# Learning Latent Representations of Music to Generate Interactive Musical Palettes

**Adam Roberts**
Google Brain
Mountain View, USA
adarob@google.com

**Jesse Engel**
Google Brain
Mountain View, USA
jesseengel@google.com

**Sageev Oore***
Dalhousie University
& Vector Institute
Canada
sageev@vectorinstitute.ai

**Douglas Eck**
Google Brain
Mountain View, USA
deck@google.com

## ABSTRACT

Advances in machine learning have the potential to radically reshape interactions between humans and computers. Deep learning makes it possible to discover powerful representations that are capable of capturing the latent structure of high-dimensional data such as music. By creating interactive latent space "palettes" of musical sequences and timbres, we demonstrate interfaces for musical creation made possible by machine learning. We introduce an interface to the intuitive, low-dimensional control spaces for high-dimensional note sequences, allowing users to explore a compositional space of melodies or drum beats in a simple 2-D grid. Furthermore, users can define 1-D trajectories in the 2-D space for autonomous, continuous morphing during improvisation. Similarly for timbre, our interface to a learned latent space of audio provides an intuitive and smooth search space for morphing between the timbres of different instruments. We remove technical and computational barriers by embedding pre-trained networks into a browser-based GPU-accelerated framework, making the systems accessible to a wide range of users while maintaining potential for creative flexibility and personalization.

### Author Keywords

musical interface; latent space; variational autoencoder; deep learning

## INTRODUCTION

Music, when treated as data, is often represented in high-dimensional spaces. Digital music formats such as CD-quality Pulse-code modulation (PCM), for example, records audible vibrations as a discrete sequence of 44.1 thousand 16-bit integers per second [11]; audio can then be modelled by treating each sample as a unique dimension and capturing correlations between them. Alternately, musical compositions can be communicated as a score; in one heavily constrained version of this, for example, we could represent sequences of monophonic 16th notes of equal intensity with approximately 7 bits for each note, or 112 per bar. That is far fewer dimensions than audio, but even there, exploring all possible variations of a score by flipping one bit at a time quickly becomes intractable, and further, would result in a large proportion of melodies being so musically unconventional that they would easily be perceived as being incoherent.

While the high dimensionality affords an exponential number of possibilities, only some of these possibilities are likely for real music, which could be seen as residing on a lower-dimensional manifold within the space. Machine learning techniques can learn the shape of such low-dimensional manifolds from data, and be used to gain better understanding of and to explore large datasets [22]. They can also be used to build creative tools within the realm of "Artificial Intelligence Augmentation" (AIA) [7]. Learning the reduction directly from the data allows us to avoid heuristics and hand-tuned features, along with the biases and preconceptions about the data that would normally accompany those.

Autoencoders and variational autoencoders [13] are models designed to learn efficient, low-dimensional representations capable of reproducing their high-dimensional inputs. The hope is that in order to effectively utilize the "latent space", they will learn a mapping that is "effective". What do we mean by this, or rather, what might be some desirable characteristics for such a mapping? First we might wish for smoothness: for example, if two points are near each other in latent space, then we would like for their corresponding points in the output space to also be near one another. In the constrained case of monophonic melodies mentioned above, this would mean that we would like for the two monophonic sequences to be perceptually similar. Second, while the original high-dimensional space allows for very unlikely points, we would like the latent space to correspond primarily to the likely ones: that is, if we map from a sampled point in the latent space to the original space, we would like the resulting point to be "feasible", i.e. not one of those unconventionally incoherent sequences that

©2018. Copyright for the individual papers remains with the authors. Copying permitted for private and academic purposes.
*MILC '18*, March 11, 2018, Tokyo, Japan
* Research conducted while this author was at Google Brain.

CEUR-WS.org/Vol-2068/milc7.pdf

we described earlier. If we can satisfy these two requirements, then that would allow interpolation in the latent space to correspond to a meaningful interpolation in the original space. For example, if $A$ and $B$ are score representations of two monophonic melodies, and $f(A)$ and $f(B)$ are their corresponding latent representations, then as we sample $N+1$ points along the line between the latent points $f(A)$ and $f(B)$:

$$c_i = \alpha_i f(A) + (1-\alpha_i) f(B)$$

where $\alpha_i = i/N$ and $i$ runs from $0 \ldots N$, then $C_i = f^{-1}(c_i)$ should always be a feasible (i.e. statistically "likely") melody, and also $C_i$ should be perceptually fairly similar to $C_{i+1}$. That is, the smoothness of the latent space with respect to the outputs makes it possible to define projections of the output space onto 1-D line segments and 2-D rectangles [14, 7]. Tracing along such low-dimensional manifolds, we can thus *morph* from one melody to another in an interesting way. Low-dimensional representations thus potentially afford interesting visualizations and natural interactions.

In this paper, we present two interfaces based on the principle of musical latent spaces:

- A musical sequence explorer for 2-bar melody loops and drum beats, using the latent space of MusicVAE [20].
- An instrument timbre explorer, using the audio-based latent space of instrument samples generated by NSynth [8].

The former is implemented using deeplearn.js [21] for GPU-accelerated inference in the browser, enabling dynamic exploration with no additional installation or setup required. It can output both audio and MIDI. The latter is implemented as a Max For Live device with pre-synthesized samples, providing access to a massive number of timbres from within Ableton Live, a widely-used, professional-grade production tool.

## RELATED WORK

There is a fair amount of prior work on the concept of "musical morphing" in the space of both audio and compositions. In the audio realm, techniques such as cross-synthesis use heuristics or machine learning to mix hand-designed features of instrument timbres [12]. For compositions, heuristics and models based on music theory are used to morph between existing pieces [5, 9].

MMorph [19] is described as a real-time tool that allows users to morph between up to four MIDI compositions by mixing different elements of each piece (such as rhythm, pitch, dynamics, timbre, harmony). Via a SmallTalk GUI, a user could specify which elements of each MIDI file to morph with checkboxes and drag a control around a 2-D space to determine the relative amounts contributed by each of the four pieces in real-time. The user could also choose from several different morphing methods such as "interpolation" and "weighting", although no description of these methods is provided.

Hamanaka et al. [9] introduce a simple copy-and-paste interface for both interpolating between and extrapolating beyond pairs of melodies by reducing the melodies to a low-dimension representations based on music theory.

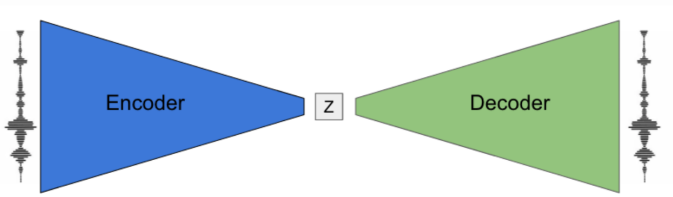


**Figure 1. Diagram of an autoencoder/VAE. Input (in this case an audio waveform) is mapped through an encoder function to a compressed latent vector $z$. Transformations, such as interpolation, can then be applied to the vector. The resulting latent vector is then fed through a decoder function to produce the output audio.**

Closely related to our melodic interpolation is work by Bretan et al. [4, 3]. The authors first extract a bag-of-features from monophonic musical phrases, and then use an autoencoder to construct a latent space. By selecting nearest-neighbors from the training set in latent space, the autoencoder can interpolate naturally between any two given 2-bar monophonic musical phrases, such that the gradual progression can be heard in both the harmonic and rhythmic elements [4].They subsequently extend this to learn small perturbations in the latent space to allow variety of real-time call-and-response interactions for polyphonic musical phrases [3]. However, the approach is somewhat less scalable and generalizable, as it can only recombine elements from the training data and the nearest-neighbor selection scales poorly as the size of the training set grows.

The Hyperviolin [12] is a physical, violin-like instrument that makes use of cross-synthesis to produce timbres mixing between various instruments such as a flute or choral vocals. The developers demonstrated multiple methods of controlling the relative weights of the high-level timbral features via "sensor shoes". The performer could control some constrained aspects of the timbre within a preset space in real-time with movements of her foot and could move between different presets throughout a piece by making larger physical movements along a path.

The Morph Table [6] uses a 3-D interface with physical cubes allowing multiple users to control the musical output of the system by morphing in both audio and compositional spaces. Movements of each cube in one dimension are associated with compositional morphing using techniques such as key-modulation, linear interpolation of note features (pitch, onset, duration, etc.) or "cross-fading" between subsections. In a second dimension, standard audio effects are are adjusted. Movements in a third dimension–accessed by rotating the cubes to expose different faces–changes the morphing endpoints between six different presets.

Existing examples of AIA interfaces using latent spaces include examples for generating faces [14], fonts [7], and generic images such as shoes and landscapes [23].

## METHODS

Both of the interfaces introduced in this paper employ autoencoders, unsupervised machine learning models each composed of three parts: an encoder, a latent vector, and a decoder (Figure 1). The latent vector, $z$, is an intermediate representation of the data examples, $x$, but is either of lower dimension or is

regularized to produce an information bottleneck. In the case of variational autoencoders (VAE), $z$ is regularized to encourage the model to learn low-entropy latent representations of $x$ close to a standard multivariate Gaussian distribution. The encoder is a neural network that produces $z$ from the input $x$ and the decoder is a separate neural network that attempts to reconstruct $x$ from $z$. Gradient-based optimization is then used to reduce the reconstruction loss (the difference between the encoded input and decoded output) and KL divergence from the Gaussian prior (if applicable) [13].

The purpose of the bottleneck is that it forces the model to distill the information content of the data to lower-entropy representations. In the process, it discovers and encodes the most important features that differentiate data examples. The model becomes specialized to produce examples like those from the training distribution, making it difficult for the decoder to produce "unrealistic" outputs that are improbable under the distribution $p(x)$. However, when trained properly it is general enough to be able to reconstruct and generate examples that are from the same distribution as $p(x)$ but do not appear in the train set. The resulting latent space is therefore optimized for various forms of exploration including interpolation within the latent space, which can "morph" between values in the output space, producing realistic intermediate outputs that combine features of the endpoints to varying degrees.

### MusicVAE

MusicVAE is a VAE for musical sequences, such as drum beats and melodies. It uses LSTM [10] recurrent neural networks as its encoder and decoder. For our purposes, we focus on models learned from either 2-bar drum beats or 2-bar melody loops. Training sets were obtained by scraping the web for MIDI files and extracting all unique 2-bar sequences of the two types, resulting in 28 million melodies and 3.8 million drum beats.

To produce good reconstructions (and interpolations), we train a VAE with a trade-off parameter that assigns a lower weight to the KL divergence, thus allowing the system to pass enough information through the bottleneck to be able to reproduce nearly any realistic input sequence in our dataset. However, models trained in this way on noisy datasets such as ours often do not produce realistic examples when sampling random points from the Gaussian prior. To produce better random samples, we train VAEs with the trade-off parameter set to a value that encourages a better KL divergence at the expense of reconstruction quality.

### NSynth

With NSynth, we are able to learn latent spaces not just of musical sequences, but of audio waveforms themselves. Like the MusicVAE, NSynth is an autoencoder architecture (though not variational) that can learn to encode and decode sequences from a compressed latent space. Since the model must learn to most efficiently use the lower-entropy representation to represent waveforms, it learns salient acoustic features among the training data. However, the model architecture differs significantly from MusicVAE as it must capture correlations over thousands of quantized waveform timesteps [8].

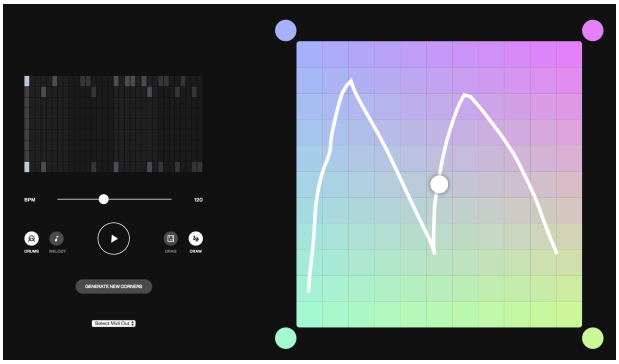

**Figure 2. The MusicVAE Sequencer. On the left is an editable sequencer for modifying the corner melodies or rhythms, along with several controls including a toggle between melody and drum modes and a drop-down selector for MIDI output. On the right is the 2-D interpolation palette where each interior square contains a sequence generated by decoding an interpolated point from the latent space between the corner sequences. The interface is shown in "Draw" mode, where the the white puck will follow along the user-drawn 1-D curve (in this case shaped like an "M"), morphing the sequence in real-time as it moves through the latent space.**

We train the NSynth model on the NSynth dataset, a collection of ~300,000 individual 4-second notes recorded from ~1,000 separate instruments and synthesizers [8]. By choosing to train on individual pitched notes, the model can then be used as a "neural synthesizer" to playback notes when receiving an input MIDI signal.

Since the model learns an expressive code for raw audio, it can be used to interpolate in this space and discover new instrument timbres that exist between pre-existing instruments. However, while the latent space is of much lower dimension than the original audio, there is no prior to sample from as was the case for MusicVAE. Therefore, there is no trivial way to randomly sample novel musical notes from the distribution, and we are limited to exploring subspaces anchored on known notes.

## INTERFACES

### MusicVAE Sequencer

The MusicVAE Sequencer (Figure 2) is an interface to the latent spaces of 2-bar drum loops and monophonic melodies, as described above. Users can toggle between these two spaces and define the four corner sequences of a 2-D, 11x11 grid by randomly sampling from the latent space (using the low-KL model), selecting from a predefined list, or inputting them manually into the sequencer at the resolution of 16th notes. Once the corner sequences are set, they are passed through the encoder of the VAE to determine their latent vectors. The latent vectors are then mixed using bi-linear interpolation in the $11 \times 11$ space, and the resulting values are decoded into sequences (from the high-KL model). With our implementation of the model architecture in deeplearn.js using weights learned via the original TensorFlow [2] model, the encoding, interpolation, and decoding can all be executed with sub-second latency on a typical consumer laptop.

Now that the palette is filled, the user can drag the white "puck" around to hear the drum beat or melody loop in each square


**Figure 3. The NSynth Instrument alongside other Ableton devices that it can be used in conjunction with to synthesize incoming MIDI events.**

of the grid. When the puck moves to a new square, the sequencer immediately updates but the play-head does not reset to the beginning, allowing the music to morph immediately but smoothly. The sound is played via the browser using preset audio samples with Tone.js [15] and can optionally be output as MIDI events via Web MIDI [1] to be further manipulated and synthesized externally.

The corner sequences can be changed at any time by the same methods mentioned above or by dragging one of the interior squares to a corner of the grid, which will set it as the new corner and cause the interior points to be updated.

If using the sequencer as a composition tool, the user can record the MIDI or audio externally, or she can right click on any square to download a MIDI file containing that particular sequence to use as a part of an arrangement.

In the case of improvisation or live performance, the user may also use the "draw" tool to define a 1-D curve within the palette, which the puck will then move through at a rate the user controls. As the puck moves into each new square, the playback sequence is updated, just as if the user had moved the puck there by hand. In this manner, a performer can set up paths in both the drum and melody palettes that will continuously evolve at potentially different rates, introducing musically interesting phasing effects.

### NSynth Instrument

The NSynth model is extremely computationally expensive (~30 minutes of GPU synthesis to generate four seconds of audio) and therefore presented a distinct challenge in creating an interactive experience for interpolating within the latent space in real-time on a laptop. Rather than generating sounds on demand, we curated a set of original sounds ahead of time and synthesized all of their interpolated latent representations. To produce even finer-grained resolution during playback, transitions are smoothed out by additionally mixing the audio in real-time from the nearest neighbor sounds on the grid. This is a straightforward case of trading off computation for memory.

We created a playable instrument integrated into Ableton Live as a Max For Live device (Figure 3). We positioned real instrument timbres at the corners of a square grid, allowing the user to mix between all four using the latent space as a palette. We created further "multigrid" modes, tiling many four-instrument grids side by side. This results in an 7x7 grid of 4x4 grids, enabling a user to explore up to 64 different instruments by dragging across a single x-y pad. Given an x-y point, the device finds the nearest four interpolated samples and plays them back with volume proportional to the their x-y distance from the point. Users can choose from 5 pre-generated grids and 3 multigrids or produce their own using the TensorFlow code released with the NSynth model [16].

Incoming MIDI notes are played back in the appropriate timbre based on the position on the palette. Additional controls include the ability to adjust where each sample the playback should start, the playback speed, and settings for an Attack Decay Sustain Release (ADSR) envelope. Furthermore, the NSynth Instrument can be combined with other Live devices to add additional effects such as an arpeggiator.

Finally, users are able to map all of these settings to an external MIDI controller for ease of use in a studio or live setting.

## EXPLORING THE INTERFACE

Interfaces provide mappings from the user's actions to the output space. In the case of the MusicVAE sequencer, the input space consists of four user-defined basis points (the corners) combined with a 2-D control surface (the palette), and the output space consists of note or drum sequences. Generally, choosing the mapping is a crucial element of designing a user interface, with numerous trade-offs at play, and where the goals often include carefully and deliberately shaping the user's cognitive model of the system [17]. In our case, the mapping is learned with a VAE, and therefore this raises the question of how the user will understand and interact with it: from the user's perspective, how does this mapping work? [18] describes some of the steps that a user may take when learning continuous mappings for complex controls over high-dimensional output spaces; we follow some of those basic approaches to explore the MusicVAE Sequencer interface. In particular, we show how to start isolating and exaggerating aspects of the mapping.

One example of exaggeration is that we set three of the basis points (bottom-left, top-left, and top-right) to all be empty, and set the bottom-right to be entirely full (i.e. every percussion instrument playing at every 16th note). In that case, our interpolation lets us check what the system does as it goes from completely sparse (silence) to completely dense, as shown in Figure 4.

As we interpolate from empty to full, we notice that the system tends to start by populating the sequencer with percussion hits on the actual beats, then on the 8th notes, and then on the 16th notes, i.e. it progresses from more likely to less likely beats. It also progresses from sparser beat sequences to denser ones. This satisfies the requirements outlined in the introduction as follows: (1) smoothness is preserved, in that each sample along the interpolated latent path corresponds to beat sequences that are similar to (i.e. a little more or less dense than) those corresponding to its adjacent samples, and (2) by first adding beats on quarter notes, then 8th notes, etc, this suggests that the system is maintaining feasible (i.e. more likely) outputs[1].

Figure 5 shows another simple experiment in which we try isolating the location (phase) of the beat in the bar. We see that

[1]For example, one could imagine a sequence that goes from sparse to dense, but in which the new percussion hits are added by simply choosing random cells in the grid from a uniform distribution over the grid. This could still be a smooth mapping, but it would have the problem that the intermediate beat sequences, during interpolation, would generally be very unlikely.

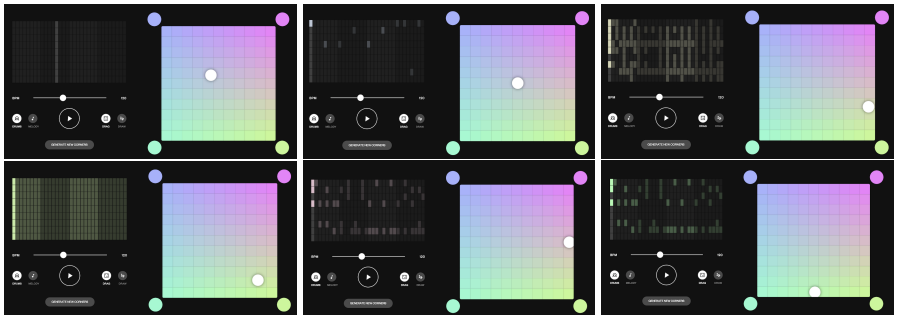

**Figure 4. Interpolating between empty and full. The dark grid on the left represents the piano roll of the percussion, and the multi-coloured square grid on the right (the palette) represents the (2-dimensional) latent space. The white puck represents a point in the latent space, and the piano roll shows that point's corresponding percussion sequence. For all six sub-figures here, the 4 corners of the latent space grid are fixed: the bottom-right corner has all drum hits enabled, and the other three corners are empty. For example, in sub-figure at (Row 2, Col 1), the puck is just next to the bottom-right corner, and indeed, the corresponding piano roll shows a very dense sequence of percussion hits. Conversely, in (Row 1, Col 2), the puck is approximately near the centre, and we see a much sparser pattern. As the puck is moved from an empty corner toward the bottom-right, drum hits are added first on quarter notes, then eighth notes, and finally 16th notes, until the sequencer eventually fills as it nears the corner. The final two images on the second row (Row 2, Col 2) and (Row 2, Col 3) illustrate the expected symmetry with respect to the diagonal for this configuration (i.e. the percussion rolls are essentially identical for these two points on either side of the (top-left, bottom-right) diagonal.**

rather than interpolating the phase (which would gradually shift the offset), there is a tendency to superpose the two rhythmic figures. This, too, is a reasonable rhythmic approach. We note that doing so in melody space would not necessarily be as natural of a choice (and in fact the system would not have that option since the melodies are constrained to be monophonic). While there is usually a lot more sophistication occurring in the interpolations aside from superposition, these results for simple cases have a natural interpretation for the user, providing landmarks on which to ground their understanding and cognitive models of the system.

In Figure 6, we explore interpolation in melody space by providing several different scales as the four basis points. As we move from top-left to top-right, we notice that at each step the output sequence is similar to the previous step, but that the combination of these smooth steps ultimately results in a very large change. Interestingly, at around a third of the way across, the output sequence includes some descending notes, even though both scales ascend. This shows that the system will sometimes move away from the basis points during the interpolation.

Finally, we note that some parameters, such as the number of grid points, were chosen in order to make the system functional, with the focus on learning an effective latent space mapping, but in future, these are design elements that would be worthwhile exploring more systematically and with user testing.

## AVAILABILITY

Supplementary resources including opensource code are available for both NSynth and MusicVAE interfaces in corresponding sub-directories of Magenta's demo github repository[2].

## CONCLUSION

This work demonstrates the use of machine learning as the basis for creative musical tools. Machine learning is often seen as a method for outsourcing mundane discriminative tasks, resulting in the desire for rigid systems that perform their duties with high accuracy. While it is likely that such systems could be used to produce satisfying music for listeners, our work indicates directions for how we might use machine learning—specifically latent-space representations— to create UI mappings that we hope will eventually provide music creators with new and interesting creativity-supporting tools.

[2] https://github.com/tensorflow/magenta-demos/

## ACKNOWLEDGMENTS

We thank Torin Blankensmith and Kyle Phillips from Google Creative Lab for implementing the MusicVAE Sequencer UI. We thank Nikhil Thorat for assistance with the deeplearn.js implementation. We thank the members of the Magenta team for helpful discussions and observations.

## REFERENCES

1. 2015. Web MIDI API. **http://webaudio.github.io/web-midi-api/**. (2015). Accessed: 2017-12-21.
2. Martín Abadi and others. 2015. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. White Paper. Google. **http://download.tensorflow.org/paper/whitepaper2015.pdf**
3. Mason Bretan, Sageev Oore, Jesse Engel, Douglas Eck, and Larry Heck. 2017a. Deep Music: Towards Musical Dialogue. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*.
4. Mason Bretan, Gil Weinberg, and Larry Heck. 2017b. A Unit Selection Methodology for Music Generation Using Deep Neural Networks. In *International Conference on Computational Creativity (ICCC 2017)*.
5. Andrew Brown and Rene Wooller. 2005. Investigating Morphing Algorithms for Generative Music. In *Proceedings of Third Iteration*, T Innocent (Ed.). Centre for Electronic Media Art, Australia, Victoria, Melbourne, 189–198. **https://eprints.qut.edu.au/24696/**
6. Andrew Brown, René Wooller, and Kate Thomas. 2007. The Morph Table: A collaborative interface for musical interaction. In *Australasian Computer Music Conference (ACMC)*. 34–39.
7. Shan Carter and Michael Nielsen. 2017. Using Artificial Intelligence to Augment Human Intelligence. *Distill* (2017). **http://distill.pub/2017/aia**
8. Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. 2017. Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders. In *International Conference on Machine Learning (ICML)*. **https://arxiv.org/abs/1704.01279**

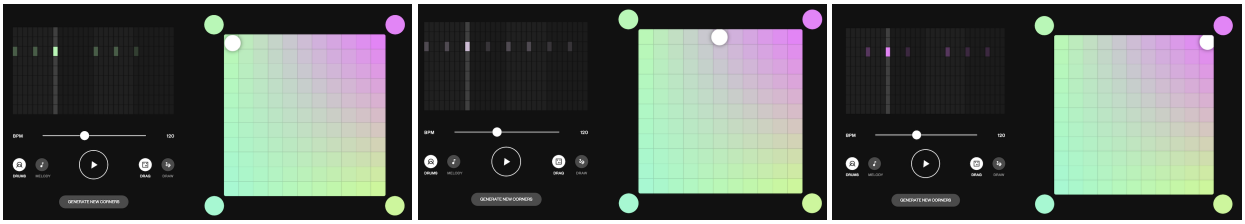

**Figure 5. Interpolation between playing "on" beats and playing "off" beats shows a moment of superposition, which we have observed with some other basic rhythmic patterns as well, and which has some natural interpretations.**

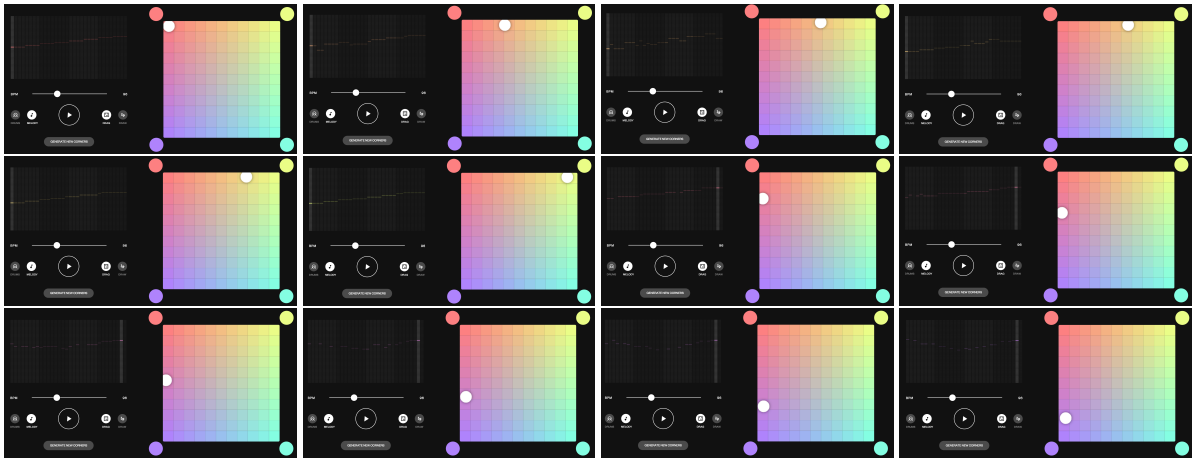

**Figure 6. Here we interpolate in melody space between four different scale patterns.**


9. Masatoshi Hamanaka, Keiji Hirata, and Satoshi Tojo. 2009. Melody morphing method based on GTTM. In *International Computer Music Conference (ICMC)*. 89–92. **http://hdl.handle.net/2027/spo.bbp2372.2009.020**
10. Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780. DOI: **http://dx.doi.org/10.1162/neco.1997.9.8.1735**
11. IEC 60908 1987. *Audio recording - Compact disc digital audio system* . Standard. International Electrotechnical Commission (IEC).
12. Tristan Jehan. 2001. *Perceptual synthesis engine: an audio-driven timbre generator*. Master's thesis. Massachusetts Institute of Technology (MIT). **http://hdl.handle.net/1721.1/61543**
13. Diederik P. Kingma and Max Welling. 2013. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations (ICLR)*. **http://arxiv.org/abs/1312.6114**
14. Ian Loh and Tom White. 2017. TopoSketch: Drawing in Latent Space. In *NIPS Workshop on Machine Learning for Creativity and Design*. **https://nips2017creativity.github.io/doc/TopoSketch.pdf**
15. Yotam Mann. 2015. Interactive Music with Tone.js. In *Web Audio Conference (WAC)*. **http://wac.ircam.fr/pdf/wac15_submission_40.pdf**
16. Parag Mital. 2017. Generate your own sounds with NSynth. **https://magenta.tensorflow.org/nsynth-fastgen**. (2017). Accessed: 2017-12-21.
17. Donald A. Norman. 2002. *The Design of Everyday Things*. Basic Books, Inc., New York, NY, USA.
18. Sageev Oore. 2005. Learning Advanced Skills on New Instruments. In *International Conference on New Interfaces for Musical Expression (NIME)*. Vancouver, Canada.
19. Daniel V. Oppenheim. 1995. Demonstrating MMorph: A System for Morphing Music in Real-time. In *International Computer Music Conference (ICMC)*. 479–480.
20. Adam Roberts, Jesse Engel, and Douglas Eck. 2017. Hierarchical Variational Autoencoders for Music. In *NIPS Workshop on Machine Learning for Creativity and Design*. **https://nips2017creativity.github.io/doc/Hierarchical_Variational_Autoencoders_for_Music.pdf**
21. Daniel Smilkov, Nikhil Thorat, and Charles Nicholson. 2017. deeplearn.js: a hardware-accelerated machine intelligence library for the web. **https://deeplearnjs.org/**. (2017). Accessed: 2017-12-21.
22. Laurens van der Maaten, Eric Postma, and Jaap van den Herik. 2009. *Dimensionality reduction: A comparative review*. Technical Report. TiCC, Tilburg University.
23. Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. 2016. Generative visual manipulation on the natural image manifold. In *European Conference on Computer Vision (ECCV)*. **https://arxiv.org/abs/1609.03552**