Parameterized Melody Generation with Autoencoders and Temporally-Consistent Noise

Aline Weber, Lucas N. Alegre
Institute of Informatics
Federal Univ. of Rio Grande do Sul
Porto Alegre, Brazil
{aweber, lnalegre}@inf.ufrgs.br

Jim Torresen
Department of Informatics; RITMO
University of Oslo
Oslo, Norway
jimtoer@ifi.uio.no

Bruno C. da Silva
Institute of Informatics
Federal Univ. of Rio Grande do Sul
Porto Alegre, Brazil
bsilva@inf.ufrgs.br

ABSTRACT

We introduce a machine learning technique to autonomously generate novel melodies that are variations of an arbitrary base melody. These are produced by a neural network that ensures that (with high probability) the melodic and rhythmic structure of the new melody is consistent with a given set of sample songs. We train a Variational Autoencoder network to identify a low-dimensional set of variables that allows for the compression and representation of sample songs. By perturbing these variables with Perlin Noise—a temporally-consistent parameterized noise function—it is possible to generate smoothly-changing novel melodies. We show that *(1)* by regulating the amount of noise, one can specify how much of the base song will be preserved; and *(2)* there is a direct correlation between the noise signal and the differences between the statistical properties of novel melodies and the original one. Users can interpret the controllable noise as a type of "creativity knob": the higher it is, the more leeway the network has to generate significantly different melodies. We present a physical prototype that allows musicians to use a keyboard to provide base melodies and to adjust the network's "creativity knobs" to regulate in real-time the process that proposes new melody ideas.

Author Keywords

Variational Autoencoders, Perlin Noise, Melody Generation

CCS Concepts

•**Applied computing** → **Sound and music computing;**
•**Computing methodologies** → *Neural networks;*

1. INTRODUCTION

In recent years we have seen a growing interest in the application of machine learning techniques to perform music generation, processing, and analysis. A few examples include automated music composition, genre classification [6], and music tagging and recommendation [2]. Automated music generation techniques, in particular, have been extensively studied in the context of methods capable of composing new songs based on a set of sample melodies. One interesting application of music generation relates to constructing computational systems for helping musicians explore a richer space of possibilities when composing—for instance, by suggesting a few possible novel variations of a base melody informed by the musician. In this paper, we propose and evaluate a machine learning technique to generate new melodies based on a given arbitrary base melody, while ensuring that it preserves both properties of the original melody and general melodic and rhythmic characteristics of a given set of sample songs.

Previous works have attempted to achieve this goal by deploying different types of machine learning techniques. Methods exist, e.g., that interpolate pairs of melodies provided by a user [9]. We, by contrast, extend this idea to generate an arbitrary number of new melodies based on one that is provided by the user; it retains a parameterized amount of the original song's properties while still sounding similar to a set of sample songs. Other existing techniques identify compressed representations of melodies and then produce new ones by randomly tweaking such representations [10]. The resulting melodies are consistent with songs in a training set but are not smooth variations of a given reference song. Other works require, e.g., that each bar of a given base song be discretized into a fixed number of intervals [12]. Our approach allows for the generation of arbitrary continuous, real-valued note durations when producing new melodies. Furthermore, our method also differs from these techniques by enabling for both novel note durations and pitches to be generated by independent processes.

In this paper, we propose a method to autonomously generate new melodies that are variations of an arbitrary base melody. These are produced by a neural network that ensures that (with high probability) the melodic and rhythmic structure of the new melody is consistent with a given set of sample songs. This is achieved by training a Variational Autoencoder (VAE) to identify a set of latent variables that describe melodic phrases drawn from a set of songs. This process indirectly discovers patterns both in the pitches and note durations of such melodies. Then, given a base melody provided by the user, our method perturbs its corresponding latent variables by applying Perlin Noise—a temporally-consistent parameterized noise function— to generate smoothly-changing novel melodies. We show *(1)* that by regulating the amount of noise, it is possible to specify how much of the base song will be preserved; and *(2)* that there is a direct correlation between the noise signal and the differences between the statistical properties of the novel melodies and the original one. Users of our system can interpret the controllable noise as a type of "creativity knob": the higher it is, the more leeway the network has to generate significantly different melodies. We also present a prototype of a physical system that allows musicians to use a keyboard to provide base melodies and to regulate (in real-time) the process of generating novel melody ideas.

NIME'19, June 3-6, 2019, Federal University of Rio Grande do Sul, Porto Alegre, Brazil.

2. BACKGROUND

We first describe two key concepts required to understand our proposed technique: Variational Autoencoders and Perlin Noise. Section 3 introduces a method combining these ideas in a way that achieves the previously-described goals.

2.1 Variational Autoencoders

An Autoencoder (AE) is a type of unsupervised learning algorithm based on neural networks. Its goal is to learn a more efficient—more compact—representation of a given set of high-dimensional inputs; for instance, of human face images. Such a new representation, called a *latent representation*, allows for an AE to reconstruct any possible input solely based on its latent representation. Identifying a compressed representation of information is generally possible because, e.g., even though different images of human faces may have different pixel values, they also share similarities: there are patterns in the distance between eyes and the mouth, in the possible types of skin colors, etc. An AE is essentially a neural network tasked with learning how to produce as outputs the same inputs that it receives; an AE processing a particular picture of a cat, for instance, will be successful if it can propagate it through its layers and provide the same picture as output.

Mathematically, an AE is a learned function φ composed of two parts: an encoder φ_{enc} mapping an input vector X to another vector, z, called a latent representation of X; and a decoder φ_{dec} mapping z to an output vector $\hat{X}$. Ideally, $\hat{X}$ should be as close as possible to the original input X. When this is possible, the latent vector z defines a set of numbers providing an alternative representation of X. If the dimensionality of z is smaller than that of X, we say that z is a *compressed* (more compact) representation of X. In an AE, z is often a set of variables representing the high-level characteristics of X. One interesting property of AEs is that after latent representations z have been identified, one can use them to generate novel synthetic data similar to the data used to train the network. To do so, one can generate random vectors z (specifying, e.g., a random combination of eye color, skin color, etc.) and have the decoder φ_{dec} map z to the corresponding predicted $\hat{X}$—e.g., a new picture of a human face.

One difficulty with the above-described process is that in a regular AE, one does not know *a priori* the typical range or magnitude of values taken by latent vectors z. Its elements could be, e.g., numbers from 1 to 10 or from 3000 to 10000. This means that when generating novel data, it is hard to know the range of random values to be used to create random z vectors. A variant of AEs known as Variational Autoencoders (VAEs) [5] solves this problem. VAEs are neural networks similar to AEs but trained under a constraint that the values of z for typical inputs X should have a known and pre-defined form; for instance, they should be distributed according to a zero-mean, unit-variance Gaussian. This allows for the generation of random vectors z with values consistent to the ones observed during training, thus facilitating the use of AEs to generate novel data—e.g., novel human faces, similar to (but different than) the ones seen during training. Mathematical details of how VAEs are trained are beyond the scope of this paper but can be found in the literature; e.g., [5].

2.2 Perlin Noise

Several applications require the generation of random numbers to, for instance, automatically generate textures, artificial/synthetic pictures, and music. Consider, for example, a game where terrain (mountains) needs to be generated on-the-fly to create a new landscape every time the game is played. One way of doing that is to pick random heights for each mountain. A simple method of generating random numbers (also known as *noise*) for this purpose is to draw them from a Gaussian or Uniform distribution. However, this kind of noise does not always provide realistic results—a terrain where each mountain has its height determined independently by drawing from a Gaussian distribution looks like the random landscape in Figure 1a. To address this problem, [7] introduced the idea of Perlin Noise, a type of gradient-based noise function widely used for generating visual effects in computer graphics. Its main property is that consecutive random numbers drawn from this function are similar—they are said to be *consistent* with the random numbers generated immediately before it. If random mountain heights are drawn from a Perlin Noise function, for instance, the height of nearby mountains will not be completely independent and random but will be similar, resulting in a sequence of heights that is spatially consistent. A terrain where each mountain has height determined by drawing numbers from a Perlin Noise function looks like the landscape in Figure 1b. By comparing these landscapes, one can observe how Perlin Noise creates random patterns that look more realistic than those generated by completely random noise functions—in particular, because the space consistency provided by Perlin Noise results in mountain heights that vary smoothly over the space.

In this paper, we use Perlin Noise not to generate noise that is spatially-consistent, but temporally-consistent—i.e., random numbers that vary smoothly over time. Having a notion of temporal consistency is important in automated music generation to ensure that the properties of newly-produced melodies change smoothly over time, thereby resulting in an idea of continuity along the melody. Concretely, we will use Perlin Noise as a disturbance applied to the latent representations of different parts (windows) of a base melody, to generate new melodies that are smooth variations (over time) of the original one.

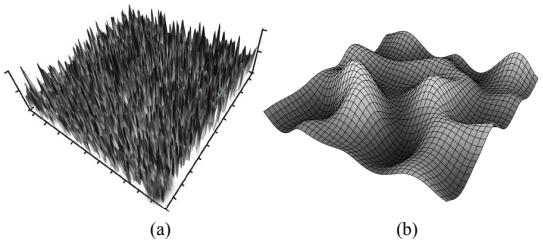

Figure 1: Landscape generated using (a) random Gaussian elevations; and (b) Perlin Noise (adapted from [11]).

3. PROPOSED METHOD

We now present a method that combines VAEs and Perlin Noise to generate new melodies that are smooth variations on a given melody. It allows the user to control how different the new melody will be w.r.t. the base one while still sounding similar (both in terms of pitch and note durations) to the general structure of songs in a training set. The high-level characteristics of pitches and note durations of the original melody are first represented as low-dimensional latent vectors computed by two independently-trained VAEs. They are then smoothly transformed over time by applying temporally-consistent Perlin Noise to the corresponding latent vectors. Applying the decoder functions of each VAE to the perturbed representations results in a novel, smoothly-varying melody based on the original one. Because we com-

pute latent representations with VAEs trained over a set of sample songs, the decoded melody also retains the general properties of the songs in the training set.

Our method receives as input a base melody described as a sequence of notes, where each note is annotated with its corresponding pitch and duration. To generate a new melody, our technique analyzes sequences of L consecutive notes—called windows—of the original melody. We give the pitches and durations of notes within each window as inputs to two different VAEs. One VAE is specialized in modeling sequences of pitches that commonly occur in the training set, while the other is specialized in modeling sequences of note durations that typically occur in that set. The VAEs first compute the latent representation of pitches and note durations within a window. Perlin noise (varying smoothly over consecutive windows of notes) is then applied to the latent representations. When these perturbed latent representations are given to the decoders, time-consistent variations of the original melody are produced as a function of the magnitude of the noise applied at each window.

3.1 Neural Network Architectures

Our technique makes use of two feed-forward fully-connected VAEs responsible for modeling the characteristics within a window: φ^P, for note pitches; and φ^D, for note durations.

- φ^P solves a classification problem: given L note pitches within a window, the k-th output of φ^P $(1 \leq k \leq L)$ is a vector containing the probabilities of note k being each one of the possible 12 pitches in the chromatic scale. Pitches within an input X are one-hot encoded, resulting in inputs to φ^P with $(L \times 12)$ dimensions. Both the encoder (φ^P_{enc}) and decoder (φ^P_{dec}) of φ^P are composed of fully-connected layers with ReLu activations. They are connected through a latent space layer z^P. The output layer of φ^P is followed by a Softmax layer to ensure that the final values form a probability distribution over the possible pitches. The overall architecture of φ^P is depicted in Figure 2a;
- φ^D solves a regression problem: given L note durations within a window, the k-th output of φ^D $(1 \leq k \leq L)$ is the predicted real-valued duration of note k. Both the encoder (φ^D_{enc}) and decoder (φ^D_{dec}) of φ^D are composed of fully-connected ReLu activated layers connected by a latent layer z^D. The output of the decoder is followed by a linearly-activated layer so that the outputs of φ^D can be arbitrary real-valued durations for each note within a window. The overall architecture of φ^D is depicted in Figure 2b.

As previously discussed, it is possible to train a VAE in a way that enforces that latent vectors z have values within some pre-defined distribution or range. Here, we enforce that the latent spaces of φ^P and φ^D are constrained to follow a Gaussian distribution $N(0, 0.1)$. φ^P is trained by analyzing a large number of windows of L pitches drawn from some set of sample songs; φ^D is trained similarly but over windows of L note durations. For more details on the construction of the dataset, see Section 4.1.

3.2 Temporally-Consistent Noise

As discussed in Section 3, we wish to use a Perlin Noise function to generate random numbers that are time-consistent; these will be used to perturb the latent representations of note pitches and durations, thus resulting in new melodies that differ from the base one in a way that varies smoothly over time windows. Let $g(t)$ be a standard Perlin Noise function as defined in [7]: it takes as input a time (in arbitrary units) and returns a time-consistent pseudo-random

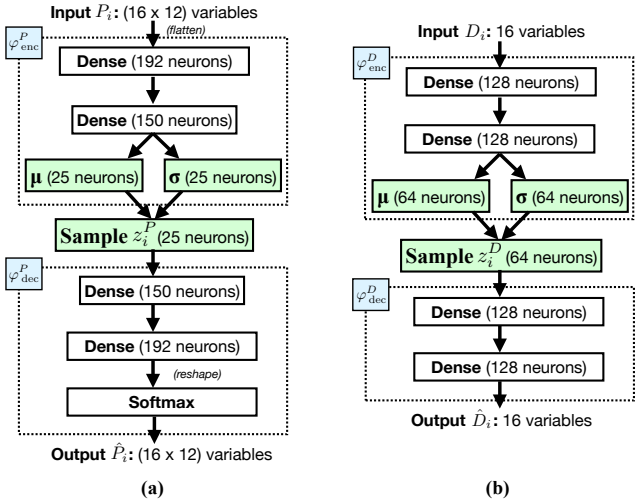

Figure 2: Network architectures of (a) φ^P; and (b) φ^D, both for the case of window length $L = 16$.

number. We define a variant of this function that is parameterized by two adjustable variables, α and β. In particular, we define our parameterized temporally-consistent Perlin Noise function $f(t\,;\alpha,\beta)$ by first using α as a scaling factor applied to the input of g: random values are then drawn from $g(t\alpha)$, instead of from $g(t)$. Larger values of α result in a less smooth sequence of random values. Secondly, we analyze a given sequence of values drawn from this modified noise function and rescale their magnitudes to the interval $[-\beta, +\beta]$. Larger values of beta imply a higher-amplitude sequence of random values. Intuitively, the α and β hyper-parameters can be used to adjust how fast and by how much (w.r.t. a given base melody) the different note windows in a new melody may change. In particular, α regulates the smoothness of noise over time; i.e., how fast the melody being generated may change over time. See Figure 3a for an example. The β hyper-parameter, on the other hand, regulates the amplitude of the random numbers being generated; it determines how much noise may be applied to each window being analyzed. See Figure 3b for an example.

3.3 Generation of Novel Melodies

After training both VAEs, we generate novel melodies by applying temporally-consistent Perlin Noise to the latent representation of note pitches and durations within each window of the base melody. We denote the base melody, composed of N notes, as $X = [(p_1, d_1), \ldots, (p_N, d_N)]$, where p_k is the pitch of the k-th note and d_k is its the duration. We first divide the melody into non-overlapping windows of size L, where the i-th window is defined as $X_i = \big[\big(p_{(i,1)}, d_{(i,1)}\big), \ldots, \big(p_{(i,L)}, d_{(i,L)}\big)\big]$, where $p_{(i,k)} = p_{iL+k}$ is the k-th note pitch in window i and $d_{(i,k)} = d_{iL+k}$ is the corresponding note duration. Our method splits each element of that window into its two components, resulting in a vector of pitches, P_i, and a vector of durations, D_i.

The encoder φ^P_{enc} of φ^P receives pitches P_i as input (for each window i of the melody) and computes their latent representation, z_i^P. It then perturbs z_i^P with Perlin Noise $\varepsilon_i^P = f(i\,;\alpha^P,\beta^P)$, thus obtaining $\tilde{z}_i^P$. The decoder φ^P_{dec} maps $\tilde{z}_i^P$ to P_i^{new}—a vector containing the new pitches generated for that time window. A similar process is also used to produce note durations: φ^D_{enc} receives as input D_i, generates z_i^D, and applies noise $\varepsilon_i^D = f(i\,;\alpha^D,\beta^D)$ to that latent representation. The resulting perturbed latent vector $\tilde{z}_i^D$ is given as input to the decoder, φ^D_{dec}, which produces the

new durations D_i^{new} for that window. Elements of P_i^{new} and D_i^{new} are merged to form X_i^{new}, a vector containing the new melody for the i-th window. The above process iterates over all length-L windows of the original melody, repeatedly appending each newly-generated window X_i^{new} to the novel melody X^{new}. See Algorithm 1 for details.

Algorithm 1 Generation of Novel Melodies

Let $X = [(p_1, d_1), (p_2, d_2), \ldots, (p_N, d_N)]$ *(Base melody)*
$X^{\text{new}} \leftarrow [\,]$ *(Initializes new melody)*
for i from $1 \ldots \lceil N/L \rceil$ **do** *(Iterates over melody windows)*
  Let $X_i = \left[\left(p_{(i,1)}, d_{(i,1)}\right), \ldots, \left(p_{(i,L)}, d_{(i,L)}\right)\right]$
  Let $P_i = \left[p_{(i,1)}, \ldots, p_{(i,L)}\right]$ *(Current window pitches)*
  Let $D_i = \left[d_{(i,1)}, \ldots, d_{(i,L)}\right]$ *(Current window durations)*
  $\varepsilon_i^P \leftarrow f(i; \alpha^P, \beta^P); \varepsilon_i^D \leftarrow f(i; \alpha^D, \beta^D)$ *(Perlin Noise)*
  $z_i^P \leftarrow \varphi_{\text{enc}}^P(P_i)$ *(Current latent pitches)*
  $\tilde{z}_i^P \leftarrow z_i^P + \varepsilon_i^P$ *(Noise-perturbed latent pitches)*
  $P_i^{\text{new}} \leftarrow \varphi_{\text{dec}}^P(\tilde{z}_i^P) = \left[p_{(i,1)}^{\text{new}}, \ldots, p_{(i,L)}^{\text{new}}\right]$ *(New pitches)*
  $z_i^D \leftarrow \varphi_{\text{enc}}^D(D_i)$ *(Current latent durations)*
  $\tilde{z}_i^D \leftarrow z_i^D + \varepsilon_i^D$ *(Noise-perturbed latent durations)*
  $D_i^{\text{new}} \leftarrow \varphi_{\text{dec}}^D(\tilde{z}_i^D) = \left[d_{(i,1)}^{\text{new}}, \ldots, d_{(i,L)}^{\text{new}}\right]$ *(New durations)*
  $X_i^{\text{new}} \leftarrow \left[\left(p_{(i,1)}^{\text{new}}, d_{(i,1)}^{\text{new}}\right), \ldots, \left(p_{(i,L)}^{\text{new}}, d_{(i,L)}^{\text{new}}\right)\right]$
  $X^{\text{new}} \leftarrow X^{\text{new}} \cup X_i^{\text{new}}$ *(Appends window to new melody)*
end for

4. EXPERIMENTS

In the following experiments, we analyze three main properties of our method: *1)* the effect of α in the generated melodies, demonstrating that it regulates the temporal consistency of the new melody w.r.t. the original one; *2)* the effect of β in the generated melodies, showing that exists a direct correlation between noise magnitude and the difference between statistical properties of novel melodies and the original one; and *3)* the advantages of using temporally-consistent noise to perturb the latent variables describing a given base melody, compared to naïvely perturbing them using i.i.d. noise drawn from a Gaussian or Uniform distribution. We also describe a physical prototype that allows musicians to use a keyboard to provide base melodies and adjust the noise hyper-parameters, thus regulating (in real-time) the generation of noise-parameterized melody variations that retain the general compositional properties of songs in a training set.

4.1 Training Process

We trained the two VAEs described in Section 3.1 using a dataset of classical songs in the MIDI format. As we are interested in processing melodies (monophonic note sequences), not harmonies, we selected only MIDI tracks that were composed of 85% or more of individual notes. We preserve only the highest note in each eventual remaining chords. This resulted in a set of 197 MIDI tracks in the key of C major. A training set was then constructed by extracting windows of $L = 16$ notes from the selected tracks. This was performed in a sliding-window fashion: the first window of a track included the notes $(1, \ldots, 16)$; the second window, notes $(2, \ldots, 17)$; and so on. The complete resulting dataset contained 131613 windows. As discussed in Section 3.3, each training window is a vector X_i composed of L tuples of pitches and durations. To obtain the data needed to train each VAE, each window X_i in the dataset was split into its components: a vector of pitches, P_i, and a vector of durations, D_i. The set of all P_i's constituted the training set of φ^P, and the set of all D_i's constituted the training set of φ^D. All note durations were standardized—a pre-processing step to help stabilize the training of neural networks. The datasets of φ^P and φ^D were split into training and validation sets according to an 80%-20% proportion. The parameters of both VAEs were optimized by the gradient descent method Adam [4] with a 0.001 learning rate. Training resulted in a 97% validation reconstruction accuracy for φ^P and a validation loss of 0.5283 for φ^D.

4.2 Novel Melody Analysis

We start our analyses by quantifying how effective our method is in generating new melodies that are variations of a base melody. We first introduce a distance metric for quantifying how different a window of pitches P_i (from the base melody) is when compared to the pitches generated by our networks. Concretely, we wish to evaluate how different a window $P_i^{\text{new}} = \left[p_{(i,1)}^{\text{new}}, \ldots, p_{(i,L)}^{\text{new}}\right]$ (produced by φ^P) is w.r.t. the original window $P_i = \left[p_{(i,1)}, \ldots, p_{(i,L)}\right]$. We first note that the VAE trained to predict note pitches, φ^P, produces as output a probability distribution over the most likely pitches to appear at each location k within a window i; i.e., it produces $\text{Prob}(p_{(i,k)}^{\text{new}} = C)$, $\text{Prob}(p_{(i,k)}^{\text{new}} = C\#)$, and so on. This implies that the VAE produces a stochastic set of notes; note pitches actually outputted by φ^P are drawn from the above distribution.

To define a distance metric $d(P_i, P_i^{\text{new}})$ that compares how different a newly-generated melody $P_i^{\text{new}} = \varphi^P(P_i)$ is w.r.t. a given base melody P_i, we first note that if a trained VAE has high accuracy, the pitch $p_{(i,k)}^{\text{new}}$ with highest probability at a location k within the new window will match the true pitch $p_{(i,k)}$ at the corresponding position in the original input window P_i. Based on this observation, one can define a distance metric between P_i^{new} (the melody produced by the network) and P_i (the original melody being perturbed) by computing the log-likelihood of the stochastic output of φ^P generating the original base melody exactly even when the latent space is perturbed by noise:

$$d(P_i, P_i^{\text{new}}) = -\log\left(\prod_{k=1}^{16} \text{Prob}(p_{(i,k)}^{\text{new}} = p_{(i,k)})\right) \quad (1)$$

$$= -\sum_{k=1}^{16} \log\left(\text{Prob}(p_{(i,k)}^{\text{new}} = p_{(i,k)})\right) \quad (2)$$

When the VAE produces an output P_i^{new} that is precisely equal to the original input P_i, $d(P_i, P_i^{\text{new}})$ is minimized: the noise applied to the latent pitch representation did not change the original melody at all. As the VAE model produces melodies P_i^{new} that differ significantly from the original one, the metric d increases. We now use the above-defined metric to quantify the effect of different values of α in the generated melodies, demonstrating that it does regulate the temporal consistency of the novel melody w.r.t. the original one (Figures 3a and 3c). We also analyze the effect of different β's in the generated melodies, showing that a direct correlation exists between the noise level and the differences between the statistical properties of novel melodies and the original one (Figures 3b and 3d).

To conduct the experiments that follow, we selected 20 random 16-note melody windows from the training set. Each one was used to create a song composed of 200 repetitions of that window. We perturbed the latent representations according to different parametrizations α and β of the Perlin

Noise function. Afterward, we used the proposed distance metric d to quantify how different each perturbed window was w.r.t. the original one, as a function of the amount of Perlin Noise being applied at that time.[1]

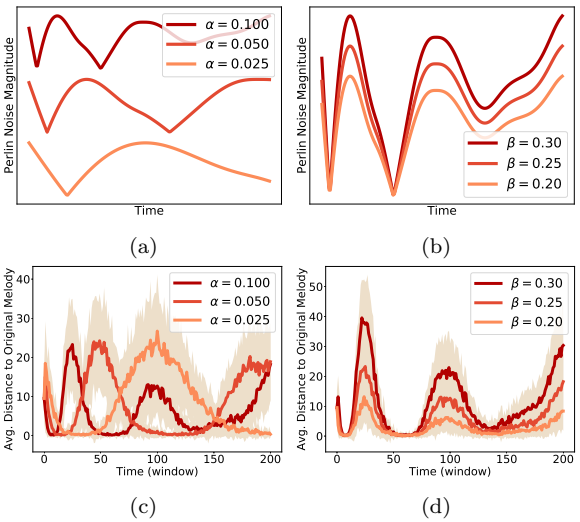

Figure 3: Sample Perlin Noise functions constructed by selecting (a) different α values; and (b) different β values. Distance metric over time between novel melodies and the original one for (c) different values of α; and (d) different values of β. Note how the distance/similarity between new melodies and the original one is *directly correlated* (and thus regulated) by the corresponding Perlin Noise function.

Figures 3a and 3b show a few sample Perlin Noise functions $f(t\,;\alpha,\beta)$ constructed by selecting different values of α and β. Note how (as expected from the discussion in Section 3.2) α regulates the smoothness of the noise function over time, and β regulates the magnitude of the noise. Figures 3c and 3d show the effect that different values of α and β (when used to parameterize the noise applied to the melody latent representations) have on the distance $d(P_i, P_i^{\text{new}})$ between newly-generated melodies and the original one. Here, the vertical axis depicts the average distance between the newly-generated windows constructed by our network and the corresponding original ones. In particular, we can compare Figures 3a and 3c and observe that the higher we set α (thereby allowing the amount of noise applied to each window to vary more rapidly), the faster $d(P_i, P_i^{\text{new}})$ changes along the melody. Note also how the smoothness and general form of the Perlin Noise function (for different values of α) is *directly correlated with* (and therefore regulates) the smoothness and general form of the distance function $d(P_i, P_i^{\text{new}})$. This can be observed by noting how changes to the Perlin noise over time cause a direct and proportional change to the distance metric d at the corresponding times. These observations imply that α does indeed regulate how rapidly the similarity between the novel melodies and the original one might change over time. A similar analysis can be performed by comparing Figures 3b and 3d: by changing β, one can regulate *how much* of the original melody will be preserved in the newly-generated melodies. When $\beta = 0.30$, for instance, the novel melodies can differ w.r.t. the original melody up to 4 times more than when $\beta = 0.20$. Hence, besides allowing for random melody generation in a temporally-consistent way, our proposed method also successfully introduces a parameterized way of controlling how different the novel melodies may be over time.

[1]Interested readers can listen to sample generated melodies at `https://www.youtube.com/watch?v=k_z54dV4WoI`.

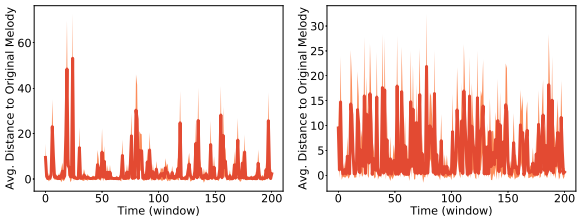

Figure 4: Distance between novel melodies and the original one when noises are sampled from naïve distributions: (a) Gaussian $N(0, \beta/2)$; and (b) Uniform $U[-\beta, +\beta]$.

Finally, we now show that naïvely perturbing the latent variables that describe a given base melody using i.i.d. noise drawn from a Gaussian or Uniform distribution does not result in the properties that we previously achieved with Perlin Noise. Figures 4a and 4b show the average distance between newly-generated melodies and the original one when their latent representations are perturbed by noises drawn, respectively, from $N(0, \beta/2)$ and from $U[-\beta, +\beta]$. In both cases, we used $\beta = 0.25$ so that it is possible to compare these results with the ones in Figure 3d. Using a Gaussian noise distribution (Figure 4a) produces melodies composed of highly-variable windows of pitches, often followed by a sequence of windows almost identical to the original ones. This results in new songs with harsh transitions between subsequent windows and no control over the smoothness of the generated melodies. Using a Uniform noise distribution (Figure 4b) produces melodies with even more abrupt variations over time. These experiments confirm that, differently from was achieved when using Perlin Noise, it is not possible to use Gaussian or Uniform noise distributions to control the temporal consistency of the melodies being generated by the networks, and that a temporally-consistent and smooth parameterized noise generator is needed.

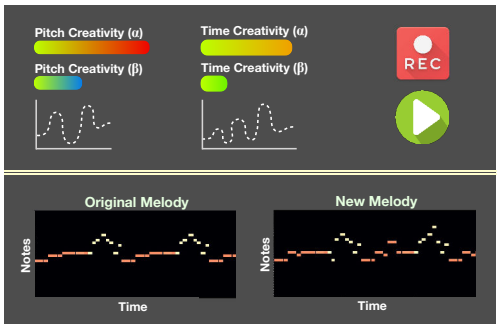

Figure 5: Graphical interface of our system prototype.

Besides performing the computational analyses above, we implemented a physical prototype of our system that allows musicians to use a keyboard to provide base melodies and to adjust the network's "creativity knobs" (α and β) to regulate, in real-time, the process that proposes new melodies that retain general compositional properties of the songs in the training set. This prototype is depicted in Figure 5. Our prototype is composed of a standard MIDI keyboard controller that allows musicians to play and generate melodies in real time. The resulting MIDI files are pre-processed according to the steps in Section 3 and are given as input to our method. By using the graphical interface in Figure 5, users can tweak the values of α and β for each of the autoencoders—the one generating variations in pitch and

the one changing note durations. The resulting melodies can be heard in real-time. This system may be used as an aid in the composition process: a musician provides a basic melody idea to the system, which then produces a given number of variations of it. The musician selects the one that is preferable and may feed it back to the system to obtain more novel ideas/variations, thereby exploiting the system's proposed variations to help in the composition process.

5. RELATED WORK

Other approaches exist that generate melodies based on applying transformations to the latent representation of a given melody. Most of these either perform simple translations of the latent vector or apply operations that interpolate between existing melodies; e.g., MusicVAE [9, 10]. Our method, by contrast, uses a more general and user-adjustable noise function to determine how the latent representation of a base melody will be modified over time. This allows the user to directly regulate how similar the produced melody will be w.r.t. the original one while retaining the general characteristics of songs in the training set. [12] also shares with us the objective of generating melodies, but it does not aim at producing interesting variations of a user-provided base melody. Instead, they generate melodies that are compatible with a given chord progression; this requires the use of a pre-defined rule-based system so that Gaussian noise (when added to the latent variables) produces feasible melodies. Other methods exist that generate melodies by exploiting properties of VAEs; e.g., [3]. Here, the authors introduce a technique that samples from the latent space, but that is limited in that each (fixed-length) portion of the melody being analyzed must be discretized into a pre-defined maximum number of fixed-duration notes. Our method, by contrast, can represent, process, and generate sequences of notes with arbitrary real-valued durations, since we deploy independent VAEs networks to model sequences of pitches and sequences of real-valued note lengths. [1] presents many different methods for music generation based on deep learning techniques, including autoencoders. These methods differ from ours in that they usually perturb points in the latent space by random sampling or by interpolating previously-specified latent vectors, instead of applying temporally-consistent noise that ensures smooth melodic changes. Finally, the idea of using Perlin Noise as a tool to generate new melodies has been studied before; e.g., in [8]. This work differs from ours in that they do not use Perlin Noise to modify a given melody. Instead, they use it to construct something akin to an effects pedal that applies Perlin Noise directly to an analog sound signal, thereby obtaining a parameterized sound effect generator.

6. CONCLUSIONS AND FUTURE WORK

We have introduced a method to generate new melodies by perturbing the latent representation of a base melody with temporally-consistent gradient noise. This ensures that the characteristics of the novel melody are smoothly transformed over time. Furthermore, the discovery of latent representations via variational autoencoders makes it possible for the generated melodies to retain general properties consistent with songs in a training set. We proposed the use of a parameterized type of Perlin Noise and showed that there is a direct correlation between the noise signal and the differences between the statistical properties of the novel melodies and the original one. Users of our system can interpret the controllable noise as a type of "creativity knob": the higher it is set, the more leeway the network has to generate significantly different melodies. We also briefly discussed a physical prototype that allows musicians to interact in real-time with our system via a MIDI keyboard and a graphical interface.

Our method currently analyzes and processes melody windows, *one at a time*, to produce smooth variations over a base melody. However, it does not take into account how the original song was split into bars. Exploiting this type of information, as well as heuristics regarding typical note durations within a bar, may allow us to generate new melodies that more closely retain higher-level rhythmic properties of the original song. We also wish to study, as future work, the possibility of extending our system to use recurrent VAEs; these may be able to more directly model the relation between successive notes instead of dealing with disjoint sequences of neighboring-note windows. Finally, we are also interested in extending the mathematical and statistical analyses discussed in Section 4.2 and performing a human-centered evaluation to measure how effective our method may be in facilitating the composition process.

7. ACKNOWLEDGMENTS

This work was partially supported by FAPERGS under grant no. 17/2551-000; by The Research Council of Norway, project no. 240862 (Engineering Predictability with Embodied Cognition project); as part of the Collaboration on Intelligent Machines (COINMAC) project no. 261645; and by Centres of Excellence scheme, project no. 262762.

8. REFERENCES

[1] J.-P. Briot, G. Hadjeres, and F. Pachet. Deep learning techniques for music generation-a survey. *arXiv preprint arXiv:1709.01620*, 2017.

[2] D. Eck, P. Lamere, T. Bertin-mahieux, and S. Green. Automatic generation of social tags for music recommendation. In *Advances in Neural Information Processing Systems 20*, pages 385–392, 2008.

[3] G. Hadjeres, F. Nielsen, and F. Pachet. Glsr-vae: Geodesic latent space regularization for variational autoencoder architectures. In *2017 IEEE Symposium Series on Computational Intelligence*, pages 1–7, 2017.

[4] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[5] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[6] T. L. Li, A. B. Chan, and A. Chun. Automatic musical pattern feature extraction using convolutional neural network. In *Proc. Int. Conf. Data Mining and Applications*. sn, 2010.

[7] K. Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985.

[8] A. Popov. Using perlin noise in sound synthesis. In *Proceedings of the Linux Audio Conference*, 2018.

[9] A. Roberts, J. Engel, and D. Eck. Hierarchical variational autoencoders for music. In *NIPS Workshop on Machine Learning for Creativity and Design*, 2017.

[10] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck. A hierarchical latent vector model for learning long-term structure in music. *arXiv preprint arXiv:1803.05428*, 2018.

[11] Scratchapixel Website. Using perlin noise to create a terrain mesh. [Online; accessed 2019-01-23].

[12] Y. Teng, A. Zhao, and C. Goudeseune. Generating nontrivial melodies for music as a service. *arXiv preprint arXiv:1710.02280*, 2017.