture have been made to achieve acceptable results and the network could still be tuned or changed to achieve better results.

The choice of having 2 different architectures with the generator and the discriminator is mainly based on the results we were able to generate by trying several combinations (LSTM/LSTM, LSTM/Dense, Dense/Dense, etc...). Usually, generator and discriminator are similar but it is not mandatory and asymmetric architecture is also possible.

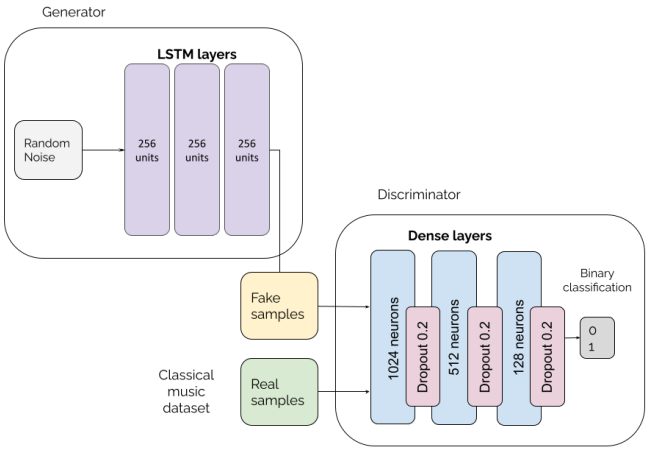

FIG. 2: Architecture of the GAN

D. GAN's training

The overall GAN model co-trains the generator and discriminator, using the Adam optimizer with a learning rate of 0.0002. One of the main problem we had while training the GAN was the discriminator learning way too fast compare to the generator, thus, the discriminator loss was going to zero while the generator loss converges towards a constant value with no further improvement after a small number of epochs.

Several solutions were considered to solve this problem, especially by changing the training loop of the GAN. Initially, both networks are trained at each epoch with the same learning rate, but the number of training loop of each network can be changed, thus we tried to train the generator more than the discriminator. An other solution was to reduce the learning rate of the discriminator and eventually train the discriminator for several epochs at each training epoch of the generator. This appear to be a good solution. Moreover, it is possible to set a threshold on the discriminator loss in order to adapt the training of the network and avoid the discriminator becoming to good compare to the generator.

The GAN was trained over 2000 epochs and results were checked regurlarly in order to see the evolution of the training and assess if the training was failing. This method was necessary since we implemented our own architecture with limited ressources (Kaggle is limited to 30 hours of GPU-use per week) and such implementation of GANs for symbolic music is quite new.

III. RESULTS

Several combinations of MIDI encoder and decoders, network architectures and configurations were tested in order to produce interpretable results for the MIDI encoder. One of the challenges using the original encoding tested, REMI, was that every bar was distinguished by a "1" in the sequence, and it was difficult to determine if the network would be able to learn the correct placement of these one's. Another challenge was determining the proper shape and format of the input data; for instance, the length of the input data produced using REMI was varied; an 8 bar sequence with no sound consisted of 7 one's, whereas the most complex 8 bar sequence had a length of 788. For this reason, the octuple tokenization was used (see "Midi tokenization"). Figure 3A shows the expected output of the network.

Initially, the shorter sequences were padded with zeros, which produced output of mostly zeros. After padding the sequences with repetitions of the sequences, instead, the network output more non-zero integers. Further regarding handling of the input data, LSTM was finally chosen over simple dense and convolutional architectures as its expected input was more reasonable for the chosen encoder output (sequences instead of square images) and the recurrence of the network handles the time-based nature of the music files well.

After training the LSTM-based GAN model using several hundred octuple-tokenized files, eventually the discriminator overtook the generator and the generator was unable to learn new patterns (Fig. 3B).

The data was then scaled to be between -1 and 1 and the learning rate of the discriminator reduced (Fig. 3C).

After training the GAN for 1000 epochs (approximately 50 times longer than before) and using twice as much data, the network produced a distribution that more closely followed the input data (Fig. 3D). The corresponding sheet music to the example output from the generator is shown in Figure 5.

The audio transcription of the results played by a virtual piano instrument are available on soundcloud through this link : **Soundcloud Playlist**. These are 5 samples obtained with the GAN trained. The samples are all different, however, some similarities can be heard which can be assimilated to mode collapse, which is one of the main problem encountered when training GANs, where generated samples are all similar (or even identical in some cases). Nevertheless, overall results are quite promising, and the network architecture can still be tuned and improved to get more "classical" melodies.

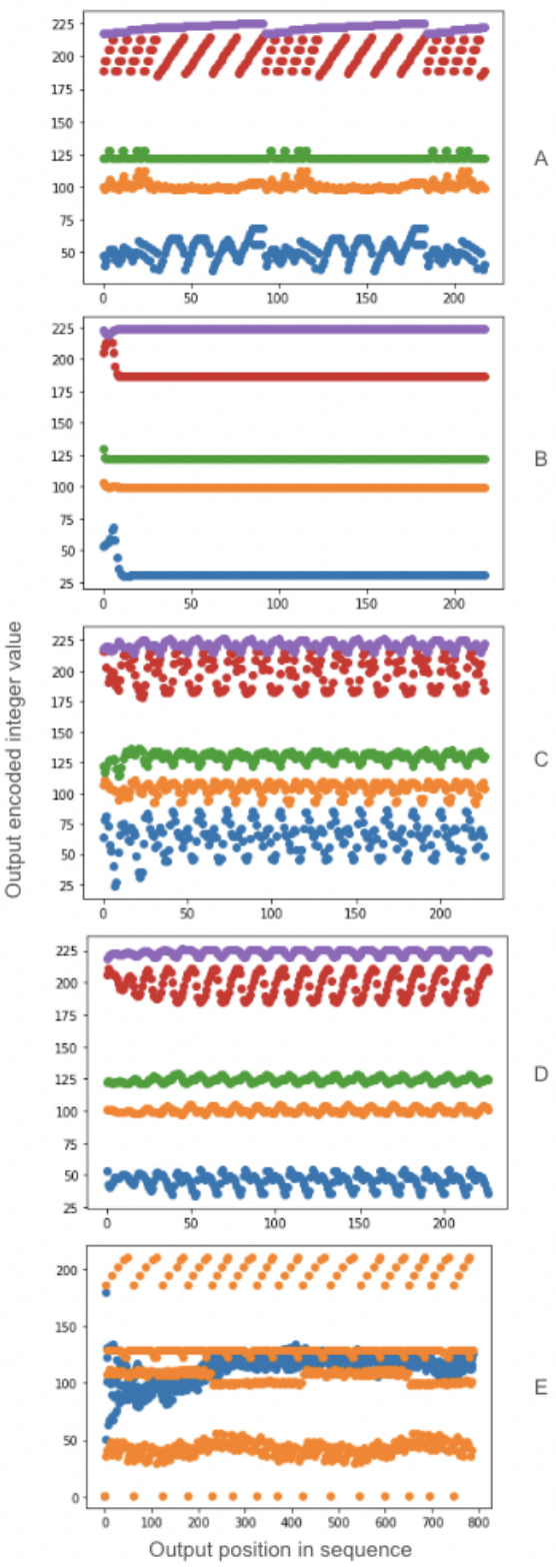

FIG. 3: Expected output of network using octuple tokenization; from bottom to top: pitch, velocity, duration, position, and bar

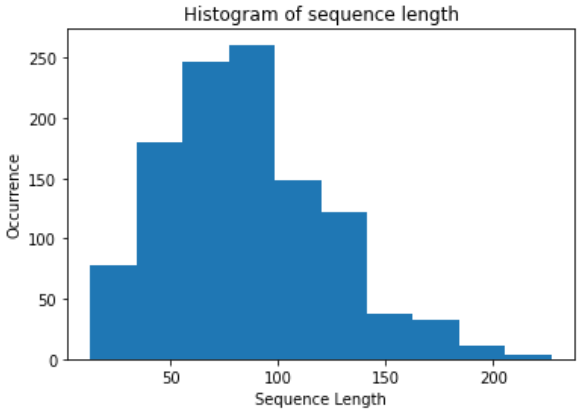

FIG. 4: Lengths of sequences used in dataset

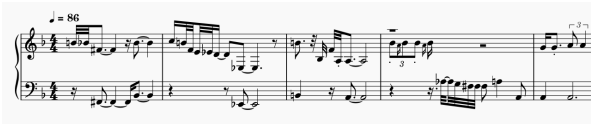

FIG. 5: Snippet of final output example, converted from MIDI to sheet music

IV. DISCUSSION

Originally inspired by the Kaggle competition "I'm Something of a Painter Myself" in which participants use a GAN to produce around 10,000 Monet-style images, as well as several examples of music generation using GAN online, the expectation for the output was that the GAN would produce sound that somewhat resembled the input music. While generating new music using RNN architecture is quite usual by training networks for prediction, GANs network for generation of symbolic music is less common and there are actually a very few example of implementations. However, after further evaluating examples online in which authors used GAN to produce music, albeit with different generator/discriminator architectures and music file encodings, the expectation that the GAN could produce Mozart-quality music seemed to be overzealous from the beginning. Even in these examples, classical music inputs resulted in quasi-jazz outputs.

However, as the network was tuned and trained for longer intervals (several hours instead of minutes), the output of the GAN evolved from monotonic (when the discriminator is too good and the output is constant) or bashing of piano keys (when the network under-trained) to surprisingly relaxing music.

To further improve results, the network could be trained on even more tokenized MIDI encodings and for many more epochs (although computing time limits on resources such as Kaggle may prove to be a constraint,

as experienced in the development of the network thus far). Additionally, since in the current GAN model, a single generator is tasked with learning the distributions of 5 different features of the music input, results may improve by training networks individually for each of the features. This issue was illuminated in one of the first outputs generated by the model. The model appeared to be minimizing the loss for the entire distribution, closely following the shape of the most dense feature (the somewhat sinusoidal band) and centered around the center of the total distribution of the input data. In Figure 3E, the input distribution is orange and the output is blue.

In this project, combining state of the art tokenization method with a LSTM-Based GAN architecture is quite new and the results seems very promising.

V. CONCLUSION

While GAN models still have a ways to go before they can reproduce and even create Mozart or Bach-quality music[12], and the GAN presented in this project could likely be fine-tuned to produce more melodic music at the level of existing examples, the evolution of the audio produced as the network improved was very exciting. Artificially generated music in the "real world" does not need only be an interesting school project; uses may range to be anything from being a hobby to help deal with anxiety or depression, to defining the next generation of music in a modern society[22].

Though the sounds generated by the model discussed in this report aren't necessarily what one would hear on the radio or at a concert, listening to the audio clips produced by the model could easily inspire new songs and accelerate, rather than undermine, the creative process. As machine learning is exponentially woven into society, it's only a matter of time before artificially generated music becomes more mainstream.

VI. CONTRIBUTIONS

All members of the team contributed equally to the project.

[1] Deep convolutional generative adversarial network nbsp;: nbsp; tensorflow core.

[2] WaveNet: A generative model for raw audio, 8 Sep. 2016.

[3] Gabora L. Bell, S. A music-generating system inspired by the science of complex adaptive systems. *Proceedings of the 4th International Workshop on Musical Metacreation (MUME 2016)*, 2016.

[4] Christine Payne. MuseNet, 25 Apr. 2019.

[5] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[6] 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, 2017.

[7] Hao-Wen Dong and Yi-Hsuan Yang. Convolutional generative adversarial networks with binary neurons for polyphonic music generation, 2018.

[8] E. R. Miranda J. A. Biles (Eds). *Evolutionary computer music*. Springer-Verlag London Limited, 2007.

[9] Angelo Genovese, Vincenzo Piuri, and Fabio Scotti. Towards explainable face aging with generative adversarial networks. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 3806–3810, 2019.

[10] G. Hadjeres and F. Pachet. Deepbach: a steerable model for bach chorales generation. techreport, December 2016. music.

[11] Mehdi Mirza Bing Xu David Warde-Farley Sherjil Ozair Aaron Courville Yoshua Bengio Ian J. Goodfellow, Jean Pouget-Abadie. Generative adversarial nets. 2014.

[12] Maximos Kaliakatsos-Papakostas, Andreas Floros, and Michael N. Vrahatis. Chapter 13 - artificial intelligence methods for music generation: a review and future perspectives. In Xin-She Yang, editor, *Nature-Inspired Computation and Swarm Intelligence*, pages 217–245. Academic Press, 2020.

[13] Bernd Krueger. Classical piano - midi page, 2022. [Online; accessed 15-May-2022].

[14] I. D. Matic, A. P. Oliveira, and A. Cardoso. Automatic melody generation using neural networks and cellular automata. Faculty of Electrical Engineering, University of Belgrade, Serbia, 2012.

[15] Rui Wang Zeqian Ju Tao Qin Tie-Yan Liu Mingliang Zeng, Xu Tan. Musicbert: Symbolic music understanding with large-scale pre-training. 2021.

[16] Fabien Chhel Amal El Fallah Seghrouchni-Nicolas Gutowski Nathan Fradet, Jean-Pierre Briot. Miditok: A python package for midi file tokenization. 2021.

[17] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[18] Yan Sun Lihong Jiang Kuo-Ming Chao Sebastian Walter, Guillaume Mougeot and Hongming Cai. Midipgan: A progressive gan approach to midi generation. 2021.

[19] Sakib Shahriar. Gan computers generate arts? a survey on visual arts, music, and literary text generation using generative adversarial network. *Displays*, 73:102237, 2022.

[20] Sigurdur Skuli. How to generate music using a lstm neural network in keras, Dec 2017.

[21] Wikipedia contributors. I–v–vi–iv progression — Wikipedia, the free encyclopedia, 2022. [Online; accessed 9-January-2022].

[22] duncan williams, victoria hodge, lina gega, damian murphy, peter cowling, and anders drachen. ai and automatic

music generation for mindfulness. *journal of the audio engineering society*, march 2019.

[23] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[24] Yi-Hsuan Yang Yu-Siang Huang. Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions. 2020.

Appendix A: MIDI

Musical Instrument Digital Interface, commonly called MIDI, is a communication protocol and a file format dedicated to music. It was firstly implemented in the 80's to normalize communication between instruments and computer with Digital Audio Workstation (i.e software for music). In this project, we use the file format MIDI as the output.
MIDI does not define audio signal but only command information. It provides information on starting and stopping a note with a certain velocity. All audible notes from C1 to G9 and velocities are encoded as a 7-bit integer (from 0 to 127). MIDI files can be read by virtual instruments (synthesizer) to create an audio signal but also by music software to get a readable music sheet to make the music actually playable by a musician.

Appendix B: Long-Short Term Memory networks

The idea of a RNN is that they might be able to remember and connect previous information to the present task. But RNNs suffer from long term dependency problem, where overtime when more information piles up then the RNNs become unable to connect the information and also become less effective in learning new things. This problem is counteracted by the use of LSTMs.

LSTMs short for Long Short Term Memory are a special kind of RNN, capable of learning long-term dependencies. LSTMs are explicitly designed to avoid the long-term dependency problem. RNNs have the form of a chain of repeating modules of neural network and this repeating module in a standard RNN will have a very simple structure, such as a single tanh layer. LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

The key to LSTMs is the cell state, it runs through the entire chain. The LSTM has the ability to remove or add information to the cell state regulated by gates. LSTMs has three gates, namely, forget gate, input gate and output gate. All these gate layers output a number between 0 and 1.

First the network should decide what information to discard from the cell state. This decision is made by a sigmoid layer called the forget gate layer. A 1 in the output represents "completely keep the information" and a 0 represents "completely discard this information".

Secondly the network should decide what new information it's going to store in the cell state. This has two parts. One part is a sigmoid layer called the input gate layer which decides what values it will update. Second part is a tanh layer which creates a vector of new candidate values that could be added to the state. The network combines these two to create an update to the state.

And finally, the network should decide what it's going to output. This is taken care by the output gate layer. First, a sigmoid layer decides what part of the cell state it's going to output and then it puts the cell state through tanh and multiply it by the output of the sigmoid gate, so it only outputs the part that it has decided.