
Learning a Latent Space of Multitrack Measures

Ian Simon, Adam Roberts, Colin Raffel, Jesse Engel, Curtis Hawthorne & Douglas Eck

Google Brain

{iansimon,adarob,craffel,jesseengel,fjord,deck}@google.com

Recent advances in machine learning have made it possible to train generative models which can accurately represent and generate many different types of objects such as images, sketches [15], and piano performances [37], to name a few. We give an overview of this field in Appendix A. Some of these models learn a *latent space*: a lower-dimensional representation that can be mapped to and from the object space. In this paper, we present a latent space model of individual measures of symbolic music with multi-instrument polyphony and dynamics. This latent space model allows us to perform a number of intuitive operations:

- Sample a measure from the prior distribution to generate novel music from scratch.
- Interpolate (i.e. slowly morph) between two measures in a semantically meaningful way.
- Apply transformations to an existing measure, e.g. “increase note density” or “add strings”.

The latent space model can also be augmented with additional conditioning variables, which we demonstrate with chords. Chord conditioning allows us to perform the above operations while holding chords constant or to change chords while keeping musical texture constant. Even though this model only represents individual measures and thus is incapable of generating long-term structure on its own, combining the latent space with chord conditioning makes it fairly easy to generate music with convincing long-term dependencies; e.g. a composer could pick a single point in the latent space and then decode that point (or slowly interpolate between two points) over a desired chord progression.

Our model is a variational autoencoder (VAE) [20], extending the architecture of MusicVAE [33] to handle up to 8 tracks and a variable number of events per track. Full details of our architecture are available in Appendix C.3. We use hierarchical LSTM neural networks for the encoder and decoder. Our bidirectional encoder consists of two levels. The first level independently consumes each of the 8 track event streams, concatenating the final outputs from the forward and backward directions to produce 8 track embeddings. The second level consumes these 8 embeddings, outputting a single embedding that is the concatenation of final states of the forward and backward directions. This embedding is then fed through two fully-connected layers parameterize the autoencoder’s latent distribution. The unidirectional decoder is also made up of two levels: The first level (called the “conductor” by Roberts et al.) is initialized by setting its state to be the latent vector and is then run for 8 steps with a null input, outputting 8 track embeddings. For each of the 8 tracks, the lower-level LSTM is initialized in the same manner as the conductor using one of the 8 track embeddings. The lower-level LSTM then produces the sequence for each track, including the choice of instrument as a MIDI program number. A diagram of our model is shown in Figure 1(a).

We further introduce chord conditioning on both the encoder and decoder, which encourages the model to “factor out” chord information from the latent representation. This allows us to control chords and other attributes independently, which supports both holding the “arrangement” of the sequence constant while changing the underlying chord progression and holding the chord progression constant while changing the arrangement. We encode a chord as a one-hot vector over 49 chord types: major, minor, augmented, and diminished triads for all 12 pitch classes, plus a “no-chord” value. This chord vector is appended to the model input at each encoding and decoding step and can vary between steps; as such we are able to model harmonic changes within a single measure. Since MIDI files do not contain chord annotations, we introduce a heuristic procedure for estimating chords described in Appendix C.8.

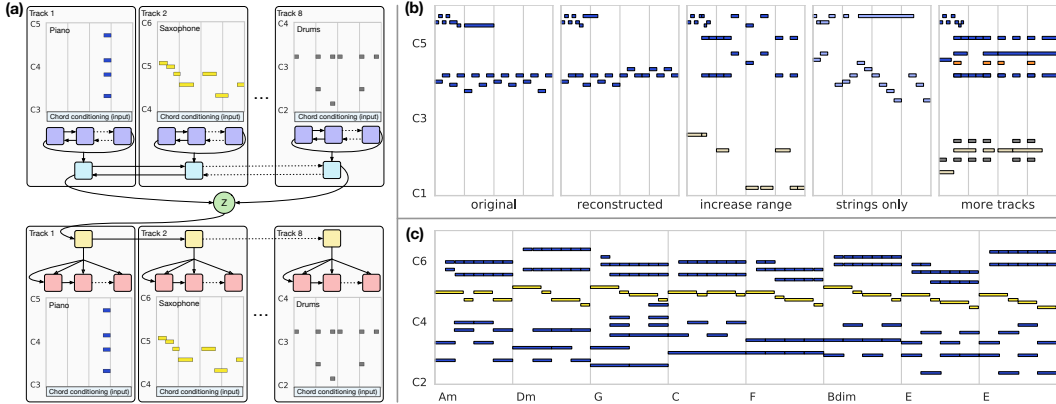


Figure 1: (a) Diagram of our multitrack MusicVAE model. (b) Examples of attribute manipulations. (c) Eight measures decoded from the same latent vector with different chord conditioning.

Our model is trained against the evidence lower bound (ELBO) of the data likelihood as is standard in VAEs. We give some background on this framework in Appendix C.1. To avoid posterior collapse, we use the “free bits” method of Kingma et al. [21]. A full description of the resulting loss function is provided in Appendix C.2. For specific details on the optimizer and training regime, refer to Appendix C.5. Our models are all trained on the Lakh MIDI Dataset [30], a collection of 176,581 MIDI files scraped from the web. A description of our data preprocessing is available in Appendix C.7. We consider measures with up to 8 tracks (see Figure 2 for an example with 4 tracks). A track is represented as a MIDI-like sequence of events from an extension of the vocabulary used by Simon and Oore [37] to handle metric timing and choice of instrument. Details about our MIDI representation can be found in Appendix B.

We now demonstrate several types of musical manipulations that can be performed via the latent space. Additional manipulations are described in Appendix D. Renderings of all figures can be heard at <https://goo.gl/s2N7dV>. Visit goo.gl/9SwfJC and goo.gl/s5MBXR for interactive demos.

Our latent space makes it possible to manipulate attributes of a note sequence. Given a particular attribute (e.g. note density), we can compute the difference between the mean latent vectors of the set of examples that have the attribute and the set of examples that do not to get an *attribute vector*. Then, given an example sequence that does not have the attribute, we can add the attribute by a) encoding the sequence, b) adding the attribute vector to the latent code, and c) decoding the translated latent code. As observed by Carter and Nielsen [6], this is a rather primitive way to learn such an attribute transformation, but often works in practice. Figure 1(b) shows several attribute transformations to an example measure: increasing the pitch range, using only string instruments, and using more instruments. Note that none of these transformations is performed in a straightforwardly mechanical way; indeed, there is often no mechanical way to perform an operation like “add more tracks”.

Figure 1(c) shows a single latent vector decoded under several different chords. Notice that the instrumental choice and rhythmic pattern remain fairly consistent, while the harmony changes. This allows us to concatenate multiple measures generated from a single latent vector to create a coherent multi-measure sequence. We find that this technique approximately matches the playing style of much popular modern music, where players shift a consistent rhythmic pattern—a “groove”—and modulate it over different repeating chord progressions. The model also naturally “vamps” by introducing small musically-related variations from the same latent vector and chord due to the autoregressive RNN sampling procedure. Figure 3 shows such two additional examples of decoding a fixed latent vector with different chord conditioning.

We have shown how to train and apply a latent space model over measures of symbolic music with multiple polyphonic instruments. We believe that ours is the first model capable of generating full multitrack polyphonic sequences with arbitrary instrumentation. On top of this, many natural musical operations are enabled by our latent space representation including interpolation, attribute manipulation, and (with side information) chord conditioning. Our source code is available at <https://github.com/tensorflow/magenta>.

Acknowledgements

We would like to thank Anna Huang and Erich Elsen for helpful reviews on drafts of the paper. SoundFont used in our audio examples created by John Nebauer.

References

- [1] Bharucha, J. and Todd, Peter M. Modeling the perception of tonal structure with neural nets. *Computer Music Journal*, 13(4):44–53, 1989.
- [2] Biolcati, Massimo. iReal Pro. *Technimo*, 2008. <https://irealpro.com/>
- [3] Boulanger-Lewandowski, Nicolas, Bengio, Yoshua, and Vincent, Pascal. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv:1206.6392*, 2012.
- [4] Briot, Jean-Pierre, Hadjeres, Gaëtan, and Pachet, François. Deep learning techniques for music generation—a survey. *arXiv:1709.01620*, 2017.
- [5] Brunner, Gino, Wang, Yuyi, Wattenhofer, Roger, and Wiesendanger, Jonas. JamBot: Music theory aware chord based generation of polyphonic music with LSTMs. *arXiv:1711.07682*, 2017.
- [6] Carter, Shan and Nielsen, Michael. Using artificial intelligence to augment human intelligence. *Distill*, 2017. doi: 10.23915/distill.00009. <https://distill.pub/2017/aia>
- [7] Chen, Chun-Chi J. and Miikkulainen, Risto. Creating melodies with evolving recurrent neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, 2001.
- [8] Chu, Hang, Urtasun, Raquel, and Fidler, Sanja. Song from PI: A musically plausible network for pop music generation. *CoRR*, abs/1611.03477, 2016. URL <http://arxiv.org/abs/1611.03477>.
- [9] Cope, David. *Computers and Musical Style*. Oxford University Press, 1991.
- [10] Dong, Hao-Wen, Hsiao, Wen-Yi, Yang, Li-Chia, and Yang, Yi-Hsuan. MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proc. AAAI*, 2018.
- [11] Eck, Douglas and Schmidhuber, Jürgen. Finding temporal structure in music: Blues improvisation with LSTM recurrent networks. In *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, 2002.
- [12] Engel, Jesse, Resnick, Cinjon, Roberts, Adam, Dieleman, Sander, Norouzi, Mohammad, Eck, Douglas, and Simonyan, Karen. Neural audio synthesis of musical notes with WaveNet autoencoders. In *Proc. ICML*, 2017.
- [13] Fernández, Jose D. and Vico, Francisco. AI methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research*, 2013.
- [14] Gannon, Peter. Band-in-a-Box. *PG Music*, 1990. <http://www.pgmusic.com/>
- [15] Ha, David and Eck, Douglas. A neural representation of sketch drawings. *arXiv:1704.03477*, 2017.
- [16] Hadjeres, Gaëtan and Pachet, François. DeepBach: a steerable model for Bach chorales generation. *arXiv:1612.01010*, 2016.
- [17] Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [18] Huang, Cheng-Zhi Anna, Cooijmans, Tim, Roberts, Adam, Courville, Aaron, and Eck, Douglas. Counterpoint by convolution. In *Proc. ISMIR*, 2017.

- [19] Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL <http://arxiv.org/abs/1412.6980>.
- [20] Kingma, Diederik P. and Welling, Max. Stochastic gradient VB and the variational auto-encoder. In *Proc. ICLR*, 2014.
- [21] Kingma, Diederik P., Salimans, Tim, Jozefowicz, Rafal, Chen, Xi, Sutskever, Ilya, and Welling, Max. Improved variational inference with inverse autoregressive flow. In *Proc. NIPS*, 2016.
- [22] Kirnberger, Johann Philipp. *Der allezeit fertige Polonaisen- und Menuettenkomponist*. Berlin, 1767.
- [23] Koren, Yehuda, Bell, Robert, and Volinsky, Chris. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.
- [24] Liang, Feynman, Gotham, Mark, Johnson, Matthew, and Shotton, Jamie. Automatic stylistic composition of Bach chorales with deep LSTM. In *Proc. ISMIR*, 2017.
- [25] Mao, Huanru Henry, Shin, Taylor, and Cottrell, Garrison W. DeepJ: Style-specific music generation. *arXiv:1801.00887*, 2018.
- [26] Masada, Kristen and Bunescu, Razvan C. Chord recognition in symbolic music using semi-Markov conditional random fields. In *Proc. ISMIR*, 2017. URL https://ismir2017.smcnus.org/wp-content/uploads/2017/10/224_Paper.pdf.
- [27] Menabrea, Luigi Federico and Augusta Ada King-Noel, Countess of Lovelace. *Sketch of the Analytical Engine Invented by Charles Babbage, Esq.* Richard and John E. Taylor, 1843.
- [28] Mozer, Michael C. Connectionist music composition based on melodic, stylistic and psychophysical constraints. *Music and Connectionism*, 1991.
- [29] Papadopoulos, George and Wiggins, Geraint. AI methods for algorithmic composition: A survey, a critical view and future prospects. In *AISB Symposium on Musical Creativity*. Edinburgh, UK, 1999.
- [30] Raffel, Colin. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. Columbia University, 2016.
- [31] Raffel, Colin and Ellis, Daniel P. W. Intuitive analysis, creation and manipulation of MIDI data with pretty_midi. In *ISMIR Late Breaking and Demo Papers*, 2014.
- [32] Raffel, Colin and Ellis, Daniel PW. Extracting ground-truth information from MIDI files: A MIDIfesto. In *Proceedings of the 17th International Society for Music Information Retrieval Conference*, 2016.
- [33] Roberts, Adam, Engel, Jesse, Raffel, Colin, Hawthorne, Curtis, and Eck, Douglas. A hierarchical latent vector model for learning long-term structure in music. *arXiv:1803.05428*, 2018.
- [34] Roure, David De, Willcox, Pip, and Weigl, David M. Numbers into notes: Cast your mind back 200 years. In *17th International Society for Music Information Retrieval Conference Late Breaking and Demo Papers*, 2016.
- [35] Roy, Pierre, Papadopoulos, Alexandre, and Pachet, François. Sampling variations of lead sheets. *CoRR*, abs/1703.00760, 2017.
- [36] Scholz, Ricardo and Ramalho, Geber. COCHONUT: Recognizing complex chords from MIDI guitar sequences. In *Proc. ISMIR*, 2008.
- [37] Simon, Ian and Oore, Sageev. Performance RNN: Generating music with expressive timing and dynamics. *Magenta*, 2017. <https://magenta.tensorflow.org/performance-rnn>
- [38] Vintch, Brett. A generative model for music track playlists. *iHeartRadio Tech Blog*, 2017. <https://tech.heart.com/a-generative-model-for-track-playlists-4dba8b8515c>

- [39] Viterbi, Andrew. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.
- [40] Wang, Yun-Sheng and Wechsler, Harry. Musical keys and chords recognition using unsupervised learning with infinite Gaussian mixture. In *Proceedings of the 2nd ACM International Conference on Multimedia Retrieval*, 2012.
- [41] White, Tom. Sampling generative networks: Notes on a few effective techniques. *arXiv:1609.04468*, 2016.
- [42] Yang, Li-Chia, Chou, Szu-Yu, and Yang, Yi-Hsuan. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. In *Proc. ISMIR*, 2017.

A Background and Related Work

We define “latent space models” as generative models which produce a lower-dimensional representation that can be mapped to and from the object space. A major advantage of latent space models is that many operations that would be difficult to perform in the object space, like morphing between two objects in a semantically meaningful way, become straightforward arithmetic in the latent space. It has even been claimed that latent space models can augment human understanding of the object domain [6]. Latent space models have already been trained for several musical concepts including raw waveforms of notes [12], melodies and drum tracks [33], and playlists [38]. Such models are also frequently used for music recommendations [23], where both user “taste” and song “style” are reasoned about in terms of latent vectors.

Our work builds on a long history of past efforts at symbolic music generation, plus more recent interest in latent spaces and modeling interplay between instruments. Algorithmic music generation has been a topic of interest for at least 200 years [22, 27, 34]. Prior to the recent neural network renaissance, most systems, such as that of Cope [9], used human-encoded rules, Markov models, or a few other method categories as described by Fernández and Vico [13] and Papadopoulos and Wiggins [29] in recent surveys.

The use of neural networks in symbolic music generation has seen a resurgence in interest, as surveyed by Briot et al. [4]. Early work on neural networks for symbolic music generation includes Bharucha and Todd [1], Mozer [28], Chen and Miikkulainen [7], and Eck and Schmidhuber [11]. One of the first effective neural network systems for generating polyphonic music is from Boulanger-Lewandowski et al. [3], who use a recurrent model over a pianoroll representation to generate classical and folk music. The pianoroll is a fairly standard representation for polyphonic music generation; we instead use an event-based representation closer to the MIDI standard itself.

Some work in music generation has been focused on specific domains. One such popular domain for polyphonic music generation is Bach chorales: DeepBach [16], CoCoNet [18], and BachBot [24] are all different generative models for polyphonic Bach chorales that can respond to user input. In contrast, our model presented in this paper works simultaneously across multiple Western music styles including classical, jazz, and pop/rock; essentially any music expressible with MIDI notes and program changes is compatible. Because of its latent space representation, our model is also able to interpolate between these different domains.

Other recent systems for generating polyphonic music include JamBot [5] which generates chords then notes in a two-step process, DeepJ [25] which generates polyphonic piano music where a user can control several style parameters, a model from Roy et al. [35] that generates variations on lead sheets, and Song from PI [8] which generates melody, chord, and drum tracks using a hierarchical recurrent network combined with hand-engineered features.

There are also commercial software systems such as PG Music’s Band-in-a-Box [14] and Technimo’s iReal Pro [2] that generate multi-instrumental music over user-specified chord progressions. However, these products appear to support a limited and fixed number of preset arrangement styles combined with rule-based modifications.

Perhaps the most similar systems to our current work are MusicVAE from Roberts et al. [33] (which we directly extend) and MuseGAN from Dong et al. [10] (which builds upon the work of Yang et al. [42]). MuseGAN [10] is based on generative adversarial networks (GANs) and is capable of modeling multiple instruments over multiple bars. Like our work, the system uses a latent space shared across tracks to handle interdependencies between instruments. However, in MuseGAN the set of instruments is a fixed quintet consisting of bass, drums, guitar, piano, and strings, whereas our system handles arbitrary instrument combinations. Separately, MuseGAN is focused on accompaniment and generation and is unable to represent or manipulate preexisting music. Our system can also generate from scratch, but in contrast with MuseGAN, it can also facilitate user-driven manipulation of existing music via the latent space.

The MusicVAE architecture introduced by Roberts et al. [33] is able to learn a latent space of musical sequences using a novel hierarchical decoder that allows it to model long-term structure and multi-instrument sequences. However, this work applies strict constraints to the sequences to reach its goals. In order to guarantee a constant number of events per measure, non-drum tracks are limited to monophonic sequences, and all tracks are represented with a single velocity and quantized at the

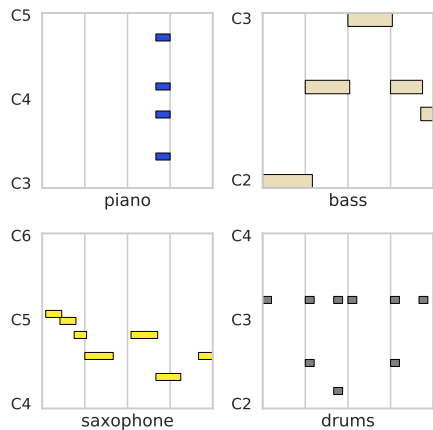


Figure 2: A single measure consisting of 4 tracks, shown as separate pianorolls. Each track is represented as a MIDI-like sequence of *note-on*, *note-off*, *time-shift*, and *velocity-change* events, with a single *program-select* event at the beginning and an *end-track* event at the end. For all other figures in this paper, multiple tracks are combined into a single pianoroll, color-coded by instrument family.

level of 16th notes. This reduces the challenge of modeling longer-term structure, but at the expense of expressiveness. Furthermore, while the “trio” MusicVAE is capable of modeling three broad instrument classes—melody, bass, and drums—it is limited to exactly three instruments, arbitrarily excluding potentially pivotal voices and disregarding the specific identity of each instrument (e.g. electric guitar and piano are both considered “melody” instruments). Further, Roberts et al. do not consider modeling fine-grained timing and velocity, nor do they develop a method for chord conditioning. Nevertheless, the MusicVAE architecture and its implementation provide a powerful basis for exploring a more expressive and complete multitrack latent space, and thus we position our work as an extension of it.

B Measure Representation

We model measures with up to 8 tracks (see Figure 2 for an example with 4 tracks). Each track consists of a single “instrument” as extracted by `pretty_midi` [31]. A track is represented as a MIDI-like sequence of events from an extension of the vocabulary used by Simon and Oore [37] to handle metric timing and choice of instrument:

- 128 **note-on** events, one for each MIDI pitch.
- 128 **note-off** events, one for each MIDI pitch.
- 8 **velocity-change** events, MIDI velocity quantized into 8 bins. These events set the velocity for subsequent note-on events.
- 96 **time-shift** events that shift the current time forward by the corresponding number of quantized time steps, where 24 steps is the length of a quarter note.
- 129 **program-select** events (128 programs plus drums) that set the MIDI program number at the beginning of each track.
- A single **end-track** event, used to mark the end of each track. For measures with fewer than 8 tracks, missing tracks consist solely of the end-track event.

For simplicity we only include measures with exactly 96 quantized time steps (4 quarter notes), as this is the most frequent measure size in our dataset.

C Model

C.1 Variational Autoencoders

An *autoencoder* is a model that learns to “compress” or encode objects to a lower-dimensional latent space and then decode these latent representations back into the original objects. Autoencoders are typically trained to optimize a *reconstruction loss* which measures how close the reconstructed

version is to the original object. This encourages the model to produce a compressed representation that captures the important variation among the objects.

The variational autoencoder extends the basic autoencoder in that it considers the latent representation z to be a *random variable* drawn from a *prior* distribution $p(z)$, usually a multivariate Gaussian with diagonal covariance. The encoder approximates the *posterior* distribution $p(z | x)$, while the decoder models the *likelihood* $p(x | z)$. The VAE is thus a generative model with the following generation process: generate a latent vector z from the prior distribution by sampling $z \sim p(z)$, then use the decoder to sample an output using the sampled z by $x \sim p(x | z)$.

In a variational autoencoder, the encoder and decoder are typically neural networks $q_\lambda(z | x)$ and $p_\theta(x | z)$ parameterized by θ and λ , respectively. In our case where we are dealing with sequences, both the encoder and decoder are hierarchical LSTM [17] models as in MusicVAE.

This architecture has two key benefits: First, it is a *latent variable model*; we can sample new measures and we can map existing measures to the latent space and transform them in various ways. Second, it is *hierarchical*; individual tracks are independent conditional on their embeddings. The use of track embeddings gives the model a way to represent complex dependencies between tracks, so that tracks will “fit together” when generated.

C.2 Loss Function

The VAE model optimizes a loss function that is the difference of two terms: the reconstruction loss, and the Kullback-Leibler (KL) divergence loss:

$$\mathbb{E} [\log p_\theta(x | z)] - D_{\text{KL}}(q_\lambda(z | x) \parallel p(z)) \quad (1)$$

The reconstruction loss term $\mathbb{E} [\log p_\theta(x | z)]$ maximizes the log-likelihood of the training data. In our model, the reconstruction loss is computed as the sum of the cross entropy between the predicted output distribution and the ground truth value over all events on all tracks in a given sequence. The KL divergence loss term $D_{\text{KL}}(q_\lambda(z | x) \parallel p(z))$ encourages $q_\lambda(z | x)$ (the distribution produced by the encoder) to be close to $p(z)$, the unit Gaussian prior.

One implication of optimizing a combination of loss terms is that there’s a core tradeoff between two desires:

1. *reconstruction*: The model should be able to faithfully represent measures from the training set in the latent space, such that these measures can be reproduced from their latent code.
2. *sampling*: The prior should be enforced; i.e. the posterior distributions for encoded measures from the training set should be close to the prior. This ensures that measures sampled from the prior are plausible.

We control this tradeoff by using the “free bits” method of Kingma et al. [21]; the KL loss term is allowed a budget τ bits of entropy per training example before it begins accruing loss. Increasing τ therefore improves reconstruction fidelity with the drawback of less realistic samples and semantically meaningless interpolation. We found $\tau = 64$ produced a good trade-off between reconstruction and sampling/interpolation in most cases. The only exception was in the attribute vector experiments, for which we found better results by using $\tau = 256$. Perhaps surprisingly, providing chords (see Appendix C.4) had little effect on reconstruction accuracy (for a given τ) even though samples from the chord-conditioned model do respect the chord conditioning.

C.3 Architecture

Here we give a brief overview of our model’s architecture; for specific details refer to [33] and our public source code¹. Our encoder consists of two levels of bidirectional LSTMs. The first level independently consumes each of the 8 track event streams, concatenating the final outputs from the forward and backward directions to produce 8 track embeddings. The second level consumes these 8 embeddings, outputting a single embedding that is the concatenation of final states of the forward and backward directions. This embedding is then fed through two fully-connected layers to produce the μ and, after a softplus activation, the σ parameter for the autoencoder’s latent distribution. A latent vector is then sampled from a multivariate Gaussian distribution with a diagonal covariance, parameterized by μ and σ .

¹<https://github.com/tensorflow/magenta>

The decoder is made up of two levels of unidirectional LSTMs. The first level (called the “conductor” by Roberts et al.) is initialized by setting its state to be the result of passing the latent vector through a linear layer with a tanh activation. This conductor LSTM is then run for 8 steps with a null input, outputting 8 track embeddings. For each of the 8 tracks, the lower-level LSTM is initialized in the same manner as the conductor using one of the 8 track embeddings. The initial input to the LSTM for each track is a zero vector concatenated with the track embedding, and subsequent inputs are the one-hot representation of the previous event concatenated with the track embedding. The outputs of the LSTM are then passed through a final softmax layer over the event vocabulary.

C.4 Conditioning

While the latent space allows for the manipulation of individual attributes, it is sometimes difficult to avoid introducing side effects on correlated attributes. By conditioning both the encoder and decoder on chords, we encourage the model to “factor out” chord information from the latent representation. This allows us to control chords and other attributes independently, which supports both holding the “arrangement” of the sequence constant while changing the underlying chord progression and holding the chord progression constant while changing the arrangement.

We encode a chord as a one-hot vector over 49 chord types: major, minor, augmented, and diminished triads for all 12 pitch classes, plus a “no-chord” value. This chord vector is appended to the model input at each encoding and decoding step and can vary between steps; as such we are able to model harmonic changes within a single measure.

C.5 Training

The model is trained with the Adam optimizer [19] using a batch size of 256. We anneal the learning rate from $1e-3$ to $1e-5$ with exponential decay rate 0.9999, for 100,000 gradient update steps. We use *teacher forcing* and feed the ground truth output value back to the model (instead of using its own output) at each sequence step during training.

For both levels of the model hierarchy (measure-tracks and track-events), we use a bidirectional LSTM encoder with 1024 nodes and a forward LSTM decoder with 3 layers of 512 nodes each. Our latent space has dimension 512.

C.6 Inference

Many strategies exist for producing a single output sequence from the softmax distributions produced by the LSTM outputs, including beam search and sampling autoregressively with a *temperature* parameter that controls the uniformity of the distribution. In all examples in this paper, we sample autoregressively with a temperature of 0.2 until an end token is returned.

C.7 Dataset

Our models are all trained on the Lakh MIDI Dataset [30], a collection of 176,581 MIDI files scraped from the web. The dataset is preprocessed as follows:

The dataset is first split into measures, and measures with length different from 4 quarter notes are discarded. Tracks are then extracted from each measure using the `pretty_midi` Python library; each track consists of notes with a single MIDI program number (or drums), though multiple tracks may use the same program number. Measures with fewer than 2 or more than 8 tracks are discarded. The tracks are then sorted by increasing program number, with drums at the end. Measures where any one track has more than 64 events (from the vocabulary in Appendix B) are discarded.

Finally, the measures are deduped, resulting in a training set of 4,092,681 examples. During training, each measure is augmented by transposing up or down within a minor third by an amount chosen uniformly at random; notes falling outside the valid MIDI pitch range are dropped. We perform this data augmentation step as the key distribution in the training data is far from uniform; around 50% of the data set is in C major or A minor.

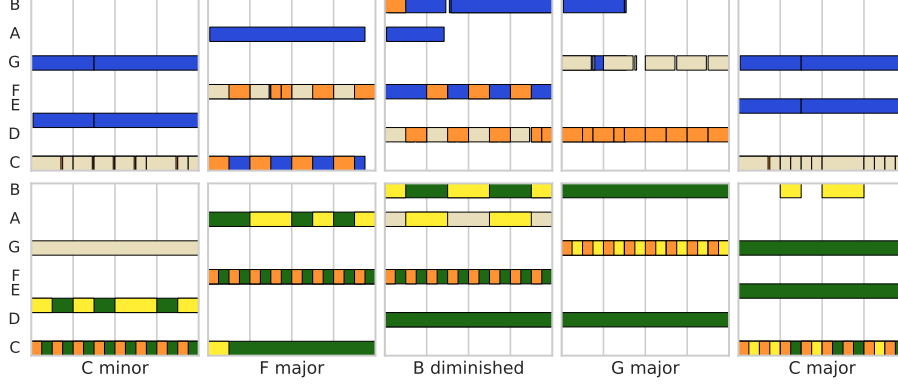


Figure 3: Two points in the latent space, each decoded over five different chords. Drums and pitch octave information have been removed from the pianorolls to show that the model is respecting the chord conditioning.

C.8 Chord Inference

When conditioning on chords, we would ideally train using ground-truth labels. Since MIDI files do not typically contain such labels [32], we automatically infer chord labels using a heuristic process.

First, each MIDI file is split into segments with a consistent tempo and time signature. For each segment, we infer chords at a frequency of 2 per measure using the Viterbi algorithm [39] over a heuristically-defined probability distribution; as a byproduct we also infer the time-varying key of the sequence. This algorithm takes time quadratic in the number of measures, so for efficiency we discard MIDI segments longer than 500 measures.

We infer 8 different chord types (major, minor, augmented, diminished, dominant-seventh, major-seventh, minor-seventh, and half-diminished) rooted at each of the 12 pitch classes plus a single “no-chord” designation, for a total of 97 chord classes. After chord inference is complete, the 8 chord types are projected down to the 4 triad types (49 total classes) used as model input.

Our chord inference computes the maximum-likelihood chord and key sequence over the following probability distribution on keys, chords, and notes:

$$p(h, y) = p(h_0)p(y_0 | h_0) \prod_{t=1}^n p(h_t | h_{t-1})p(y_t | h_t) \quad (2)$$

where h is the “harmony” sequence (key and chord at each step) and y is a sequence of unit-normalized pitch class vectors over the duration-weighted notes at each step.

For simplicity this heuristic approach was designed to minimize the number of parameters while penalizing key changes, chord changes, and key/chord/note pitch mismatches. Besides the chord change frequency of 2 per bar we use 4 other parameters:

- $\gamma = 0.5$, the probability of a chord change
- $\rho = 0.001$, the probability of a key change
- $\psi = 0.01$, the probability that a chord note will be drawn from outside the current key
- $\kappa = 100$, the “concentration” of the pitch class distribution under a chord; lower values are more forgiving of pitch mismatches

We define the harmony transition distribution as follows:

$$p(h_t | h_{t-1}) = \begin{cases} (1 - \gamma)(1 - \rho) & \text{if no change} \\ \gamma(1 - \rho)g(h_t, h_{t-1}) & \text{if chord change} \\ \frac{\rho}{11}f(h_t) & \text{if key change} \end{cases} \quad (3)$$

where $f(h_t)$ is a binomial distribution on the number of chord pitches belonging to the key, and

$$g(h_t, h_{t-1}) = f(h_t) + \frac{f(h_{t-1})}{48} \quad (4)$$

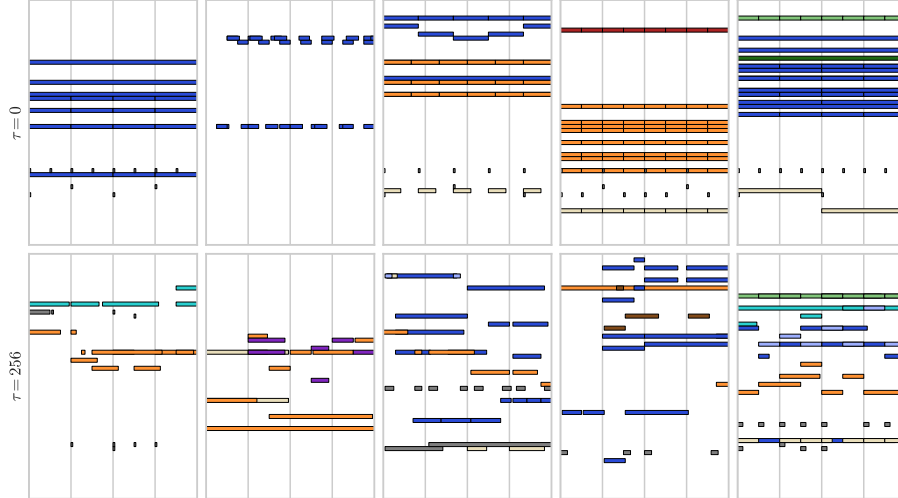


Figure 4: 5 different measures sampled from (top) a VAE model with 0 free bits and (bottom) a VAE model with 256 free bits.

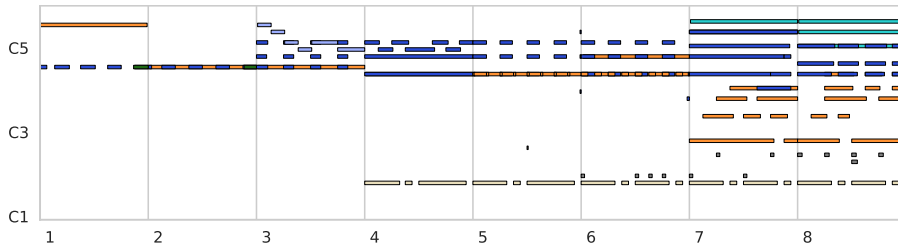


Figure 5: An interpolation between two measures generated by our model.

(11 is the number of keys minus the current key, and 48 is the number of chords minus the current chord.) The note observation distribution is defined as:

$$p(y_t | h_t) \sim \kappa \times (y_t \cdot c(h_t)) \quad (5)$$

where $c(h_t)$ is a unit-normalized vector representing the (uniformly-weighted) pitch classes in the chord for h_t .

More complex MIDI-to-chord techniques exist in the literature [36, 40, 26], but we find our heuristic approach satisfactory for model conditioning even though it ignores many relevant cues and likely makes basic errors. For example, even though our heuristic chord inference does not use the fact that the chord root is often played by the bass, the trained VAE model learns this pattern and usually generates bass parts that play the root.

D Additional Manipulations

Most straightforwardly, we can sample from the model. By sampling latent codes from the prior distribution and then feeding them through the decoder, we obtain new measures of multitrack music. Because our model uses a flexible representation and is trained on a large corpus, the samples it generates can be quite diverse. Some example sampled measures are shown in Figure 4.

As demonstrated by Roberts et al. [33], a latent space can be used to interpolate between two musical sequences in a more semantically meaningful way compared to naively blending the notes together. Given two measures x_0 and x_1 , we can interpolate between them by applying the encoder to obtain latent codes z_0 and z_1 , then for any $0 \leq \alpha \leq 1$ constructing z_α using spherical linear interpolation [41], as most of the probability mass of the Gaussian prior lies very close to the unit hypersphere. We then decode z_α into x_α to obtain the interpolated measure. Figure 5 shows an 8-step interpolation between two measures constructed in the above manner.