MuseGAN: Symbolic-domain Music Generation and Accompaniment with Multi-track Sequential Generative Adversarial Networks

Hao-Wen Dong* [1], Wen-Yi Hsiao* [1,2], Li-Chia Yang [1], Yi-Hsuan Yang [1]
[1] Research Center for Information Technology Innovation, Academia Sinica, Taipei, Taiwan
[2] Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
salu133445@citi.sinica.edu.tw, s105062581@m105.nthu.edu.tw, {richard40148,yang}@citi.sinica.edu.tw
* These authors contributed equally to this work.

Abstract

Generating music has a few notable differences from generating images and videos. First, music is an art of time, necessitating a temporal model. Second, music is usually composed of multiple instruments/tracks, with close interaction with one another. Each track has its own temporal dynamics, but collectively they unfold over time interdependently. Lastly, for symbolic domain music generation, the targeted output is sequences of discrete musical events, not continuous values. In this paper, we propose and study three generative adversarial networks (GANs) for symbolic-domain multi-track music generation, using a data set of 127,731 MIDI bars of pop/rock music. The three models, which differ in the underlying model assumption and accordingly the network architecture, are referred to as the jamming model, composer model, and hybrid model, respectively. We propose a few intra-track and inter-track objective metrics to examine and compare their generation result, in addition to a subjective evaluation. We show that our models can learn from the noisy MIDI files and generate coherent music of four bars right from scratch (i.e. without human inputs). We also propose extensions of our models to facilitate human-AI cooperative music creation: given the piano track composed by human we can generate four additional tracks in return to accompany it.

Introduction

Generating realistic and aesthetic pieces have been considered as one of the most exciting tasks in AI. Thanks to the revolution of deep neural networks, recent years have seen major progress in generating images, videos and text, notably using a generative model called generative adversarial networks (GANs) (Goodfellow et al. 2014; Radford, Metz, and Chintala 2015; Vondrick, Pirsiavash, and Torralba 2016; Saito and Matsumoto 2016; Tulyakov et al. 2017; Yu et al. 2017). Similar attempts have also been made to generate music (see the Related Work section), but the task remains challenging due to the following reasons.

First, music is an art of time. As shown in Figure 1, music has a hierarchical structure, with higher-level building blocks (e.g. a phrase) made up of smaller recurrent patterns (e.g. a bar). People pay attention to structural patterns related to coherence, rhythm, tension, and the emotion flow (Herremans and Chew 2017) while listening to music. Therefore, a mechanism to account for the temporal structure is critical.

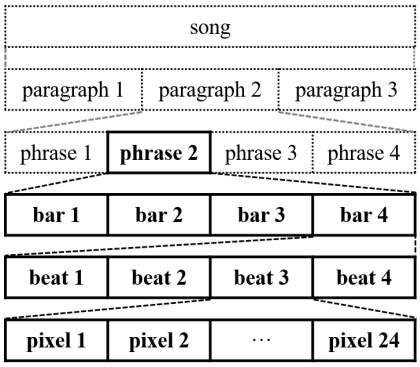


Figure 1: Hierarchical structure of a music piece.

Second, music is usually composed of multiple tracks. A symphony orchestra contains a large ensemble of string and brass instruments, and a rock band often consists of vocal, guitar, bass and drums. These tracks interact with one another closely and unfold over time interdependently. In the music theory, we can find extensive discussion on composition disciplines for relating sounds, such as harmony (Tymoczko 2010) and counterpoint (Jeppesen 2013).

Lastly, while the target output for image and video generation is usually continuous pixel values, in symbolic domain music generation the target output is discrete (more specifically, binary), for music represents a sequence of discrete musical events corresponding to the on and off of musical notes from different instruments (Raffel and Ellis 2016). In consequence, success in image and video generation may not be readily generalizable to music generation.

As a result, existing works on music generation are usually limited in certain aspects. Examples include: enforcing temporal structure with the more expensive recurrent neural network (RNN) design (Mogren 2016), generating only single-track music (Jaques et al. 2016; Roberts et al. 2016; Bretan, Weinberg, and Heck 2016; Yang, Chou, and Yang 2017), requiring user input (Chu, Urtasun, and Fidler 2016), focusing on one specific genre (Hadjeres and Pachet 2016), or dealing with audio-domain, continuous-valued generation (van den Oord et al. 2016; Blaauw and Bonada 2017).

The goal of this work is to build a neural music generator that is free of these limitations, based on state-of-the-art

convolutional GANs (Gulrajani et al. 2017). Specifically, we propose the idea of *intra-track* and *inter-track* discriminative feedbacks, and develop a multi-loss training procedure to train the adversarial network for generating multi-track music. The intra-track feedbacks are used to generate each individual track, just like a music teacher specialized in one specific instrument. And, the inter-track feedback serves as a composer or a band leader, who tries to evaluate the joint performance of all the musicians (tracks) in the play. To investigate their effectiveness, we build a 'jamming' GAN that uses only the intra-track loss, a 'composer' GAN that uses only the inter-track loss, and a 'hybrid' GAN that uses both.

Moreover, we propose two methods to incorporate a temporal model to our networks, either by viewing time as an additional dimension to be generated by the networks, or by learning to follow the temporal structure of a track given *a priori* by human. The latter one can be applied to human-AI cooperative music generation, or music accompaniment (Raphael 2010; Roberts et al. 2016).

In our implementation, all the parameters of the neural networks are learned from a collection of discrete-valued MIDI files (Raffel 2016), which are composed of mainly pop/rock music containing possibly noisy score of the following five tracks: bass, drum, guitar, piano, and strings.

We further propose a few intra-track and inter-track objective measures and use them to monitor the learning process and to evaluate the generation result quantitatively. We also report a user study involving 144 listeners for a subjective evaluation of the result of different networks.

To the best of our knowledge, this work represents the first attempt to develop a GAN model that can deal with the aforementioned three challenges at the same time, even not seen in other domains: sequential, multi-track, and discrete-valued generation. Moreover, because our model is one type of convolutional neural networks (CNNs), it is supposed to run faster than RNN-based models and is more easily parallelized (van den Oord et al. 2016; Gehring et al. 2017).

We dub our model as the multi-track sequential generative adversarial network, or MuseGAN for short. Although we focus on music generation in this paper, the design is fairly generic and we hence hope it will be adapted to generate multi-track sequences in other domains as well.

In what follows, we give a brief introduction to GANs first, and then present details of the proposed model.

Generative Adversarial Networks

The core concept of GAN is to achieve adversarial learning by constructing two networks: the generator 'G' and the discriminator 'D' (Goodfellow et al. 2014). The generator aims at converting a random noise sampled from a prior distribution into a realistic artificial data, whereas the discriminator has to distinguish real data from those generated by the generator. While training, both networks will be updated simultaneously by solving the following min-max game:

$$\min_G \max_D \mathbf{E}_{\mathbf{x}\sim p_{\text{data}}}[\log(D(\mathbf{x}))] + \mathbf{E}_{\tilde{\mathbf{x}}\sim p_G}[1-\log(D(\tilde{\mathbf{x}}))]\,, \tag{1}$$

where $\mathbf{x}$ and $\tilde{\mathbf{x}} = G(\mathbf{z})$ represent the real and artificial data sampled respectively from the distributions p_{data} and p_G, and $\mathbf{z}$ is a vector of random values. By solving Eq. (1), the distributions p_{data} and p_G would become closer and optimally identical when they reach the Nash equilibrium.

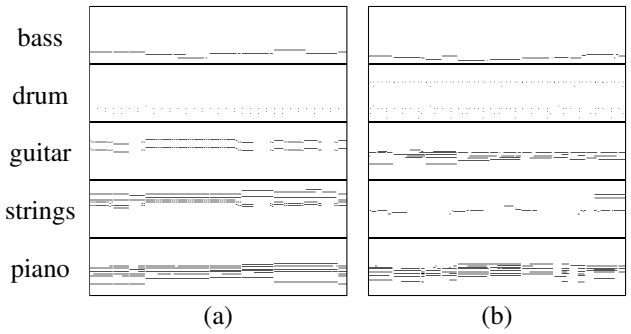


Figure 2: Piano-roll (time step–note) representation of two MIDI fragments, each with five tracks. A black dot indicates that a specific note is played at that time step. Both fragments use bass to play the melody, and fragment (b) uses fewer strings. We use piano-roll as our data representation.

Follow-up research found major issues in Eq. (1) and suggested the following alternative formulation to stabilize the training process and to avoid the issue of mode collapsing (Arjovsky, Chintala, and Bottou 2017):

$$\min_G \max_D \mathbf{E}_{\mathbf{x}\sim p_{data}}[D(\mathbf{x})] - \mathbf{E}_{\tilde{\mathbf{x}}\sim p_G}[D(\tilde{\mathbf{x}})]\,. \tag{2}$$

Eq. (2) minimizes the Earth Movers distance, or Wasserstein distance, between p_{data} and p_G, giving rise to the name Wasserstein GAN (WGAN). Later on, (Gulrajani et al. 2017) proposed to add a gradient penalty (GP) term to the objective function of the discriminator, for some favorable properties of the learning process. The objective function of the discriminator becomes:

$$\mathbf{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})] - \mathbf{E}_{\tilde{\mathbf{x}}\sim p_G}[D(\tilde{\mathbf{x}})] + \mathbf{E}_{\mathbf{x}\sim p_D}[(\nabla\|\mathbf{x}\|-1)^2]\,, \tag{3}$$

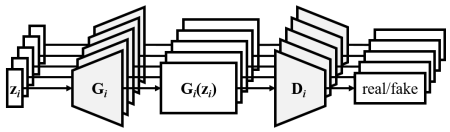
where $\nabla(\cdot)$ is the gradient operator. Model learning with Eq. (3) is found to generate high-quality result with faster convergence and less parameters tuning. Therefore, we resort to this WGAN-GP model as our generative adversarial model.

Proposed Model

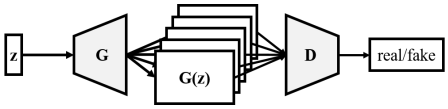
Data Representation

Figure 2 shows the so-called *piano-roll* representation of two MIDI fragments, each having five tracks. As can be seen, a piano-roll representation is a binary-valued, scoresheet-like matrix representing the presence of notes over different time steps. The horizontal axis represents time, and the vertical axis represent notes (from low-pitched to high-pitched ones). Each MIDI fragment in Figure 2 has multiple bars, but the piano-roll of a bar is still a (narrower) matrix.

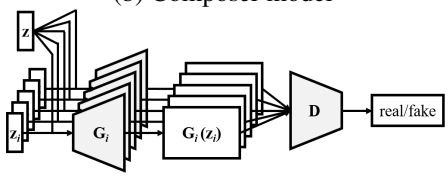
Following (Yang, Chou, and Yang 2017), we take bars as the basic unit of music. Harmonic changes (e.g. chord changes) usually occur at the boundary of bars. Moreover, when composing songs, human beings often use bars as the building blocks. There is a piano-roll for the music of each



(a) Jamming model



(b) Composer model



(c) Hybrid model

Figure 3: Three GAN models for generating multi-track data. Note that we do not show the real data $\mathbf{x}$, which will also be fed to the discriminator(s).

bar, each track, both for the real and the generated data (i.e. the desired output). As this matrix has a fixed size, we can use convolutions in our D and G to learn patterns.

Modeling the Multi-track Interdependency

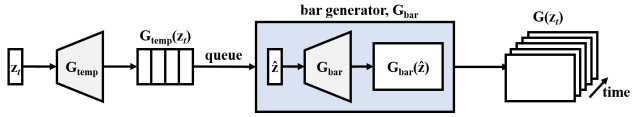
For human beings, there are typically two ways to create pop music. Given a group of musicians playing different instruments, they can create music by improvising music without a predefined arrangement, a.k.a. jamming. Or, we can have a composer who arranges different instruments with knowledge of harmonic structure and instrumentation. Musicians will then follow the composition and play the music.

We design two generative models corresponding to these two composition approaches. In the **jamming model**, multiple generators work independently and generate music of its own track from a different random vector $\mathbf{z}_i$, $i \in \{1, 2, \ldots, M\}$, where M denotes the number of generators (or tracks). Moreover, these generators receive critics (i.e. backpropogated supervisory signal used to update model parameters) from different discriminators, whom we refer to as **intra-track discriminative feedbacks**. As illustrated in Figure 3(a), to generate music of $M = 5$ tracks, we need five generators and five discriminators (or '5G5D' for short).

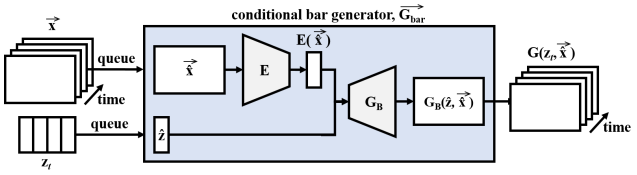
In contrast, in the **composer model**, one single generator creates a multi-channel piano-roll, with each channel representing a specific track, as shown in Figure 3(b). This model requires only one input random vector $\mathbf{z}$ (which may be viewed as the intention of the composer) and one discriminator. This discriminator examines the M tracks collectively to tell whether the input music is real or fake, and then provides **inter-track discriminative feedback** to the generator. Regardless of the value of M, we always need only '1G1D.'

Combining the idea of jamming and composing, we further propose a **hybrid model** that exploits both intra- and inter-track discriminative feedbacks. As illustrated in Figure 3(c), the hybrid model uses M generators to create music of M tracks. Each generator takes as inputs an *inter-track* random vector $\mathbf{z}$ and an *intra-track* random vector $\mathbf{z}_i$. We expect that the inter-track random vector can coordinate the generation of different musicians (G_i), just like a composer does. Moreover, we use only one discriminator to evaluate the M tracks collectively. For $M = 5$, we need five generators and just one discriminator, or '5G1D.'

A major difference between the composer model and the hybrid model lies in flexibility—in the hybrid model we can use different network architectures (e.g. number of layers, filter size) and different inputs for the M generators. Therefore, we can for example vary the generation of one specific track without losing the inter-track interdependency.



(a) Generation from scratch



(b) Track-conditional generation

Figure 4: Two temporal models employed in our work. Note that only the generators are shown.

Modeling the Temporal Structure

The models presented above can only generate multi-track music bar by bar, with possibly no coherence among the bars. We need a temporal model to generate music of a few bars long, such as a musical phrase (cf. Figure 1). We design two methods to achieve this, as described below.

Generation from scratch (T1) The first method aims to generate fixed-length musical phrases by viewing bar progression as another dimension to grow the generator, as shown in Figure 4(a). The generator G consists of two parts, the **temporal structure generator** G_{temp} and the **bar generator** G_{bar}. Let $\mathbf{z}_t$ be the **time-dependent random vectors** and T the number of bars per phrase. G_{temp} takes $\mathbf{z}_t$ as input and generates (using 1D convolutions) for each bar a *latent vector*, which carries temporal information. The latent vector is then used by G_{bar} to generate music sequentially:

$$G(\mathbf{z}_t)^{(t)} = G_{\text{bar}}\left(G_{\text{temp}}\left(\mathbf{z}_t\right)^{(t)}\right), \quad \forall\, t = 1, 2, \ldots, T\,. \tag{4}$$

The collection of $G(\mathbf{z}_t)^{(t)}$ over time becomes a tensor. We note that similar idea has been used by (Saito and Matsumoto 2016) for video generation.

Track-conditional generation (T2) The second method assumes that one specific track $\overrightarrow{\mathbf{x}}$ (of T bars) is given by human, and tries to learn the temporal structure underlying that track and to generate the remaining tracks and complete the

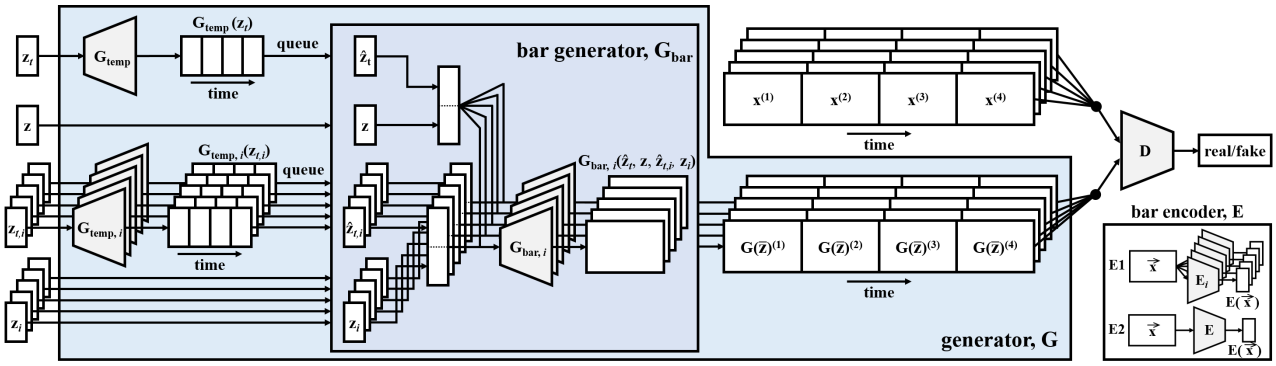


Figure 5: System diagram of the proposed MuseGAN model for multi-track sequential data generation.

song. As shown in Figure 4(b), the generator G now sequentially generates bars one after another with the **conditional bar generator** $\overrightarrow{G_{\text{bar}}}$, which takes two inputs, a random noise $\mathbf{z}$ and one bar of the given track $\overrightarrow{\mathbf{x}}^{(t)}$, and generates a bar conditioned on that track. We achieve such a conditional generation by projecting $\overrightarrow{\mathbf{x}}^{(t)}$ to the space of $\mathbf{z}$ (and then concatenate them) via an encoder E, which is another CNN that functions as the *conditioner* named by (Yang, Chou, and Yang 2017). We have accordingly,

$$G(\mathbf{z}_t)^{(t)} = \overrightarrow{G_{\text{bar}}}\left(\mathbf{z}^{(t)} \circ E\left(\overrightarrow{\mathbf{x}}^{(t)}\right)\right), \quad \forall\, t = 1, 2, \ldots, T\,. \tag{5}$$

where $\circ$ denotes vector concatenation. Notably, the encoder is forced to extract inter-track features instead of intra-track features from the given track, since intra-track features are supposed to be not useful for generating the other tracks.

To our knowledge, incorporating a temporal model in this way is new. It requires human input but it can find applications in human-AI cooperative generation (Raphael 2010).

MuseGAN

We now present MuseGAN, an integration and extension of the proposed multi-track and temporal structure models. As shown in Figure 5, the input to MuseGAN, denoted as $\bar{\mathbf{z}}$, is composed of four parts: an inter-track time-independent random vectors $\mathbf{z}$, an intra-track time-independent random vectors $\mathbf{z}_i$, an inter-track time-dependent random vectors $\mathbf{z}_t$ and an intra-track time-dependent random vectors $\mathbf{z}_{t,i}$. Firstly, the temporal structure generators, G_{temp} and $G_{\text{temp},i}$, take the time-dependent vectors $\mathbf{z}_t$ and $\mathbf{z}_{t,i}$ respectively, as their inputs and both output a series of latent features containing temporal information. Secondly, the latent features from G_{temp} and $G_{\text{temp},i}$ are concatenated respectively with the time-independent random vectors $\mathbf{z}$ and $\mathbf{z}_i$, and then fed to the bar generator G_{bar}, who generates bars sequentially:

$$G_i\left(\bar{\mathbf{z}}\right)^{(t)} = G_{\text{bar},,i}\left(\left(\mathbf{z} \circ G_{\text{temp}}(\mathbf{z}_t)^{(t)}\right) \circ \left(\mathbf{z}_i \circ G_{\text{temp},i}(\mathbf{z}_{t,i})^{(t)}\right)\right). \tag{6}$$

Lastly, we use the tensor $\{G_i\left(\bar{\mathbf{z}}\right)^{(t)}\}_{i,t=1}^{M,T}$ as the artificial data $\tilde{\mathbf{x}}$ to train the discriminator D along with real data $\mathbf{x}$, based on the WGAN-GP (Gulrajani et al. 2017).

For the track-conditional scenario, an additional encoder E is responsible for extracting useful inter-track features from the user-provided track. This can be done analogously as T2 so we omit the details due to space limitation.

Implementation

Dataset and Preprocessing

We employed the *matched* subset of the Lakh MIDI dataset (LMD)[1] (Raffel 2016) as the training dataset, for it provides a rich collection of real-world MIDI files and some associated meta-data. We used the python library pretty_midi (Raffel and Ellis 2014) to extract the piano-rolls from the files. To our best knowledge, this work may represent the first one that uses this dataset for music generation. We refer readers to (Raffel and Ellis 2016) for an introduction of the MIDI format and some noted issues in LMD.

According to our experience, using LMD for music generation has a few pitfalls, mostly because of the mapping between tracks and MIDI channels. As these MIDI files are often user-generated, the mapping is error-prone (Raffel and Ellis 2016). We performed three steps of data cleansing.

First, because there is no clear way to identify which track plays the melody and which plays the chords (Raffel and Ellis 2016), we cannot categorize the piano-rolls into melody, rhythm and drum tracks as some previous work did (Chu, Urtasun, and Fidler 2016) and (Yang, Chou, and Yang 2017). We instead divided each MIDI file into five tracks: bass (B), drum (D), guitar (G), piano (P) and strings (S). Moreover, we only picked songs that are tagged as 'Rock' songs. Although the genre tags are also error-prone, doing so reduces the number of empty bars in these tracks.

Second, even by filtering the songs with genre tag, some of the tracks still tend to play only a few notes in the entire songs. This increases data sparsity and impedes the learning process. We dealt with such a data imbalance issue by merging similar instruments into one of the aforementioned tracks, by summing the corresponding piano-rolls. For instruments that are hard to be categorized, we simply considered them as part of the strings, for strings represents the largest minority. Doing so introduces noises to our data, but empirically we found it better than having empty bars.

[1] http://colinraffel.com/projects/lmd/

		empty bar (EB; %)				used pitch classes (UPC)				qualified note (QN; %)				note in scale (NIS; %)			
		B	G	S	P	B	G	S	P	B	G	S	P	B	G	S	P
training data		8.06	19.4	10.1	24.8	1.71	3.08	3.38	3.28	90.0	81.9	89.6	88.4	47.2	67.8	62.0	65.0
from scratch	jam	6.59	18.3	6.10	22.6	1.53	3.69	4.09	4.13	71.5	56.6	63.1	62.2	38.7	67.2	61.2	64.8
	comp	0.01	1.34	0.01	0.02	2.51	4.20	5.19	4.89	49.5	47.4	52.5	49.9	45.0	41.4	53.9	64.0
	hybrid	2.14	11.7	6.04	17.8	2.35	4.76	5.24	5.45	44.6	43.2	52.0	45.5	43.3	70.8	61.0	64.5
	ablated	92.4	12.5	0.00	0.68	1.00	2.88	4.72	2.32	0.00	22.8	26.2	31.1	0.00	87.9	59.3	94.9
track conditional	jam	4.60	13.3	3.44	—	2.05	3.79	4.23	—	73.9	58.8	62.3	—	43.9	68.2	64.3	—
	comp	0.65	1.97	1.49	—	2.51	4.57	5.10	—	53.5	48.4	59.0	—	53.7	67.2	60.5	—
	hybrid	2.09	10.3	4.05	—	2.86	4.43	4.32	—	43.3	55.6	67.1	—	43.5	61.2	63.8	—

Table 1: Intra-track evaluation for bass (B), guitar (G), strings (S) and piano (P) (values closer to the first row are better)

Finally, we discarded songs that are not in C key or do not use four-beat time signatures. For each bar, we set the width (time resolution) to 96 for modeling common temporal patterns such as triplets, 16th notes and (the minimal one) 32th notes. We set the height to 84 to cover pitches from C1 to C8. The size of a data tensor per bar is hence 96 (time step) $\times$ 84 (note) $\times$ 5 (track). We got totally 127,731 bars for training.

Training Data for the Temporal Model

We need to combine the bars into phrases to train our temporal model. However, this cannot be done arbitrarily, to ensure that the phrases are musically meaningful. Therefore, we instead obtained phrases by using a state-of-the-art music segmentation algorithm called the "structural features" (Serrà et al. 2012), available in the MSAF toolbox[2] (Nieto and Bello 2016) to segment the MIDI files of LMD. The segmentation worked quite well. We considered four bars as a phrase, and accordingly pruned longer segments into proper size. Finally, we got 50,266 phrases in total.

Model Implementation Details

We refer readers to the appendix for details of the network architecture. We only note here a few important settings:

- As suggested by (Arjovsky, Chintala, and Bottou 2017), we found it beneficial to use batch normalization (BN) only for G. Moreover, we update D more times than G.
- Similar to MidiNet (Yang, Chou, and Yang 2017), in our G, the first few layers learn to grow the temporal dimension, whereas the latter layers grow the notes.
- By the end of G we used tanh as the activation function, and then binarized the result by setting the threshold to 0.

The training time for each model is less than 24 hours with a Tesla K40m GPU.

Objective Metrics for Evaluation

To evaluate our models, we design several metrics that can be calculated from both the training (real) data and the generated data. By comparing the values computed from the real and generated data, we can get an idea of how good our generator is. This concept is the same as the one in GANs—the distributions of real and fake data should be close. We have four intra-track and one (the last one) inter-track metrics:

- **EB**: ratio of empty bars (in %).
- **UPC**: number of used pitch classes per bar (from 1 to 12).
- **QN**: ratio of "qualified" notes (in %). If the length of one notes is less than three time steps (i.e. a 32th note), we consider it as noise, or an unqualified note. QN shows whether the generated music is overly fragmented.
- **NIS**: ratio of notes in the C scale (in %).
- **TD**: or tonal distance (Harte, Sandler, and Gasser 2006). It measures the hamornicity between a pair of tracks. Larger TD implies weaker harmonic inter-track relations.

Analysis of Training Data

We apply these metrics to the training data to gain a greater understanding of the LMD. The result is shown in the first rows of Tables 1 and 2. From **UPC**, we find that bass tends to play the melody, and guitar, strings and piano tend to play the chords (UPC around 3). The values of **TD** is around 1.50 when measuring the distance between melody-like track (i.e. bass) and chord-like track (e.g. piano), and around 1.00 for two chord-like tracks. If we shuffle the train data by pairing five tracks of one bar randomly, TD will slightly increase by 0.10. The other metrics reveal how noisy the data is. For example, because we require our data in C key, it is unexpected to see that the **NIS** is only around 40–70%. This suggests that the key information in LMD is also error-prone, which also increases the difficulty of our work.

Experiments

Generation From Scratch

Training We firstly study the training process of the composer model for generation from scratch, to gain insights. Figure 7(a) shows the training loss of D (solid line) as a function of training steps. We see that it decreases rapidly in the beginning and then saturates. However, there is a mild growing trend after the point 'B' marked on the graph, suggesting that G starts to learn something after that. We show in Figure 6 the generated piano-rolls at the five points marked on Figure 7(a). We see an interesting evolution of the generated piano-rolls. For example, G seems to learn the proper interval and number of pitches per track quite early. At 'B,'

[2] https://github.com/urinieto/msaf

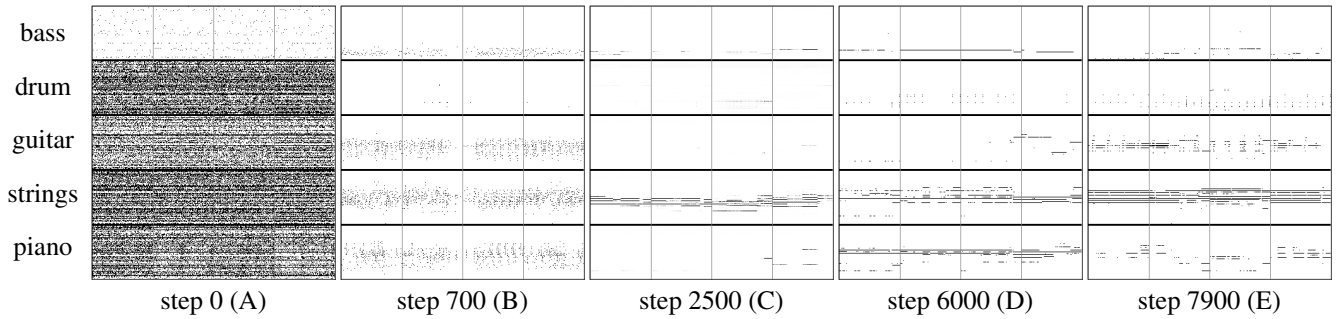


Figure 6: Evolution of the generated piano-rolls as a function of update steps, for the composer model, generating from scratch.

		tonal distance (TD)					
		B-G	B-S	B-P	G-S	G-P	S-P
training data		1.59	1.59	1.56	1.14	1.12	1.13
from scratch	jam	1.56	1.60	1.54	1.05	0.99	1.05
	comp	1.37	1.36	1.30	0.95	0.98	0.91
	hybrid	1.34	1.35	1.32	0.85	0.85	0.83

Table 2: Inter-track evaluation (smaller values are better)

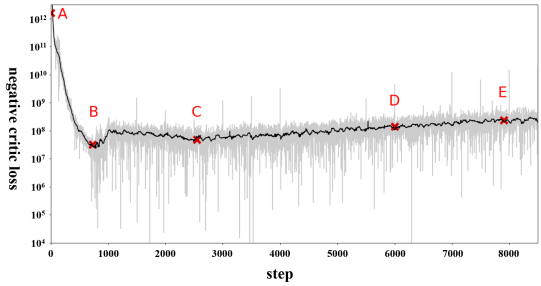
we can already see cluster of points gathering at the bottom of the piano-roll for bass. For guitar, strings and piano, they also learn the duration of the notes. These results show that G indeed becomes better along with the training process.

We also show the values of two objective metrics in Figure 7. From subfigure (b) we see that G can ultimately learn the proper number of pitch classes; and from (c) we see that QN remains lower than that of the training data even at the end, suggesting room for further improving our G. These show that a researcher can employ these metrics to study the generated result, before launching a subjective test.

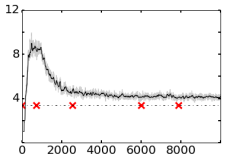
Testing After training, we use our models to generate 20,000 bars and evaluate their quality in terms of the objective metrics. The result is shown in the middle rows of Tables 1 and 2. For comparison, we also show in Table 1 the result of an **ablated** version of the composer model, which does not use BN at all. This ablated model barely learns anything, so its values can be taken as a reference.

For the intra-track metrics, we see that the jamming model tends to perform the best. This is possibly because it is originally designed to focus on each individual tracks. From **UPC**, we see that all our models tend to produce more than four notes at the same time, suggesting that some noise might have been produced. Another important observation is that all the three models obtain fairly low **QN**, compared to that of the training set. This suggests that the generated music still contains too many fragmented notes. This is related to the way we binarize continuous-valued output of G to create the sequence of discrete notes. We do not have a smart solution yet and leave this as a future work.

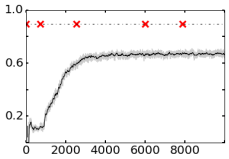
For the inter-track metric **TD** (Table 2), on the other hand, we see that the music generated by the jamming model has weaker harmonic relation cross tracks. The TD for the composer and hybrid models are much lower, suggesting that



(a) Training loss of the discriminator



(b) UPC of strings



(c) QN of strings

Figure 7: (a) Training loss of the discriminator, (b) UPC and (c) QN for the strings track, for the composer model, generating from scratch. The gray and black curves are the raw values and the smoothed ones, respectively. The dashed lines in (b), (c) indicate the value calculated from the training set.

these two may be more appropriate for multi-track generation. Moreover, we see that these two models perform similarly across different combinations of tracks. This is encouraging, because we know that the hybrid model may not have traded performance for its flexibility.

Example Result Figure 8 shows the piano-rolls of six phrases generated by the hybrid model. Some observations:

- The bass usually plays the lowest pitches and it is mostly monophonic (i.e. playing the melody).
- The drum often has clear 8- or 16-beat rhythmic patterns.
- The other three tracks tend to play the chords, and their pitches sometimes overlap (creating the black lines), indicating nice harmonic relations.

More examples of the generated piano-rolls and MIDIs can be found in the appendix and our project webpage—
`https://salu133445.github.io/musegan/`.

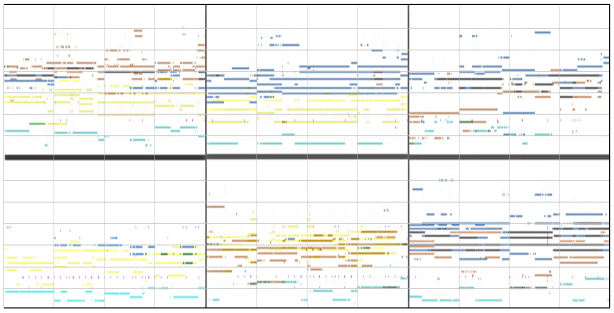


Figure 8: Example result of the hybrid model, by generation from scratch (best viewed in color—**cyan**: bass, **purple**: drum, **yellow**: guitar, **blue**: strings, **orange**: piano)

from scratch		**H**	**R**	**MS**	**C**	**OR**
non-pro	jam	2.83	3.29	2.88	2.84	2.88
	comp	3.12	**3.36**	2.95	3.13	3.12
	hybrid	**3.15**	3.33	**3.09**	**3.30**	**3.16**
pro	jam	2.31	3.05	2.48	2.49	2.42
	comp	2.66	3.13	2.68	2.63	2.73
	hybrid	**2.92**	**3.25**	**2.81**	**3.00**	**2.93**
track-conditional		**H**	**R**	**MS**	**C**	**OR**
non-pro	jam	**2.89**	**3.44**	2.97	**3.01**	**3.06**
	comp	2.70	3.29	**2.98**	2.97	2.86
	hybrid	2.78	3.34	2.93	2.98	3.01
pro	jam	2.44	**3.32**	2.67	2.72	2.69
	comp	2.35	3.21	2.59	2.67	2.62
	hybrid	**2.49**	3.29	**2.71**	**2.73**	**2.70**

Table 3: Result of user study (**H**: harmonious, **R**: rhythmic, **MS**: musically structured, **C**: coherent, **OR**: overall rating)

Track-conditional Generation

We use the piano tracks from the training data as the condition to generate the other four tracks per song. In terms of either the intra-track (shown in Table 1) or the inter-track (Table 5) metrics, we do not see much difference between the result of track-conditional generation and generation from scratch. This suggests that the two temporal models are comparable. The only notable difference lies in the result of **NIS**: the provided piano track seems to inform well (via the encoder) the intended scale information to the other tracks and improves the NIS for all the three models.

User Study

Finally, we conducted a listening test. 144 subjects were recruited from the Internet via our social circles. 44 of them are deemed 'pro user,' according to a simple questionnaire probing their musical background. Each subject had to listen to nine music clips generated by our models in random order, after being quantized by sixteenth notes. Each clip consists of three four-bar phrases. The subject rated the phrases in terms of whether they 1) have pleasant harmony, 2) have unified rhythm, 3) have clear musical structure, 4) are coherent, and 5) the overall rating, in Likert scale from 1 to 5. Result shown in Table 3 leads to the following observations:

- Our model performs slightly better in 'from scratch' than in 'track-conditional.' The scores fall within 2.30 to 3.44, which is not bad but suggests room for improvement.
- We receive higher scores in '**R**'hythmic, showing that we learn better in generating drums.
- For pro users, the ordering of the three models is usually jammer$\prec$composer$\prec$hybrid. The hybrid model is also preferred by non-pros for generation from scratch.

Related Work

Video Generation Similar to music generation, video generation also requires a temporal model. VGAN (Vondrick, Pirsiavash, and Torralba 2016) applied 3D convolutions and decomposed a video frame into background and foreground to facilitate modeling the dynamic and static scenes respectively. TGAN (Saito and Matsumoto 2016) used two CNN-based generators, one for generating images and the other for generating a set of latents to account for the scene dynamics. The second generator in TGAN can be seen as a temporal model. By concatenating prior random noise with the generated latents as input to the generator, it also can produce convincing results. More recently, MocoGAN (Tulyakov et al. 2017) further extended this concept and used an RNN for the temporal model. Despite of the differences between video and music generation, our model design is inspired by these prior arts. We are interested in testing our model on video generation in future work.

Music Generation As reviewed by (Briot, Hadjeres, and Pachet 2017), a surging number of neural networks have been proposed lately for music generation. Many are based on RNNs, such as DeepBach (Hadjeres and Pachet 2016), MelodyRNN (Waite et al. 2016), folk-rnn (Sturm et al. 2016) and Song_from_PI (Chu, Urtasun, and Fidler 2016). DeepBach can generate four-track music, but only in the specific style of J. S. Bach. Song_from_PI used a hierarchical RNN model to generate multi-track music involving melody, chord and percussion, but it requires human input to provide the temporal model. WaveNet, which directly generated raw waveforms, is a CNN-based model. However, audio-domain generation is fairly different from symbolic-domain generation. Some recent work have started to explore using GANs for generating music, such as CRNN-GAN (Mogren 2016) and MidiNet (Yang, Chou, and Yang 2017). Yet, these two prior arts only generated a single melody track.

Conclusion

In this work, we have presented a novel convolutional GAN model for multi-track sequence generation. We have also implemented such a model for generating MIDIs of pop/rock music using a large-scale yet noisy dataset. In addition, we showed that we can gain insights into the model learning process via a few objective metrics. The model we trained can start to learn something about music, but musically and aesthetically it is still far behind the level of human musicians. However, the model still has a few desirable properties, and we hope follow-up research can further improve it.

References

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*.

Blaauw, M., and Bonada, J. 2017. A neural parametric singing synthesizer. *arXiv preprint arXiv:1704.03809*.

Bretan, M.; Weinberg, G.; and Heck, L. 2016. A unit selection methodology for music generation using deep neural networks. *arXiv preprint arXiv:1612.03789*.

Briot, J.-P.; Hadjeres, G.; and Pachet, F. 2017. Deep learning techniques for music generation: A survey. *arXiv:1709.01620*.

Chu, H.; Urtasun, R.; and Fidler, S. 2016. Song from PI: A musically plausible network for pop music generation. *arXiv preprint arXiv:1611.03477*.

Gehring, J.; Auli, M.; Grangier, D.; Yarats, D.; and Dauphin, Y. N. 2017. Convolutional sequence to sequence learning. *arXiv/1705.03122*.

Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *Proc. Advances in Neural Information Processing Systems*, 2672–2680.

Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; and Courville, A. 2017. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*.

Hadjeres, G., and Pachet, F. 2016. DeepBach: A steerable model for Bach chorales generation. *arXiv preprint arXiv:1612.01010*.

Harte, C.; Sandler, M.; and Gasser, M. 2006. Detecting harmonic change in musical audio. In *Proc. Int. Soc. Music Information Retrieval Conf.*

Herremans, D., and Chew, E. 2017. MorpheuS: generating structured music with constrained patterns and tension. *IEEE Trans. Affective Computing* PP(99):1–1.

Jaques, N.; Gu, S.; Turner, R. E.; and Eck, D. 2016. Tuning recurrent neural networks with reinforcement learning. *arXiv preprint arXiv:1611.02796*.

Jeppesen, K. 2013. *Counterpoint: The polyphonic vocal style of the sixteenth century*. Courier Corporation.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Mogren, O. 2016. C-RNN-GAN: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*.

Nieto, O., and Bello, J. P. 2016. Systematic exploration of computational music structure research. In *Proc. Int. Soc. Music Information Retrieval Conf.*, 547–553.

Radford, A.; Metz, L.; and Chintala, S. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Raffel, C., and Ellis, D. P. W. 2014. Intuitive analysis, creation and manipulation of MIDI data with pretty_midi. In *Proc. Int. Soc. Music Information Retrieval Conf., Late Breaking and Demo Papers*.

Raffel, C., and Ellis, D. P. W. 2016. Extracting ground truth information from MIDI files: A MIDIfesto. In *Proc. Int. Soc. Music Information Retrieval Conf.*

Raffel, C. 2016. *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. Ph.D. Dissertation, Columbia University.

Raphael, C. 2010. Music plus one and machine learning. In *Proc. Int. Conf. Machine Learning*, 21–28.

Roberts, A.; Engel, J.; Hawthorne, C.; Simon, I.; Waite, E.; Oore, S.; Jaques, N.; Resnick, C.; and Eck, D. 2016. Interactive musical improvisation with Magenta. In *Proc. Neural Information Processing Systems*.

Ronneberger, O.; Fischer, P.; and Brox, T. 2015. U-net: Convolutional networks for biomedical image segmentation. *arXiv:1505.04597*.

Saito, M., and Matsumoto, E. 2016. Temporal generative adversarial nets. *arXiv preprint arXiv:1611.06624*.

Serrà, J.; Mller, M.; Grosche, P.; and Arcos, J. L. 2012. Unsupervised detection of music boundaries by time series structure features. *Proc. AAAI Conference on Artificial Intelligence*.

Sturm, B. L.; Santos, J. F.; Ben-Tal, O.; and Korshunova, I. 2016. Music transcription modelling and composition using deep learning. *arXiv preprint arXiv:1604.08723*.

Tulyakov, S.; Liu, M.; Yang, X.; and Kautz, J. 2017. MoCoGAN: Decomposing motion and content for video generation. *arXiv preprint arXiv:1707.04993*.

Tymoczko, D. 2010. *A geometry of music: Harmony and counterpoint in the extended common practice*. Oxford University Press.

van den Oord, A.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; and Kavukcuoglu, K. 2016. WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Vondrick, C.; Pirsiavash, H.; and Torralba, A. 2016. Generating videos with scene dynamics. In *Proc. Neural Information Processing Systems*.

Waite, E.; Eck, D.; Roberts, A.; and Abolafia, D. 2016. Project Magenta: Generating long-term structure in songs and stories. `https://magenta.tensorflow.org/blog/2016/07/15/lookback-rnn-attention-rnn/`.

Yang, L.-C.; Chou, S.-Y.; and Yang, Y.-H. 2017. MidiNet: A convolutional generative adversarial network for symbolic-domain music generation. In *Proc. Int. Soc. Music Information Retrieval Conf.*

Yu, L.; Zhang, W.; Wang, J.; and Yu, Y. 2017. SeqGAN: Sequence generative adversarial nets with policy gradient. *Proc. AAAI Conf. Artificial Intelligence*.

Appendix A Model Implementation Details

The network architecture of the proposed model is tabulated in Table 4. Some details can be found below.

Latent vector The input latent vector of G_{temp} has the same length as its output latent vectors. The length of the input latent vector of G_{bar} is fixed to 128. In MuseGAN, $\mathbf{z}$ is divided into four parts, each having length 32.

Generator G consists of four 1-D transposed convolutional layers along time (to grow the temporal dimension) and two 1-D transposed convolution layers along the pitch axis (to grow the notes) successively. Following (Gulrajani et al. 2017), we use batch normalization (BN) only in G.

Discriminator D consists of five 1-D convolutional layers and one fully-connected layer. For LReLU (leakly rectified linear unit), the negative slope is set to 0.2.

Encoder E has a reverse architecture as the generator. To compress the representation of inter-track interdependency, we constrain the number of filters of each layer to 16. To speed up the training of the network, we also apply *skip connections* between hidden layers of E and G, as that proposed in U-Net (Ronneberger, Fischer, and Brox 2015).

Training For the generation from scratch model (T1), the training procedure can be represented as:

$$\theta_D = \theta_D + \lambda \nabla_{\theta_D} L_D \,, \theta_G = \theta_G + \lambda \nabla_{\theta_D} L_G \,, \quad (7)$$

where θ_D and θ_G denote the parameters of discriminators and generators, respectively, and the λ is the learning rate. Following (Arjovsky, Chintala, and Bottou 2017), we update our discriminator more times than generator. For the track-conditional generation model (T2), we update the encoder in a way similar to the generators:

$$\theta_E = \theta_E + \lambda \nabla_{\theta_E} L_G \,, \quad (8)$$

where θ_E is the parameters of the encoder. Moreover, we use Adam (Kingma and Ba 2014) optimizer with $\alpha = 0.001$, $\beta_1 = 0.5$, $\beta_2 = 0.9$.

Post-processing Before feeding to a MIDI synthesizer, we reduce the temporal resolution of the generated piano-rolls by quantizing at 16 beats to avoid fragmented notes.

Appendix B Supplementary Result

Table 5 shows the result of inter-track evaluation for track-conditional generation, which is not shown as part of Table 2 in the paper due to space limit.

Appendix C Samples of Generation Result

We provide some randomly-chosen piano-rolls of generation result of our models. Some audio files (MIDIs) can be found at `https://salu133445.github.io/musegan/`.

- Figures 9 and 10 are results of the composer and the hybrid models, respectively, by generating from scratch.
- Figures 11 shows the track-conditional generation results of the composer model. Different from what described in the paper, we take strings track as conditions here to show the flexibility of our models.
- Figure 12 provides samples of the training data.

Input: $\mathbf{z} \in \Re^{32}$					
reshaped to (1)×32 channels					
transconv	1024	2	2	BN	ReLU
transconv	K_{temp}	3	1	BN	ReLU
Output: $G_{\text{temp}}(\mathbf{z}) \in \Re^{32 \times K_{\text{temp}}}$ (K_{temp}-track latent vector)					

(a) the temporal generator G_{temp}

Input: $\mathbf{z} \in \Re^{128}$					
reshaped to (1, 1)×128 channels					
transconv	1024	2×1	(2, 1)	BN	ReLU
transconv	256	2×1	(2, 1)	BN	ReLU
transconv	256	2×1	(2, 1)	BN	ReLU
transconv	256	2×1	(2, 1)	BN	ReLU
transconv	128	3×1	(3, 1)	BN	ReLU
transconv	64	1×7	(1, 7)	BN	ReLU
transconv	K_{bar}	1×12	(1, 12)	BN	tanh
Output: $G_{\text{bar}}(\mathbf{z}) \in \Re^{96 \times 84 \times K_{\text{bar}}}$ (K_{bar}-track piano-roll)					

(b) the bar generator G_{bar}

Input: $\widetilde{\mathbf{x}} \in \Re^{T \times 96 \times 84 \times M}$ (multi-track piano-roll)				
reshaped to (T, 96, 84) $\times M$ channels				
conv	128	$2 \times 1 \times 1$	(1, 1, 1)	LReLU
conv	128	$3 \times 1 \times 1$	(1, 1, 1)	LReLU
conv	128	$1 \times 1 \times 12$	(1, 1, 12)	LReLU
conv	128	$1 \times 1 \times 7$	(1, 1, 7)	LReLU
conv	128	$1 \times 2 \times 1$	(1, 2, 1)	LReLU
conv	128	$1 \times 2 \times 1$	(1, 2, 1)	LReLU
conv	256	$1 \times 4 \times 1$	(1, 2, 1)	LReLU
conv	512	$1 \times 3 \times 1$	(1, 2, 1)	LReLU
fully-connected	1024			LReLU
fully-connected	1			
Output: $D(\widetilde{\mathbf{x}}) \in \Re$				

(c) the discriminator D

Input: $\overrightarrow{\mathbf{y}} \in \Re^{96 \times 84}$ (piano-roll)					
conv	16	1×12	(1, 12)	BN	LReLU
conv	16	1×7	(1, 7)	BN	LReLU
conv	16	3×1	(3, 1)	BN	LReLU
conv	16	2×1	(2, 1)	BN	LReLU
conv	16	2×1	(2, 1)	BN	LReLU
conv	16	2×1	(2, 1)	BN	LReLU
Output: $E(\overrightarrow{\mathbf{y}}) \in \Re^{16}$					

(d) the encoder E

Table 4: Architecture of the (a) temporal generator, (b) bar generator, (c) discriminator and (d) encoder. For convolutional (conv) and transposed conv (transconv) layers, the values represent (from left to right): number of filters, kernel size, strides, batch normalization (BN), and activation function. $(K_{\text{temp}}, K_{\text{bar}}) = (1, 1), (1, M), (M, 1)$ for the jamming model, composer model, and hybrid model, respectively.

		tonal distance (TD)					
		B-G	B-S	B-P	G-S	G-P	S-P
track condi-tional	jam	1.51	1.53	1.50	1.04	0.95	1.00
	comp	1.41	1.36	1.40	0.96	1.01	0.95
	hybrid	1.39	1.36	1.38	0.96	0.94	0.95

Table 5: Inter-track evaluation (smaller values are better)

Figure 9: Randomly-picked generation result (piano-rolls) of the composer model, generating from scratch (best viewed in color—**cyan**: bass, **purple**: drum, **yellow**: guitar, **blue**: strings, **orange**: piano)

Figure 10: Randomly-picked generation result (piano-rolls) of the hybrid model, generating from scratch (best viewed in color—**cyan**: bass, **purple**: drum, **yellow**: guitar, **blue**: strings, **orange**: piano)

Figure 11: Randomly-picked generation result (piano-rolls) of the composer model with the strings track as the conditions (best viewed in color—**cyan**: bass, **purple**: drum, **yellow**: guitar, **blue**: strings(conditions), **orange**: piano)

Figure 12: Randomly-picked samples (piano-rolls) of the training data (best viewed in color—**cyan**: bass, **purple**: drum, **yellow**: guitar, **blue**: strings, **orange**: piano)