CS230

GenreGAN: Using GANs for Music Genre Style Transfer

Naveen Kumar
navee2@stanford.edu

Lasha Gigitelashvili
lasha@stanford.edu

Abstract

Our goal was to make a model that changes genre in symbolic music. We had MIDI database of 3 Genres (Classic, Jazz and Pop) and we implemented a model based on CycleGAN[1] and then extended it to StarGAN[4] to change Genres among them.

1 Introduction

Genre Transferring from one domain to another can have interesting usage in real world. For instance, it can be used by musicians who create cover songs. They can user genre transfer to roughly see how a song will be heard in other style and choose most appropriate genre to continue their work.

The transfer should be clearly noticeable, while retaining enough of the original melody and structure such that the source piece is still recognizable. While style transfer has been studied in detail for images, its still novel for music. We have used CycleGAN[1] to transfer music genre such as pop and classic to one another as a baseline. We extended this work using StarGAN[4] architecture.

To input MIDI files to a our generator and discriminator models, they must first be converted into a matrix, the so-called piano roll representation, where columns represents time-steps and each record in this column shows if note is played or not. We omit the volume information by setting all values to 127, such that every note has the same loudness. At the and we input 4 piano rolls of size 16x84 and for generator we add target domain information as well. At the end generator generates same size piano roll but in target genre and discriminator tries to guess input song's genre.

2 Related work

There were several works of transferring object from one domain to another. Mostly those works are for transferring images. Gatys et al. [7] showed concept of neural style transfer from one image to another. In CycleGAN[8] pair of generators transforms images from domain A to domain B. Brunner et al. [1] used CycleGAN approach for music domains.

Main downside of this approach is that it is somehow inefficient. To learn all mappings among n domains $n(n-1)$ generators must be trained. To approach this issue Choi et al.[4] proposed StarGAN, a scalable approach capable of learning mappings among multiple domains. As demonstrated in Fig. 1, StarGAN model is trained on multiple domains, and with a single generator learning the mappings between all of them. The generator is not fixed to transforming from domain A to domain B, but instead it takes target domain information as well as source data and generates new data that belongs to target domain.

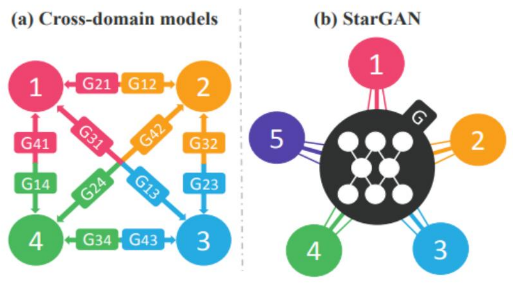

Figure 1: Comparison of cross-domain models and StarGAN. (a) To handle multiple domains, cross domain models should be built for every pair of domains. (b) StarGAN is capable of learning mappings among multiple domains using a single generator.

3 Dataset and Features

We are using Dataset [2] provided by Brunner et al. in their music genre transfer paper [1]. They have pre-processed MIDI files into numpy arrays. MIDI is a symbolic music representation that resembles sheet music. Sampling rate is 16 time steps per bar. 84 notes between C1 and C8 have been used. The piano-roll for one bar is of size 16 × 84. Since music has temporal structures we need to consider the relation of consecutive bars. Phrases consisting of 4 consecutive bars will be used as training samples. Thus, the resulting samples are of size 64 × 84. There are about 10,000 samples each for jazz, pop and classic.

For generator model we have to pass target domain label as well. Our approach for target labeling is one hot encoder with length n_d - number of domains.

4 Methods

Our goal is to train a single generator G that learns mappings among multiple domains. To achieve this, we train G to translate an input data x into an output data y conditioned on the target domain label $c, G(x,c) \to y$. We randomly generate the target domain label c so that G learns to flexibly translate the input data. We also introduce a classifier that allows a single discriminator to control multiple domains. Our discriminator produces probability distributions over both sources and domain labels $D : x \to \{D_{src}(x), D_{cls}(x)\}$.

Adversarial Loss. We use following adversarial loss:

$$\mathcal{L}_{adv} = \mathbb{E}_x[\log D_{src}(x)] + \mathbb{E}_{x,c}[\log(1 - D_{src}(G(x,c)))] \tag{1}$$

G generates music conditioned on input data x and target domain class c . $D_{src}(x)$ is probability distribution over sources. The generator wants to minimize this objective, while the discriminator wants to maximize it.

Domain Classification Loss. If we have input music data x and target domain c, our goal is to generate $y = G(x,c)$, which is classified as c.

We use following Classification loss for real data:

$$\mathcal{L}^r_{cls} = \mathbb{E}_{x,c'}[-\log D_{cls}(c'|x)] \tag{2}$$

where $D_{cls}(c'|x)$ is probability distribution over domain labels computed by D. When minimizing it D learns to classify real music x to its corresponding original domain c'. Loss function for for fake music data is following:

$$\mathcal{L}^f_{cls} = \mathbb{E}_{x,c}[-\log D_{cls}(c|G(x,c))] \tag{3}$$

G minimizes this objective and generates music data that will be classified as target domain c.

Reconstruction Loss. When using only adversarial and classification losses described above, G generates music data that should be realistic and classified as correct target domain. But, this does not guarantee that translated data preserve the musical content of its input data while changing only the domain-related part of the inputs. To address this problem, we apply following cycle consistency loss to the generator:

$$\mathcal{L}_{rec}^{r} = \mathbb{E}_{x,c,c'}[||x - G(G(x,c),c')||_1] \tag{4}$$

where G takes the generated data $G(x,c)$ and the original domain label c' and tries to reconstruct the original data x. We use $L1$ norm as our reconstruction loss. We use a same generator twice, first to translate an original data into the target domain and then to reconstruct the original data from the translated data.

Full Objective Final losses are written as following

$$\mathcal{L}_D = -\lambda_{adv}\mathcal{L}_{adv} + \lambda_{cls}\mathcal{L}_{cls}^{r} \tag{5}$$

$$\mathcal{L}_G = \lambda_{adv}\mathcal{L}_{adv} + \lambda_{cls}\mathcal{L}_{cls}^{f} + \lambda_{rec}\mathcal{L}_{rec} \tag{6}$$

where λ_{adv}, λ_{cls} and λ_{rec} are hyper-parameters for relative weighting of corresponding losses. We used λ_{adv} =1, λ_{cls} = 10 and λ_{rec} = 10 in our experiments.

5 Experiments/Results/Discussion

Our first attempt to make a working instance of described model had following problem. We monitored generator loss and discriminator loss with tensorboard and had results shown on figure 2.

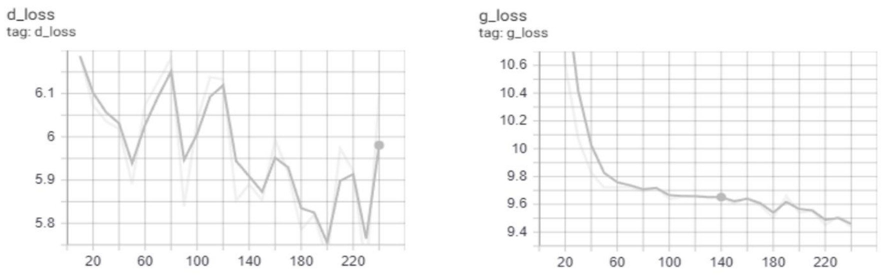

Figure 2: discriminator and generator loss

We periodically saved generated samples with following method: if note value is more then some threshold (in this case 0.5) we had note at full volume, if note value was below threshold we set note volume to 0 which means no note. We saw that after some point midi player was not playing generated files. We found out that all notes were set to 0. To monitor this we added $number\_of\_notes$ parameter, which is average number of notes in current batch and saw following result (figure 3)

As we can see generator tends to generate all 0 values. We could not figure out the reason of this behavior. We tried other GAN type (Wasserstein GAN objective with gradient penalty [9]) and various hyper parameters but the result was the same.

At some point we added another term to generator loss

$$\mathcal{L}_{diff} = |N_{notes} - N_{avg\_real\_notes}| \tag{7}$$

where N_{notes} is number of generated notes and $N_{avg\_real\_notes}$ is number of notes in average real data. The closer number of generated notes is to average number of real notes, the less is loss. So we thought this term will help model to generate desired number of notes.

After adding this term we had following picture (figure 4)

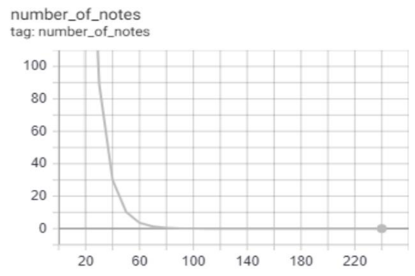

Figure 3: average number of notes generated by generator

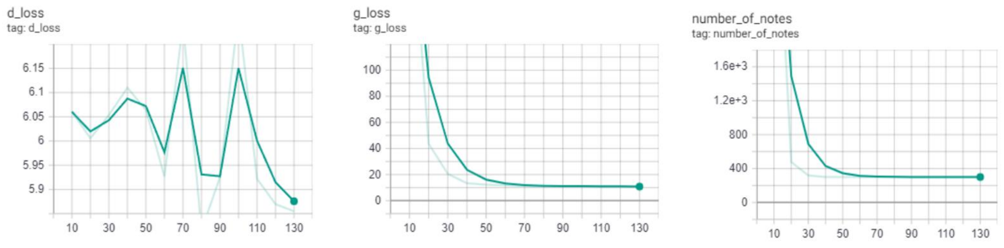

Figure 4: average number of notes generated by generator

This helped generator to generate certain number of notes but unfortunately it converged to generate same data for every input. We can say that there is some fundamental problem in implementation of our algorithm. One possible candidate for this problem is concatenating music data with target music label. Our approach was to flatten music data into $(batch\_size, time\_step * pitch\_range + number\_of\_domains)$ neurons and add dense layer to reduce it to $(batch\_size, time\_step * pitch\_range)$, after that convert back to $(batch\_size, time\_step, pitch\_range, 1)$ tensor and on top of that apply convolutions. We think that this dense layer losses music structure information and we need to implement other approach for music data and target label merging. Unfortunately we spent all time on working on structure with dense layer and had not time to experiment with other variants.

6 Conclusion/Future Work

At this point we can summarize that StarGAN approach for music generating is theoretically sound. This method proved itself with image generation and as we think is better approach than algorithms with cross-domain generator models. Main advantage of StarGAN is single generator that reduces training time during training stage and required memory during inference stage. We are sure that having sufficient time and resources StarGAN can be successfully applied to music generation and we hope our current work will be at least very small step towards achieving this goal.

7 Contributions

Lasha Gigitelashvili - Working on code, final paper, final presentation.

Naveen Kumar - Working on code, final paper, final presentation.

References

[1] Brunner, Gino and Wang, Yuyi and Wattenhofer, Roger and Zhao, Sumu (2018) Symbolic Music Genre Transfer with CycleGAN

[2] Music Dataset: https://drive.google.com/file/d/1zyN4IEM8LbDHIMSwoiwB6wRSgFyz7MEH/view?usp=sharing

[3] Coupled Generative Adversarial Networks: https://arxiv.org/abs/1606.07536

[4] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, Jaegul Choo, StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation,https://arxiv.org/abs/1711.09020

[5] M. Data, "Intuitive analysis, creation and manipulation of midi data with pretty midi," 2014.

[6] H. Dong, W. Hsiao, L. Yang, and Y. Yang, "MuseGAN: Multitrack sequential generative adversarial networks for symbolic music generation and accompaniment," in Proceedings of the ThirtySecond AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018, 2018. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17286

[7] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on. IEEE, 2016, pp. 2414– 2423.

[8] J. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017, 2017, pp. 22

[9] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In Proceedings of the 34th International Conference on Machine Learning (ICML), pages 214–223, 2017. 5