JazzGAN: Improvising with Generative Adversarial Networks

Nicholas Trieu and **Robert M. Keller**
Harvey Mudd College
Claremont, California, USA
ntrieu@hmc.edu, keller@cs.hmc.edu

Abstract

For the purpose of creating a jazz teaching tool in the open-source Impro-Visor (Improvisation Advisor) application, we trained JazzGAN, a generative adversarial network (GAN) using recurrent neural networks (RNN) to improvise monophonic jazz melodies over chord progressions. Improvising jazz melodies creates several challenges not addressed by previous generative adversarial neural networks for music generation, including (1) frequent and diverse key changes; (2) unconventional and off-beat rhythms; (3) flexibility with off-chord notes. To address these issues, we compare the performance of several data representations with JazzGAN and propose the use of harmonic bricks for phrase segmentation. We define metrics to quantify the aforementioned issues, compare several data encodings of rhythm, and show that JazzGAN compares favorably against Magenta's ImprovRNN.

I. Introduction

Many deep neural network models for music generation have been proposed (Johnson, Keller, and Weintraut 2017; Bretan, Weinberg, and Heck 2017; Yang, Chou, and Yang 2017; Dong et al. 2018; Sturm, Santos, and Korshunova 2015; Yu et al. 2017; Mogren 2016). Of these models, the majority are based on Recurrent Neural Networks (RNN) with gating mechanisms such as Long Short-Term Memory (Hochreiter and Schmidhuber 1997).

While these models are powerful enough to represent universal Turing machines (Siegelmann and Sontag 1995), their power creates several shortcomings for creativity. (Zhang et al. 2017) shows the remarkable capacity of deep neural networks to memorize their corpus and easily fit to random noise. For the purposes of constructing a *creative* music generator, it is therefore important to understand how much the model generalizes beyond rote memorizing the training corpus.

In particular, it is not well-understood how different model structures and data representations determine the musical traits learned. In part, this is due to the difficulty of defining a concrete measure for musical quality. Most neural network models have relied primarily on user studies in lieu of a performance metric (Yang, Chou, and Yang 2017; Bretan, Weinberg, and Heck 2017). While informative to some extent, these studies lack exact analysis about the generated music. Furthermore, the time constraints of the studies limit the number of sampled generations, which may represent only a fraction of the model's desirable or undesirable capabilities.

Our work focuses on training and evaluating JazzGAN, a Generative Adversarial Network (GAN) that uses RNNs to improvise monophonic jazz melodies over chord progressions. Our corpus comes from the Impro-Visor (Keller 2018) collection of transcribed jazz solos. Improvising jazz melodies to a given chord progression creates several challenges not addressed by previous GAN models, including: (1) frequent and diverse key changes; (2) unconventional and off-beat rhythms; (3) flexibility with off-chord notes. To address these issues, we compare the performance of several data representations with JazzGAN and propose use of harmonic bricks for phrase segmentation. Our contributions are as follows:

1. We propose several metrics for evaluating musical features that especially pertain to jazz music.
2. We evaluate several data representations of rhythm under these metrics.
3. Using these representations and a novel musical phrasing method, we construct GAN-based models for monophonic sequential jazz generation.
4. With the proposed metrics and models, we compare the effect of different rhythm representations on model generations.
5. To validate our models, we show that their learned chord conformity compares favorably against a similar model (Google Magenta's ImprovRNN).

II. Background

RNNs and GANs

RNNs have the capacity to memorize lengthy sequences by using self-looping or recurrent connections to pass indefinitely a hidden state of features through time steps. However, a vanilla RNN may suffer from the "vanishing gradient" problem, in which the feedback gradient used to train the RNN shrinks exponentially fast with time (Bengio, Simard,

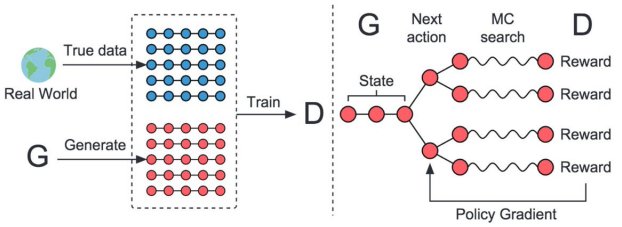

Figure 1: Illustration of SeqGAN training algorithm from (Yu et al. 2017). Left: The discriminator D is trained to distinguish between real and generated sequences. Right: The generator G is trained by the REINFORCE (Williams 1992) policy gradient where the final reward signal is provided by D.

and Frasconi 1994). To remedy this issue, gating cells such as LSTM were developed as a memory mechanism to save states through multiple time steps (Hochreiter and Schmidhuber 1997). As shown by (Zhang et al. 2017), RNNs have the capacity to memorize lengthy sequences through maximizing the log predictive likelihood of each token in the training sequence given the previous observed tokens. This presents two problems for sequential generation: (1) creativity suffers as the generator model rote memorizes the training corpus; (2) maximum-likelihood approaches suffer from exposure bias, where discrepancies between training data and generated sequences may throw off the generation for future tokens (Bengio et al. 2015).

To avoid the above limitations of a fixed-sized corpus, we use LSTM-gated RNN GANs (Goodfellow et al. 2014) trained with the REINFORCE algorithm as proposed by SeqGAN (Yu et al. 2017), consisting of (1) a discriminative neural network D to distinguish given data from real data in the training corpus, and (2) a generative neural network G that attempts to produce new sequences that fool D into classifying generations as real. By training D and G in tandem, the GAN effectively accumulates new training examples by using new generations from G to train D. Furthermore, by priming the generator with random noise, the model is not limited to the same probability distribution for equivalent prior information as noted by (Yang, Chou, and Yang 2017).

One difficulty in using GANs to model sequences of discrete tokens is that subtle changes in the generator weights may not translate to changes in the generator output, reducing the effectiveness of gradient feedback. Unfortunately, this problem is not simply remedied by using probability distributions in place of the tokens, as the discriminator will immediately pick up on non-extreme distributions. (Yu et al. 2017) instead propose the REINFORCE (Williams 1992) algorithm to alleviate difficulties in the gradient feedback of sequences of discrete tokens by treating the generator as an agent of reinforcement learning with a real-valued reward. We adopt their procedure to train our RNN GANs.

Related Work

Four other published GANs for music are SeqGAN (Yu et al. 2017), C-RNN-GAN (Mogren 2016), MidiNet (Yang, Chou, and Yang 2017), and MuseGAN (Dong et al. 2018). For the purposes of comparison with another chord-conditioning model, we also review the ImprovRNN model proposed by the Magenta Project from the Google Brain team (Google 2018). A brief description of each is provided below.

SeqGAN (Yu et al. 2017) first introduced the application of the REINFORCE (Williams 1992) algorithm to GANs generating sequences of discrete tokens. While it was built mainly for text sequences, we apply the same reinforcement learning model to music encoded as sequences of discrete tokens. Because SeqGAN was focused on text sequences, it used only the BLEU metric (Papineni et al. 2002) to evaluate performance on music sequences, and it incorporated no chord conditioning.

C-RNN-GAN (Mogren 2016) is the first published GAN model constructed specifically for (polyphonic) music generation. It used RNNs and represented notes as real-valued quadruplets of frequency, length, intensity, and timing. By using real-valued pitches, C-RNN-GAN can be trained with standard backpropagation, in contrast to the reinforcement policy gradient methods used by (Yu et al. 2017). We choose discrete pitch classes over real-valued frequencies due to C-RNN-GAN's difficulty of representing rests and inability to predict *probability distributions* of pitch classes. For similar reasons, we also adopt discrete classes of length rather than real-valued lengths. C-RNN-GAN used several metrics applicable to monophonic melodies: (1) scale consistency, (2) repetition counts, and (3) tone spans. We included these metrics in our own experiments. Unfortunately, C-RNN-GAN had no chord conditioning.

MidiNet (Yang, Chou, and Yang 2017) is a GAN that uses convolutional neural networks (CNN) for (monophonic) music generation. Though it used CNNs for G and D, it could condition on previous bars through a third conditioner CNN. It could also condition on an accompanying chord channel consisting of a one-hot twelve-dimensional vector over the twelve keys plus a major/minor bit. The authors noted their struggle in getting MidiNet to generate notes beyond those in the chord triad. MidiNet evaluated its music quality through a user study. The public MidiNet repository only contains a trained model without chord-conditioning, so we were unable to compare MidiNet against JazzGAN.

The authors of MidiNet published a second CNN-based GAN called MuseGAN, which used three different systems of CNN-GANs and a reverse-CNN encoder to generate multi-tracks of bass, drums, guitar, strings, and polyphonic piano. Unlike MidiNet, MuseGAN had no explicit chord conditioning. However, it offered two metrics applicable to monophonic melodies: (1) number of used pitch classes per sequence, and (2) qualified note frequency, defining "qualified notes" as lasting longer than a 32nd note. We include both of these metrics in our experiments, with a stronger definition of qualified notes, one eliminating unconventional note durations such as seven timesteps out of 48, where 48 timesteps represents a whole note.

Due to the difficulty of finding other usable GANs with chord-conditioning, we compare against Magenta's pre-trained ImprovRNN for chord-conditioning experiments. ImprovRNN uses the same LSTM model as Magenta's

monophonic MelodyRNN, but also conditions the melodies on an underlying chord progression. Chords are represented by both a root pitch class and a binary vector of notes included in the chord, allowing ImprovRNN to condition on more than the 24 basic triads.

One salient feature about these related models is that, of the few that have chord-conditioning, none have metrics to measure how well the model adheres to those chords. Our model is also the first to use discretized sequences with RNNs in the context of GANs specifically for music generation.

III. Proposed Metrics

As detailed in the related work section, none of the related GAN models have metrics to directly evaluate chord-conditioning. In addition, none of the related GAN models have metrics to evaluate how the learned model understands rhythms and beat positions. While these metrics may be of lesser importance in corpora with few variations in chords and rhythms, they are essential in evaluating and understanding jazz music. Finally, we note the lack of plagiarism metrics in the related works.

We introduce three categories of metrics meant to address these concerns. **Mode Collapse** metrics serve as a check against the phenomenon wherein the GAN generator may collapse to a parameter setting that always emits the same output (Salimans et al. 2016). In the context of music generation, mode collapse of the generator may be observed by many repeated notes or incoherent intervals and note durations. **Creativity** metrics measure the amount of copied sequences from the training corpus and the variety within generated sequences. **Chord Harmony** metrics evaluate how well the generated sequences adhere to a given chord progression.

Mode Collapse Metrics

We use the following metrics to evaluate the general quality of musical generations. The QR and TS metrics, as described below, have been adapted from (Dong et al. 2018). We propose additional metrics (CPR, DPR, OR) to address concerns of repeated notes and observational bias in generated rhythms. To see how well the generator model performs, we compare these metrics on a set of generated sequences against the training corpus.

Qualified Rhythm frequency (QR): QR measures the frequency of note durations within valid beat ratios of $\{1, 1/2, 1/4, 1/8, 1/16\}$, their dotted and triplet counterparts, and any tied combination of two valid ratios. This generalizes beyond MuseGAN's qualified note metric, which only measures the frequency of durations greater than a 32nd note.

Consecutive Pitch Repetitions (CPR): For a specified length l, CPR measures the frequency of occurrences of l consecutive pitch repetitions. We do not want the generator to repeat the same pitch many times in a row.

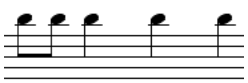

Figure 2: Example of five repeated pitches measured in CPR.

Durations of Pitch Repetitions (DPR): For a specified duration d, measures the frequency of pitch repetitions that last at least d long in total. We do not want the generator to repeat the same pitch multiple times for a long time. For example, three whole notes of the same pitch in a row are worse than three triplets of the same pitch. We only consider repetitions of two or more notes.

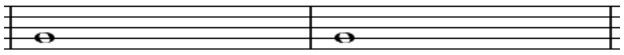

Figure 3: Example of repeated pitches that last over two bars measured in DPR.

Tone Spans (TS): For a specified tone distance d, TS measures the frequency of pitch changes that span more than d half-steps. Example: setting $d = 12$ counts the number of pitch leaps greater than an octave.

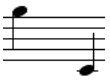

Figure 4: Example of a nineteen half-step span measured in TS.

Off-beat Recovery frequency (OR): Given an offset d, OR measures how frequently the model can recover back onto the beat after being forced to be off by d timesteps. For example, with a 48-timestep encoding for a bar, we run experiments with an offset of seven timesteps, which corresponds to no conventional beat position. We define recovery onto the beat as generating a note on a beat position corresponding to a multiple of an eighth note.

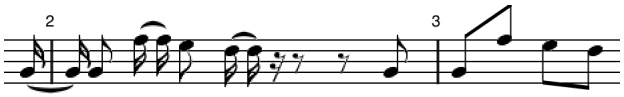

Figure 5: Bar 2: Off-beat rhythm; Bar 3: On-beat rhythm.

Creativity Metrics

We propose the following metrics to evaluate the creativity of the model.

Rote Memorization frequencies (RM): Given a specified length l, RM measures how frequently the model copies note sequences of length l from the corpus.

Pitch Variations (PV): PV measures how many distinct pitches the model plays within a sequence.

Rhythm Variations (RV): RV measures how many distinct note durations the model plays within a sequence.

These metrics are meant to evaluate how frequently the model simply mimics the training set contents, and how diverse the model generations are. Ideally, PV and RV should be close to the actual values for the training corpus.

Chord Harmony Metric

We propose the following metric to evaluate how well the model interacts with the chord progression.

Harmonic Consistency (HC): The harmonic consistency metric is based on the Impro-Visor (Keller 2018) note categorization, represented visually by coloration, which measures the frequency of black, green, blue, and red notes. Black notes are pitches that are part of the current chord, green notes (called "color tones") are tones sympathetic to the chord, blue notes are approach (by a half-step) tones to chord or color tones, and red notes are all other tones, which generally clash with the accompanying chord. The Impro-Visor vocabulary file defines these categories. We did not modify the standard file specifically for the current corpus.

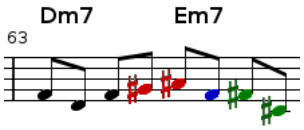

Figure 6: Notes with Impro-Visor coloration.

The HC metric turns on-chord and off-chord tones into more nuanced categories based on the surrounding context. This allows us to capture stylistic features such as approach tones, which are off-chord but resolve in the next note. Ideally, these frequencies should be close to the actual values for the training corpus.

IV. Comparing Rhythm Representations

Experimental Setup

Our first experiment compares the effect of three different rhythm encodings on generated outputs.

Time-step encoding: This is the encoding used by (Johnson, Keller, and Weintraut 2017; Google 2018; Sturm, Santos, and Korshunova 2015; Yang, Chou, and Yang 2017; Dong et al. 2018). Instead of predicting durations note-by-note, the model divides the measure into timesteps and predicts the pitch at each timestep. Notes are sustained by repeating pitches for multiple timesteps. Some studies include an additional $attack$ bit to indicate when repeated pitches are played again versus sustained (Johnson, Keller, and Weintraut 2017).

Notably, no published RNN *GAN* for music has been implemented with the timestep encoding. For our jazz corpus, we found that while vanilla RNNs are capable of learning both the pitch and attack sequences over timesteps, RNN GANs struggle to learn the attack sequence. We found that even after pre-training the RNN to generate proper attack sequences, the GAN unlearns the attack sequences during adversarial training. This may be due to the relative sparsity of attacks in sequences over timesteps.

Note duration encoding: This note-by-note encoding trains the models on sequences over notes instead of sequences over timesteps, and was used by C-RNN-GAN (Mogren 2016) and SeqGAN (Yu et al. 2017). At each step of generation, the model simultaneously predicts the pitch and duration of the next note.

The note duration encoding offers two major advantages: (1) sequences are compressed in length, (2) sparse attack sequences no longer need to be generated. In particular, sequence compression makes it easier for the RNN to recall past notes without going back through several timesteps. For example, if the model generates a whole note (equivalent to 48 timesteps), it no longer needs to remember information from 48 timesteps previously to condition on notes before the whole note.

One disadvantage of the note duration encoding is that some rhythms tend to dominate the corpus, meaning that models are susceptible to the *exposure bias* of predicting the same duration (i.e. eighth notes) over and over.

Note beat position encoding: This note-by-note encoding trains the model to predict each note's ending beat position. Note durations can then be calculated as the difference between ending beat positions.

By predicting a constantly changing beat position instead of note duration, the model is less susceptible to predicting the same duration over and over. In particular, this avoids the exposure bias of proper rhythms. Should the model ever accidentally go severely off-beat, a beat-position encoding will be better equipped to recover than a duration encoding.

Network Structure and Training Procedure

Three separate RNN-GAN models were trained with the different rhythm representations. We compare the three models using the proposed metrics.

Our neural network models were implemented in TensorFlow (Abadi et al. 2016). We used a single LSTM layer of 300 nodes, with a 72-dimension (six octave) shared embedding matrix to encode distinct pitches. To get the generator past the early stages of outputting noise, we first conducted a pre-training phase in which the generator G is trained by maximum-likelihood on training sequences. Adherence to the training sequence during pre-training was enforced through *teacher forcing* (Williams and Zipser 1989). Once the mean likelihood error fell below a threshold, we switched to training G via the REINFORCE (Williams 1992) algorithm, and training D via the cross-entropy between the predicted probability and the actual probability that the sequence is real.

Each model was trained on the same dataset for 1000 epochs with a learning rate of 10^{-3}. We updated D once every seven updates of G, and we froze D during pre-training.

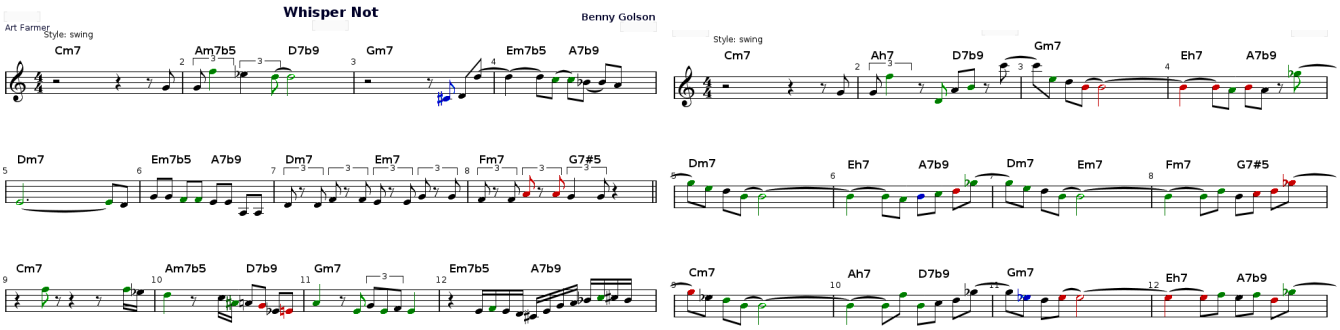

Figure 7: Left: Sample leadsheet from the corpus. Right: Sample beat-model generation over same chords, primed with the first four notes of the corpus sequence. [h7 (half-diminished seventh) is an abbreviation for m7b5 in Impro-Visor notation.]

Dataset and Feature Representation

We used the Impro-Visor (Keller 2018) corpus of monophonic jazz transcriptions with accompanying chord progressions. This collection of 44 leadsheets consisted of about 1700 total bars. A sample leadsheet snippet is given in Figure 7. Each bar was segmented into 48 timesteps (i.e. twelve timesteps represents a quarter note) to allow for sixteenth-note triplet rhythms. Note durations and beat positions were therefore encoded in a 48-dimensional one-hot vector; the model was capable of generating any of the 48 classes. The note pitches in the corpus ranged from MIDI 44 to MIDI 106. We encoded rests as just another pitch class, so there were 64 total pitch classes.

Each half-bar had a corresponding chord consisting of a tuple $(ckey, cnotes)$ where $ckey$ denotes a root pitch class from C to B (one-hot over 0-11) and $cnotes$ denote the actual notes in the chord (multi-hot from 0-11) starting from the root key of the chord. In particular, this means that the chords in the corpus span beyond the 24 basic triads, unlike in (Yang, Chou, and Yang 2017). To better enable chord conditioning, we transposed the corpus to all 12 keys by shifting all notes and chords by 0 to 11 half-steps. After transposing to the 12 different keys, we ended up with 20,000 bars.

Musical Phrasing with Harmonic Bricks

The term harmonic $brick$ follows the work of (Cork 1988) and (Elliott 2009). A brick is a chord progression of a few measures used idiomatically. Common examples of bricks are cadences and turnarounds, both of which occur in several varieties with differing frequencies. Impro-Visor automates the analysis of bricks from the chord progression in a leadsheet (Keller et al. 2013). In a previous paper (Keller et al. 2012), we indicated how bricks could be used as the basis for creative improvisation. The present paper offers an additional use of bricks in automating improvisation.

(Bretan, Weinberg, and Heck 2017) show that compelling music generation models can be developed by combining small unit sequences to form an overall musical sequence. We aim to emulate this insight in segmenting based off of chord sequences. To avoid over-fitting to a whole leadsheet, we segmented each leadsheet into musical phrases that we assume to be semi-independent. Due to our choice of a note-by-note, instead of timestep, encoding, a single bar may not be enough to capture a significant sequence of notes. So while (Yang, Chou, and Yang 2017) chose to segment by bars, we instead segmented by harmonic bricks determined by the background chord progression. Unlike (Yang, Chou, and Yang 2017), this segmentation does not limit us to a fixed look-back length nor does it require new structure to explicitly condition on past phrases. To generate continuous sequences, the generator RNN can pass its hidden state from the end of one phrase to the beginning of the next.

After segmenting into bricks, we preprocessed each sequence by eliminating rests from the start and end of the sequence. We then enforced that the resulting sequence lasted longer than half a measure and no longer than four measures. These constraints cut the number of sequences from 20,000 bars to about 13,000 bricks.

Experimental Results

We follow (Dong et al. 2018) in generating 20,000 sequences with each model and then evaluating the generations with our proposed metrics. We primed each sequence with the backing chords of a randomly chosen brick sequence from the training corpus. Tables 1, 2, and 3 show the performance of each model on the proposed metrics. In our experiments, the timestep model did not effectively learn the attack bit sequence, so we segmented notes by consecutive pitches. This automatically results in frequencies of 0% for the timestep CPR and DPR consecutive pitch scores, so we omit those scores from Table 1.

Bold entries in the table denote which model performed best on each metric. Metric parameters were chosen to provide the most information about possible mode collapse. The models were then ranked by similarity to corpus values. From the results, we see that the beat position encoding outperformed the other two models in most metrics. In particular, we see a drastic difference in the OR off-beat recovery metric. We discuss the results for each metric below.

Qualified Rhythm frequency (QR):
We calculated the frequencies of note durations within the valid beat ratios of $\{1, 1/2, 1/4, 1/8, 1/16\}$, their dotted and triplet counterparts, and any tied combination of two valid ratios. All three models generated over 90% qualified durations, and the timestep model nearly always generated valid

Table 1: Mode Collapse Metrics for JazzGAN

Entries denote mean frequency scores for each model on each metric. Bold entries denote which model obtained the best score (closest to the corpus value). Standard deviation value key: $\dagger$*:* < 0.3*,* $\dagger\dagger$*:* < 0.2*,* $\dagger\dagger\dagger$*:* < 0.1*,* $\star$*:* < 0.05*,* $\star\star$*:* < 0.03*,* $\star\star\star$*:* < 0.01*.*

	QR	CPR2	DPR24	TS12	OR7
Corpus	$0.997^{\star\star}$	$0.013^{\star}$	$0.005^{\star}$	$0.001^{\star\star}$	$0.97^{\star\star\star}$
Timestep	$\mathbf{0.999}^{\star\star}$	-	-	$0.22^{\dagger\dagger}$	$0.06^{\star\star\star}$
Duration	$0.91^{\dagger\dagger}$	$0.105^{\dagger\dagger}$	$0.045^{\dagger}$	$0.097^{\dagger\dagger}$	$0.002^{\star\star\star}$
Beat	$0.95^{\dagger\dagger}$	$\mathbf{0.063}^{\star}$	$\mathbf{0.017}^{\star}$	$\mathbf{0.026}^{\star}$	$\mathbf{0.96}^{\star\star\star}$

rhythms. We note that the QR score is inversely correlated with the RV score, suggesting that models with more diversity in rhythms naturally generate more invalid rhythms.

For comparison, MuseGAN defined their own qualified note metric encapsulating all durations greater than a 32nd note. Despite using this weaker definition of qualification, their best model achieved only 62% qualified duration frequency. MuseGAN used a bigger timestep division (96 versus our 48) and their corpus had fewer qualified notes (88.4%). However, our models achieved a smaller gap between generated frequencies and corpus frequencies. This discrepancy may be due to MuseGAN's usage of CNNs, which must generate sequences in simultaneous chunks, as opposed to RNNs.

Consecutive Pitch Repetitions (CPR2):
We calculated the frequency of occurrences of two consecutive pitch repetitions. While the duration and beat position models had about 10% or lower frequencies, there remains a gap between the model and corpus frequencies. Observations of early training epochs suggest that the GAN is susceptible to predicting repeated pitches due to the RNN passing the hidden state from step to step. Future work could investigate whether CNNs also produce higher frequencies of repeated pitches; the single-step characteristic of their music generation suggests that they would not.

Durations of Pitch Repetitions (DPR24):
We calculated the frequency of pitch repetitions of two or more notes that lasted for at least 24 timesteps, or a half-note. The corpus has even fewer of these instances, and so do the model generations. Interestingly, the duration encoding had higher frequencies for both the CPR and DPR scores. It is unclear why this would be the case, since both models are predicting note-by-note.

Tone Spans (TS12):
We calculated the frequency of tone spans greater than an octave. It becomes apparent that the timestep model struggled to generate cohesive sequences of pitches, as the TS frequency was 22%. We suspect that the GAN's inability to predict the sparse attack sequence threw off the pitch predictions as well, since the GAN was trained by a single reward value per timestep.

Off-beat Recovery frequency (OR7):
We calculated the frequency of times that the model recovered back to a beat position divisible by an eighth note, after being primed seven timesteps off-beat. To ensure that all models could keep track of the beat position, we fed the beat position as a 48-dimensional one-hot feature vector at each step. The corpus score denotes the frequency of sequences that had no notes at beat positions divisible by an eighth note.

As expected, the duration encoding utterly fails to recover since it had less need to keep track of the beat position during training. While the timestep model performed marginally better, it appears that the model was predicting based off of note duration rather than beat position. The beat position model achieves a surprisingly high recovery rate of 96%, nearly matching the corpus score of 97%.

Pitch Variations (PV):

Table 2: Creativity Metrics for JazzGAN

Entries denote mean frequency scores for each model on each metric. Bold entries denote which model obtained the best score (closest to the corpus value). Standard deviation value key: $\dagger$*:* < 0.3*,* $\dagger\dagger$*:* < 0.2*,* $\dagger\dagger\dagger$*:* < 0.1*,* $\star$*:* < 0.05*,* $\star\star$*:* < 0.03*,* $\star\star\star$*:* < 0.01*.*

	PV	RV	RM3	RM4	RM5	RM6
Corpus	$0.76^{\dagger\dagger}$	$0.32^{\dagger}$	-	-	-	-
Timestep	$0.67^{\dagger\dagger}$	$\mathbf{0.24}^{\dagger}$	$0.49^{\dagger}$	$0.16^{\dagger}$	$0.01^{\dagger\dagger}$	$0^{\dagger\dagger\dagger}$
Duration	$0.79^{\dagger}$	$0.79^{\dagger}$	$0.48^{\dagger}$	$0.19^{\dagger}$	$0.05^{\dagger\dagger\dagger}$	$0.01^{\dagger\dagger\dagger}$
Beat	$\mathbf{0.76}^{\dagger\dagger}$	$0.59^{\dagger}$	$0.85^{\dagger}$	$0.39^{\dagger}$	$0.08^{\dagger\dagger}$	$0.01^{\star}$

We calculated the average ratio across all sequences of the number of distinct pitches to the total number of notes in the sequence. For the timestep model, we evaluate the generated sequences segmented note-by-note rather than timestep-by-timestep to avoid artificially increasing the note count. All models achieved within 10% of the corpus frequency, indicating that they have learned to emulate the corpus variety.

Rhythm Variations (RV):
We calculated the average ratio across all sequences of the number of distinct note durations to the total number of notes in the sequence. Again, we segment the timestep model generations note-by-note. Unlike the PV scores, the models differ drastically from the corpus frequency of 32%. It is unclear why the note-by-note models would have increased frequencies relative to the corpus, but we note that higher RV frequencies correlate with more unqualified rhythms based on the QR score.

Rote Memorization frequencies (RM):
We calculated the frequency of copied pitch subsequences of three to six notes from the corpus. The rote-memorization frequency drops exponentially with the subsequence length, and the models do not rote-memorize past five notes. We interpret the high memorization frequency for up to four-note subsequences as indication that the model may be learning building-blocks for longer sequences, while avoiding copying longer sequences altogether. This is reminiscent of

the unit selection strategy proposed by (Bretan, Weinberg, and Heck 2017). Interestingly, the beat position encoding achieves nearly double the rote-memorization frequencies of the other models; we are unsure why there would be such a discrepancy between the note-by-note models. **Harmonic**

Table 3: Chord Metrics for JazzGAN

Entries denote frequency scores for each model on each metric. Bold entries denote which model obtained the best score (closest to the corpus value).

	HC Black	HC Red	HC Green	HC Blue
Corpus	0.56	0.06	0.34	0.04
Timestep	0.41	0.24	0.31	**0.04**
Duration	**0.44**	**0.20**	**0.34**	0.03
Beat	**0.44**	**0.20**	0.33	**0.04**

Consistency (HC):
We calculated the frequency of black (chord tones), green (sympathetic tones), blue (approach tones), and red (clashing tones) notes. The models generate similar frequencies of green and blue notes as compared to the corpus. However, they generate about 15% fewer black notes and more red notes, indicating a slightly worse harmonic consistency than the corpus. Interestingly, despite the discrepancies in PV, TS, and RM scores, the models generate much more similar HC scores. This may be a sign that the timestep model, which failed the TS metric, may be producing the right pitch keys but at the wrong octave.

Future work could investigate where the red notes occur in the beat position of the measure. It is plausible that the surplus of red notes occurs at the chord change every half-bar.

V. Comparison with ImprovRNN

Our second experiment compares how well JazzGAN learns chord conformity compared to Magenta's ImprovRNN (IRNN).

Experimental Setup

We use the same experimental setup for JazzGAN as in Experiment 1 with the rhythm representations. We use pre-trained weights for ImprovRNN as given on the Magenta repository (Google 2018). The pre-trained ImprovRNN neural network had several differences with our representation of music. Many of the sequences in our jazz corpus cannot be represented with ImprovRNN's sixteen timestep bar encoding, which only allows for beat positions that are multiples of 16th-notes. Furthermore, it was unclear how to customize the chord note vectors for ImprovRNN sequence generation, which limited our usage of the model to the basic triads. For these reasons, we did not train ImprovRNN on our jazz corpus.

Experimental Results

We reuse JazzGAN's HC statistics from the previous experiment. To evaluate the HC metric on ImprovRNN, we generated 20,000 sequences primed with the backing chords of a randomly chosen brick sequence from the jazz training corpus. In lieu of customizing the chord notes for the ImprovRNN chord vectors, we used the basic triads corresponding to the root keys of the backing chords. This also allows for a fairer comparison in case the Magenta corpus may not have included all the varieties of chords in our corpus. Table 4 shows the performance of each model on the HC metric.

Table 4: Chord Metrics for JazzGAN vs ImprovRNN

Entries denote frequency scores for each model on each metric. Bold entries denote which model obtained the best score: higher frequencies are better for all colors except red.

	HC Black	HC Red	HC Green	HC Blue
Corpus	0.56	0.06	0.34	0.04
Timestep	0.41	0.24	0.31	0.04
Duration	**0.44**	**0.20**	**0.34**	0.03
Beat	**0.44**	**0.20**	0.33	**0.04**
IRNN	0.39	0.32	0.24	0.03

Harmonic Consistency (HC):
We calculated the frequency of black (chord tones), green (sympathetic tones), blue (approach tones), and red (clashing tones) notes. ImprovRNN had the highest frequency of clashing red notes, and the lowest frequency of black chord tones and green sympathetic tones. This indicates that JazzGAN may have learned a more sophisticated chord model than ImprovRNN, as it seems to adhere to the chords better.

It must be noted that ImprovRNN trained on a different corpus than JazzGAN, and we do not know the HC frequencies for Magenta's corpus. It is possible that Magenta's corpus had more clashing tones than our jazz corpus. Nonetheless, it is promising that JazzGAN outperforms ImprovRNN even though it was conditioned on chords beyond the basic triads.

VI. Conclusion

We have introduced **Mode Collapse**, **Creativity**, and **Chord Harmony** metrics to better analyze and understand the musical quality of generated sequences. With these metrics, we have compared several representations of note duration, showing the vulnerabilities of duration encodings to off-beat collapse and the robustness of beat position encodings. Furthermore, we have demonstrated the performance of JazzGAN's RNN-based GANs for monophonic jazz melody generation with chord conditioning in comparison to Magenta's ImprovRNN. Our experiments show that RNN-based GANs trained on discretized sequences are still capable of learning complex chord conditioning and rhythms. Sample MIDI tracks can be accessed at the ImproVisor repository (Trieu 2018).

We hope that future work may utilize the proposed metrics to provide insight into other models of autonomous music generation. For example, while we have compared the qualified note metric from MuseGAN against our QR score, it would be interesting to compare other musical traits

learned by CNNs versus RNNs. We also advocate the usage of Impro-Visor's colored note metric as a measurement of chord conformity. We expect that more metrics will be needed to tease out the differences between an increasing variety of musical models.

Acknowledgment

This work was supported in part by NSF CISE REU award number 1359170 to Harvey Mudd College.

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, 265–283.

Bengio, S.; Vinyals, O.; Jaitly, N.; and Shazeer, N. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in Neural Information Processing Systems* 1:1171–1179.

Bengio, Y.; Simard, P.; and Frasconi, P. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5(2):157–166.

Bretan, M.; Weinberg, G.; and Heck, L. 2017. A unit selection methodology for music generation using deep neural networks. *Eighth International Conference on Computational Creativity, ICCC, Atlanta* 72–79.

Cork, C. 1988. *Harmony by LEGO bricks: A new approach to the use of harmony in jazz improvisation*. Tadley Ewing Publications.

Dong, H.-W.; Hsiao, W.-Y.; Yang, L.-C.; and Yang, Y.-H. 2018. Musegan: Symbolic-domain music generation and accompaniment with multi-track sequential generative adversarial networks. *AAAI* 34–41.

Elliott, J. 2009. *Insights in Jazz: An Inside View of Jazz Standard Chord Progressions*. http://www.dropback.co.uk.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. *Advances in neural information processing systems* 27:2672–2680.

Google. 2018. Improv rnn. `github.com/tensorflow/magenta/tree/master/magenta/models/`.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Johnson, D. D.; Keller, R. M.; and Weintraut, N. 2017. Learning to create jazz melodies using a product of experts. *Eighth International Conference on Computational Creativity, ICCC, Atlanta* 151–158.

Keller, R. M.; Schofield, A.; Toman-Yih, A.; and Merritt, Z. 2012. A creative improvisational companion based on idiomatic harmonic bricks. *Third International Conference on Computational Creativity, ICCC, Dublin* 155–159.

Keller, R. M.; Schofield, A.; Toman-Yih, A.; Merritt, Z.; and Elliott, J. 2013. Automating the explanation of jazz chord progressions using idiomatic analysis. *Computer Music Journal* 37(4):54–69.

Keller. 2018. Impro-visor. `www.impro-visor.com`.

Mogren, O. 2016. C-rnn-gan: Continuous recurrent neural networks with adversarial training. *Constructive Machine Learning Workshop*.

Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W.-J. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, 311–318. Association for Computational Linguistics.

Salimans, T.; Goodfellow, I.; Zaremba, W.; Cheung, V.; Radford, A.; and Chen, X. 2016. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, 2234–2242.

Siegelmann, H. T., and Sontag, E. D. 1995. On the computational power of neural nets. *Journal of computer and system sciences* 50(1):132–150.

Sturm, B.; Santos, J. F.; and Korshunova, I. 2015. Folk music style modelling by recurrent neural networks with long short term memory units. *16th International Society for Music Information Retrieval Conference*.

Trieu, N. 2018. Jazzgan examples. `github.com/Impro-Visor/sequence_gan/mume2018`.

Williams, R. J., and Zipser, D. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* 1(2):270–280.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*. Springer. 5–32.

Yang, L.-C.; Chou, S.-Y.; and Yang, Y.-H. 2017. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR2017), Suzhou, China* 324–331.

Yu, L.; Zhang, W.; Wang, J.; and Yu, Y. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. *AAAI* 2852–2858.

Zhang, C.; Bengio, S.; Hardt, M.; Recht, B.; and Vinyals, O. 2017. Understanding deep learning requires rethinking generalization. *ICLR*.