GENERATING NONTRIVIAL MELODIES FOR MUSIC AS A SERVICE

Yifei Teng
U. of Illinois, Dept. of ECE
teng9@illinois.edu

Anny Zhao
U. of Illinois, Dept. of ECE
anzhao2@illinois.edu

Camille Goudeseune
U. of Illinois, Beckman Inst.
cog@illinois.edu

ABSTRACT

We present a hybrid neural network and rule-based system that generates pop music. Music produced by pure rule-based systems often sounds mechanical. Music produced by machine learning sounds better, but still lacks hierarchical temporal structure. We restore temporal hierarchy by augmenting machine learning with a temporal production grammar, which generates the music's overall structure and chord progressions. A compatible melody is then generated by a conditional variational recurrent autoencoder.

The autoencoder is trained with eight-measure segments from a corpus of 10,000 MIDI files, each of which has had its melody track and chord progressions identified heuristically.

The autoencoder maps melody into a multi-dimensional feature space, conditioned by the underlying chord progression. A melody is then generated by feeding a random sample from that space to the autoencoder's decoder, along with the chord progression generated by the grammar. The autoencoder can make musically plausible variations on an existing melody, suitable for recurring motifs. It can also reharmonize a melody to a new chord progression, keeping the rhythm and contour.

The generated music compares favorably with that generated by other academic and commercial software designed for the music-as-a-service industry.

1. INTRODUCTION

Computer-generated music has started to expand from its pure artistic and academic roots into commerce. Companies such as Jukedeck and Amper offer so-called music as a service, by analogy with software as a service. However, their melodies, when present at all, often just arpeggiate the underlying chord.

We extend this approach by generating music with both chord progressions and interesting, nontrivial melodies. We expand a song structure such as $AA'BA$ into a harmonic plan, and then add a melody compatible with this structure and harmony. This compatibility uses a chord-melody relationship found by applying machine learning to a corpus of MIDI transcriptions of pop music (Figure 1).

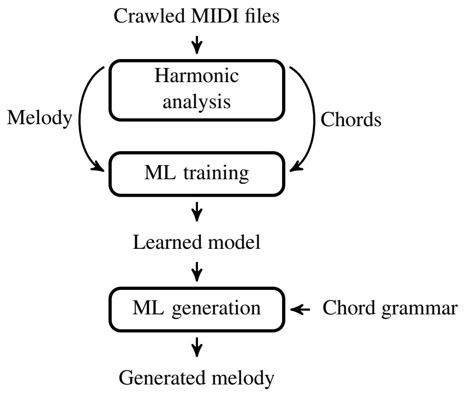

Figure 1: Machine learning (ML) workflow for generating music from a MIDI corpus.

Prior research is discussed in section 2. Harmonic analysis is detailed in sections 3 and 4. Hierarchy generation and melody generation are described in section 5.

2. RELATED WORK

Recent approaches to machine composition use neural networks (NNs), hoping to approximate how humans compose. Chu et al [5] generate a melody with a hierarchical NN that encodes a composition strategy for pop music, and then accompany the melody with chords and percussion. However, this music lacks hierarchical temporal structure. Boulanger-Lewandowski et al [3] investigate hierarchical temporal dependencies and long-term polyphonic structure. Inspired by how an opening theme often recurs at a song's end, they detect patterns with a recurrent temporal restricted Boltzmann machine (RTRBM). This can represent more complicated temporal distributions of notes. Similarly, Huang and Wu [10] generate structured music with a 2-layer Long Short Term Memory (LSTM) network. Although the resulting music often sounds plausible, it cannot produce clearly repeated melodic themes, just like a Markov resynthesis of the text of the famous poem "Jabberwocky" is unlikely to replicate the identical opening and closing stanzas of the original. Despite the LSTM network's theoretical capability of long-term memory, it fails to generalize to arbitrary time lengths [8], and its generated melodies remain unimaginative.

In these approaches, tonic chords dominate, and melody is little more than arpeggiation. To avoid this banality, we work in reverse. We first create structure and chords, and then fit melody to that. This mimics how classical western Roman-numeral harmony is taught to beginners: only after one has the underlying chord sequence, can one explain the melody in terms of chord tones, passing tones, appoggiaturas, and so on.

3. MELODY IDENTIFICATION

For pop music, a catchy and memorable melody is crucial. To generate melodies that sound less robotic than those generated by other algorithms, we use machine learning. To create a learning database, we started with a corpus of 10,000 MIDI files [16], from which we extracted useful training data (melodies that sound vivid or fun). In particular, the training data was eight-measure excerpts labelled as melody and chords. We thus had to identify which of a MIDI file's several tracks contained the melody. To do so, we assigned each track the sum of a rubric score and an entropy score. Whichever track scored highest was declared to be the melody. (Ties between high-scoring tracks were broken arbitrarily, because they were usually due to several tracks having identical notes, differing only in which instrument played them.)

3.1 Rubric score

Our rubric considered attributes such as instrumentation, note density, and pitch range.

We first considered a track's instrument name (MIDI meta-event `FF 04`). Certain instruments are more common for melody, such as violin or flute. Others are more likely to be applied as accompaniment or long sustained notes, such as low brass. A third category is likely used as unpitched percussion. The instrument's category then adjusted the rubric's score.

We also considered the track's note density, how often at least one note is sounding (between corresponding MIDI note-on and note-off events), as a fraction of the track's full duration. A track scored higher if this was between 0.4 and 0.8, a typical value for pop melodies.

Finally we considered pitch range, because we observed that pop melodies often lie between C3 and C5. The score was higher for a pitch range between C3 and C6, to exclude bass tracks from consideration.

The values for these attributes were chosen based on manual inspection of 100 files in the corpus.

3.2 Entropy score

We empirically observed that melody tracks often have a greater variety of pitches than other tracks. Thus, to quantify how varied, complex, and dynamic a track was, we calculated each track's entropy

$$H(X) = -\sum_{i=1}^{12} P(x_i) \log P(x_i) \tag{1}$$

where x_i represents the event that a particular note in the octave is i semitones from the pitch C, and $P(x_i)$ represents that event's probability. Higher entropy corresponds to a greater number of distinct pitches.

3.3 Evaluation

To measure how well this scoring identified melody tracks, we manually tagged the melody track of 160 randomly selected MIDI files. Comparing the scored prediction to this ground truth showed that the error rate was 15%.

4. CHORD DETECTION

To identify the chords in a MIDI file, we considered three aspects of how pop music differs from genres like classical music. First, chord inversions (where the lowest note is not the chord's root) are rare. When a new chord is presented, it is often in root position: most pop songs have a clear melody line and bass line [14], and the onset of a new chord is marked with the chord's root in that bass line. Second, chords may contain extensions (seventh), substitutions (flattened fifth), doublings, drop voicings (changing which octave a pitch sounds in), and omissions (third or fifth). Although such modifications complicate the task of functional harmony analysis, this is not a concern for our application. Third, new chord onsets are often at the start of a measure; rarely are there more than two chords per measure. Combining these observations led us to the following chord detection algorithm.

We first partition the song into segments with constant time signatures. (these are explicitly stated as MIDI meta messages). Then each segment is evenly divided into bins, where we try to match the entire bin to a chord. Because chords have different durations, we try different bin lengths: half a measure, one measure, and two measures. Then for each bin, containing all the notes sounding during that time interval, we add all these notes to a set that is matched against a fixed collection of chords, based on how close the pitches are, with a cost function:

Chord Detection: COST

```
1: function BESTCHORDINBIN($Pitches$)
2:     $Root \leftarrow$ Lowest note starting before first upbeat
3:     $Chords \leftarrow$ All chords, as array of intervals
4: return $\text{argmin}_{C \in Chords}\{\text{COST}(Pitches, C, Root)\}$
5: function COST($Pitches, Chord, Root$)
6:     $PitchCost \leftarrow 0$
7:     for $P \in Pitches$ do
8:         $interval \leftarrow$ No. semitones of $P$ from $Root$
9:         $d \leftarrow \min_{voice \in Chord}\{dist(interval, voice)\}$
10:        $PitchCost \leftarrow PitchCost + d$
11:    $ChordCost \leftarrow 0$
12:    for $voice \in Chord$ do
13:        $d \leftarrow \min_{P \in Pitches}\{dist(P - Root, voice)\}$
14:        $ChordCost \leftarrow ChordCost + d$
15: return $PitchCost + ChordCost$
```

Distance in semitones	Compatibility distance
0	0
1	6
2	2
3	2
4	2
5	1
6	4
7	1

Table 1: Interval compatibility.

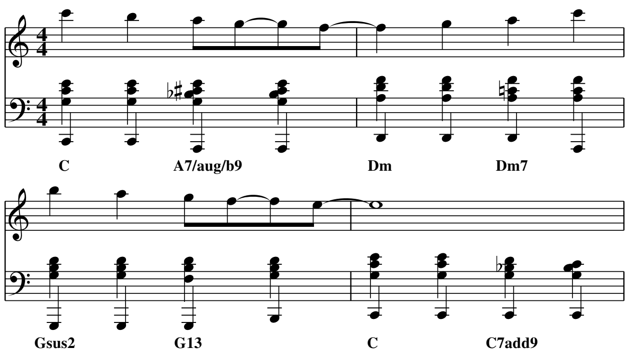

Figure 2: Example of chord detection.

Each chord's cost is the sum of the distance of the nearest interval in the chord (from the root) to each interval in the input pitches, and the distance of the nearest interval in the input pitches (from its "root") to each interval in the chord, based on some definition of distance. The cost function then returns the lowest-cost chord.

Defining the distance in terms of mere pitch difference in semitones would be simple, but performs poorly. For example, matching the pitch set $[C, E, G]$ to the chord $[C, E\flat, G\flat]$ would yield a cost of two, which is far too low. Instead, our distance function reflects how compatible intervals are. The unison is the most compatible, with distance zero; fourths and fifths are next, with distance one (Table 1). This conveniently handles omitted-fifth chords, because the chord's root matches the omitted fifth with a distance of only one.

Figure 2 demonstrates chord detection on the song *Fly me to the moon*. The bin size is half a measure, yielding 8 identified chords. The $A7/aug/\flat9$ chord resulted from the accompaniment notes $A, G, B\flat(\text{flat ninth}), C\sharp, E$, and the melody notes $A, G, F(\text{augmented fifth})$.

5. WRITING MUSIC

To output pieces with audibly hierarchical structure, we start with the harmonic structure produced by a temporal generative grammar. Then an autoencoder recurrent neural network (RNN) generates a melody compatible with this harmonic scaffold. The RNN learns to play using the chord's notes, with occasional surprising non-chord tone decorations such as passing tones and appoggiaturas.

5.1 Generating Melody

We first search for a representation of the melody using ML. This is traditionally done by an autoencoder, a pair of NNs that maps high-dimensional input data to and from a lower-dimensional space. Although this dimensionality reduction can eliminate perfect mappings, this turns out not to be a problem because the subspace of "pleasant" music within all possible musics is sufficiently small. Thus, the autoencoder can extract the pleasant content and map only that into the representation space.

It is tempting to feed a random point from the representation space to the autoencoder's decoder, and observe how much sense it makes of that point. However, because one cannot control the shape of the distribution of melody representations, one cannot guarantee that a given point from the representation space would be similar to those seen by the decoder during training. Thus, the vanilla autoencoder architecture [2] is not viable as a generative model. We propose the following improvements for generating melodies:

1. *Condition the NN on the chord progression.* The chord progression is provided to the NN at every level, so when reproducing a melody, the decoder has access to both the representation and the chord progression. This is useful because a melody has rhythmic information, intervallic content, and contour. The decoder can ignore the separately provided harmonic information, and use only the melody's other aspects. This also lets the representation remain constant while altering the chord progression, so the NN can adapt a melody to a changed chord progression, such as what happens when a key changes from minor to major.

2. *Add a stochastic layer.* Autoencoders which learn a stochastic representation are called variational autoencoders, and perform well in generative modelling of images [11]. The representation is not deterministic. We assume a particular (Gaussian) distribution in the representation space, and then train the NN to transform this distribution to match the distribution of input melodies in their high dimensional space. This ensures that we can take a random sampling of the representation space following its associated probability distribution, then feed it through the decoder and expect a melody similar to the set of musically sensible melodies.

3. *Use recurrent connections.* Pop music has many time-invariant elements, especially at time scales below a few beats. A recurrent NN shares the same processing infrastructure for note sequences starting at different times, and thereby accelerates learning.

4. *Normalize all other notes relative to the tonic.* Pop music is also largely pitch invariant, insofar as a song transposed by a few semitones still sounds perceptually similar. The NN ignores the song's key and considers the tonic pitch to be abstract, as far as pitches in melody and chords are concerned.

16x	-16	…	16	Silent	Attack	x8
	⋮	⋮	⋮	⋮	⋮	
	I	…	VI	VII	Silent	
	Pwr	Maj	Min	Dim	Aug	
	⋮	⋮	⋮	⋮	⋮	
	Major	Dorian	…	Locrian	Jazz Minor	

Table 2: An encoding of 8 measures (see section 5.1.1).

5.1.1 Implementation

The input melody is quantized to sixteenth notes. Only sections with an unchanging duple or quadruple meter are kept. The melody is converted to a series of one-hot vectors, whose slots represent offsets from the tonic in the range of -16 to 16 semitones, with one more slot representing silence. There is also an attack channel, where a value of 1 indicates that the note is being rearticulated at the current time step. The encoding for chords supports up to two chords per measure, and uses a one-hot vector for scale degrees and separate boolean channels for chord qualities (Table 2). (Note that because this encoding uses just seven Roman-numeral symbols, it does not try to represent chords outside the current mode. Before training, we removed from the corpus the few songs that contained significant occurrences of this.) We use the basic triad form for each chord identified using techniques from section 4, marking compatible chord qualities. For example, G^7 is encoded by marking a 1 in the Maj and Pwr columns. (The chord quality encoding could be extended to seventh and ninth chords.) The table's gray rows are data the network is conditioned on, while the other rows are input data that the network tries to reproduce. For an 8-measure example, the input and output vector size is $35 \times 8 \times 16 = 4480$, and the conditional vector size is $8 \times 16 + 5 \times 16 + 8 = 216$.

The network has 24 recurrent layers, 12 each for the encoder and decoder (Figure 3). Drawing on ideas of deep residual learning from computer vision [9], we make additional connections from the input to every third hidden layer. To improve learning, the network accesses both the original melody and the transformed results from previous layers during processing. The conditional part (chords and mode) is also provided to the network at every recurrent layer, as extra incoming connections.

The network is implemented in Tensorflow, a machine learning library for rapid prototyping and production training [1]. It was trained for four days on an Nvidia Tesla K80 GPU. We used Gated Recurrent Units [4] to build the bidirectional recurrent layer and Exponential Linear Units [7] as activation functions. These significantly accelerate training while simplifying the network [6, 7]. Figure 4 shows the training error (the sum of model reproduction errors) and the difference of the latent distribution from a unit Gaussian distribution, as measured by Kullback-Leibler divergence [12]. The network's input data (available at `https://goo.gl/VezNNA`) is a set of MIDI

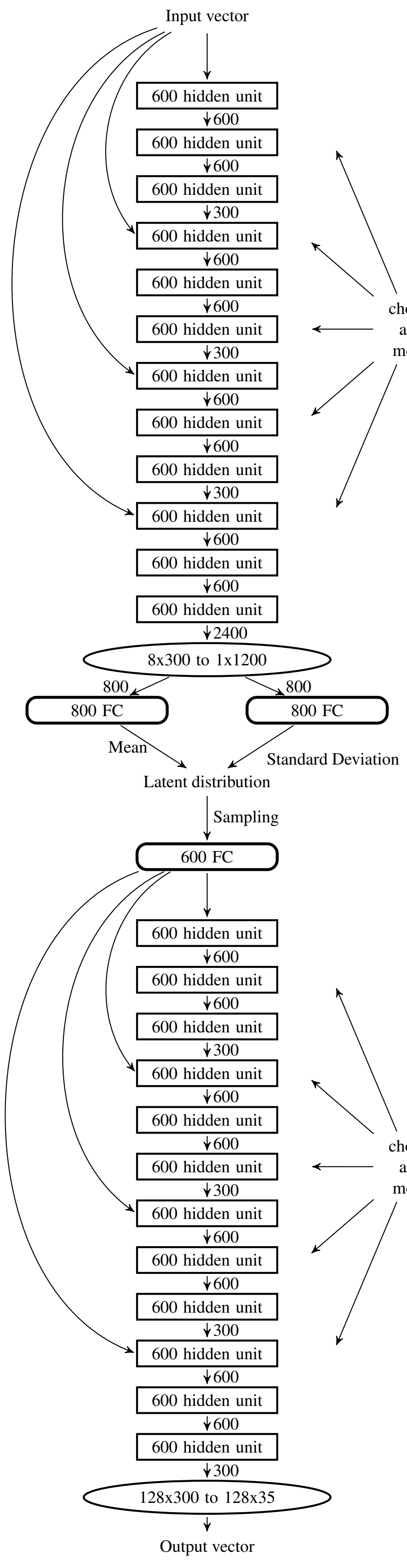

Figure 3: Network architecture. Rectangles are bidirectional recurrent neural network cells. Ellipses are strided time-convolution cells. Rounded rectangles are fully connected (FC) layers. Numbered arrows indicate a connection's dimension.

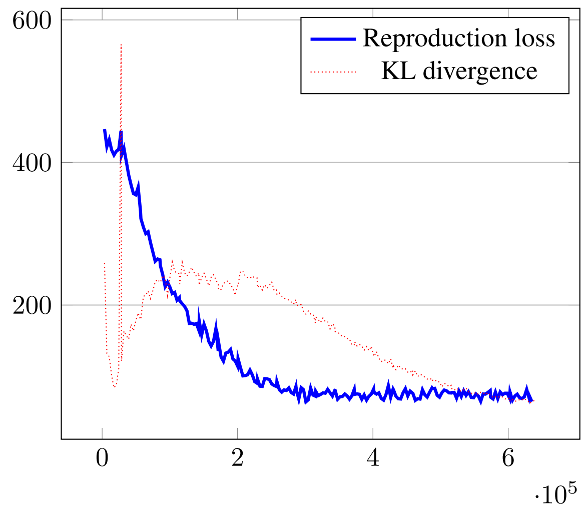

Figure 4: Training error and Kullback-Leibler divergence of the NN. The horizontal axis indicates how many training segments have elapsed ($\times 10^5$). Initial outliers have been removed.

songs from various online sources. Our harmonic analysis converted this to 1.9×10^6 measures of melodies and corresponding chords. We implemented KL warm up, because that is crucial to learning for a variational autoencoder [15]. But instead of linearly scaling the KL term for this, we found that a sigmoid reduced the network's reproduction loss.

5.2 Generating hierarchy and chords

Hierarchy and chords are generated simultaneously, using a temporal generative grammar [13], modified to suit the harmonies of pop music, and extended to enable repeated motifs with variations. The original temporal generative grammar has notions of sharing by binding a section to a symbol. For example, the rule

$$\text{let } x = \text{I in I } M5(x) \text{ I } M5(x) \text{ I}, \qquad (2)$$

where $M5$ indicates modulating to the 5th degree, would expand to five sections, with the second and fourth identical because x is reused. We extend this by having symbols x carry along a number: $x_1, x_2, \ldots$. Different subscripts of the same symbol still expand to the same chord progression, but denote slightly different latent representations when generating corresponding melodies for those sections. The latent representations corresponding to $x_{i>1}$ are derived from that of x_1 by adding random Gaussian perturbations. This yields variations on the original melody.

5.3 Training examples in the representation space

We randomly chose 130 songs from the training set, fed them through the network, and performed t-SNE analysis on the resulting 130 locations in the representation space. Although a melody maps to a distribution in the representation space, Figure 5 plots only each distribution's mean, for visual clarity. This t-SNE analysis effectively reduces the 800-dimensional representation

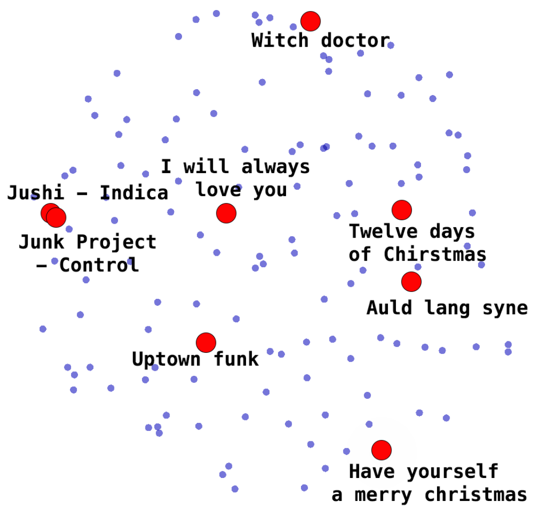

Figure 5: Example melodies in a t-SNE plot of the representation space.

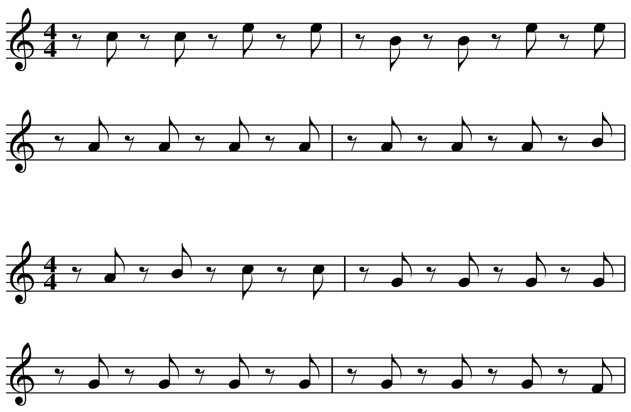

Figure 6: Four-bar excerpts from the songs *Indica* (top) and *Control* (bottom).

space into a low-dimensional human-readable format [17]. (A larger interactive visualization of 1,680 songs is at `https://composing.ai/tsne`.)

Two songs that are both in the techno genre, *Indica* by Jushi and *Control* by Junk Project, are indeed very near in the t-SNE plot, almost overlapping. Excerpts from them show that both have a staccato rhythm with notes landing on the upbeat, and have similar contours (Figure 6).

5.4 Reharmonizing melody

We hypothesized that, when building the neural network architecture, providing the chord progression to both the encoder and the decoder would not preserve that information in the representation space, thus saving space for rhythmic nuances and contour. To test this hypothesis, we gave the network songs disjoint from the training set and collected their representations. We then fed these representations along with a new chord progression to the network. We hoped that it would respond by generat-

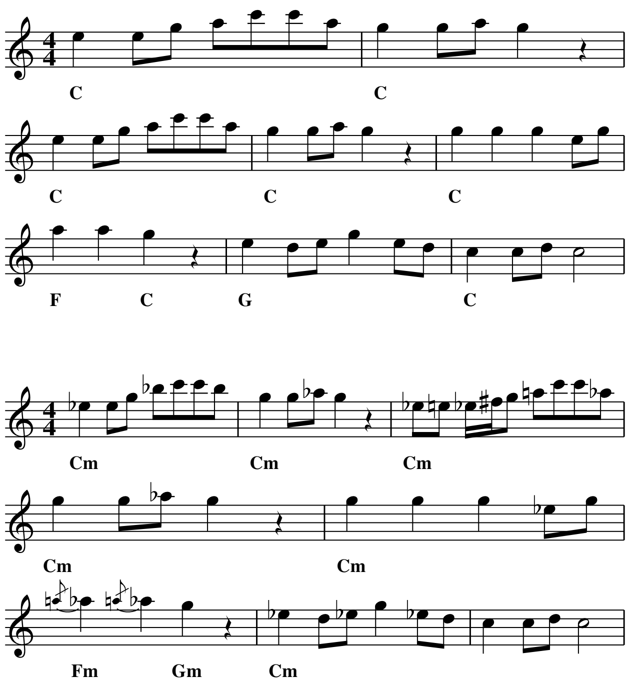

Figure 7: The song *Jasmine Flower* with original chords (top), and adapted to a new chord progression (bottom).

ing a melody that was harmonically compatible with the new chord progression, while still resembling the original melody. We demonstrate this with the Chinese folk song *Jasmine Flower*, in a genre unfamiliar to the NN (Figure 7). Note that we supplied the chords in Figure 7 (bottom), for which the NN filled in the melody. The network flattened the E, A, and B, by observing that the chord progression looked minor. This is typically how a human would perform the reharmonization, demonstrating the network's comprehension of how melody and harmony interact.

Although the NN struggled to reproduce the melody, it provided interesting modifications. The grace notes in measure 6 could be due to similar ones in the training set, or due to vacillation between the A♮ from the representation and the A♭ from the chord conditioning.

5.5 Examples of generated melodies

Because an entire multi-section composition cannot fit here, we merely show excerpts from two shorter examples.

Figure 8 and Figure 9 demonstrate melodies generated from points in the representation space that are not near any particular previously known melody. Structure is evident in Figure 8: measures 1–3 present a short phrase, and measure 4 leads to the next four measures, which recapitulate the first three measures with elaborate variation. Figure 9 shows an energetic melody where the grammar only produced C minor chords. Although the final two measures wander off, the first six have consistent style and natural contour.

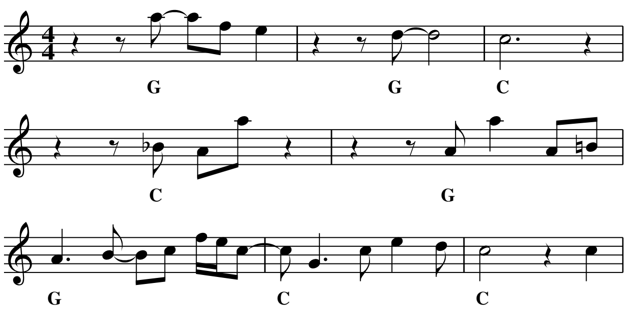

Figure 8: Generated melody for a grammar-generated chord progression.

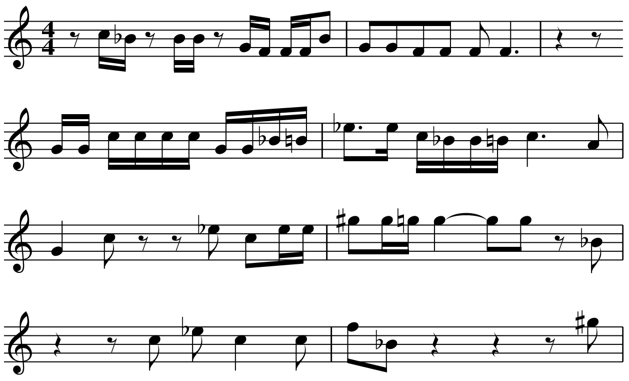
Figure 9: Generated melody for an extended C minor chord.

6. CONCLUSION AND FUTURE WORK

We have combined generative grammars for structure and harmony with a NN, trained on a large corpus, to emit melodies compatible with a given chord progression. This system generates compositions in a pop music style whose melody, harmony, motivic development, and hierarchical structure all fit the genre.

This system is currently limited by assuming that the input data's chords are in root position. More sophisticated chord detection would still let it exploit the relative harmonic rigidity of popular music. Also, by investigating the representation found by the NN, meaning could be assigned to some of its 800 dimensions, such as intensity, consonance, and contour. This would let us boost or attenuate a given melody along those dimensions.

Acknowledgements. The authors are grateful to Mark Hasegawa-Johnson for overall guidance, and to the anonymous reviewers for insights into both philosophical issues and minute details.

7. REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: a system for large-scale machine learning. In *Proc. USENIX*

Conf. Operating Systems Design and Implementation, pages 265–283. USENIX Association, 2016.

[2] Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[3] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. `arxiv.org/abs/1206.6392`, 2012.

[4] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. `arxiv.org/abs/1406.1078`, 2014.

[5] H. Chu, R. Urtasun, and S. Fidler. Song from PI: a musically plausible network for pop music generation. `arxiv.org/abs/1611.03477`, 2016.

[6] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. `arxiv.org/abs/1412.3555`, 2014.

[7] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). `arxiv.org/abs/1511.07289`, 2015.

[8] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. `arxiv.org/abs/1410.5401`, 2014.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. `arxiv.org/abs/1512.03385`, 2015.

[10] A. Huang and R. Wu. Deep learning for music. `arxiv.org/abs/1606.04930`, 2016.

[11] D. P. Kingma and M. Welling. Auto-encoding variational bayes. `arxiv.org/abs/1312.6114`, 2013.

[12] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[13] D. Quick and P. Hudak. A temporal generative graph grammar for harmonic and metrical structure. In *Proc. International Computer Music Conference*, pages 177–184, 2013.

[14] M. P. Ryynänen and A. P. Klapuri. Automatic transcription of melody, bass line, and chords in polyphonic music. *Computer Music Journal*, 32(3):72–86, 2008.

[15] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther. Ladder variational autoencoders. `arxiv.org/abs/1602.02282`, 2016.

[16] Y. Teng and A. Zhao. Composing.AI. `http://composing.ai/`, 2017.

[17] L. van der Maaten and G. E. Hinton. Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.