MIDINET: A CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORK FOR SYMBOLIC-DOMAIN MUSIC GENERATION

Li-Chia Yang, Szu-Yu Chou, Yi-Hsuan Yang
Research Center for IT innovation, Academia Sinica, Taipei, Taiwan
{richard40148, fearofchou, yang}@citi.sinica.edu.tw

ABSTRACT

Most existing neural network models for music generation use recurrent neural networks. However, the recent WaveNet model proposed by DeepMind shows that convolutional neural networks (CNNs) can also generate realistic musical waveforms in the audio domain. Following this light, we investigate using CNNs for generating melody (a series of MIDI notes) one bar after another in the symbolic domain. In addition to the generator, we use a discriminator to learn the distributions of melodies, making it a generative adversarial network (GAN). Moreover, we propose a novel conditional mechanism to exploit available prior knowledge, so that the model can generate melodies either from scratch, by following a chord sequence, or by conditioning on the melody of previous bars (e.g. a priming melody), among other possibilities. The resulting model, named MidiNet, can be expanded to generate music with multiple MIDI channels (i.e. tracks). We conduct a user study to compare the melody of eight-bar long generated by MidiNet and by Google's MelodyRNN models, each time using the same priming melody. Result shows that MidiNet performs comparably with MelodyRNN models in being realistic and pleasant to listen to, yet MidiNet's melodies are reported to be much more interesting.

1. INTRODUCTION

Algorithmic composition is not a new idea. The first computational model for algorithmic composition dates back to 1959 [16], according to the survey of Papadopoulos and Wiggins [23]. People have also used (shallow) neural networks for music generation since 1989 [30]. It was, however, only until recent years when deep neural networks demonstrated their ability in learning from big data collections that generating music by neural networks became a trending topic. Lots of deep neural network models for music generation have been proposed just over the past two years [4, 7, 10, 15, 18, 19, 21, 22, 26, 28, 31, 33].

The majority of existing neural network models for music generation use recurrent neural networks (RNNs) and their variants, presumably for music generation is inherently about generating sequences [2, 3, 9, 14]. These models differ in the model assumptions and the way musical events are represented and predicted, but they all use information from the previous events to condition the generation of the present one. Famous examples include the MelodyRNN models [33] for *symbolic-domain* generation (i.e. generating MIDIs) and the SampleRNN model [19] for *audio-domain* generation (i.e. generating WAVs).

Relatively fewer attempts have been made to use deep convolutional neural networks (CNNs) for music generation. A notable exception is the WaveNet model [31] proposed recently for audio-domain generation. It generates one audio sample at a time, with the predictive distribution for each sample conditioned on previous samples through dilated causal convolutions [31]. WaveNet shows it possible for CNNs to generate realistic music. This is encouraging, as CNNs are typically faster to train and more easily parallelizable than RNNs [32].

Following this light, we investigate in this paper a novel CNN-based model for symbolic-domain generation, focusing on melody generation. [1] Instead of creating a melody sequence continuously, we propose to generate melodies *one bar (measure) after another*, in a successive manner. This allows us to employ convolutions on a 2-D matrix representing the presence of notes over different time steps in a bar. We can have such a score-like representation for each bar for either a real or a generated MIDI.

Moreover, to emulate *creativity* [23] and encourage diverse generation result, we use random noises as input to our *generator* CNN. The goal of the generator is to transform random noises into the aforementioned 2-D score-like representation, that "appears" to be from real MIDI. This transformation is achieved by a special convolution operator called transposed convolution [8]. Meanwhile, we learn a *discriminator* CNN that takes as input a 2-D score-like representation and predicts whether it is from a real or a generated MIDI, thereby informing the generator how to appear to be real. This amounts to a generative adversarial network (GAN) [11–13, 24, 27], which learns the generator and discriminator iteratively under the concept of minimax two-player game theory.

This GAN alone does not take into account the temporal dependencies across different bars. To address this issue, we propose a novel conditional mechanism to use

[1] In general, a melody may be defined as a succession of (monophonic) musical notes expressing a particular musical idea.

	MelodyRNN [33]	Song from PI [7]	DeepBach [15]	C-RNN-GAN [21]	MidiNet (this paper)	WaveNet [31]
core model	RNN	RNN	RNN	RNN	CNN	CNN
data type	symbolic	symbolic	symbolic	symbolic	symbolic	audio
genre specificity	—	—	Bach chorale	—	—	—
mandatory prior knowledge	priming melody	music scale & melody profile	—	—	—	priming wave
follow a priming melody	√	√			√	√
follow a chord sequence					√	
generate multi-track music		√	√		√	√
use GAN				√	√	
use versatile conditions					√	
open source code	√			√	√	

Table 1. Comparison between recent neural network based music generation models

music from the previous bars to condition the generation of the present bar. This is achieved by learning another CNN model, which we call the *conditioner* CNN, to incorporate information from previous bars to intermediate layers of the generator CNN. This way, our model can "look back" without a recurrent unit as used in RNNs. Like RNNs, our model can generate music of arbitrary number of bars.

Because we use random noises as inputs to our generator, our model can generate melodies *from scratch*, i.e. without any other prior information. However, due to the conditioner CNN, our model has the capacity to exploit whatever prior knowledge that is available and can be represented as a matrix. For example, our model can generate music by following a chord progression, or by following a few starting notes (i.e. a priming melody). Given the same priming melody, our model can generate different results each time, again due to the random input.

The proposed model can be extended to generate different types of music, by using different conditions. Based on an idea called *feature matching* [27], we propose a way to control the influence of such conditions on the generation result. We can then control, for example, how much the current bar should sound like the previous bars. Moreover, our CNNs can be easily extended to deal with tensors instead of matrices, to exploit multi-channel MIDIs and to generate music of multiple tracks or parts. We believe such a highly adaptive and generic model structure can be a useful alternative to RNN-based designs. We refer to this new model as the MidiNet.

In our experiment, we conduct a user study to compare the melodies generated by MidiNet and MelodyRNN models [33]. For fair comparison, we use the same priming melodies for them to generate melodies of eight-bar long (including the primers), without any other prior information. To demonstrate the flexibility of MidiNet, we provide the result of two additional settings: one uses additionally chord progressions of eight-bar long to condition the generation, and the other uses a slightly different network architecture to generate more creative music. For reproducibility, the source code and pre-trained models of MidiNet are released online [2] .

[2] https://github.com/RichardYang40148/MidiNet

2. RELATED WORK

A large number of deep neural network models have been proposed lately for music generation. This includes models for generating a melody sequence or audio waveforms by following a few priming notes [10,18,19,22,31,33], accompanying a melody sequence with music of other parts [15], or playing a duet with human [4, 26].

Table 1 compares MidiNet with a number of major related models. We briefly describe each of them below.

The MelodyRNN models [33] proposed by the Magenta Project from the Google Brain team are possibly among the most famous examples of symbolic-domain music generation by neural networks. In total three RNN-based models were proposed, including two variants that aim to learn longer-term structures, the lookback RNN and the attention RNN [33]. Source code and pre-trained models for the three models are all publicly available. [3] As the main function of MelodyRNN is to generate a melody sequence from a priming melody, we use the MelodyRNN models as the baseline in our evaluation.

Song from PI [7] is a hierarchical RNN model that uses a hierarchy of recurrent layers to generate not only the melody but also the drums and chords, leading to a multi-track pop song. This model nicely demonstrates the ability of RNNs in generating multiple sequences simultaneously. However, it requires prior knowledge of the musical scale and some profiles of the melody to be generated [7], which is not needed in many other models, including MidiNet.

DeepBach [15], proposed by Sony CSL, is specifically designed for composing polyphonic four-part chorale music in the style of J. S. Bach. It is an RNN-based model that allows enforcing user-defined constraints such as rhythm, notes, parts, chords and cadences.

C-RNN-GAN [21] is to date the only existing model that uses GAN for music generation, to our best knowledge. It also takes random noises as input as MidiNet does, to generate diverse melodies. However, it lacks a conditional mechanism [17, 20, 25] to generate music by following either a priming melody or a chord sequence.

[3] https://github.com/tensorflow/magenta/tree/master/magenta/models/melody_rnn (accessed 2017-4-26)

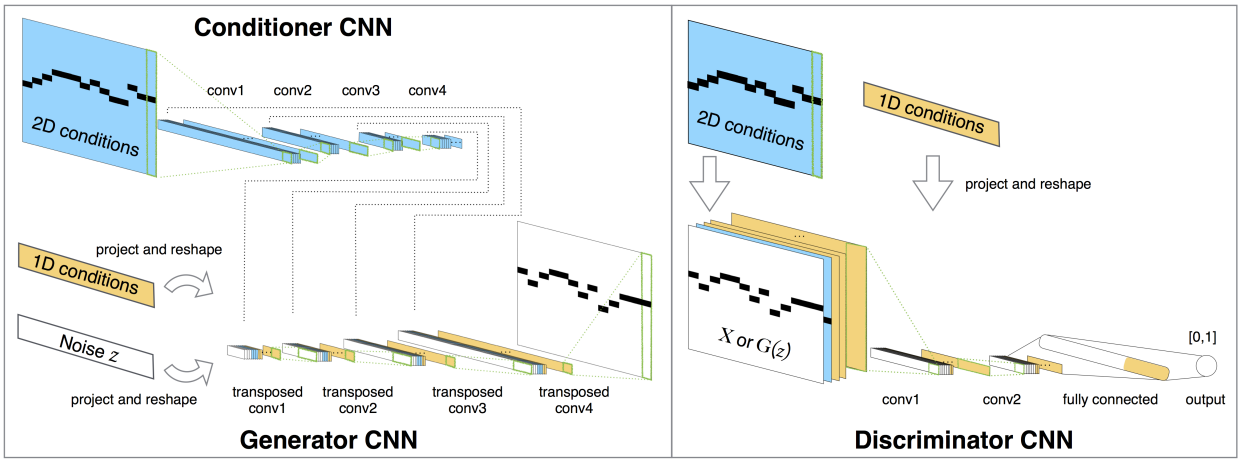

Figure 1. System diagram of the proposed MidiNet model for symbolic-domain music generation.

WaveNet [10, 31] is a CNN-based model proposed by DeepMind for creating raw waveforms of speech and music. The advantage of audio-domain generation is the possibility of creating new sounds, but we choose to focus on symbolic-domain generation in this paper.

3. METHODS

A system diagram of MidiNet is shown in Figure 1. Below, we present the technical details of each major component.

3.1 Symbolic Representation for Convolution

Our model uses a symbolic representation of music in fixed time length, by dividing a MIDI file into bars. The note events of a MIDI channel can be represented by an h-by-w real-valued matrix $\mathbf{X}$, where h denotes the number of MIDI notes we consider, possibly including one more dimension for representing silence, and w represents the number of time steps we use in a bar. For melody generation, there is at most one active note per time step. We use a binary matrix $\mathbf{X} \in \{0,1\}^{h \times w}$ if we omit the velocity (volume) of the note events. We use multiple matrices per bar if we want to generate multi-track music.

In this representation, we may not be able to easily distinguish between a long note and two short repeating notes (i.e. consecutive notes with the same pitch). Future extensions can be done to emphasize the note onsets.

3.2 Generator CNN and Discriminator CNN

The core of MidiNet is a modified deep convolutional generative adversarial network (DCGAN) [24], which aims at learning a discriminator D to distinguish between real (authentic) and generated (artificial) data, and a generator G that "fools" the discriminator. As typical in GANs, the input of G is a vector of random noises $\mathbf{z} \in \mathbb{R}^l$, whereas the output of G is an h-by-w matrix $\widehat{\mathbf{X}} = G(\mathbf{z})$ that "appears" to be real to D. GANs learn G and D by solving:

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

where $\mathbf{X} \sim p_{\text{data}}(\mathbf{X})$ denotes the operation of sampling from real data, and $\mathbf{z} \sim p_z(\mathbf{z})$ the sampling from a random distribution. As typical in GANs, we need to train G and D iteratively multiple times, to gradually make a better G.

Our discriminator is a typical CNN with a few convolution layers, followed by fully-connected layers. These layers are optimized with a cross-entropy loss function, such that the output of D is close to 1 for real data (i.e. $\mathbf{X}$) and 0 for those generated (i.e. $G(\mathbf{z})$). We use a sigmoid neuron at the output layer of D so its output is in [0,1].

The goal of the generator CNN, on the other hand, is to make the output of D close to 1 for the generated data. For generation, it has to transform a vector $\mathbf{z}$ into a matrix $\widehat{\mathbf{X}}$. This is achieved by using a few fully connected layers first, and then a few transposed convolution layers [8] that "upsamples" smaller vectors/matrices into larger ones.

Owing to the nature of minimax games, the training of GANs is subject to issues of instability and mode collapsing [12]. Among the various possible techniques to improve the training of GANs [1, 5, 27], we employ the so-called feature matching and one-sided label smoothing [27] in our model. The idea of feature matching is to add additional L2 regularizers to Eq. 1, such that the distributions of real and generated data are enforced to be close. Specifically, we add the following two terms when we learn G:

$$\lambda_1 \|\mathbb{E}\, \mathbf{X} - \mathbb{E}\, G(\mathbf{z})\|_2^2 + \lambda_2 \|\mathbb{E}\, f(\mathbf{X}) - \mathbb{E}\, f(G(\mathbf{z}))\|_2^2, \quad (2)$$

where f denotes the first convolution layer of the discriminator, and λ_1, λ_2 are parameters to be set empirically.

3.3 Conditioner CNN

In GAN-based image generation, people often use a vector to encode available prior knowledge that can be used to condition the generation. This is achieved by reshaping the vector and then adding it to different layers of G and D, to provide additional input [20]. Assuming that the conditioning vector has length n, to add it to an intermediate layer of shape a-by-b we can duplicate the values ab times

to get a tensor of shape a-by-b-by-n, and then concatenate it with the intermediate layer in the feature map axis. This is illustrated by the light orange blocks in Figure 1. We call such a conditional vector *1-D conditions*.

As the generation result of our GAN is an h-by-w matrix of notes and time steps, it is convenient if we can perform conditioning directly on each entry of the matrix. For example, the melody of a previous bar can be represented as another h-by-w matrix and used to condition the generation of the present bar. We can have multiple such matrices to learn from multiple previous bars. We can directly add such a conditional matrix to the input layer of D to influence all the subsequent layers. However, to exploit such *2-D conditions* in G, we need a mechanism to reshape the conditional matrix to smaller vectors of different shapes, to include them to different intermediate layers of G.

We propose to achieve this by using a conditioner CNN that can be viewed as a *reverse* of the generator CNN. As the blue blocks in Figure 1 illustrates, the conditioner CNN uses a few convolution layers to process the input h-by-w conditional matrix. The conditioner and generator CNNs use exactly the same filter shapes in their convolution layers, so that the outputs of their convolution layers have "compatible" shapes. In this way, we can concatenate the output of a convolution layer of the conditioner CNN to the input of a corresponding transposed convolution layer of the generator CNN, to influence the generation process. In the training stage, the conditioner and generator CNNs are trained simultaneously, by sharing the same gradients.

3.4 Tunning for Creativity

We propose two methods to control the trade-off between creativity and discipline of MidiNet. The first method is to manipulate the effect of the conditions by using them only in part of the intermediate transposed convolution layers of G, to give G more freedom from the imposed conditions. The second method capitalizes the effect of the feature matching technique [27]: we can increase the values of λ_1 and λ_2 to make the generated music sounds closer to existing music (i.e. those observed in the training set).

4. IMPLEMENTATION

4.1 Dataset

As the major task considered in this paper is melody generation, for training MidiNet we need a MIDI dataset that clearly specifies per file which channel corresponds to the melody. To this end, we crawled a collection of 1,022 MIDI tabs of pop music from TheoryTab, [4] which provides exactly two channels per tab, one for melody and the other for the underlying chord progression. With this dataset, we can implement at least two versions of MidiNets: one that learns from only the melody channel for fair comparison with MelodyRNN [33], which does not use chords, and the other that additionally uses chords to condition melody generation, to test the capacity of MidiNet.

[4] `https://www.hooktheory.com/theorytab`

	dimensions 1–12	13
major	C, C#, D, D#, E, F, F#, G, G#, A, A#, B	0
minor	A, A#, B, C, C#, D, D#, E, F, F#, G, G#	1

Table 2. 13-dimensional chord representation

For simplicity, we filtered out MIDI tabs that contain chords other than the 24 basic chord triads (12 major and 12 minor chords). Next, we segmented the remaining tabs every 8 bars, and then pre-processed the melody channel and the chord channel separately, as described below.

For melodies, we fixed the smallest note unit to be the sixteenth note, making $w = 16$. Specifically, we prolonged notes which have a pause note after them. If the first note of a bar is a pause, we extended the second note to have it played while the bar begins. There are other exceptions such as triplets and shorter notes (e.g. 32nd notes), but we chose to exclude them in this implementation. Moreover, for simplicity, we shifted all the melodies into two octaves, from `C4` to `B5`, and neglected the velocity of the note events. Although our melodies would use only 24 possible notes after these preprocessing steps, we considered all the 128 MIDI notes (i.e. from `C0` to `G10`) in our symbolic representation. In doing so, we can detect model collapsing [12] more easily, by checking whether the model generates notes outside these octaves. As there are no pauses in our data after preprocessing, we do not need a dimension for silence. Therefore, $h = 128$.

For chords, instead of using a 24-dimensional one-hot vector, we found it more efficient to use a chord representation that has only 13 dimensions— the first 12 dimensions for marking the key, and the last for the chord type (i.e. major or minor), as illustrated in Table 4.1. We pruned the chords such that there is only one chord per bar.

After these preprocessing steps, we were left with 526 MIDI tabs (i.e. 4,208 bars). [5] For data augmentation, we circularly shifted the melodies and chords to any of the 12 keys in equal temperament, leading to a final dataset of 50,496 bars of melody and chord pairs for training.

4.2 Network Specification

Our model was implemented in TensorFlow. For the generator, we used as input random vectors of white Gaussian noise of length $l = 100$. Each random vector go through two fully-connected layers, with 1024 and 512 neurons respectively, before being reshaped into a 1-by-2 matrix. We then used four transposed convolution layers: the first three use filters of shape 1-by-2 and two strides [8], and the last layer uses filters of shape 128-by-1 and one stride. Accordingly, our conditioner has four convolution layers, which use 128-by-1 filters for the first layer, and 1-by-2 filters for the other three. For creating a monophonic note sequence, we added a layer to the end of G to turn off per time step all but the note with the highest activation.

As typical in GANs, the discriminator is likely to overpower the generator, leading to the so-called vanishing gra-

[5] In contrast, MelodyRNN models [33] were trained on *thousands* of MIDI files, though the exact number is not yet disclosed.

dient problem [1,12]. We adopted two strategies to weaken the discriminator. First, in each iteration, we updated the generator and conditioner twice, but the discriminator only once. Second, we used only two convolution layers (14 filters of shape 128-by-2, two strides, and 77 filters of shape 1-by-4, two strides) and one fully-connected layer (1,024 neurons) for the discriminator.

We fine-tuned the other parameters of MidiNet and considered the following three variants in our experiment.

4.2.1 Model 1: Melody generator, no chord condition

This variant uses the melody of the previous bar to condition the generation of the present bar. We used this 2-D condition in all the four transposed convolution layers of G. We set the number of filters in all the four transposed convolution layers of G and the four convolution layers of the conditioner CNN to 256. The feature matching parameters λ_1 and λ_2 are set to 0.1 and 1, respectively. We did not use the 2-D condition for D, requiring it to distinguish between real and generated melodies from the present bar.

In the training stage, we firstly added one empty bar before all the MIDI tabs, and then randomly sampled two consecutive bars from any tab. We used the former bar as an instance of real data (i.e. X) and the input to D, and the former bar (which is a real melody or all zeros) as a 2-D condition and the input to the conditioner CNN Once the model was trained, we used G to generate melodies of 8-bar long in the following way: the first bar was composed of a real, priming melody sampled from our dataset; the generation of the second bar was made by G, conditioned by this real melody; starting from the third bar, G had to use the (artificial) melody it generated previously for the last bar as the 2-D condition. This process repeated until we had all the eight bars. [6]

4.2.2 Model 2: Melody generator with chord condition, stable mode

This variant additionally uses the chord channel. Because our MIDI tabs use one chord per bar, we used the chord (a 13-dimensional vector; see Table 4.1) of the present bar as a 1-D condition for generating the melody for the same bar. We can say that our model is generating a melody sequence that fits the given chord progression.

To highlight the chord condition, we used the 2-D previous-bar condition only in the last transposed convolution layer of G. In contrast, we used the 1-D chord condition in all the four transposed convolution layer of G, as well as the input layer for D. Moreover, we set $\lambda_1 = 0.01$, $\lambda_2 = 0.1$, and used 128 filters in the transposed convolution layers of G and only 16 filters in the convolution layers of the conditioner CNN. As a result, the melody generator is more *chord-dominant* and *stable*, for it would mostly follow the chord progression and seldom generate notes that violate the constraint imposed by the chords.

[6] It is also possible to use multiple previous bars to condition our generation, but we leave this as a future extension.

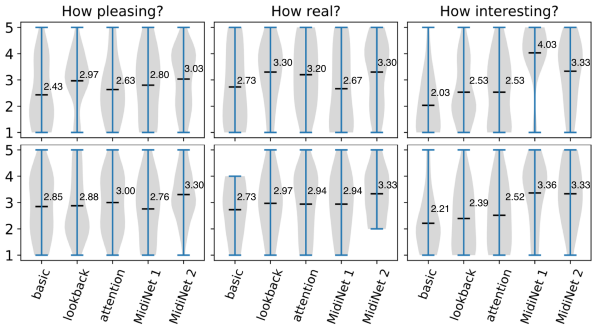

Figure 2. Result of a user study comparing MelodyRNN and MidiNet models, for people (top row) with musical backgrounds and (bottom) without musical backgrounds. The middle bars indicate the mean values. Please note that MidiNet Model 2 takes the chord condition as additional information.

4.2.3 Model 3: Melody generator with chord condition, creative mode

This variant realizes a slightly more creative melody generator by placing the 2-D condition in every transposed convolution layer of G. In this way, G would sometimes violate the constraint imposed by the chords, to somehow adhere to the melody of the previous bar. Such violations sometimes sound unpleasant, but can be sometimes creative. Unlike the previous two variants, we need to listen to several melodies generated by this model to handpick good ones. However, we believe such a model can still be useful for assisting and inspiring human composers.

5. EXPERIMENTAL RESULT

To evaluate the aesthetic quality of the generation result, a user study that involves human listeners is needed. We conducted a study with 21 participants. Ten of them understand basic music theory and have the experience of being an amateur musician, so we considered them as people with musical backgrounds, or *professionals* for short.

We compared MidiNet with three MelodyRNN models pre-trained and released by Google Magenta: the basic RNN, the lookback RNN, and the attention RNN [33]. We randomly picked 100 priming melodies from the training data [7] and asked the models create melodies of eight bars by following these primers. We considered two variants of MidiNet in the user study: model 1 (Section 4.2.1) for fair comparison with MelodyRNN, and model 2 (Section 4.2.2) for probing the effects of using chords. Although the result of model 2 was generated by additionally following the chords, we did not playback the chord channel in the user study.

We randomly selected the generation result of three out of the 100 priming melodies for each participant to listen to, leading to three sets of music. To avoid bias, we randomly shuffled the generation result by the five considered

[7] Even though these priming melodies are in the training data, MidiNet generates melodies that are quite different from the existing ones.

(a) MidiNet model 1

(b) MidiNet model 2

(c) MidiNet model 3

Figure 3. Example result of the melodies (of 8 bars) generated by different implementations of MidiNet.

models, such that in each set the ordering of the five models is different. The participants were asked to stay in a quiet room separately and used a headphone for music listening through the Internet, one set at a time. We told them that some of the music "might be" real, and some might be generated by machine, although all of them were actually automatically generated. They were asked to rate the generated melodies in terms of the following three metrics: *how pleasing*, *how real*, and *how interesting*, from 1 (low) to 5 (high) in a five-point Likert scale.

The result of the user study is shown in Figure 2 as violin plots, where the following observations can be made. First, among the MelodyRNN models, lookback RNN and attention RNN consistently outperform basic RNN across the three metrics and two user groups (i.e. people with and without musical backgrounds), which is expected according to the report of Magenta [33]. The mean values for lookback RNN are around 3 (medium) for being pleasant and realistic, and around 2.5 for being interesting.

Second, MidiNet model 1, which uses only the previous bar condition, obtains similar ratings as the MelodyRNN models in being pleasant and realistic. This is encouraging, as MelodyRNN models can virtually exploit all the previous bars in generation. This result demonstrates the effectiveness of the proposed conditioner CNN in learning temporal information. Furthermore, we note that the melodies generated by MidiNet model 1 were found much more interesting than those generated by MelodyRNN. The mean value in being interesting is around 4 for people with musical backgrounds, and 3.4 for people without musical backgrounds. The violin plot indicates that the ratings of the professionals are mostly larger than 3.

Third, MidiNet model 2, which further uses chords, obtains the highest mean ratings in being pleasant and realistic for both user groups. In terms of interestingness, it also outperforms the three MelodyRNN models, but is inferior to MidiNet model 1, especially for professionals.

According to the feedback from the professionals, a melody sounds artificial if it lacks variety or violates principals in (Western) music theory. The result of MidiNet model 1 can sound artificial, for it relies on only the previous bar and hence occasionally generates "unexpected" notes. In contrast, the chord channel provides a musical context that can be effectively used by MidiNet model 2 through the conditional mechanism. However, occasional violation of music theory might be a source of interestingness and thereby creativity. For example, the professionals reported that the melodies generated by MelodyRNN models are sometimes too repetitive, or "safe," making them artificial and less interesting. It might be possible to further fine tune our model to reach a better balance between being real and being interesting, but we believe our user study has shown the promise of MidiNet.

Figure 3 shows some melodies generated by different implementations of MidiNet, which may provide insights into MidiNet's performance. Figure 3(a) shows that MidiNet model 1 can effectively exploit the previous bar condition—most bars start with exactly the same first two notes (as the priming bar) and they use similar notes in between. Figure 3(b) shows the result of MidiNet model 2, which highlights the chord condition. Figure 3(c) shows that MidiNet can generate more creative result by making the chord condition and previous bar condition equally strong. We can see stronger connections between adjacent bars from the result of this MidiNet model 3. For more audio examples, please go to `https://soundcloud.com/vgtsv6jf5fwq/sets`.

6. CONCLUSION

We have presented MidiNet, a novel CNN-GAN based model for MIDI generation. It has a conditional mechanism to exploit versatile prior knowledge of music. It also has a flexible architecture and can generate different types of music depending on input and specifications. Our evaluation shows that it can be a powerful alternative to RNNs.

For future work, we would extend MidiNet to generate multi-track music, to include velocity and pauses by training the model by using richer and larger MIDI data. We are also interested in using ideas of reinforcement learning [29] to incorporate principles of music theory [18], and to take input from music information retrieval models such as genre recognition [6] and emotion recognition [34].

7. REFERENCES

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

[2] Jamshed J. Bharucha and Peter M. Todd. Modeling the perception of tonal structure with neural nets. *Computer Music Journal*, 13(4):44–53.

[3] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

[4] Mason Bretan, Gil Weinberg, and Larry Heck. A unit selection methodology for music generation using deep neural networks. *arXiv preprint arXiv:1612.03789*, 2016.

[5] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In *Proc. Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.

[6] Keunwoo Choi, George Fazekas, Mark B. Sandler, and Kyunghyun Cho. Convolutional recurrent neural networks for music classification. *arXiv preprint arXiv:1609.04243*, 2016.

[7] Hang Chu, Raquel Urtasun, and Sanja Fidler. Song from PI: A musically plausible network for pop music generation. *arXiv preprint arXiv:1611.03477*, 2016.

[8] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[9] Douglas Eck and Juergen Schmidhuber. Finding temporal structure in music: Blues improvisation with LSTM recurrent networks. In *Proc. IEEE Workshop on Neural Networks for Signal Processing*, pages 747–756, 2002.

[10] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with WaveNet autoencoders. *arXiv preprint arXiv:1704.01279*, 2017.

[11] Jon Gauthier. Conditional generative adversarial nets for convolutional face generation. *Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter semester*, 2014:5, 2014.

[12] Ian J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2017.

[13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proc. Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[14] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[15] Gaëtan Hadjeres and François Pachet. DeepBach: a steerable model for bach chorales generation. *arXiv preprint arXiv:1612.01010*, 2016.

[16] Lejaren Hiller and Leonard M. Isaacson. *Experimental Music: Composition with an Electronic Computer*. New York: McGraw-Hill, 1959.

[17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.

[18] Natasha Jaques, Shixiang Gu, Richard E. Turner, and Douglas Eck. Tuning recurrent neural networks with reinforcement learning. *arXiv preprint arXiv:1611.02796*, 2016.

[19] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron C. Courville, and Yoshua Bengio. SampleRNN: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*, 2016.

[20] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

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

[22] Tom Le Paine, Pooya Khorrami, Shiyu Chang, Yang Zhang, Prajit Ramachandran, Mark A. Hasegawa-Johnson, and Thomas S. Huang. Fast WaveNet generation algorithm. *arXiv preprint arXiv:1611.09482*, 2016.

[23] George Papadopoulos and Geraint Wiggins. AI methods for algorithmic composition: A survey, a critical view and future prospects. In *Proc. AISB Symposium on Musical Creativity*, pages 110–117, 1999.

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

[25] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

[26] Adam Roberts, Jesse Engel, Curtis Hawthorne, Ian Simon, Elliot Waite, Sageev Oore, Natasha Jaques, Cinjon Resnick, and Douglas Eck. Interactive musical improvisation with Magenta. In *Proc. Neural Information Processing Systems*, 2016.

[27] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *Proc. Advances in Neural Information Processing Systems*, pages 2226–2234, 2016.

[28] Zheng Sun, Jiaqi Liu, Zewang Zhang, Jingwen Chen, Zhao Huo, Ching Hua Lee, and Xiao Zhang. Composing music with grammar argumented neural networks and note-level encoding. *arXiv preprint arXiv:1611.05416*, 2016.

[29] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[30] Peter M. Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4):27–43.

[31] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[32] Aäron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelCNN decoders. In *Proc. Advances in Neural Information Processing Systems*, pages 4790–4798, 2016.

[33] Elliot Waite, Douglas Eck, Adam Roberts, and Dan Abolafia. Project Magenta: Generating long-term structure in songs and stories, 2016. `https://magenta.tensorflow.org/blog/2016/07/15/lookback-rnn-attention-rnn/`.

[34] Yi-Hsuan Yang and Homer H. Chen. *Music Emotion Recognition*. CRC Press, 2011.