Unrolled Creative Adversarial Network Generating Novel Musical Pieces

Shyma Alhuwaider
King Abdullah University for Science and Technology
Thuwal, Saudi Arabia
shyma.alhuwdier@kaust.edu.sa

Pratik Nag
pratik.nag@kaust.edu.sa

Abstract

Music generation has become a hot topic in Artificial intelligence and Machine learning ([2] ,[13]) over recent years. In most of the recent works ([4], [7], [3]) RNN-based Neural network methods has been applied for sequence generation. Whereas very few researchers [10] have explored Generative Adversarial Networks and its counter parts for music generation. In this project We make use of a classical system and testing a new system for generating creative music. Both systems are based on adversarial networks that generate music by learning from examples. The classical system learns a set of music pieces without differentiating between the classes. Where as for the new system the network learns the different composers and their different styles to generate a creative music piece by deviating from the learned composers' styles. Our base structure is Generative Adversarial Networks (GAN), that are capable of generating novel outputs given a set of inputs to learn from and mimic it's distribution. It has been presented in previous work that GAN are limited with respect to creativity products in their original design [5]. We build on their work using Creative Adversarial Networks (CAN) by applying it on the music domain rather than the visual art domain. In addition to that we introduce unrolled CAN to prevent mode collapse. We conducted experiments on both GAN and CAN in generating music and measured their capabilities on deviating form the input set.

1. Introduction

Machines have always been shown as devices that can replicate certain learned behaviours. It has been a dream to machines produce creativity and is of much desire in the machine learning scope and artificial intelligence. After all if we want machines to mimic human behaviour, then one of the most important traits is creativity. Take driving for example; human are capable of using their creativity and creative problem solving skills, to avoid unexpected road events, preventing accidents. That is a skill that all self driving car developers hope to accomplish. The art of generating music is a very complex problem [10] and is a different challenge than that of text generation. It includes finding a rhythm between the musical notes and learning polyphony and many more challenges.

To study style in music lets take a look at what style means in language. When one speaks or writes their style is defined by their selection of words an vocabulary, their choice of grammar and linguistics governed within constraints of the language and dialect one has learned, and have not created. However, since constraints can allow for interpretations of different realizations it produces variety. Therefore, a pattern in a style of writing or a piece of music may not relate in every aspect, only some aspects are selected to convey pattern replication. In 1,2 we can see two examples of musical pieces replicating a pattern of using pitches (8-7/2-1) typically chosen between classical music composers [9].

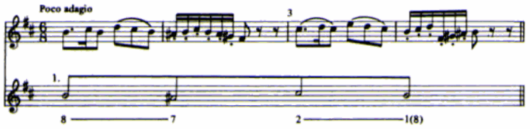

Figure 1. Symphony No.46 in B major Haydn

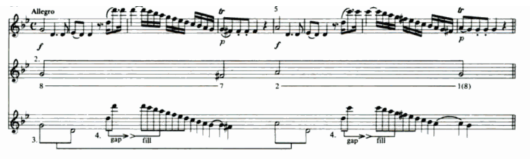

Figure 2. Mozart Piano Quartet in G minor

There are two clefs in musical sheets (3) that are used to differentiate between high and bass notes and are called the treble clef and the bass clef. The treble clef is usually played with the right hand on the piano and the bass clef is usually played with the left hand on the piano.

In music sheets some note may be represented different graphically yet representing the same note. And sometimes

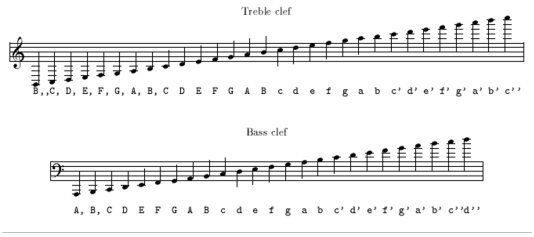

Figure 3. Treble clef and Bass clef

certain music sheets may look identical graphically while actually are representing different notes that is due to the change in the clef (Check figure 4).

A professional musician must be able to identify the similarities and differences in the notes. That comes naturally since professional musicians are regularly reading sheet music.

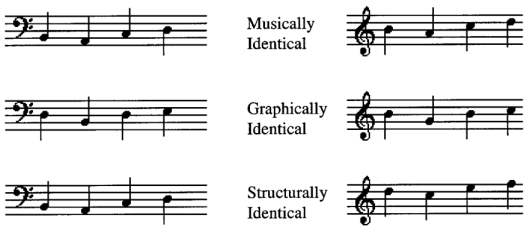

Figure 4. Same-Different exercise example [12]

2. Methodology

In this section the methodology of music GAN and CAN is discussed.

2.1. Data set

Three data sets were used in this project; jazz midi data set, classical music midi data set classified by composers, and Arabic music midi data set. All the data sets are included in our github repository. Both the jazz data set and the classical music data set were retrieved from kaggle. The Arabic data set was collected manually and cleaned to cancel out some of the repetitive notes such as drums.

2.2. Data Preprocessing

The data used in this project contain midi files. In order to pass them to the GAN or the CAN we proceeded with converting the midi files to images. Converting each note to a pixel location. which can be converted back to midi file without losing any of the information. (see figure 5)

2.3. Music GAN architecture

Here we show the architecture used for the music GAN. Which consists of two networks; a generator network and a discriminator network. The generator takes in a random

Figure 5. Sample of musical piece by the composer Albeniz in image format

vector Z and generates an image representing a midi file. The generated image is then passed to the discriminator which tries to distinguish between the real images data set and the fake (generated) images. The GAN is trained over the three data sets separately creating three different generators. One that is capable of producing jazz musical pieces, another that is capable of generating classical music pieces, and lastly an experimental generator trained on manually collected and cleaned data is capable of generating Arabic musical pieces.

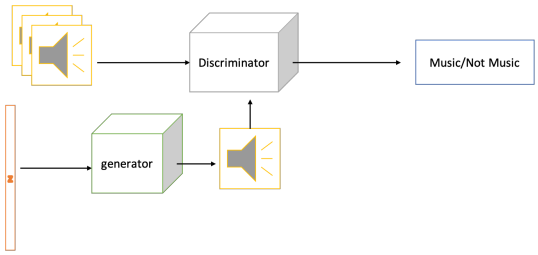

Figure 6. GAN architecture

A detailed architecture of Discriminator and Generator. 7,8

```
Discriminator(
  (net): Sequential(
    (0): Conv2d(1, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): Dropout(p=0.3, inplace=False)
    (4): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (5): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): LeakyReLU(negative_slope=0.01)
    (7): Dropout(p=0.3, inplace=False)
    (8): Flatten()
    (9): Linear(in_features=524288, out_features=1, bias=True)
    (10): Sigmoid()
  )
)
```

Figure 7. GAN Discriminator block

2.4. Music CAN and unrolled CAN architecture

In the music CAN the architecture[11] is still made up of two neural networks that compete against each other. However, the loss here stresses on the generator motivating it to produce a musical piece that cannot be classified as belonging to one class. In other words, the discriminator should be capable of classifying the original real data to their composers or class label. The discriminator should not be able

```
Generator(
  (net): Sequential(
    (0): ConvTranspose2d(100, 1024, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU()
    (9): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU()
    (12): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU()
    (15): ConvTranspose2d(64, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (16): Tanh()
  )
)
```

Figure 8. GAN Generator block

to find a suitable class for the generated piece of music. Thus the input data must be labeled by the class it belongs to in order to reduce the loss on real data classification. The generator architecture is similar to that of the GAN where it takes a random z vector as an input and generates an image representing the midi file.

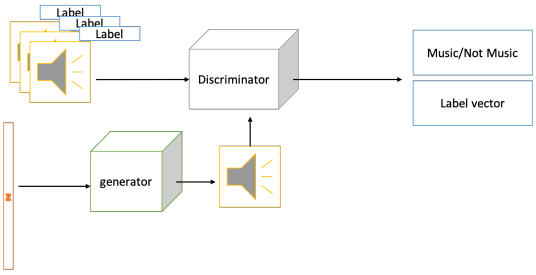

Figure 9. CAN architecture

A detailed architecture of Discriminator and Generator. 10, 11 For our CAN generator architecture we are convert-

```
Discriminator(
  (net): Sequential(
    (0): Conv2d(1, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): Dropout(p=0.3, inplace=False)
    (4): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (5): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): LeakyReLU(negative_slope=0.01)
    (7): Flatten()
  )
  (disc): Sequential(
    (0): Linear(in_features=524288, out_features=1, bias=True)
    (1): Sigmoid()
  )
  (classify): Sequential(
    (0): Linear(in_features=524288, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.01)
    (2): Linear(in_features=512, out_features=19, bias=True)
  )
)
```

Figure 10. CAN Discriminator block

```
Generator(
  (net): Sequential(
    (0): ConvTranspose2d(100, 1024, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU()
    (9): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU()
    (12): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU()
    (15): ConvTranspose2d(64, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (16): Sigmoid()
  )
)
```

Figure 11. CAN Generator block

ing a size 100 random z vector using convolution layer and batch normalization layer to $512 * 4 * 4$ tensor, apply convolution and batch normalization to produce a $256 * 8 * 8$ tensor, apply convolution and batch normalization to produce a $128 * 32 * 32$ tensor, apply convolution and batch normalization to produce a $64 * 64 * 64$ tensor, finally apply convolution and batch normalization again to produce a $1 * 128 * 128$ tensor representing a graphical representation of the midi file.

As for the discriminator we use two convolution layers applying batch normalization on both and a leaky relu activation, followed by the fully connected layer.

3. Technical Details

3.1. Generative Adversarial Networks

GAN was first present by Ian J. Goodfellow on 2014 where two deep neural networks play a minmax game over the value function $V(G, D)$ [6]. Where G is the Generator and D is the Discriminator. And z is a random vector supplied to the generator as input.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z)))] \quad (1)$$

3.2. Creative Adversarial Networks

CAN was presented recently where a modification to the value function to include cross entropy loss over the classes of the generated output $(G(z))$[5]. Thus now the architecture includes classification and therefore, the loss also includes classification loss. And in order to convey creativity the authors used class ambiguity loss. The class ambiguity loss is achieved by maximizing the class posterior entropy. And hence, the value function is as follows

$$\min_G \max_D V(D,G) = \mathbb{E}_{x,\hat{c}\sim p_{data}}[logD_r(x) + logD_c(c = \hat{c}|x)] + \mathbb{E}_{z\sim p_z}[log(1 - D_r(G(z))) - \sum_{k=1}^{K}(\frac{1}{K}log(D_c(c_k|G(z))) + (1 - \frac{1}{K})log(1 - D_c(c_k|G(z))))] \quad (2)$$

Where as above z here is the random vector. And c here represents the class label.

3.3. Unrolling Creative Adversarial Networks

The unrolled CAN is a combination between unrolled GAN and CAN architecture where this structure allows the generator network to look into the future of the discriminator network and optimize over the future discriminator

but not updating the discriminator just yet[8]. In other words, the generator is a number (unrolling step numbers may vary) of steps ahead than the discriminator. The result is a reduction in the strength of the discriminator allowing the generator to catch up. In figure 12 we show the result of the regular CAN network after 20 epochs and the results after the modification was introduced making it an unrolled CAN.

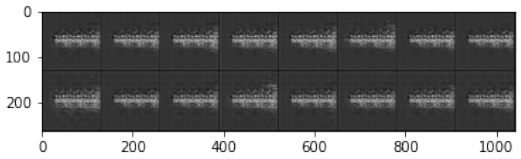

Figure 12. Example of CAN output converging to almost identical result after 20 epochs

The way the parameters are updated in GAN is described by the set of equations below

$$G \leftarrow G - \eta \frac{dV(D(G), G)}{dG} \tag{3}$$

$$D \leftarrow D - \eta \frac{dV(D, G)}{dD} \tag{4}$$

where $V(D, G)$ is our value function. The updates in parameters after unrolling will be as follows

$$G \leftarrow G - \eta \frac{dV(D^k(G), G)}{dG} \tag{5}$$

$$D \leftarrow D - \eta \frac{dV(D, G)}{dD} \tag{6}$$

Where η here represents the learning rate. and k is the number of unrolling steps. See figure 13 for the changes due to Unrolled-CAN.

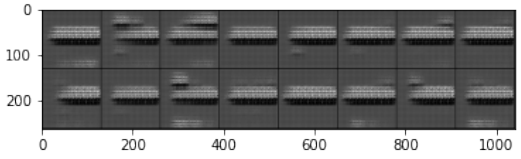

Figure 13. Example of unrolled CAN, with k=1, output converging to varying result after 20 epochs

4. Results

In this section we will discuss and compare the results of the three architectures; GAN, CAN, and unrolled CAN. The results are presented after post-processing the images and converting them to musical notes extracted from their corresponding midi files.

4.1. Scoring novelty generated musical pieces

In this paper we followed the expert gate [1] technique where an auto-encoder is trained on the reconstruction of the real data set. This expert gate becomes an expert in reconstruction of the real data set, which is used to test the novelty of the generated image. The Mean Squared Error (MSE) is then used as the score. Higher scores mean higher creativity and deviation from the training set. After training the expert gate the reconstruction error of the real data set has an average of (0.01 MSE). 14 for network architecture.

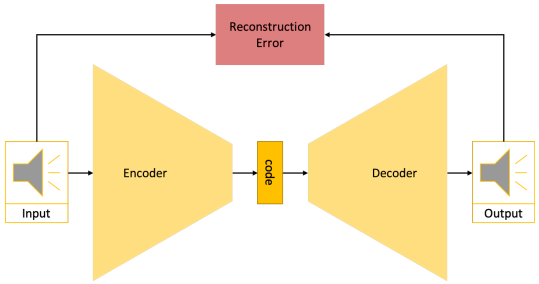

Figure 14. Auto-encoder network architecture

4.2. Music GAN results

Here the music GAN was capable of generating a musical piece that sounds and scores close to the data set. The output presented below is a result of training on the Jazz data set. The resulting generated data files have very low reconstruction with an average of (0.02 MSE). The generated music produced by the GAN architecture was only abke to capture the treble clef it was incapable of producing the bass clef.

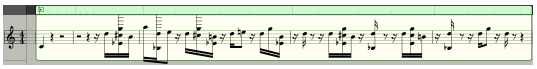

Figure 15. Output musical note from classical GAN architecture trained over Jazz data set

4.3. Music CAN results

The CAN network was trained on the classical music data set where each composer was considered a class label the goal was to generate new classical music that deviates from all the classical music composer styles. From the results below we can see an improvement over the GAN network where the bass clef is being captured and more coherence between the notes. The average score of the generated music set is (0.013 MSE) which means although the music is novel and has a high entropy between composers' styles it is still generating music files that are close to the data set.

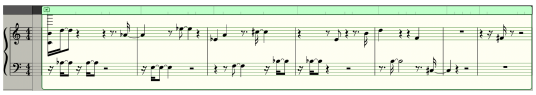

Figure 16. Output musical note from CAN architecture trained over classical music data set

4.4. Unrolled Music CAN results

Unrolled CAN architecture mixed the best of both GAN and CAN and presenting higher scores whilst maintaining high entropy between composers' styles. The average score of unrolled CAN generated music is (0.043 MSE).

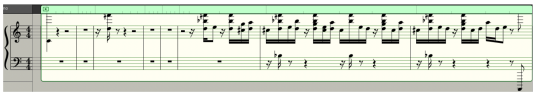

Figure 17. Output musical note from Unrolled CAN architecture trained over classical music data set

4.5. Discussion

It is observed that unrolled CAN is capable of deviating from learned styles and also deviating from the data set while still being able to trick the discriminator network.

Network	Max Novelty Score
GAN	0.029
CAN	0.013
Unrolled CAN	**0.043**

Table 1. Results. Ours is better.

Non the less the CAN architecture was very good at generating music that deviates from the composers' styles but converging to the generation of similar results leading to mode collapse which could be the reason why the average reconstruction error was very low.

4.6. Reproducing the results

All of our codes related to the work has been uploaded in the following github link: GITHUB REPO. To train the Unrolled CAN model on dataset one only needs to run the file GAN_reconstructed.ipynb which can be found in the folder named CAN_models. sample data has been given in classical_data_midi folder. The files named training.py and model.py has all the model related configurations defined. The midi datasets can be processed to images using the data pre processing.ipynb file.

To train our second best model that is the DCGAN model, one need to navigate to GAN_models folder and run the GAN-Copy2.ipynb file. Note that before running the file make sure to specify the proper folder path containing the image representation of the midi datasets in the GAN-Copy2.ipynb file. Some generated results for both the models has been uploaded in the respective folders.

The novelty_score.ipynb file has been used here to get the novelty score as discussed in [1]. The file can be found in the home directory.

The datasets used for this work can be found here : Jazz Midi Dataset, and Classical Midi Dataset.

5. Future improvements

In future improvements we hope to have a wider variety of the data set and introduce different instruments within one piece creating unrolled concert CAN capable of orchestrating different pieces of music using multiple instruments. We hope to deploy this model online using amazon web services. Deployed model will have multiple optionalities including choice of data set, length of musical piece and much more. Generated music is present in midi files allowing users to modify speed and instrument selection and much more.

6. Conclusion

now is the new artistic era that introduces machines into the playground. Machines are now capable of contributing to the artistic and imaginative sector which they were and for a long time not contributing to it. It is always desired to see machines evolving in artificial intelligence and perceive new ideas that humans might have a harder time to capture. Humans require years to learn music theory and play fluently. Rather here in this paper we present an algorithm capable of learning to play musical instruments fluently within few minutes and produce music within seconds. To conclude we have seen the different way musical styles are classified, looked into multiple architectures. We presented in this paper that the GAN structure [6], and CAN in [5] are fully capable of generating music. We also bring forward a novel modification to the CAN by unrolling it using techniques shown in [8]. This project was a combination of art and science. Using the capabilities of machine learning to produce novel and new generated music. This could be the next playlist hotels are going to play in their lobbies, it could help singers and music composers look for new inspirations. After all machines are made for us to benefit from.

Acknowledgement

We would like to thank prof. Mohamed Hamdy Elhoseiny for his guidence throughout this project. We would also like to thank the TA's and other members of the course for giving us impactful insights and suggestions on our model architecture and the performance evaluation of our work. We have also taken help from the homework assignment codes to reproduce our own pytorch model for DCGAN as well as CAN architecture.

Project contribution

We both have worked simultaniously on all the project deliverables. To be specific : Pratik Nag mainly worked and researched on midi to image transformation, DCGAN and CAN architecture development, data preprocessing architecture development (midi to image transition),conducting local small scale experients, while Shyma Alhuwaider worked on data collection, data preprocessing, data post processing, Unrolled CAN architecture, conducting experiments, collecting results and evaluating the models. As well as running the final versions of the models on ibex for final evaluation. Both have worked together on implementation of the novelty score.

References

[1] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts, 2017.

[2] Jean-Pierre Briot and François Pachet. Deep learning for music generation: challenges and directions. *Neural Computing and Applications*, 32(4):981–993, 2020.

[3] C-CJ Chen and Risto Miikkulainen. Creating melodies with evolving recurrent neural networks. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, volume 3, pages 2241–2246. IEEE, 2001.

[4] Hannah Davis and Saif M Mohammad. Generating music from literature. *arXiv preprint arXiv:1403.2124*, 2014.

[5] Ahmed M. Elgammal, Bingchen Liu, Mohamed Elhoseiny, and Marian Mazzone. CAN: creative adversarial networks, generating "art" by learning about styles and deviating from style norms. *CoRR*, abs/1706.07068, 2017.

[6] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[7] Yu-Siang Huang and Yi-Hsuan Yang. Pop music transformer: Generating music with rhythm and harmony. *arXiv preprint arXiv:2002.00212*, 2020.

[8] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks, 2017.

[9] Leonard B Meyer. *Style and music: Theory, history, and ideology*. University of Chicago Press, 1996.

[10] Aashiq Muhamed, Liang Li, Xingjian Shi, Rahul Suresh, and Alexander J Smola. Symbolic music generation with transformer-gans. 2021.

[11] Othman Sbai, Mohamed Elhoseiny, Antoine Bordes, Yann LeCun, and Camille Couprie. Design: Design inspiration from generative networks. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018.

[12] Daniele Schön, Carlo Semenza, and Gianfranco Denes. Naming of musical notes: A selective deficit in one musical clef. *Cortex*, 37(3):407–421, 2001.

[13] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint arXiv:1703.10847*, 2017.