Deep recurrent music writer: memory-enhanced variational autoencoder-based musical score composition and an objective measure

Romain Sabathe, Eduardo Coutinho, Björn Schuller

Deep Recurrent Music Writer: Memory-enhanced Variational Autoencoder-based Musical Score Composition and an Objective Measure

Romain Sabathé*§, Eduardo Coutinho†*, and Björn Schuller*‡

*Department of Computing, Imperial College London, London, United Kingdom
†Department of Music, University of Liverpool, Liverpool, United Kingdom
‡Chair of Complex & Intelligent Systems, University of Passau, Passau, Germany
§Corresponding author. Email: e.coutinho@liverpool.ac.uk.

Abstract—In recent years, there has been an increasing interest in music generation using machine learning techniques typically used for classification or regression tasks. This is a field still in its infancy, and most attempts are still characterized by the imposition of many restrictions to the music composition process in order to favor the creation of "interesting" outputs. Furthermore, and most importantly, none of the past attempts has focused on developing objective measures to evaluate the music composed, which would allow to evaluate the pieces composed against a predetermined standard as well as permitting to fine-tune models for better "performance" and music composition goals. In this work, we intend to advance state-of-the-art in this area by introducing and evaluating a new metric for an objective assessment of the quality of the generated pieces. We will use this measure to evaluate the outputs of a truly generative model based on Variational Autoencoders that we apply here to automated music composition. Using our metric, we demonstrate that our model can generate music pieces that follow general stylistic characteristics of a given composer or musical genre. Additionally, we use this measure to investigate the impact of various parameters and model architectures on the compositional process and output.

I. INTRODUCTION

Machine learning (ML) has endowed computers with the capacity to learn from data without being explicitly programmed to do so (Arthur Samuel, 1959). Historically, this capacity has been used mainly to deal with two kinds of tasks – classification and regression problems. These two classes of problems were, and still are, particularly useful in a myriad of data science areas and applied in various contexts to answer financial, industrial, marketing or strategical questions. The recent competitions from the website Kaggle [1], which hosts online data science challenges, confirm this trend – at the time of writing, Bosch is trying to predict internal failures on its production lines, the Chinese company TalkingData is looking for predicting users' demographic characteristics, where as another competition aims at recognizing species of tree leaves from pictures. All these problems involve a classification or regression task. More recently however, new techniques, often involving Deep Learning (DL), breached beyond the scope of these two problems – nowadays ML is also interested in

generating new data. Striking examples are the creation of fake Wikipedia articles and hand-written sentences [1], upscaling images to recover lost information [2], creating images based on a sentence [3], voice synthesis [4] or even beating the world-champion of Go [5].

In this paper, we follow this recent trend and will focus on generating new music. In particular, we aim to infer the underlying rules characterizing the composition process of a set of musical pieces (sometimes referred to as musical grammars, see for instance [6], [7]) that are fundamental aspects of musical organization and appreciation, and generate new music pieces on that style or compositional principles. To this end, we will rely on a class of models called Variational Autoencoders (VAEs) [8], [9], which has become popular lately due to their generative capabilities (as well as their solid theoretical foundations). VAEs aim to model the underlying and complex joint distribution of a given data, to sample from it, and to generate new examples that fit the same distribution. An important advantage of VAEs is that the input data can be of any kind (e.g., images, sound, video, text). For example, a VAE could be used to learn the distribution of pictures of sunflowers. Then, by sampling from this distribution, we would obtain pictures whose content fundamentally follows all the "rules" that make a sunflower what it is – its color, its shape, etc. In our case, we want to learn the distribution of musical pieces belonging to a given style by allowing the VAE to capture the relevant musical rules that underlie the composition process.

Given the largely subjective task that we address in this paper, another core focus of the work presented here is the development and application of an objective measure that can allow to understand how different models, architecture and parameters lead to different musical outcomes, and their meaningfulness in a specific musical context. This is a fundamental aspect of generating new data that has been largely ignored in previous research, and constitutes a crucial step forward in the direction of an objective evaluation of generative musical models in machine learning.

The remaining of this paper is structured as follows. In Section II, we review past works on automatic music generation

using ML. In section III, we introduce a new performance measure developed to objectively compare the music generated by different models, and we describe our experimental setup. In Section IV we describe and analyze the results of our experiments using VAEs, including the effects of various parameters and architectures. Our conclusions, and thoughts for further work are presented in Section V.

II. RELATED WORK

Only a few works have focused on music generation with ML techniques. Nonetheless, all faced similar challenges: how to represent the data (i.e., music), which examples to learn from, which types of models to use, and how to assess the quality of the generated music. In this section, we briefly sketch the approaches that have been used to address each of these problems.

In [10], the authors employed a Simple Recurrent Network (SRN) [11]. The SRN was trained using genetic algorithms to, as stated by the author, maximize the chances of generating "good melodies" one measure at a time. The music data was discretized in pitch – ranging from C2 to C5 – and time – whole notes, half notes, quarter notes, eighth notes and sixteenth notes were allowed within a measure. The results were evaluated using musical rules considerations such as pitch diversity, rhythmic diversity, measure diversity or the capability of the network to stay coherent regarding one or several pitch scales. This evaluation, however, is designed specifically for melodies whereas we would like to assess the quality of an automatic composer as a whole. Results were promising but the authors admit the resulting pieces lack of overall structure. Besides, the model is particularly constrained in its generation capabilities, in terms of pitch and duration ranges.

The popular Long-Short Term Memory (LSTM) RNNs were used for the first time in music composition by Eck and Schmidhuber [12]. A LSTM-RNN is similar to traditional feed-forward neural networks except that the non-linear hidden units are replaced by a special kind of units — the LSTM memory blocks. These blocks consist of special memory cells that permit the RNN to access a long-range temporal context and predict the outputs based on such information, a characteristic that is particularly interesting for music composition, given the fact that music is at is mots basic level a set of relationships between music elements at different temporal scales (e.g., phrases, melody, sections, movements). In [12], chords and melodies were theoretically split. While both were generated using LSTM-RNNs, the learning process was not identical for both. Chords were learned by the network by seeing sequences of predefined chords in random order. On the other hand, melodies were learned by seeing a random sequence of notes sampled from a jazz scale, along with a chord at each bar. The network was provided with 13 notes it was allowed to play on in order to produce both chords and melodies. The architecture was designed such that the melody was conditioned by the chord progression rules, and only contained one note at a time. No performance measure

was used. Nevertheless, subjectively speaking, it is hard not to be impressed by the results achieved by this method. However, the approach is limited in that it imposes very restrictive pre-defined musical rules such as a reasonable scale to play on. For instance, the chord progression was very similar at each iteration, and no original chords were played apart from those appearing in the training set. Ideally, one would like to minimize preprocessing and restrictions imposed on the model and still achieve satisfactory results.

In order to model the high variety of simultaneous note patterns (harmonies) characteristic of polyphonic music, [13] used a model called RNN - Restricted Boltzmann Machine (RRN-RBM). Since RBM models are trained by fitting their (so-called) hidden units to the underlying distribution of the input data, they show some theoretical resemblance to VAEs. Whilst some parameters were tuned using an RNN and gradient descent, some others were tuned using Gibbs Sampling [14]. No priors were imposed on the range of playable notes, and time was discretized to the quarter of a beat. The authors used a database comprising three datasets of classical piano and another dataset of folk music, amounting to a total of 67 hours of music. The performance of the model was assessed on these 4 different datasets of varying complexity. In a preprocessing stage, music pieces were transposed to a common tonality (C major for major pieces and C minor for minor pieces. This helped reducing the variance of the training set). Some generated music pieces were provided as examples, but no performance measures were used in order to assess their quality.

Liu and colleagues [15] revisited and extended the LSTM-RNN architecture introduced by [12]. They used a sequence of two LSTM layers without preprocessing or specific encoding (in that sense, the problem setting is similar to [13]). The learning material consisted of a database of Bach's chorales (the same used in [13]). Their innovation consisted of using resilient propagation as optimization algorithm and square mean distance as error function instead of the log likelihood. No clear results are presented regarding music generation, and the paper emphasizes the lack of a proper evaluation metric.

Huang and colleagues [16] used a very similar approach with a two-layer LSTM-RNN, a quarter-note discretization of time, and a larger dataset of 2000 classical music pieces (417 of them being from Bach's repertoire and the remaining being from various artists and obtained via web crawling). In this work an interesting comparison was conducted between the music generated by this model, and those generated in [13]. Twenty-six volunteers were asked to rate the quality of pieces generated by both of these works. Results were surprisingly good with the majority of the subjects giving a mark of 7+ out of 10 (where 1 stands for "random" and 10 stands for "composed by a novice composer"). The proposed LSTM-RNN did obtain higher ratings than the RNN-RBM model, but the sample size was not large enough to draw statistically significant decisions. Also, the assessment method is dubious.

III. METHODS

In the works presented in the previous section, only in [10] the issue of evaluation of the generated pieces was addressed by including rules of tonality and rhythm in the selection of the best pieces. However, in order to assess the similarity of the generated music to specific styles, knowledge related to a wider range of structural and formal parameters should ideally be included. In this section, we will address this issue and propose a new, objective measure that enables to assess the musical similarity between music pieces.

A. Performance measure: the Malahanobis distance

Since we usually train models on a corpus of music pieces and we are interested in reproducing styles rather specific pieces, we decided to use a metric that permits comparing a set of pieces (e.g., the training corpus) against a single piece (e.g., a newly generated one). Therefore, the aim of this metric is to quantify the musical similarity between a specific music piece and a given distribution of pieces. The first step was to develop a quantity that characterizes a given music piece. We did so by using high-level, symbolic musical descriptors. Each of these descriptors is gathered in a vector that we call the *piece signature vector*. This signature vector must be descriptive enough so that it becomes unlikely for two different music pieces to have the same signature vector. Naturally, the most accurate signature vector should use a detailed set of music theoretical parameters, but previous works indicate that even using very simple metrics it is possible to describe complex musical characteristics, such as music styles [17]. We base our signature vector on this work, which consists of 17 high-level features:

Number of notes Number of notes in the piece divided by the length of the piece. A note is counted when we observe a succession of timesteps for which a specific pitch is always played. The scaling is necessary given that, for instance, a 10-second-long musical piece composed of only five notes will sound normal, whereas a 1-hour-long piece composed of the same number of notes will sound empty.
Occupation rate The ratio between the number of non-null values in the piano roll representation and the length of the piece. The piano roll representation is described in Figure 3.
Polyphonic rate The number of time steps where two or more notes were played simultaneously, divided by the total number of notes in the piece.
Pitch range descriptors The maximum, minimum, mean and standard deviation of the non-null pitches in the piece. As pitch values in MIDI format are encoded between 0 (minimum; C-2) and 127 (maximum; G8), all values were divided by 127 in order to force these descriptors to be bounded between 0 and 1 (for simplicity).
Pitch interval range An interval is a difference in pitch between two consecutive notes (with or without being separated by a period of silence) . All intervals were scaled between 0 and 1 (i.e., divided by 127) and the maximum, minimum, mean and standard deviation were computed.

Duration range The duration is the number of time steps during which a note is held. As before, the maximum, minimum, mean and standard deviation of all durations in the piece were computed (no scaling was performed).

As described, some descriptors had to be scaled in order to allow the creation of an adequate comparison between pieces. In order to compute the similarity (or distance) between a musical piece x and a given corpus $\mathcal{D} = \{T_1, T_2, \ldots, T_n\}$ characterized by their mean signature vector μ and covariance matrix Σ (which would represent a particular musical grammar), we use the Mahalanobis distance [18]. Formally,

$$D(x, \mathcal{D}) = \sqrt{(x - \mu)^T \Sigma^{-1} (x - \mu)}.$$

It should be noted that, in practice, the various features are not scaled to the same range. This could have been a problem if we have used the Euclidean distance and a one-versus-one piece comparison, since we would have large difference in amplitudes for some features and small difference in amplitudes for some others (leading to an unbalanced weighting between the different descriptors). Nonetheless, the use of the covariance matrix and the Mahalanobis distance alleviates this problem.

As a preliminary evaluation of the adequateness of our measure, we chose a reference distribution $\mathcal{D}$ of Beethoven pieces and compared it to a set of individual music pieces. Note that the Beethoven distribution $\mathcal{D}$ is also the dataset we used for training our models (more details in Section III-C). A subset of pieces from the same Beethoven distribution was used to build the reference distribution $\mathcal{D}$. Then, we created another distribution composed of 62 jazz pieces from different artists (e.g., Nat King Cole, Bill Evans; we will later refer to this distribution as the *Jazz distribution*. Finally, we obtained samples from two random distributions. Both are constructed to have a sensible number of notes in the musical piece (450 non-null values for a piece of 300 time steps long). However, whilst the first one uses the full spectrum of pitch (ranging from 1 to 128), the second one, that we called "sensible random distribution" uses a spectrum of pitch ranging from 20 to 100, which is more common in modern music. This second random distribution will therefore give us a better insight as to what is a reasonable score, and what is just random.

The plot in Figure 1 follows our expectations. The Beethoven samples are closer from $\mathcal{D}$ than pieces from the Jazz distribution. Yet, if the difference is noticeable, the two distributions are not separated by a wide gap. We can therefore infer that a distance of 6 and lower indicates a coherent piece of music. A distance of 3 to 4 could indicate something that generally comprises musical material that generally resembles the Beethoven corpus used. Our expectations are also met regarding the random distributions. The fully random distribution is farther from the Beethoven distribution, compared to the sensible random distribution. We therefore conclude that a distance of approximately 10 or higher approximated a random piece and, as a result, should be discarded.
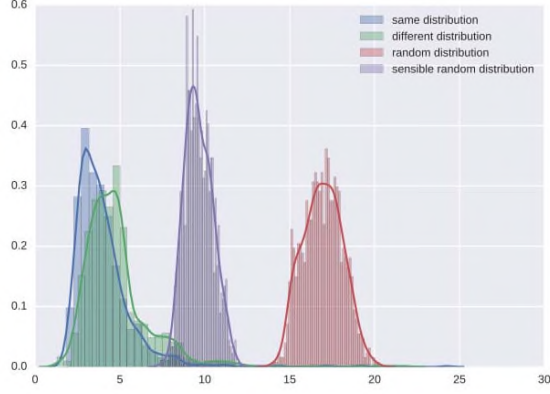
Fig. 1. Distribution of distances using the Mahalanobis distance. The distribution of reference (from which we obtained μ and Σ) was built using 100 samples from the Beethoven corpus. Each sample was 300 time steps long. The shown distributions were obtained by sampling 1000 samples from, in order, the Beethoven corpus, the Jazz corpus and a random corpus. Each of these samples were also 300 time steps long. The random corpus was created by using binomial sampling with $p = 0.012$. p was chosen to have roughly 450 notes per sample. The sensible random distribution is created as the random one, except that the possible active pitches range from 20 up to 100, which corresponds to the distribution of most real music pieces.

B. VAE – a generative model

As we have suggested in Section II, LSTM-RNN has offered the best performance so far in terms of generation quality from a subjective point of view. However, this model is not generative by nature and still needs to be provided with a few time steps of music to start improvising. On the contrary, we would like to build upon a truly generative model, capable of generating music without any priors. Besides, witnessing the recent successes of VAEs [19], [20], we decided to evaluate their music generation capabilities.

We first present a brief summary of the theory behind VAEs. A VAE is a non-trivial model $\mathcal{M}(f_\theta, g_\lambda, Q_\omega)$ that takes an input x and tries to output x with as little distortion and loss as possible. f_θ is a function whose parameters are θ, similarly g_λ is a function whose parameters are λ and Q is a probability distribution over the *latent space* that we describe just below. Q is parametrized by ω. The output of a VAE is computed as follows:

1) Compute $\omega = g_\lambda(x)$.
2) Sample $z \sim Q_\omega$. z is said to be a latent vector and has been sampled from the latent space.
3) Compute $\tilde{x} = f_\theta(z)$. $\tilde{x}$ is supposed to be a close approximation of x.

In order to train a VAE, one has to decide on the shape of the probability distribution Q, the shape of the functions f and g and tune the parameters θ and λ. In general, we want f and g to be extremely flexible functions such as neural networks. Besides, if we set Q to be a Gaussian distribution, then we can use the *reparameterization trick* mentioned in [21] which makes it possible to use gradient descent to tune the model in an end-to-end fashion. In that case, ω becomes the traditional parameters (μ, Σ) of a normal distribution.

To train the model, two loss functions are used:

1) During encoding, we want to ensure that the relevant latent vectors are mostly located within a ball centered at 0 and of radius 1, so we spatially concentrate all the encoding information in the ball. We use the Kullback-Leibler divergence for this purpose:

$$\mathcal{L}^z = \frac{\partial \text{KL}\left[Q\left(z|g_\lambda(X)\right)||\mathcal{N}\left(0, I\right)\right]}{\partial \lambda},$$

where $\mathcal{L}^z$ is called the latent loss and X is the input distribution.

2) During decoding, we want to ensure that the retrieved vector $\tilde{x}$ is close to x. This loss is more intuitive since we can take the log-likelihood between x and $\tilde{x}$. This is particularly suitable since the values of x are constrained between 0 and 1:

$$\mathcal{L}^r = -\sum_i x_i \log\left(\tilde{x}_i\right),$$

where $\mathcal{L}^r$ is called the reconstruction loss.

The final loss is taken to be the sum of the latent loss and of the reconstruction loss.

$$\mathcal{L} = \mathcal{L}^z + \mathcal{L}^r.$$

Different varieties of VAEs can therefore be created by appropriately choosing the functions f, g and the distribution Q. In this paper, we use a specific kind of VAE called Deep Recurrent Attentive Writer (DRAW) [20]. This model demonstrated promising capabilities for image generation by defining f and g functions to be single-layer LSTM-RNN (traditional VAEs use standard feed-forward neural networks instead). The use of an LSTM-RNN allows to introduce temporal information in the VAE. As a result, the final output of DRAW is computed sequentially, time step after time step, by updating a so-called "canvas". Figure 2 presents a graphical view of this model. Theoretically, this endows the model the ability to refine its output, by adding local details to it or withdrawing some along the different time steps [2].

Once the model has been trained, we can use it for music generation. To do so, we bypass the role of g_λ and simply sample a z from $\mathcal{N}(0, I)$ and decode it using f_θ. This makes sense since the training forced the relevant latent vectors to be packed in the ball centered at 0 and of radius 1 in the latent space. Since the output of the model is not computed exactly the same way during training and generation, we will hereafter distinguish between the "training phase" and the "generation phase". We refer the reader to [20] for more information about the implementation of this model. Note that we did not use the attention mechanism mentioned in this paper since it surprisingly yielded very poor results in preliminary experiments. More work should be done to investigate this.

---

[2]We could for instance imagine, that the first time steps would be dedicated to creating a bass line while the remaining steps are dedicated to creating a melody.
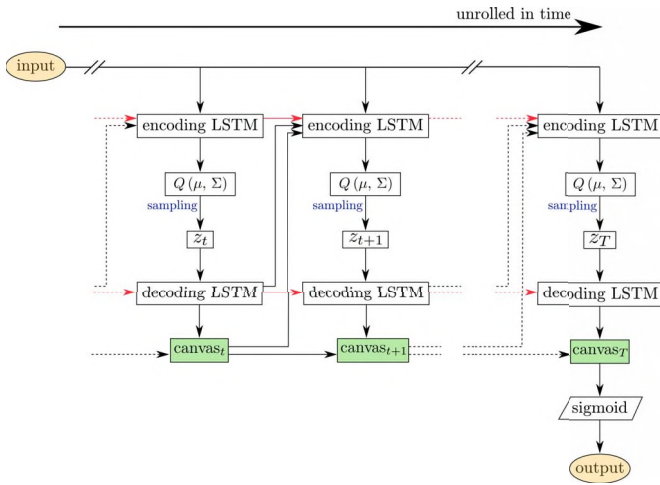
Fig. 2. Graphical view of the DRAW model as presented in [20]. In our explanations, the "encoding LSTM" is the f_θ function, the "decoding LSTM" is the g_λ function and the distribution Q is parameterized by $\omega = (\mu, \Sigma)$. The red arrows represent the usual recurrent connection of LSTM-RNN whilst the black arrows represent recurrent connections created by design.

C. Music corpus and representation

We used a corpus a 37 free-of-rights MIDI files of Beethoven's piano pieces obtained by crawling the web. As a result, the corpus gathers a lot of various unrelated works. Nevertheless, we chose the pieces that only contained one playing piano. This is motivated so that our training set includes all the variety of a full musical composition (bass line, chords, melody, etc.) with a single instrument (to avoid the complexities arising from multiple instruments playing together at this stage).

We followed the raw encoding explored in previous works, because it provides an intuitive and flexible music representation and it does not require preprocessing or normalization. In order to represent the time dimension, we split time in eights of a beat. Note that this quantization still allows the model to represent virtually any musically-relevant duration patterns since we can arbitrarily modulate the tempo. Also, it does not interfere with the range of notes or chords which the model can generate (unlike previous works; see Introduction). We record what happens within each time step in a binary vector of length 128. As we mentioned earlier, the reason behind this length is related to the encoding of a MIDI file itself where each "signal" is encoded using an 8-bit signed integer. A MIDI signal can be as diverse as "beginning of piece", "end of piece", "note C5 is played" etc. Besides, most pianos only feature 88 keys, thus they all can be encoded using a vector of length 128 (the full range of MIDI notes). A value of 1 indicates that a note is played at this time step, and 0 indicates that the note is absent (consecutive cells with values 1 indicate a sustained note). A graphical representation of this encoding (a.k.a. piano roll) is shown in Figure 3.

An important requirement of the DRAW model, along with other known VAEs, is that they can only process fixed-size inputs. We are therefore forced to use sections of music pieces
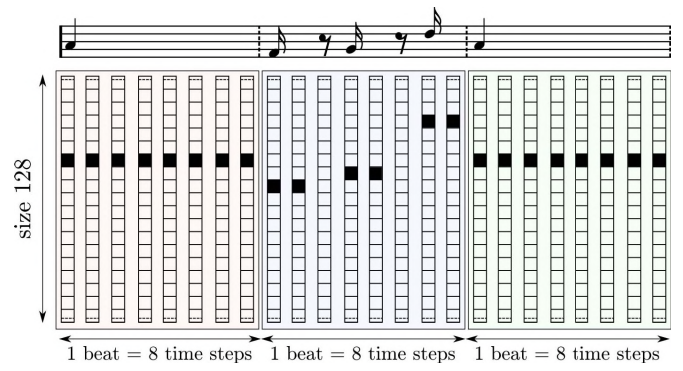

Fig. 3. The correspondence between a real sheet music and the encoding used in this paper ("piano roll" representation). Note that a bar is filled with 8 binary vectors. In this figure, a blank square indicates a value 0 and a filled square indicates a value 1.

TABLE I
SEARCH SPACE USED WHEN EXPLORING DRAW ARCHITECTURES. FOR MORE INFORMATION ON THE PARAMETERS, REFER TO III-B AND [20].

Parameter	Min. value	Max. value	Sampling method
Chunk size (L)	64	160	uniform (multiples of 8)
Dimensionality of the latent space ($\#z$)	5	1000	uniform
Number of sequential updates before outputting (T)	3	75	uniform
Number of units in the LSTM layers	128	1024	uniform

as inputs, and train the model to output similarly sized chunks. Therefore, the pieces used for training had to be split. We did this by creating non-overlapping chunks of 13 beats (i.e., 104 time steps). This number was determined experimentally since, as we show later, it offers a good compromise between duration and perceived quality of the generated music. Naturally, this imposes a limitation on the length of the generated tracks. Nonetheless, this is an issue that could be solved and that we intend to address in future work.

D. Optimization of the model parameters

In our work, we use a modified version of the DRAW model [20] that is able to handle rectangular patches, instead of square patches in the original work (this was necessary so that pitch representation was independent of the music segments size). Additionally, so as to work with a reasonably good performing DRAW architecture, we performed several experiments where we varied the parameters of the DRAW model and assessed the performance on music generation of each resulting architecture using our performance measure. In total, 136 architectures were randomly generated using uniform sampling from the search space described in Table I.

Out of these experiments, we selected the architecture for detailed analysis that, according to our measure, yielded the best and most stable results. Such model had a latent space $\#z$ dimension of 22, 167 LSTM units in both the encoding function g_λ and decoding function f_θ, and the number of steps

T required to perform the sequential autoencoding (which corresponds to the memory length of g_λ and f_θ) was 23. Note that the memory of the LSTM units in g_λ and f_θ were reset after each minibatch. This enforced the autoencoding of independent chunks of music, yet each patch being well-sounding by itself. We briefly discuss in Section V ideas to alleviate this problem. We used batches of size 37, corresponding to the number of music pieces in the data set. This choice was motivated by the way TensorFlow handles LSTM units, even though this is not strictly required since we reset the LSTM units at each iteration.

The output of the model, during training phase or generation phase, is a matrix of dimension 104×128 (104 time steps with vectors of length 128 associated to each time step), whereby each value is in $[0, 1]$. In other terms, these are probabilities that the notes should be played. To properly generate a piece, we first normalized the matrix so that the maximum equals 1 and the minimum equals 0, and then applied a binary threshold α. The appropriate value of α is discussed in the next Section.

$$\forall i \in [\![1, 104]\!], \ \forall j \in [\![1, 128]\!], \ x_{ij} \leftarrow \begin{cases} 1 & \text{if } x_{ij} > \alpha \\ 0 & \text{otherwise.} \end{cases}$$

As for the optimizer algorithm, we used the Adam algorithm with a learning rate $l = 0.001$, and parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$, which are common parameters in the literature. The training was stopped after running for 24 hours on a low-end graphic card (NVIDIA GT750M). Note that when we tested the 136 different architectures, the training was stopped after only 30 minutes in order to perform many tests in a restricted time span.

IV. RESULTS

The model with the parameters optimized as described in the previous section leads to general musical pieces having a mean Mahalanobis distance to the Beethoven distribution of 3.47 ± 1.47, which is a very good score if we compare to the graph presented in Figure 1. To complete this comparison, we performed a t-test between the generated distribution (corresponding to the blue curve in Figure 4) and the Beethoven reference distribution (corresponding to the blue curve in Figure 1). Results ($t = 0.81$, $p = .42$) show that these two distributions are not significantly different. To obtain these results, we generated 100 musical pieces from our trained model and computed the Mahalanobis distance for each of these to the Beethoven dataset. In addition to the only DRAW architecture we presented, we show in Figure 4 additional results obtained during the optimization stage of the DRAW architecture and parameters (as detailed in Table I). We took the 3 architectures that performed the best out of these experiments, and plotted the distribution of Mahalanobis distances obtained by comparing 100 generated pieces from each architecture to the Beethoven reference distribution.

In order to evaluate of the importance of each optimized parameter for music generation, we gathered the results we obtained when testing the 136 different architectures (see the
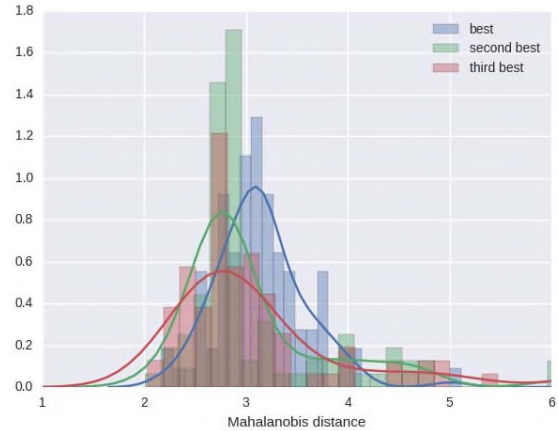

Fig. 4. Distribution of Mahalanobis distances between generated musical pieces and the Beethoven dataset. For each of the 3 best architecture we obtained when performing random search, we generated 100 pieces, computed their Mahalanobis distance to the Beethoven dataset for each of them, and reported the results in this plot. Note that the Mahalanobis distances are centered around 3 which is a strong score. However, the mode of the distribution does not correspond to its mean since a fat tail and a few failed generations – of score 10 and higher – tend to increase the value of the mean. This is the reason why the blue distribution is actually the best, it yields the most stable results.

TABLE II
CATEGORIZATION USED TO RATE THE PERFORMANCE OF EACH
ARCHITECTURE. THE VALUES ARE TAKEN IN ACCORDANCE TO FIGURE 1.

Category	Range of valid values
Like Beethoven metric-wise	$0 \le x < 4.5$
Coherent music	$4.5 \le x < 7$
Sensible random	$7 \le x < 10$
Random or worse	$10 \le x < \infty$

beginning of this section) and used a decision tree to visualize the impact of each parameters on the distance measure. Instead of using a regression tree to regress the Mahalanobis score directly, we assigned a category to each score (and therefore to each model) to shift to a classification problem. The categorization is described in Table II. We enforced a maximum depth of 3 and 10 examples at least were required to perform a split.

The analysis of the decision tree has provided some insights regarding what makes a relevant DRAW architecture for generating music. The number of LSTM units used to encode and decode the data appears to be the most determining factor. If it is too high (i.e. greater than 300), only a latent space with lots of dimensions (i.e. greater than 550) can lead to coherent pieces. However, as the tree shows, better results were obtained using lower number of LSTM units. Therefore, it seems pointless to use an architecture with that many LSTM units and number of dimensions since the training becomes harder and the training time is considerably increased. Interestingly enough, the tree tells us that with less than 300 LSTM units, it is better to have a latent space with few dimensions (i.e. less than 65). We interpret this observation in terms of degrees of liberties. Since the latent space contains all the encodings of musical pieces, if we increase the dimension of the latent
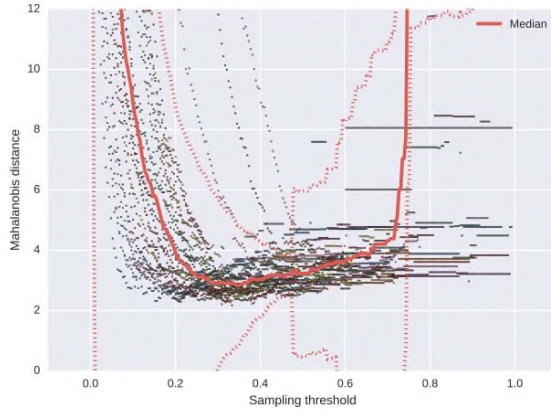
Fig. 5. Different Mahalanobis distances between generated musical pieces and the Beethoven dataset. We generated 30 outputs from our baseline DRAW architecture and varied the binary threshold α 200 times uniformly between 0 and 1, leading to $6,000$ measurements, i.e. $6,000$ points in this plot. The solid line shows the median of the distances and the dotted line is 1.96 times the standard deviation. Notice that, for low values of α, few notes are actually played resulting in empty pieces and therefore a poor score (e.g. above 4). On the contrary, at higher values of α, *all* notes with non-null probabilities have been played, resulting in these characteristic horizontal lines.
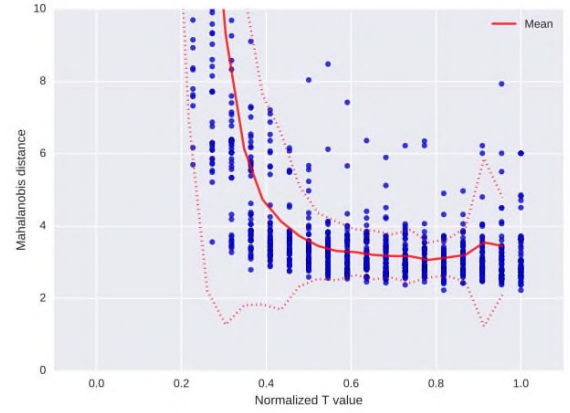
Fig. 6. Different Mahalanobis distances between generated musical pieces and the Beethoven dataset. We generated 50 outputs from our baseline DRAW architecture and, instead of keeping on the very last iteration, we evaluated every one of them, from the first to the last. The $x-$axis uses a normalized scale between 0 and 1 but we remind that the baseline architecture uses $T = 23$. The solid line shows the mean of the distances and the dotted line is 1.96 times the standard deviation. We limited the range of the $y-$axis which hides some points with distances higher than 10.

space, we will dilute the relevant encodings (that decode into coherent pieces) with non-relevant encodings (that decode into random pieces). It is likely that, with more training data and therefore more relevant encodings, a higher dimension of the latent space becomes necessary. Further works should investigate this. At last, the number T of steps required to generate an output should be, according to the tree, limited to at most 35. It is likely that, above this value, the model just keeps adding notes leading to over-dense pieces, however this should be confirmed experimentally.

We further investigated the ideal binary thresholding α. We generated 30 outputs from our baseline architecture, from each output, we used 200 different thresholds uniformly distributed between 0 and 1, eventually creating $30 \times 200 = 6,000$ music pieces. We computed the Mahalanobis distance between each one of them and the Beethoven distribution. The results are presented in Figure 5. If we only look at the median results, a value of $\alpha = 0.25$ yields the best performance with a median Mahalanobis distance of about 3. However, if we take into account the variability of the results (via the dotted lines of standard deviation), then a value of $\alpha = 0.45$ provides the most stable results with a slightly worse median Mahalanobis distance of about 3.5.

We also evaluated the influence of the parameter T on the quality of the generation. More specifically, we assessed the quality of the pieces at each step of their generation (from the first step, up to the last one, i.e., the 23rd). Following the same methodology used to determine the ideal α, we gathered 50 outputs from the baseline architecture, and generated a piece at each step of the generation, leading to $50 \times 23 = 1,150$ music pieces. We computed the evaluation score for each of them. The resulting plot is shown in Figure 6. As we expected, the higher the generation step, the lower the Mahalanobis distance.

This shows the expected behavior of the DRAW model: as the number $t \in [1, T]$ of output iteration increases, the model adds more and more relevant information to the output its currently generating, leading to an overall decrease of the Mahalanobis distance. However, we would expect this to be the case up the last generation step, but it rather seems that the quality of generated pieces becomes a bit more volatile for the very last steps. Results were the best and more stable for a normalized T value of around 0.75 which, in the case of our baseline, corresponds to the 17th step. Additional research should be done to determine whether the same architecture with $T = 17$ would yield similar results or if every DRAW architecture tend to generate an optimal output at an intermediate stage of the generation.

V. CONCLUSIONS AND OUTLOOK

In this work, we proposed a new measure for assessing the quality of generated music pieces using Variational Autoencoders. In relation to the metric, we exemplified its usefulness in two scenarios. First, we used it to automatically and systematically fine-tune the generative models' parameters and architectures for optimizing the musical outputs in terms of proximity to a specific musical style/context (in this paper a set of piece by Ludwig van Beethoven). Second, we used the metric to select the most interesting output generated by the model, i.e., those that resembled the most the original corpus in terms of the musical characteristics measured. In both cases, we have demonstrated its usefulness in identifying newly generated new musical pieces that are objectively close in musical terms to the original corpus, and fine-tuning generative models (in our case Variational Autoencoders) to maximize that same outcome.

In relation to generative power of Variational Autoencoders, and considering the non-general and abstract rules used in

these initial explorations, we were able to create a wide variety of original music segments and listening sensations (fast and slow, monophonic and polyphonic pieces, using high and low registers, etc.). Although we would not argue that the generated pieces are in the style of the original corpus (some piece from Beethoven's portfolio), we demonstrated that our metric is good enough for distinguishing structured music from noise and other styles[3]. In this respect, further work is necessary to investigate the use of other music descriptors so as to measure relevant properties of a given musical style in order to drive the model training and piece selection processes. This is a straightforward extension of this work given that our metric is simple and quick to compute, and can be applied to a variety of descriptors that can incorporate music and non-musical knowledge. It would be interesting though, that future work includes a perceptual evaluation of our metric, both in terms of the quality of the outputs generated as well as the similarity to a given style.

The major and most obvious drawback of our method is that Variational Autoencoders cannot generate longer sequences than what it has been trained for. One way of addressing this problem in future work would be to train the model to output short pieces, conditioned to the previous outputted short pieces. More concretely, a second distribution over the latent space could be used where the parameters of this distribution would be determined by a recurrent model over the previous generated short-pieces. By merging the distribution Q we mentioned in this paper and this new distribution, the model could be able to output pieces, that, when concatenated, form a coherent musical piece.

Finally, on a more general note, automatic music composition is a field still in its infancy, and it faces many challenges. Indeed, and unlike other perceptual domains (e.g., vision) people are extremely sensitive to static and temporal sound patterns, and possess complex innate and acquired mental schemata that rule the perception and appreciation of music. Future research needs to understand how such complex musical knowledge can be incorporated into generative models. Possible ways are to explore recently developed methods for automatic feature and knowledge extraction (e.g., [22]) and the automatic analysis of the emotional impact of generated musical pieces given that this is one of the main reason why people choose to listen to music and closely tied to music structure (e.g., [23]).

ACKNOWLEDGEMENT

REFERENCES

[1] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, pp. 1–43, 2013. [Online]. Available: http://arxiv.org/abs/1308.0850

---

[3]A set of examples generated by the best model reported in this paper, and selected with our metric can be found at http://www.openaudio.eu/ and http://www.eadward.org/.

[2] D. Garcia, "Image super-resolution through deep learning," 2016. [Online]. Available: https://github.com/david-gpu/srez

[3] E. Mansimov, E. Parisotto, J. L. Ba, and R. Salakhutdinov, "Generating Images from Captions with Attention," *arXiv preprint*, pp. 1–12, 2015. [Online]. Available: http://arxiv.org/abs/1511.02793

[4] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[6] W. D. Mario Baroni, Simon Maguire, "The concept of musical grammar," *Music Analysis*, vol. 2, no. 2, pp. 175–208, 1983. [Online]. Available: http://www.jstor.org/stable/854248

[7] F. Lerdahl and R. Jackendoff, *A generative theory of tonal music*. MIT Press, 1983.

[8] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[9] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[10] C.-C. Chen and R. Miikkulainen, "Creating melodies with evolving recurrent neural networks," *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, vol. 3, pp. 2241–2246, 2001. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=938515

[11] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[12] D. Eck and J. Schmidhuber, "Finding temporal structure in music: Blues improvisation with LSTM recurrent networks," *Neural Networks for Signal Processing - Proceedings of the IEEE Workshop*, pp. 747–756, 2002.

[13] N. Boulanger-Lewandowski, P. Vincent, and Y. Bengio, "Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription," *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, no. Cd, pp. 1159–1166, 2012. [Online]. Available: http://arxiv.org/abs/1206.6392

[14] G. Hinton, "A practical guide to training restricted boltzmann machines," *Momentum*, vol. 9, no. 1, p. 926, 2010.

[15] I.-T. Liu and B. Ramakrishnan, "Bach in 2014: Music Composition with Recurrent Neural Network," *arXiv:1412.3191*, vol. 5, pp. 1–9, 2014. [Online]. Available: http://arxiv.org/abs/1412.3191

[16] A. Huang and R. Wu, "Deep Learning for Music," 2016. [Online]. Available: http://arxiv.org/abs/1606.04930

[17] D. Rizo, P. D. León, J. Pedro, C. Pérez-Sancho, A. Pertusa, and J. Iñesta, "A pattern recognition approach for melody track selection in MIDI files," *Distribution*, pp. 61–66, 2006. [Online]. Available: http://eprints.pascal-network.org/archive/00005787/

[18] P. C. Mahalanobis, "On the generalized distance in statistics," *Proceedings of the National Institute of Sciences (Calcutta)*, vol. 2, pp. 49–55, 1936.

[19] D. P. Kingma, T. Salimans, and M. Welling, "Variational Dropout and the Local Reparameterization Trick," *arXiv*, no. Mcmc, pp. 1–13, jun 2015. [Online]. Available: http://arxiv.org/abs/1506.02557

[20] K. Gregor, I. Danihelka, A. Graves, and D. Wierstra, "DRAW: A Recurrent Neural Network For Image Generation," *Icml-2015*, pp. 1–16, 2014.

[21] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *ICLR*, no. Ml, 2013, pp. 1–14. [Online]. Available: http://arxiv.org/abs/1312.6114

[22] G. Trigeorgis, F. Ringeval, R. Brückner, E. Marchi, M. Nicolaou, B. Schuller, and S. Zafeiriou, "Adieu Features? End-to-End Speech Emotion Recognition using a Deep Convolutional Recurrent Network," in *Proceedings 41st IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2016*, IEEE. Shanghai, P. R. China: IEEE, March 2016, pp. 5200–5204.

[23] E. Coutinho, G. Trigeorgis, S. Zafeiriou, and B. Schuller, "Automatically Estimating Emotion in Music with Deep Long-Short Term Memory Recurrent Neural Networks," in *Proceedings of the MediaEval 2015 Multimedia Benchmark Workshop, satellite of Interspeech 2015*, M. Larson, B. Ionescu, M. Sjöberg, X. Anguera, J. Poignant, M. Riegler, M. Eskevich, C. Hauff, R. Sutcliffe, G. J. Jones, Y.-H. Yang, M. Soleymani, and S. Papadopoulos, Eds., vol. 1436. Wurzen, Germany: CEUR, September 2015, 3 pages.