A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music

Adam Roberts [1] Jesse Engel [1] Colin Raffel [1] Curtis Hawthorne [1] Douglas Eck [1]

Abstract

The Variational Autoencoder (VAE) has proven to be an effective model for producing semantically meaningful latent representations for natural data. However, it has thus far seen limited application to sequential data, and, as we demonstrate, existing recurrent VAE models have difficulty modeling sequences with long-term structure. To address this issue, we propose the use of a *hierarchical* decoder, which first outputs embeddings for subsequences of the input and then uses these embeddings to generate each subsequence independently. This structure encourages the model to utilize its latent code, thereby avoiding the "posterior collapse" problem, which remains an issue for recurrent VAEs. We apply this architecture to modeling sequences of musical notes and find that it exhibits dramatically better sampling, interpolation, and reconstruction performance than a "flat" baseline model. An implementation of our "MusicVAE" is available online.[2]

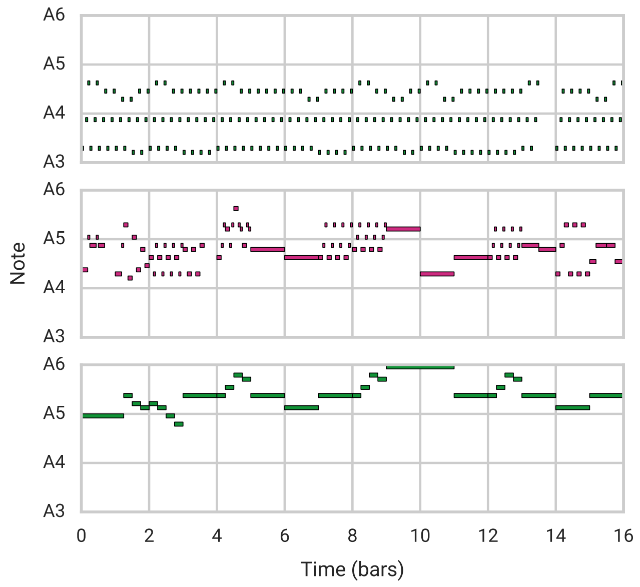

Figure 1. Demonstration of latent-space averaging using MusicVAE. The latent codes for the top and bottom sequences are averaged and decoded by our model to produce the middle sequence. The latent-space mean involves a similar repeating pattern to the top sequence, but in a higher register and with intermittent pauses like the bottom sequence. Audio for this example is available in the online supplement.[3] See Figs. 12 and 13 in Appendix E for a baseline comparison.

1. Introduction

Generative modeling describes the framework of estimating the underlying probability distribution $p(x)$ used to generate data x. This can facilitate a wide range of applications, from sampling novel datapoints to unsupervised representation learning to estimating the probability of an existing datapoint under the learned distribution. Much recent progress in generative modeling has been expedited by the use of deep neural networks, producing "deep generative models," which leverage the expressive power of deep networks to model complex and high-dimensional distributions. Practical achievements include generating realistic images with millions of pixels (Karras et al., 2017), generating synthetic audio with hundreds of thousands of timesteps (van den Oord et al., 2016a), and achieving state-of-the-art performance on semi-supervised learning tasks (Wei et al., 2018).

A wide variety of methods have been used in deep generative modeling, including implicit models such as Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) and explicit deep autoregressive models such as PixelCNN (van den Oord et al., 2016b) and WaveNet (van den Oord et al., 2016a). In this work, we focus on deep latent variable models such as the Variational Autoencoder (VAE) (Kingma & Welling, 2014; Rezende et al., 2014). The advantage of these models is that they explicitly model both $p(z|x)$ and $p(z)$, where z is a latent vector that can either be inferred from existing data or sampled from a distribution over the

[1]Google Brain, Mountain View, CA, USA. Correspondence to: Adam Roberts <adarob@google.com>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

[2]https://goo.gl/magenta/musicvae-code
[3]https://goo.gl/magenta/musicvae-examples

latent space. Ideally, the latent vector captures the pertinent characteristics of a given datapoint and disentangles factors of variation in a dataset. These autoencoders also model the likelihood $p(x|z)$, which provides an efficient way of mapping the latent vector back to the data space.

Our interest in deep latent variable models primarily comes from their increasing use in creative applications of machine learning (Carter & Nielsen, 2017; Ha & Eck, 2018; Engel et al., 2017a). This arises from surprising and convenient characteristics of the latent spaces typically learned by these models. For example, averaging the latent codes for all datapoints that possess a given attribute produces a so-called attribute vector, which can be used to make targeted changes to data examples. Encoding a datapoint with some attribute (say, a photograph of a person with brown hair) to obtain its latent code, subtracting the corresponding attribute vector (the "brown hair" vector), adding another attribute vector ("blond hair"), and decoding the resulting latent vector can produce a realistic manifestation of the initial point with the attributes swapped (the same person with blond hair) (Larsen et al., 2016; Mikolov et al., 2013). As another example, interpolating between latent vectors and decoding points on the trajectory can produce realistic intermediate datapoints that morph between the characteristics of the ends in a smooth and semantically meaningful way.

Most work on deep latent variable models has focused on continuous-valued data with a fixed dimensionality, e.g., images. Modeling sequential data is less common, particularly sequences of discrete tokens such as musical scores, which typically require the use of an autoregressive decoder. This is partially because autoregression is often sufficiently powerful that the autoencoder ignores the latent code (Bowman et al., 2016). While they have shown some success on short sequences (e.g., sentences), deep latent variable models have yet to be successfully applied to very long sequences.

To address this gap, we introduce a novel sequential autoencoder with a hierarchical recurrent decoder, which helps overcome the aforementioned issue of modeling long-term structure with recurrent VAEs. Our model encodes an entire sequence to a single latent vector, which enables many of the creative applications enjoyed by VAEs of images. We show experimentally that our model is capable of effectively autoencoding substantially longer sequences than a baseline model with a "flat" decoder RNN.

In this paper, we focus on the application of modeling sequences of musical notes. Western popular music exhibits strong long-term structure, such as the repetition and variation between measures and sections of a piece of music. This structure is also hierarchical–songs are divided into sections, which are broken up into measures, and then into beats, and so on. Further, music is fundamentally a multi-stream signal, in the sense that it often involves multiple players with strong inter-player dependencies. These unique properties, in addition to the potential creative applications, make music an ideal testbed for our sequential autoencoder.

After covering a background of work our approach builds on, we describe our model and its novel architectural enhancements. We then provide an overview of related work on applying latent variable models to sequences. Finally, we demonstrate the ability of our method to model musical data through quantitative and qualitative evaluations.

2. Background

Fundamentally, our model is an autoencoder, i.e., its goal is to accurately reconstruct its inputs. However, we additionally desire the ability to draw novel samples and perform latent-space interpolations and attribute vector arithmetic. For these properties, we adopt the framework of the Variational Autoencoder. Successfully using VAEs for sequences benefits from some additional extensions to the VAE objective. In the following subsections, we cover the prior work that forms the backbone for our approach.

2.1. Variational Autoencoders

A common constraint applied to autoencoders is that they compress the relevant information about the input into a lower-dimensional latent code. Ideally, this forces the model to produce a compressed representation that captures important factors of variation in the dataset. In pursuit of this goal, the Variational Autoencoder (Kingma & Welling, 2014; Rezende et al., 2014) introduces the constraint that the latent code z is a random variable distributed according to a prior $p(z)$. The data generation model is then $z \sim p(z), x \sim p(x|z)$. The VAE consists of an encoder $q_\lambda(z|x)$, which approximates the posterior $p(z|x)$, and a decoder $p_\theta(x|z)$, which parameterizes the likelihood $p(x|z)$. In practice, the approximate posterior and likelihood distributions ("encoder" and "decoder") are parameterized by neural networks with parameters λ and θ respectively. Following the framework of Variational Inference, we do posterior inference by minimizing the KL divergence between our approximate posterior, the encoder, and the true posterior $p(z|x)$ by maximizing the evidence lower bound (ELBO)

$$\mathbb{E}[\log p_\theta(x|z)] - \mathrm{KL}(q_\lambda(z|x)||p(z)) \le \log p(x) \tag{1}$$

where the expectation is taken with respect to $z \sim q_\lambda(z|x)$ and $\mathrm{KL}(\cdot||\cdot)$ is the KL-divergence. Naively computing the gradient through the ELBO is infeasible due to the sampling operation used to obtain z. In the common case where $p(z)$ is a diagonal-covariance Gaussian, this can be circumvented by replacing $z \sim \mathcal{N}(\mu, \sigma I)$ with

$$\epsilon \sim \mathcal{N}(0, I), z = \mu + \sigma \odot \epsilon \tag{2}$$

2.1.1. β-VAE and Free Bits

One way of interpreting the ELBO used in the VAE is by considering its two terms, $\mathbb{E}[\log p_\theta(x|z)]$ and $\mathrm{KL}(q_\lambda(z|x)||p(z))$, separately. The first can be thought of as requiring that $p(x|z)$ is high for samples of z from $q_\lambda(z|x)$–ensuring accurate reconstructions. The second encourages $q_\lambda(z|x)$ to be close to the prior–ensuring we can generate realistic data by sampling latent codes from $p(z)$. The presence of these terms suggests a trade-off between the quality of samples and reconstructions–or equivalently, between the rate (amount of information encoded in $q_\lambda(z|x)$) and distortion (data likelihood) (Alemi et al., 2017).

As is, the ELBO has no way of directly controlling this trade-off. A common modification to the ELBO introduces the KL weight hyperparameter β (Bowman et al., 2016; Higgins et al., 2017) producing

$$\mathbb{E}[\log p_\theta(x|z)] - \beta\, \mathrm{KL}(q_\lambda(z|x)||p(z)) \tag{3}$$

Setting $\beta < 1$ encourages the model to prioritize reconstruction quality over learning a compact representation.

Another approach for adjusting this trade-off is to only enforce the KL regularization term once it exceeds a threshold (Kingma et al., 2016):

$$\mathbb{E}[\log p_\theta(x|z)] - \max(\mathrm{KL}(q_\lambda(z|x)||p(z)) - \tau, 0) \tag{4}$$

This stems from the interpretation that $\mathrm{KL}(q_\lambda(z|x)||p(z))$ measures the amount of information required to code samples from $p(z)$ using $q_\lambda(z|x)$. Utilizing this threshold therefore amounts to giving the model a certain budget of "free bits" to use when learning the approximate posterior. Note that these modified objectives no longer optimize a lower bound on the likelihood, but as is custom we still refer to the resulting models as "Variational Autoencoders."

2.1.2. Latent Space Manipulation

The broad goal of an autoencoder is to learn a compact representation of the data. For creative applications, we have additional uses for the latent space learned by the model (Carter & Nielsen, 2017; Roberts et al., 2018). First, given a point in latent space that maps to a realistic datapoint, nearby latent space points should map to datapoints that are semantically similar. By extrapolation, this implies that all points along a continuous curve connecting two points in latent space should be decodable to a series of datapoints that produce a smooth semantic interpolation in data space. Further, this requirement effectively mandates that the latent space is "smooth" and does not contain any "holes," i.e., isolated regions that do not map to realistic datapoints. Second, we desire that the latent space disentangles meaningful semantic groups in the dataset.

Ideally, these requirements should be satisfied by a VAE if the likelihood and KL divergence terms are both sufficiently small on held-out test data. A more practical test of these properties is to interpolate between points in the latent space and test whether the corresponding points in the data space are interpolated in a semantically meaningful way. Concretely, if z_1 and z_2 are the latent vectors corresponding to datapoints x_1 and x_2, then we can perform linear interpolation in latent space by computing

$$c_\alpha = \alpha z_1 + (1 - \alpha) z_2 \tag{5}$$

for $\alpha \in [0, 1]$. Our goal is satisfied if $p_\theta(x|c_\alpha)$ is a realistic datapoint for all α, $p_\theta(x|c_\alpha)$ transitions in a semantically meaningful way from $p_\theta(x|c_0)$ to $p_\theta(x|c_1)$ as we vary α from 0 to 1, and that $p_\theta(x|c_\alpha)$ is perceptually similar to $p_\theta(x|c_{\alpha+\delta})$ for small δ. Note that because the prior over the latent space of a VAE is a spherical Gaussian, samples from high-dimensional priors are practically indistinguishable from samples from the uniform distribution on the unit hypersphere (Huszár, 2017). In practice we therefore use spherical interpolation (White, 2016) instead of Eq. (5).

An additional test for whether our autoencoder will be useful in creative applications measures whether it produces reliable "attribute vectors." Attribute vectors are computed as the average latent vector for a collection of datapoints that share some particular attribute. Typically, attribute vectors are computed for pairs of attributes, e.g., images of people with and without glasses. The model's ability to discover attributes is then tested by encoding a datapoint with attribute A, subtracting the "attribute A vector" from its latent code, adding the "attribute B vector", and testing whether the decoded result appears like the original datapoint with attribute B instead of A. In our experiments, we use the above latent space manipulation techniques to demonstrate the power of our proposed model.

2.2. Recurrent VAEs

While a wide variety of neural network structures have been considered for the encoder and decoder network in a VAE, in the present work we are most interested in models with a recurrent encoder and decoder (Bowman et al., 2016). Concretely, the encoder, $q_\lambda(z|x)$, is a recurrent neural network (RNN) that processes the input sequence $x = \{x_1, x_2, \ldots, x_T\}$ and produces a sequence of hidden states $h_1, h_2, \ldots, h_T$. The parameters of the distribution over the latent code z are then set as a function of h_T. The decoder, $p_\theta(x|z)$, uses the sampled latent vector z to set the initial state of a decoder RNN, which autoregressively produces the output sequence $y = \{y_1, y_2, \ldots, y_T\}$. The model is trained both to reconstruct the input sequence (i.e., $y_i = x_i, i \in \{1, \ldots, T\}$) and to learn an approximate posterior $q_\lambda(z|x)$ close to the prior $p(z)$, as in a standard VAE.

There are two main drawbacks of this approach. First, RNNs

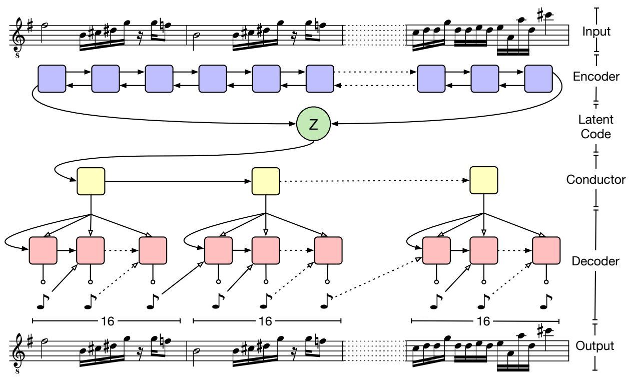

Figure 2. Schematic of our hierarchical recurrent Variational Autoencoder model, MusicVAE.

are themselves typically used on their own as powerful autoregressive models of sequences. As a result, the decoder in a recurrent VAE is itself sufficiently powerful to produce an effective model of the data and may disregard the latent code. With the latent code ignored, the KL divergence term of the ELBO can be trivially set to zero, despite the fact that the model is effectively no longer acting as an autoencoder. Second, the model must compress the entire sequence to a single latent vector. While this approach has been shown to work for short sequences (Bowman et al., 2016; Sutskever et al., 2014), it begins to fail as the sequence length increases (Bahdanau et al., 2015). In the following section, we present a latent variable autoencoder that overcomes these issues by using a hierarchical RNN for the decoder.

3. Model

At a high level, our model follows the basic structure used in previously-proposed VAEs for sequential data (Bowman et al., 2016). However, we introduce a novel hierarchical decoder, which we demonstrate produces substantially better performance on long sequences in Section 5. A schematic of our model, which we dub "MusicVAE," is shown in Fig. 2.

3.1. Bidirectional Encoder

For the encoder $q_\lambda(z|x)$, we use a two-layer bidirectional LSTM network (Hochreiter & Schmidhuber, 1997; Schuster & Paliwal, 1997). We process an input sequence $\mathbf{x} = \{x_1, x_2, \ldots, x_T\}$ to obtain the final state vectors $\overrightarrow{h}_T, \overleftarrow{h}_T$ from the second bidirectional LSTM layer. These are then concatenated to produce h_T and fed into two fully-connected layers to produce the latent distribution parameters μ and σ:

$$\mu = W_{h\mu}h_T + b_\mu \tag{6}$$

$$\sigma = \log\left(\exp(W_{h\sigma}h_T + b_\sigma) + 1\right) \tag{7}$$

where $W_{h\mu}, W_{h\sigma}$ and b_μ, b_σ are weight matrices and bias vectors, respectively. In our experiments, we use an LSTM state size of 2048 for all layers and 512 latent dimensions. As is standard in VAEs, μ and σ then parametrize the latent distribution as in Eq. (2). The use of a bidirectional recurrent encoder ideally gives the parametrization of the latent distribution longer-term context about the input sequence.

3.2. Hierarchical Decoder

In prior work, the decoder in a recurrent VAE is typically a simple stacked RNN. The decoder RNN uses the latent vector z to set its initial state, and proceeds to generate the output sequence autoregressively. In preliminary experiments (discussed in Section 5), we found that using a simple RNN as the decoder resulted in poor sampling and reconstruction for long sequences. We believe this is caused by the vanishing influence of the latent state as the output sequence is generated.

To mitigate this issue, we propose a novel hierarchical RNN for the decoder. Assume that the input sequence (and target output sequence) $\mathbf{x}$ can be segmented into U nonoverlapping subsequences y_u with endpoints i_u so that

$$y_u = \{x_{i_u}, x_{i_u+1}, x_{i_u+2}, \ldots, x_{i_{u+1}-1}\} \tag{8}$$

$$\rightarrow \mathbf{x} = \{y_1, y_2, \ldots, y_U\} \tag{9}$$

where we define the special case of $i_{U+1} = T$. Then, the latent vector z is passed through a fully-connected layer[4] followed by a $\tanh$ activation to get the initial state of a "conductor" RNN. The conductor RNN produces U embedding vectors $\mathbf{c} = \{c_1, c_2, \ldots, c_U\}$, one for each subsequence. In our experiments, we use a two-layer unidirectional LSTM for the conductor with a hidden state size of 1024 and 512 output dimensions.

Once the conductor has produced the sequence of embedding vectors $\mathbf{c}$, each one is individually passed through a shared fully-connected layer followed by a $\tanh$ activation to produce initial states for a final bottom-layer decoder RNN. The decoder RNN then autoregressively produces a sequence of distributions over output tokens for each subsequence y_u via a softmax output layer. At each step of the bottom-level decoder, the current conductor embedding c_u is concatenated with the previous output token to be used as the input. In our experiments, we used a 2-layer LSTM with 1024 units per layer for the decoder RNN.

In principle, our use of an autoregressive RNN decoder still allows for the "posterior collapse" problem where the model effectively learns to ignore the latent state. Simliar to (Chen et al., 2017), we find that it is important to limit the scope of the decoder to force it to use the latent code to model long-term structure. For a CNN decoder, this is as simple as reducing the receptive field, but no direct analogy exists

[4]Throughout, whenever we refer to a "fully-connected layer," we mean a simple affine transformation as in Eq. (6).

for RNNs, which in principle have an unlimited temporal receptive field. To get around this, we reduce the effective scope of the bottom-level RNN in the decoder by only allowing it to propagate state within an output subsequence. As described above, we initialize each subsequence RNN state with the corresponding embedding passed down by the conductor. This implies that the only way for the decoder to get longer-term context is by using the embeddings produced by the conductor, which in turn depend solely on the latent code. We experimented with an autoregressive version of the conductor where the decoder state was passed back to the conductor at the end of each subsequence, but found it exhibited worse performance. We believe that these combined constraints effectively force the model to utilize the conductor embeddings, and by extension the latent vector, in order to correctly autoencode the sequence.

3.3. Multi-Stream Modeling

Many common sources of sequential data, such as text, consist solely of a single "stream," i.e., there is only one sequence source which is producing tokens. However, music is often a fundamentally multi-stream signal–a given musical sequence may consist of multiple players producing notes in tandem. Modeling music therefore may also involve modeling the complex inter-stream dependencies.

We explore this possibility by introducing a "trio" model, which is identical to our basic MusicVAE except that it produces 3 separate distributions over output tokens–one for each of three instruments (drum, bass, and melody). In our hierarchical decoder model, we consider these separate streams as an orthogonal "dimension" of hierarchy, and use a separate decoder RNN for each instrument. The embeddings from the conductor RNN initialize the states of each instrument RNN through separate fully-connected layers followed by $\tanh$ activations. For our baseline with a "flat" (non-hierarchical) decoder, we use a single RNN and split its output to produce per-instrument softmaxes.

4. Related Work

A closely related model is the aforementioned recurrent VAE of (Bowman et al., 2016). Like ours, their model is effectively a VAE that uses RNNs for both the encoder and decoder. With careful optimization, (Bowman et al., 2016) demonstrate the ability to generate and interpolate between sentences which have been modeled at the character level. A very similar model was also proposed by (Fabius & van Amersfoort, 2015), which was applied with limited success to music. This approach was also extended to utilize a convolutional encoder and decoder with dilated convolutions in (Yang et al., 2017). The primary difference between these models and ours is the decoder architecture; namely, we use a hierarchical RNN. The flat RNN decoder we use as a baseline in Section 5 exhibits significantly degraded performance when dealing with very long sequences.

Various additional VAE models with autoregressive decoders have also been proposed. (Semeniuta et al., 2017) consider extensions of the recurrent VAE where the RNNs are replaced with feed-forward and convolutional networks. The PixelVAE (Gulrajani et al., 2017) marries a VAE with a PixelCNN (van den Oord et al., 2016b) and applies the result to the task of natural image modeling. Similarly, the Variational Lossy Autoencoder (Chen et al., 2017) combines a VAE with a PixelCNN/PixelRNN decoder. The authors also consider limiting the power of the decoder and using a more expressive Inverse Autoregressive Flow (Kingma et al., 2016) prior on the latent codes. Another example of a VAE with a recurrent encoder and decoder is SketchRNN (Ha & Eck, 2018), which successfully models sequences of continuously-valued pen coordinates.

The hierarchical paragraph autoencoder proposed in (Li et al., 2015) has several parallels to our work. They also consider an autoencoder with hierarchical RNNs for the encoder and decoder, where each level in the hierarchy corresponds to natural subsequences in text (e.g., sentences and words). However, they do not impose any constraints on the latent code, and as a result are unable to sample or interpolate between sequences. Our model otherwise differs in its use of a flat bidirectional encoder and lack of autoregressive connections in the first level of the hierarchy.

More broadly, our model can be considered in the sequence-to-sequence framework (Sutskever et al., 2014), where an encoder produces a compressed representation of an input sequence which is then used to condition a decoder to generate an output sequence. For example, the NSynth model learns embeddings by compressing audio waveforms with a downsampling convolutional encoder and then reconstructing audio with a WaveNet-style decoder (Engel et al., 2017b). Recurrent sequence-to-sequence models are most often applied to sequence *transduction* tasks where the input and output sequences are different. Nevertheless, sequence-to-sequence autoencoders have been occasionally considered, e.g., as an auxiliary unsupervised training method for semi-supervised learning (Dai & Le, 2015) or in the paragraph autoencoder described above. Again, our approach differs in that we impose structure on the compressed representation (our latent vector) so that we can perform sampling and interpolation.

Finally, there have been many recurrent models proposed where the recurrent states are themselves stochastic latent variables with dependencies across time (Chung et al., 2015; Bayer & Osendorfer, 2014; Fraccaro et al., 2016). A particularly similar example to our model is that of (Serban et al., 2017), which also utilizes a hierarchical encoder and decoder. Their model uses two levels of hierarchy and generates a stochastic latent variable for each subsequence of the input sequence. The crucial difference between this class of models and ours is that we use a *single* latent variable to represent the entire sequence, which allows for creative applications such as interpolation and attribute manipulation.

5. Experiments

To demonstrate the power of the MusicVAE, we carried out a series of quantitative and qualitative studies on music data. First, we demonstrate that a simple recurrent VAE like the one described in (Bowman et al., 2016) can effectively generate and interpolate between short sequences of musical notes. Then, we move to significantly longer note sequences, where our novel hierarchical decoder is necessary in order to effectively model the data. To verify this assertion, we provide quantitative evidence that it is able to reconstruct, interpolate between, and model attributes from data significantly better than the baseline. We conclude with a series of listening studies which demonstrate that our proposed model also produces a dramatic improvement in the perceived quality of samples.

5.1. Data and Training

For our data source, we use MIDI files, which are a widely-used digital score format. MIDI files contain instructions for the notes to be played on each individual instrument in a song, as well as meter (timing) information. We collected $\approx$1.5 million unique files from the web, which provided ample data for training our models. We extracted the following types of training data from these MIDI files: 2- and 16-bar melodies (monophonic note sequences), 2- and 16-bar drum patterns (events corresponding to playing different drums), and 16-bar "trio" sequences consisting of separate streams of a melodic line, a bass line, and a drum pattern. For further details on our dataset creation process, refer to Appendix A. For ease of comparison, we also evaluated reconstruction quality (as in Section 5.3 below) on the publicly available Lakh MIDI Dataset (Raffel, 2016) in Appendix B.

We modeled the monophonic melodies and basslines as sequences of 16th note events. This resulted in a 130-dimensional output space (categorical distribution over tokens) with 128 "note-on" tokens for the 128 MIDI pitches, plus single tokens for "note-off" and "rest". For drum patterns, we mapped the 61 drum classes defined by the General MIDI standard (International MIDI Association, 1991) to 9 canonical classes and represented all possible combinations of hits with 512 categorical tokens. For timing, in all cases we quantized notes to 16th note intervals, such that each bar consisted of 16 events. As a result, our two-bar data (used as a proof-of-concept with a flat decoder) resulted in sequences with $T = 32$ and 16-bar data had $T = 256$. For our hierarchical models, we use $U = 16$, meaning each subsequence corresponded to a single bar.

All models were trained using Adam (Kingma & Ba, 2014) with a learning rate annealed from 10^{-3} to 10^{-5} with exponential decay rate 0.9999 and a batch size of 512. The 2- and 16-bar models were run for 50k and 100k gradient updates, respectively. We used a cross-entropy loss against the ground-truth output with scheduled sampling (Bengio et al., 2015) for 2-bar models and teacher forcing for 16-bar models.

5.2. Short Sequences

As a proof that modeling musical sequences with a recurrent VAE is possible, we first tried modeling 2-bar ($T = 32$) monophonic music sequences (melodies and drum patterns) with a flat decoder. The model was given a tolerance of 48 free bits ($\approx$33.3 nats) and had the KL cost weight, β, annealed from 0.0 to 0.2 with exponential rate 0.99999. Scheduled sampling was introduced with an inverse sigmoid rate of 2000.

We found the model to be highly accurate at reconstructing its input (Table 1 discussed below in Section 5.3). It was also able to produce compelling interpolations (Fig. 14, Appendix E) and samples. In other words, it learned to effectively use its latent code without suffering from posterior collapse or exposure bias, as particularly evidenced by the relatively small difference in teacher-forced and sampled reconstruction accuracy (a few percent).

Despite this success, the model was unable to reliably reconstruct 16-bar ($T = 256$) sequences. For example, the discrepancy between teacher-forced and sampled reconstruction accuracy increased by more than 27%. This motivated our design of the hierarchical decoder described in Section 3.2. In the following sections, we provide an extensive comparison of our proposed model to the flat baseline.

5.3. Reconstruction Quality

	Teacher-Forcing		Sampling	
Model	Flat	Hierarchical	Flat	Hierarchical
2-bar Drum	0.979	-	0.917	-
2-bar Melody	0.986	-	0.951	-
16-bar Melody	0.883	**0.919**	0.620	**0.812**
16-bar Drum	0.884	**0.928**	0.549	**0.879**
Trio (Melody)	0.796	**0.848**	0.579	**0.753**
Trio (Bass)	0.829	**0.880**	0.565	**0.773**
Trio (Drums)	0.903	**0.912**	0.641	**0.863**

Table 1. Reconstruction accuracies calculated both with teacher-forcing (i.e., next-step prediction) and full sampling. All values are reported on a held-out test set. A softmax temperature of 1.0 was used in all cases, meaning we sampled directly from the logits.

To begin, we evaluate whether the hierarchical decoder produces better reconstruction accuracy on 16-bar melodies and drum patterns. For 16-bar models, we give a tolerance of 256 free bits ($\approx$177.4 nats) and use $\beta = 0.2$. Table 1 shows the per-step accuracies for reconstructing the sequences in our test set. As mentioned above, we see signs of posterior collapse with the flat baseline, leading to reductions in accuracy of $\approx 27 - 32\%$ when teacher forcing is removed for inference. Our hierarchical decoder both increases the next-step prediction accuracy and further reduces the exposure bias by better learning to use its latent code. With the hierarchical model, the decrease in sampling accuracy versus teacher forcing only ranges between $\approx 5 - 11\%$. In general, we also find that the reconstruction errors made by our models are reasonable, e.g., notes shortened by a beat or the addition of notes in the appropriate key.

We also explored the performance of our models on our multi-stream "trio" dataset, consisting of 16-bar sequences of melody, bass, and drums. As with single-stream data, the hierarchical model was able to achieve higher accuracy than the flat baseline while exhibiting a much smaller gap between teacher-forced and sampled performance.

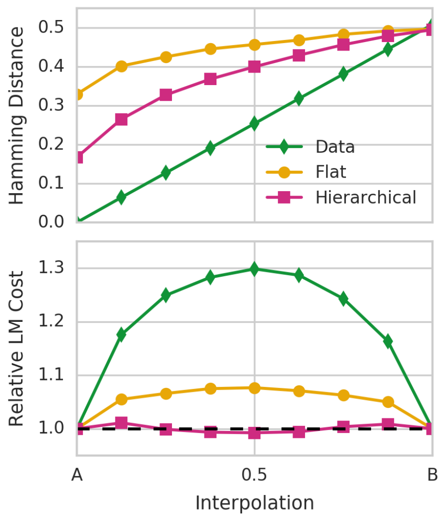

Figure 3. Latent-space interpolation results. All values are averaged over 1024 interpolated sequences. X-axis denotes interpolation between sequence A to B from left to right. *Top:* Sequence-normalized Hamming distance between sequence A and interpolated points. The distance from B is symmetric to A (decreasing as A increases) and is not shown. *Bottom:* Relative log probability according to an independently-trained 5-gram language model.

5.4. Interpolations

For creative purposes, we desire interpolations that are smoothly varying and semantically meaningful. In Fig. 3, we compare latent-space interpolations from a flat decoder (yellow circles) and hierarchical decoder (magenta squares) to a baseline of naive blending of the two sequences (green diamonds). We averaged the behavior of interpolating between 1024 16-bar melodies from the evaluation dataset (A) and 1024 other unique evaluation melodies (B), using a softmax temperature of 0.5 to sample the intermediate sequences. We constructed baseline "Data" interpolations by sampling a Bernoulli random variable with parameter α to choose an element from either sequence $\mathbf{a}$ or $\mathbf{b}$ for each time step, i.e., $p(x_t = b_t) = \alpha$, $p(x_t = a_t) = 1 - \alpha$.

The top graph of Fig. 3 shows that the (sequence length-normalized) Hamming distance, i.e., the proportion of timestep predictions that differ between the interpolation and sequence A, increases monotonically for all methods. The data interpolation varies linearly as expected, following the mean of the Bernoulli distribution. The Hamming distances also vary monotonically for latent space interpolations, showing that the decoded sequences morph smoothly to be less like sequence A and more like sequence B. For example, reconstructions don't remain on one mode for several steps and then jump suddenly to another. Samples have a non-zero Hamming distance at the endpoints because of imperfect reconstructions, and the hierarchical decoder has a lower intercept due to its higher reconstruction accuracy.

For the bottom graph of Fig. 3, we first trained a 5-gram language model on the melody dataset (Heafield, 2011). We show the normalized cost for each interpolated sequence given by $C_\alpha / (\alpha C_B + (1-\alpha) C_A)$, where C_α is the language model cost of an interpolated sequence with interpolation amount α, and C_A and C_B are the costs for the endpoint sequences A and B. The large hump for the data interpolation shows that interpolated sequences in data space are deemed by the language model to be much less probable than the original melodies. The flat model does better, but produces less coherent interpolated sequences than the hierarchical model, which produces interpolations of equal probability to the originals across the entire range of interpolation.

Fig. 1 shows two example melodies and their corresponding midpoint in MusicVAE space. The interpolation synthesizes semantic elements of the two melodies: a similar repeating pattern to A, in a higher register with intermittent sparsity like B, and in a new shared musical key. On the other hand, the baseline data interpolation just mixes the two resulting in harmonic and rhythmic dissonance (Fig. 12, Appendix E).

5.5. Attribute Vector Arithmetic

We can also exploit the structure of the latent space to use "attribute vectors" to alter the attributes of a given sequence. Apart from the score itself, MIDI contains limited annotations (Raffel & Ellis, 2016), so we defined five attributes which can be trivially computed directly from the note sequence: C diatonic membership, note density, average interval, and 16th and 8th note syncopation. See Appendix C for their full definitions. Ideally, computing the difference

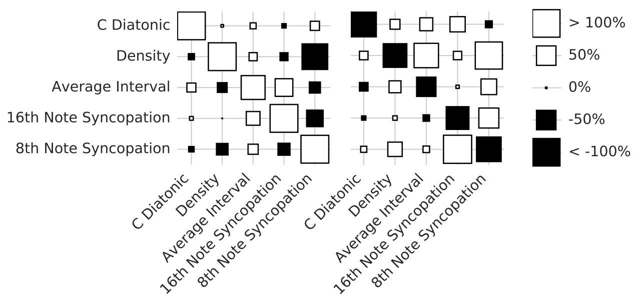

Figure 4. Adding (left) and subtracting (right) different attribute vectors in latent space and decoding the result produces the intended changes with few side effects. The vertical axis denotes the attribute vector applied and the horizontal axis denotes the attribute measured. See Figs. 6 and 8 to 11 (Appendix E) for example piano rolls and Appendix C for descriptions of each attribute.

between the average latent vector for sequences which exhibit the two extremes of each attribute and then adding or subtracting it from latent codes of existing sequences will produce the intended semantic manipulation.

To test this, we first measured these attributes in a set of 370k random training examples. For each attribute, we ordered the set by the amount of attribute exhibited, partitioned it into quartiles, and computed an attribute vector by subtracting the mean latent vector of the bottom quartile from the mean latent vector of the top quartile. We then sampled 256 random vectors from the prior, added and subtracted vectors for each attribute, and measured the average percentage change for all attributes against the sequence decoded from the unaltered latent code. The results are shown in Fig. 4.

In general, we find that applying a given attribute vector consistently produces the intended change to the targeted attribute. We also find cases where increasing one attribute decreases another (e.g. increasing density decreases 8th note syncopation), which we believe is largely because our heuristics capture overlapping characteristics. We are interested in evaluating attribute vector manipulations for labels that are non-trivial to compute (e.g., ornamentation, call/response, etc.) in future work.

5.6. Listening Tests

Capturing whether samples from our model sound realistic is difficult to do with purely quantitative metrics. To compare the perceived sample quality of the different models, we therefore carried out listening studies for melodies, trio compositions, and drum patterns.

Participants were presented with two 16-bar (≈30s) compositions that were either sampled from one of the models or extracted from our evaluation dataset. They were then asked which they thought was more musical on a Likert scale. For each study, 192 ratings were collected, with each source involved in 128 pair-wise comparisons. All samples were generated using a softmax temperature of 0.5.

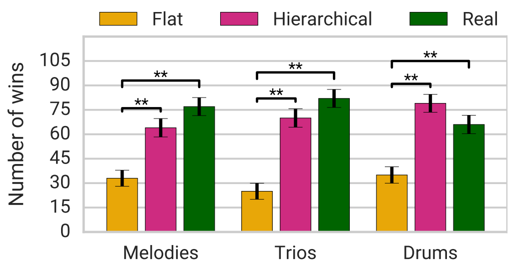

Figure 5. Results of our listening tests. Black error bars indicate estimated standard deviation of means. Double asterisks for a pair indicate a statistically significant difference in ranking.

Fig. 5 shows the number of comparisons in which a composition from each model was selected as more musical. Our listening test clearly demonstrates the improvement in sample quality gained by using a hierarchical decoder–in all cases the hierarchical model was preferred dramatically more often than the flat model and at the same rate as the evaluation data. In fact, the hierarchical drum model was preferred more often than real data, but the difference is not statistically significant. This was likely due to a listener bias towards variety, as the true drum data, while more realistic, was also more repetitive and perhaps less engaging.

Further, a Kruskal-Wallis H test of the ratings showed that there was a statistically significant difference between the models: $\chi^2(2) = 37.85, p < 0.001$ for melodies, $\chi^2(2) = 76.62, p < 0.001$ for trios, and $\chi^2(2) = 44.54, p < 0.001$ for drums. A post-hoc analysis using the Wilcoxon signed-rank test with Bonferroni correction showed that participants rated samples from the 3 hierarchical models as more musical than samples from their corresponding flat models with $p < 0.01/3$. Participants also ranked real data as more musical than samples from the flat models with $p < 0.01/3$. There was no significant difference between samples from the hierarchical models and real data.

Audio of some of the examples used in the listening tests is available in the online supplement.[3]

6. Conclusion

We proposed MusicVAE, a recurrent Variational Autencoder which utilizes a hierarchical decoder for improved modeling of sequences with long-term structure. In experiments on music data, we thoroughly demonstrated through quantitative and qualitative experiments that our model achieves substantially better performance than a flat baseline. In future work, we are interested in testing our model on other types of sequential data. To facilitate future research on recurrent latent variable models, we make our code and pre-trained models publicly available.[2]

Acknowledgements

The authors wish to thank David Ha for inspiration and guidance. We thank Claire Kayacik and Cheng-Zhi Anna Huang for their assistance with the user study analysis. We thank Erich Elsen for additional editing.

References

Alemi, A. A., Poole, B., Fischer, I., Dillon, J. V., Saurous, R. A., and Murphy, K. An information-theoretic analysis of deep latent-variable models. *arXiv preprint arXiv:1711.00464*, 2017.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *Third International Conference on Learning Representations*, 2015.

Bayer, J. and Osendorfer, C. Learning stochastic recurrent networks. In *NIPS Workshop on Advances in Variational Inference*, 2014.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, 2015.

Bowman, S., Vilnis, L., Vinyals, O., Dai, A., Jozefowicz, R., and Bengio, S. Generating sentences from a continuous space. In *Proceedings of the Twentieth Conference on Computational Natural Language Learning*, 2016.

Carter, S. and Nielsen, M. Using Artificial Intelligence to Augment Human Intelligence. *Distill*, 2017.

Chen, X., Kingma, D. P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., and Abbeel, P. Variational lossy autoencoder. In *Fifth International Conference on Learning Representations*, 2017.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A., and Bengio, Y. A recurrent latent variable model for sequential data. In *Advances in Neural Information Processing Systems*, pp. 2980–2988, 2015.

Dai, A. M. and Le, Q. V. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, 2015.

Engel, J., Hoffman, M., and Roberts, A. Latent constraints: Learning to generate conditionally from unconditional generative models. *arXiv preprint arXiv:1711.05772*, 2017a.

Engel, J., Resnick, C., Roberts, A., Dieleman, S., Eck, D., Simonyan, K., and Norouzi, M. Neural audio synthesis of musical notes with wavenet autoencoders. In *International Conference on Machine Learning*, 2017b.

Fabius, O. and van Amersfoort, J. R. Variational recurrent auto-encoders. In *Third International Conference on Learning Representations*, 2015.

Fraccaro, M., Sønderby, S. K., Paquet, U., and Winther, O. Sequential neural models with stochastic layers. In *Advances in Neural Information Processing Systems*, pp. 2199–2207, 2016.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

Gulrajani, I., Kumar, K., Ahmed, F., Taiga, A. A., Visin, F., Vazquez, D., and Courville, A. PixelVAE: A latent variable model for natural images. In *Fifth International Conference on Learning Representations*, 2017.

Ha, D. and Eck, D. Oph a neural representation of sketch drawings. In *Sixth International Conference on Learning Representations*, 2018.

Heafield, K. Kenlm: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pp. 187–197. Association for Computational Linguistics, 2011.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. beta-vae: Learning basic visual concepts with a constrained variational framework. In *Fifth International Conference on Learning Representations*, 2017.

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

Huszár, F. Gaussian distributions are soap bubbles. *inFERENCe*, 2017.

International MIDI Association, T. General MIDI level 1 specification. 1991.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2014.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *Second International Conference on Learning Representations*, 2014.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, 2016.

Larsen, A. B. L., Sønderby, S. K., Larochelle, H., and Winther, O. Autoencoding beyond pixels using a learned similarity metric. In *International Conference on Machine Learning*, 2016.

Li, J., Luong, M.-T., and Jurafsky, D. A hierarchical neural autoencoder for paragraphs and documents. In *ACL*, 2015.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 2013.

Raffel, C. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. PhD thesis, Columbia University, 2016.

Raffel, C. and Ellis, D. P. W. Extracting ground-truth information from MIDI files: A MIDIfesto. In *Proceedings of the 17th International Society for Music Information Retrieval Conference*, 2016.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, 2014.

Roberts, A., Engel, J., Oore, S., and Eck, D. (eds.). *Learning Latent Representations of Music to Generate Interactive Musical Palettes*, 2018.

Schuster, M. and Paliwal, K. K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Semeniuta, S., Severyn, A., and Barth, E. A hybrid convolutional variational autoencoder for text generation. *arXiv preprint arXiv:1702.02390*, 2017.

Serban, I. V., Sordoni, A., Lowe, R., Charlin, L., Pineau, J., Courville, A., and Bengio, Y. A hierarchical latent variable encoder-decoder model for generating dialogues. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, 2014.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016a.

van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. Conditional image generation with PixelCNN decoders. In *Advances in Neural Information Processing Systems*, 2016b.

Wei, X., Liu, Z., Wang, L., and Gong, B. Improving the improved training of Wasserstein GANs. In *Sixth International Conference on Learning Representations*, 2018.

White, T. Sampling generative networks: Notes on a few effective techniques. *arXiv preprint arXiv:1609.04468*, 2016.

Yang, Z., Hu, Z., Salakhutdinov, R., and Berg-Kirkpatrick, T. Improved variational autoencoders for text modeling using dilated convolutions. In *International Conference on Machine Learning*, 2017.

A. Dataset Creation Details

The datasets were built by first searching the web for publicly-available MIDI files, resulting in $\approx$1.5 million unique files. We removed those that were identified as having a non-4/4 time signature and used the encoded tempo to determine bar boundaries, quantizing to 16 notes per bar (16th notes).

For the 2-bar (16-bar) drum patterns, we used a 2-bar (16-bar) sliding window (with a stride of 1 bar) to extract all unique drum sequences (channel 10) with at most a single bar of consecutive rests, resulting in 3.8 million (11.4 million) examples.

For 2-bar (16-bar) melodies, we used a 2-bar (16-bar) sliding window (with a stride of 1 bar) to extract all unique monophonic sequences with at most a single bar of consecutive rests, resulting in 28.0 million (19.5 million) unique examples.

For the trio data, we used a 16-bar sliding window (with a stride of 1 bar) to extract all unique sequences containing an instrument with a program number in the piano, chromatic percussion, organ, or guitar interval, [0, 31], one in the bass interval, [32, 39], and one that is a drum (channel 10), with at most a single bar of consecutive rests in any instrument. If there were multiple instruments in any of the three categories, we took the cross product to consider all possible combinations. This resulted in 9.4 million examples.

In all cases, we reserved a held-out evaluation set of examples which we use to report reconstruction accuracy, interpolation results, etc.

B. Lakh MIDI Dataset Results

For easier comparison, we also trained our 16-bar models on the publicly available Lakh MIDI Dataset (LMD) (Raffel, 2016), which makes up a subset of the our dataset described above. We extracted 3.7 million melodies, 4.6 million drum patterns, and 116 thousand trios from the full LMD. The models were trained with the same hyperparameters as were used for the full dataset.

We first evaluated the LMD-trained melody model on a subset of the full evaluation set made by excluding any examples in the LMD train set. We found less than a 1% difference in reconstruction accuracies between the LMD-trained and original model.

In Table 2 we report the reconstruction accuracies for all 3 16-bar models trained and evaluated on LMD. While the accuracies are slightly higher than Table 1, the same conclusions regarding the relative performance of the models hold.

	Teacher-Forcing		Sampling	
Model	Flat	Hierarchical	Flat	Hierarchical
16-bar Melody	0.952	**0.956**	0.685	**0.867**
16-bar Drum	0.937	**0.955**	0.794	**0.908**
Trio (Melody)	0.866	**0.868**	0.660	**0.760**
Trio (Bass)	0.906	**0.912**	0.651	**0.782**
Trio (Drums)	0.943	**0.946**	0.641	**0.895**

Table 2. Reconstruction accuracies for the Lakh MIDI Dataset calculated both with teacher-forcing (i.e., next-step prediction) and full sampling. All values are reported on a held-out test set. A softmax temperature of 1.0 was used in all cases, meaning we sampled directly from the logits.

C. Attribute Definitions

The following definitions were used to measure the amount of each attribute.

C Diatonic

The fraction of notes in the note sequence whose pitches lay in the diatonic scale on C (A-B-C-D-E-F-G, i.e., the "white keys").

Note Density

The number of note onsets in the sequence divided by the total length of the sequence measured in 16th note steps.

Average Interval

The mean absolute pitch interval between consecutive notes in a sequence.

16th Note Syncopation

The fraction of (16th note) quantized note onsets landing on an odd 16th note position (1-indexed) with no note onset at the previous 16th note position.

8th Note Syncopation

The fraction of (16th note) quantized note onsets landing on an odd 8th note position (1-indexed) with no note onset at either the previous 16th or 8th note positions.

D. Audio Samples

Synthesized audio for all examples here and in the main text can be found in the online supplement.[5]

[5] https://goo.gl/magenta/musicvae-examples

E. Additional Figures and Samples

Subsequent pages include additional figures, referenced from the main text.

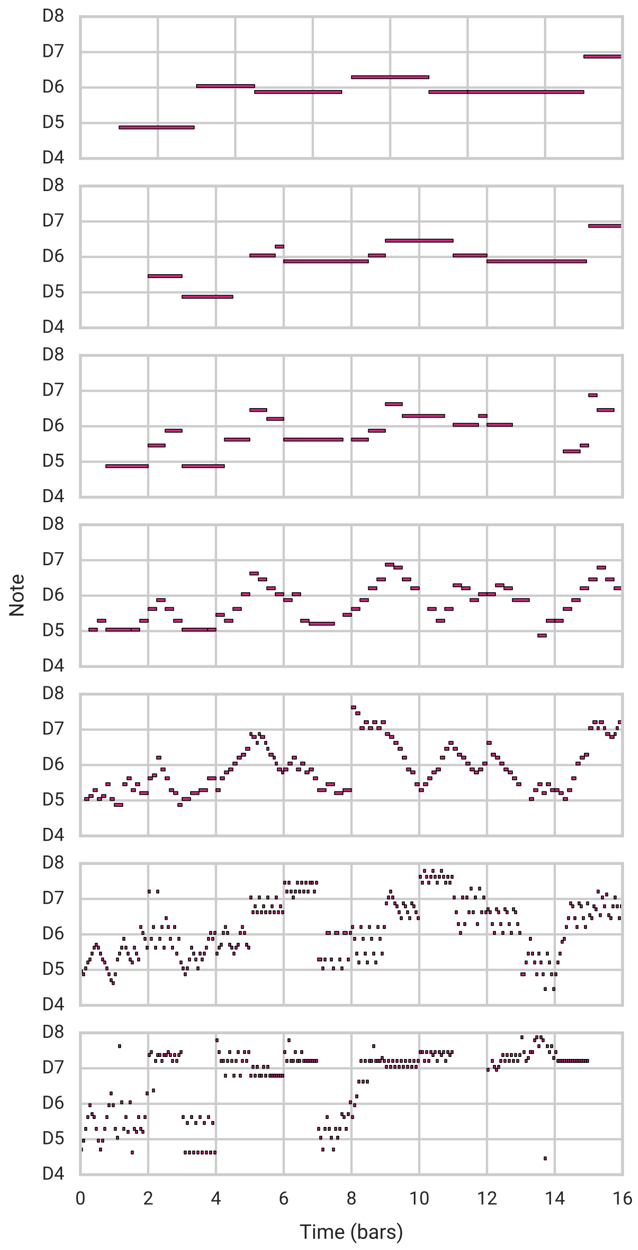

Figure 6. Varying the amount of the "Note Density" attribute vector. The amount varies from -1.5 to 1.5 in steps of 0.5, with the central sequence corresponding to no attribute vector. Audio for this example is available in the online supplement.[5]

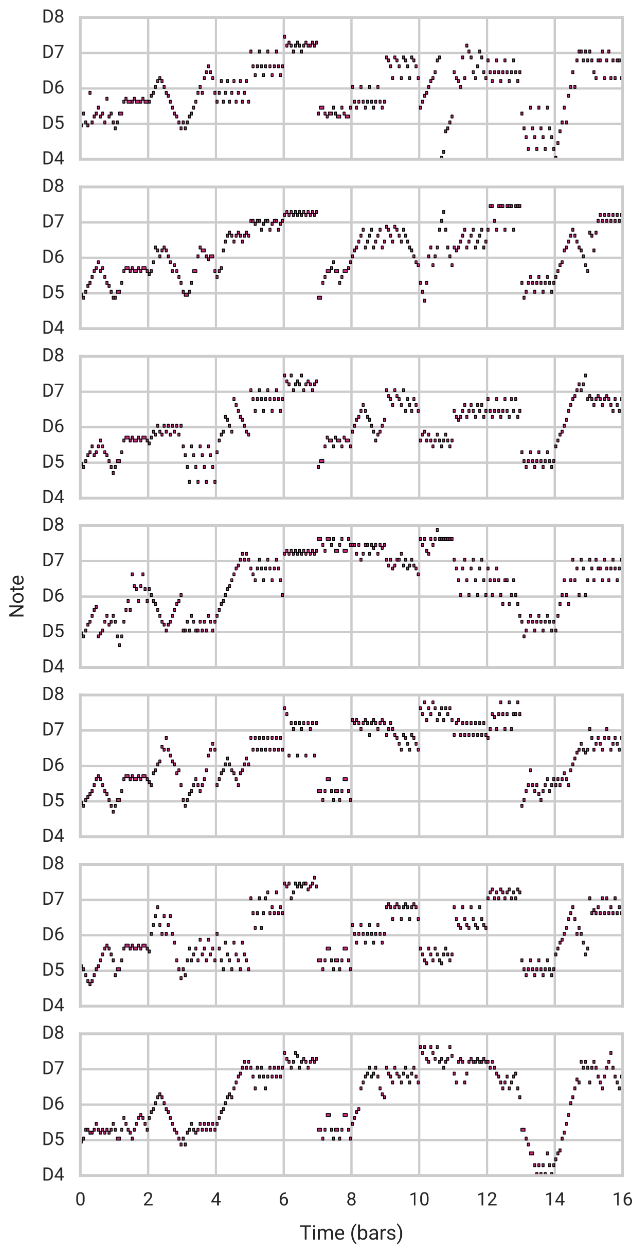

Figure 7. Additional resamplings of the same latent code (corresponding to the second-to-the-bottom in Fig. 6). While semantically similar, the specific notes vary due to the sampling in the autoregressive decoder. Audio for this example is available in the online supplement.[5]

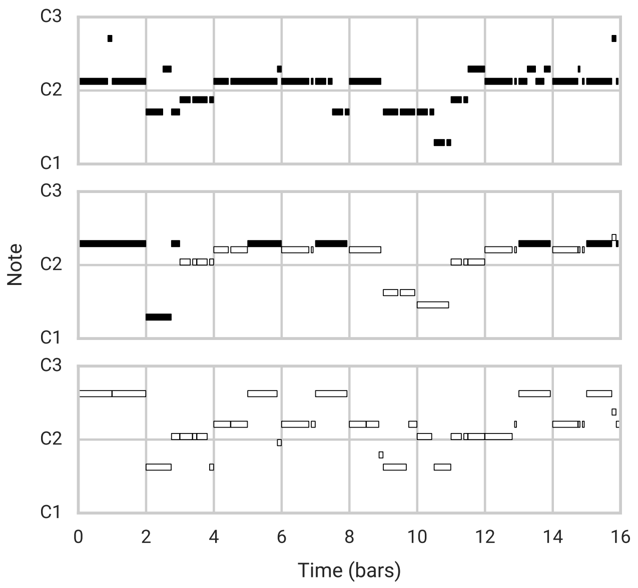

Figure 8. Subtracting (top) and adding (bottom) the "C Diatonic" attribute vector from the note sequence in the middle. For ease of interpretation, notes in the C diatonic scale are shown in white and notes outside the scale are shown in black. Audio for this example is available in the online supplement.[5]

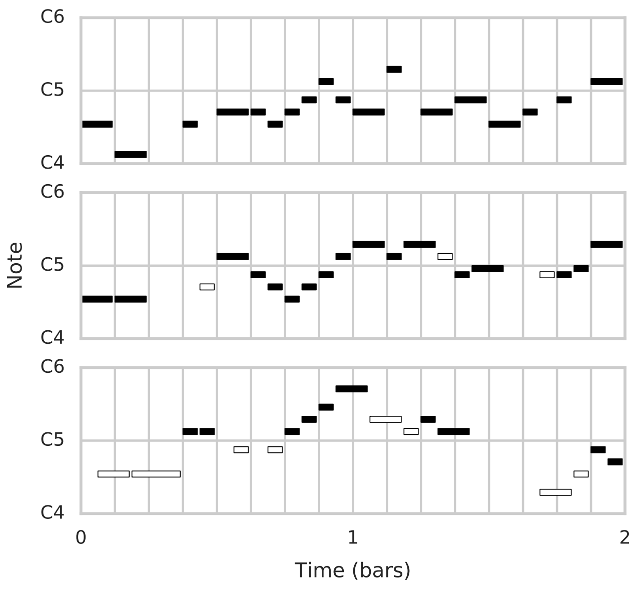

Figure 10. Subtracting (top) and adding (bottom) the "16th Note Syncopation" attribute vector from the note sequence in the middle. For ease of interpretation, only the first 2 of each sequence's 16 bars are shown. Vertical lines indicate 8th note boundaries. White and black indicate syncopated and non-syncopated notes, respectively. Audio for this example is available in the online supplement.[5]

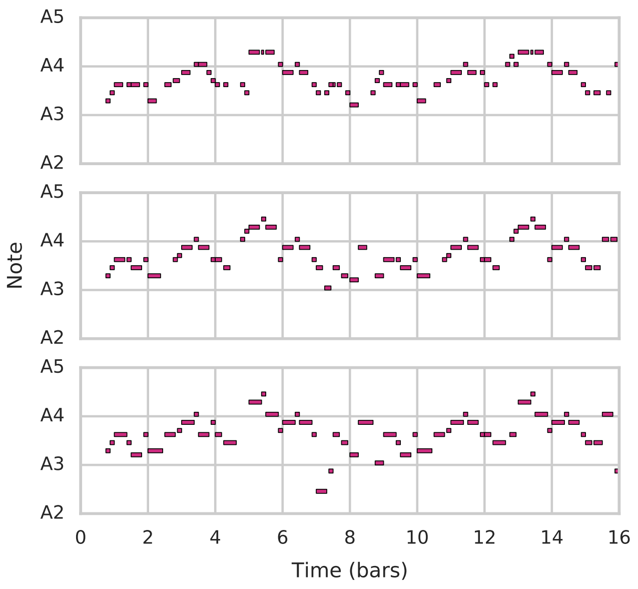

Figure 9. Subtracting (top) and adding (bottom) the "Average Interval" attribute vector from the note sequence shown in the middle. Audio for this example is available in the online supplement.[5]

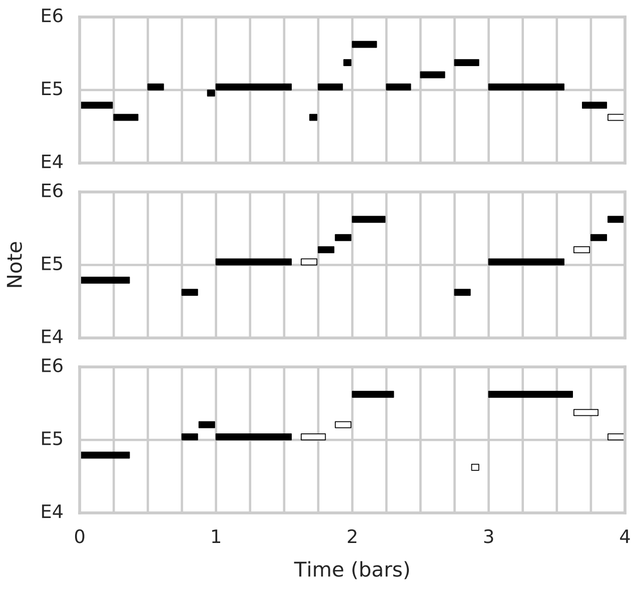

Figure 11. Subtracting (top) and adding (bottom) the "8th Note Syncopation" attribute vector from the note sequence in the middle. For ease of interpretation, only the first 4 of each sequence's 16 bars are shown. Vertical lines indicate quarter note boundaries. White and black indicate syncopated and non-syncopated notes, respectively. Audio for this example is available in the online supplement.[5]

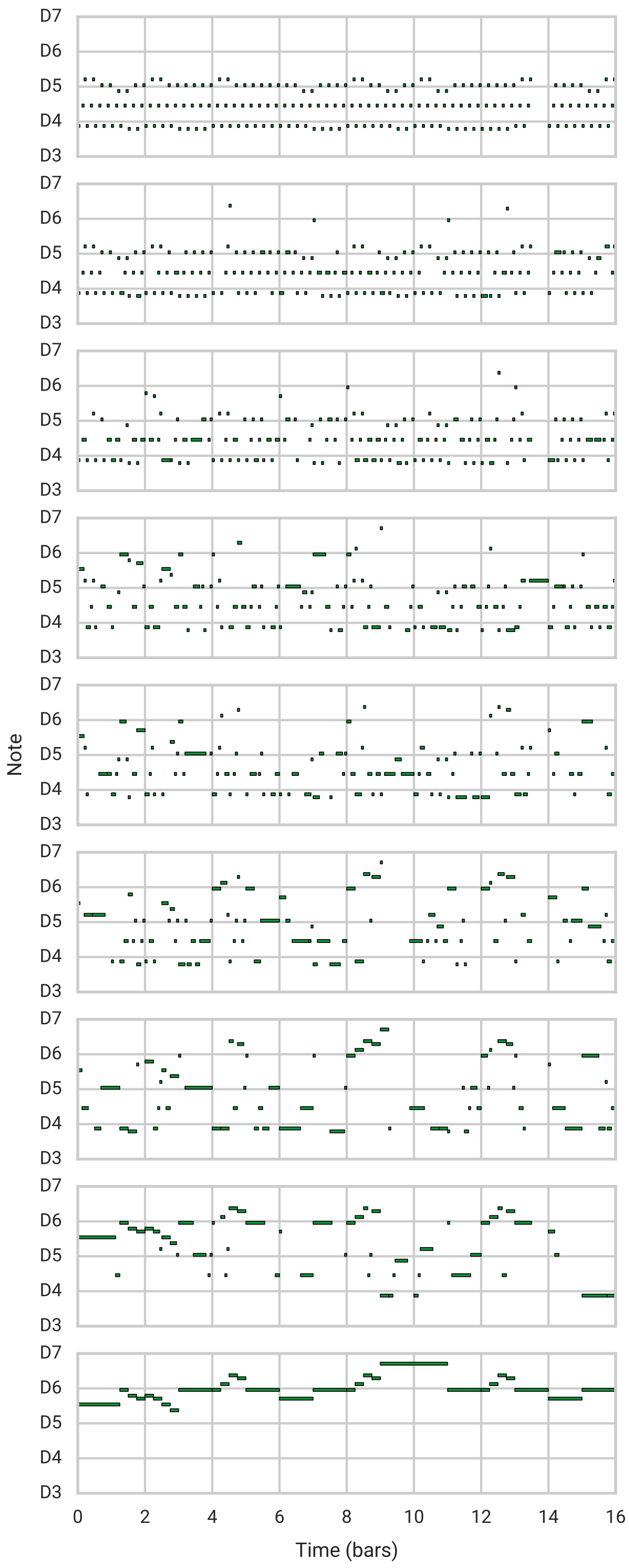

Figure 12. Interpolating between the top and bottom sequence in data space. Audio for this example is available in the online supplement.[5]

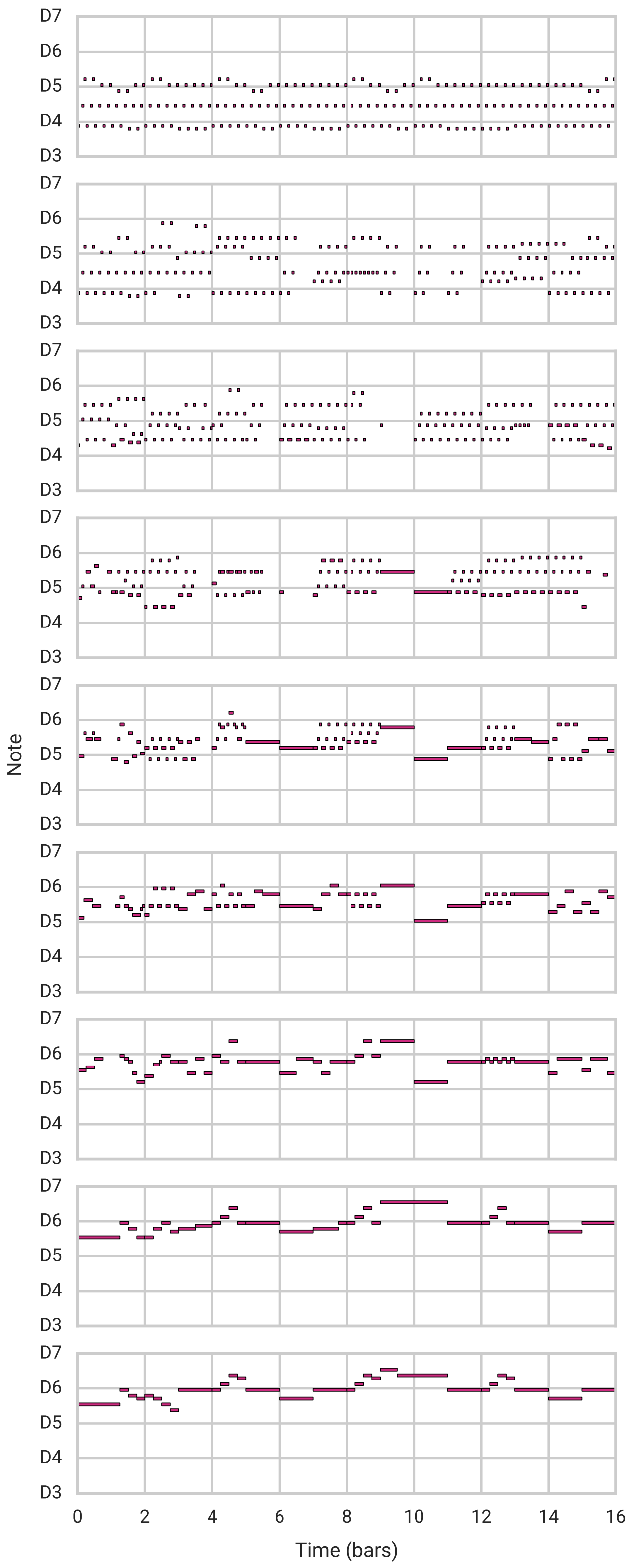

Figure 13. Interpolating between the top and bottom sequence (same as Fig. 12) in MusicVAE's latent space. Audio for this example is available in the online supplement.[5]

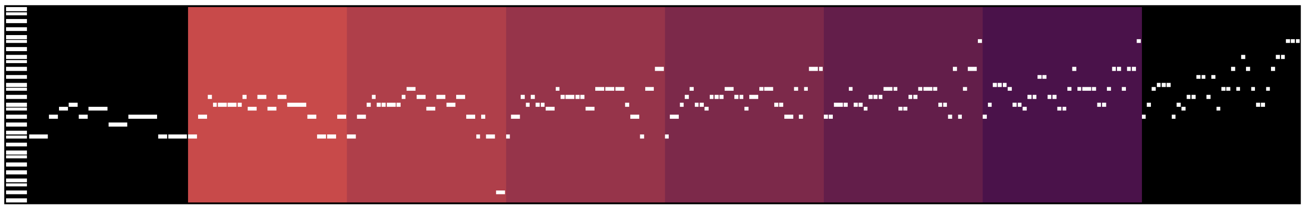

Figure 14. Example interpolation in the 2-bar melody MusicVAE latent space. Vertical axis is pitch (from A_3 to C_8) and horizontal axis is time. We sampled 6 interpolated sequences between two test-set sequences on the left and right ends. Each 2-bar sample is shown with a different background color. Audio of an extended, 13-step interpolation between these sequences is available in the online supplement.[5]

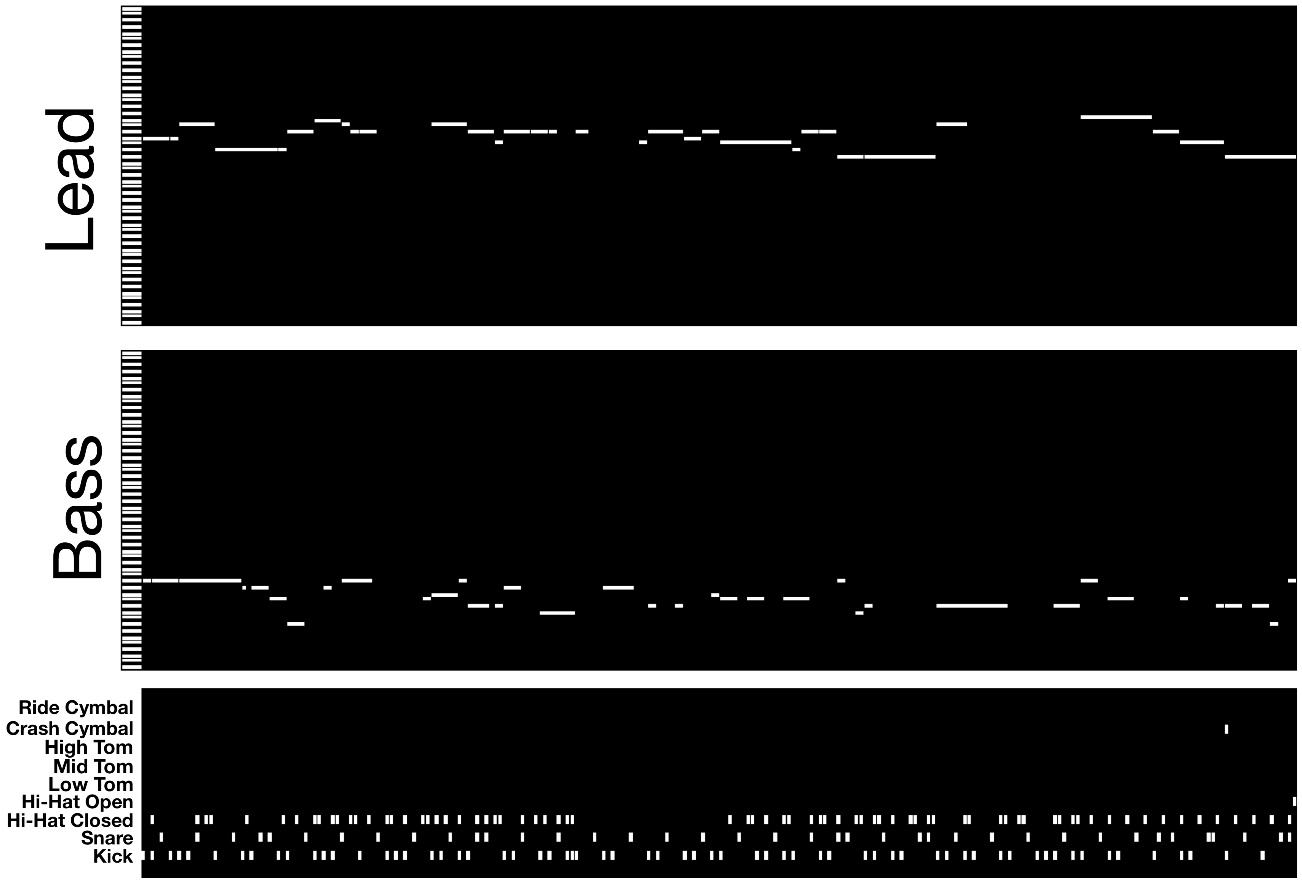

Figure 15. Selected example 16-bar trio sample generated by MusicVAE. Audio for this and other samples is available in the online supplement.[5]

.