Compress to Create

Jean-Pierre Briot

Compress to Create

Jean-Pierre Briot
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
UNIRIO, Rio de Janeiro, RJ 22290-250, Brazil
Jean-Pierre.Briot@lip6.fr

Abstract: *The current tsunami of deep learning has already conquered new areas, such as the generation of creative musical content. The motivation is in using the capacity of modern deep learning architectures and associated training and generation techniques to automatically learn musical styles from arbitrary musical corpora and then to generate musical samples from the estimated distribution, with some degree of control over the generation. In this article, we analyze the use of autoencoder architectures and how their ability for compressing information turns out to be an interesting source for generation. Autoencoders are good at representation learning, that is at extracting a compressed and abstract representation (a set of latent variables) common to the set of training examples. By choosing various instances of this abstract representation (i.e., by sampling the latent variables), one may efficiently generate various instances within the style which has been learnt. Furthermore, one may use more sophisticated ways of controlling the generation, like interpolation, recursion, or objective optimization, as will be illustrated by various examples.*

Keywords*: Deep learning, Autoencoder, Latent variables, Music generation, Control.*

I. Introduction

One of the first documented case of algorithmic composition (i.e., using a formal process, including steps (algorithm) and components, to compose music), long before computers, is the Musikalisches Wurfelspiel (Dice Music), attributed to Mozart. A musical piece is generated by concatenating randomly selected (by throwing dices) predefined music segments composed in a given style (Austrian waltz in a given key).

The first musics generated by computer appeared in the late 1950s, shortly after the invention of the first computers. The Illiac Suite is the first score composed by a computer [HI59] and was an early example of algorithmic music composition, making use of stochastic models (Markov chains) for generation, as well as rules to filter generated material according to desired properties. Note that, as opposed to the previous case which consists in rearranging predefined material, abstract models (transitions and constraints) are used to guide the generation.

One important limitation is that the specification of such abstract models, being rules, grammar, or automata, etc., is difficult (reserved to experts) and error prone. With the advent of machine learning techniques, it became natural to apply them to learn models from a corpus of existing music. In addition, the method becomes, in principle, independent of a specific musical style (e.g., classical, jazz, blues, serial). More precisely, the style is actually defined *extensively* by (and learnt from) the various examples of music curated as the training examples.

The two main abstract models which can be induced by machine learning techniques are Markov models and artificial neural networks[1]. Since the mid 2010s, deep learning – the 3rd wave

[1]Some tentative comparison, pros and cons, of neural networks and Markov models for learning musical style and for generation may be found in [BHP19, Section 1.2.3].

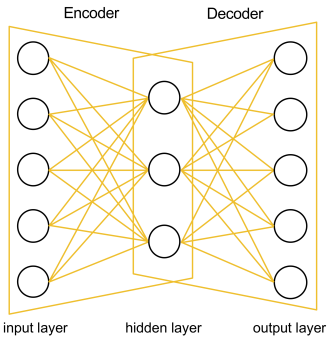

Figure 1: *Autoencoder architecture*

of artificial neural networks – has been producing striking successes and is now used routinely for applications such as image recognition, voice recognition and translation. A growing area of application of deep learning techniques is the *generation of content*, notably music but also images. Various types of artificial neural network architectures (feedforward, recurrent, etc.) are being used[2]. In this article, we will focus on a specific type of artificial neural network, *autoencoders*, and how they turn out an interesting approach for generating music.

II. Related Work and Organization

A recent book about deep learning techniques for music generation is [BHP19], with some shorter introduction and survey in [Bri20]. Some general surveys about of AI-based methods for algorithmic music composition are [PW99] and [FV13], as well as books [Cop00] and [Nie09]. A complete book about deep learning is [GBC16]. There are various introductory books (and courses) about artificial neural networks, including a very good course by Andrew Ng on Coursera. For the following of the article, actually there is no need of much previous knowledge about neural networks, besides viewing an artificial neural network as successive layers of multinomial (multiclass) logistic regression.

In this article, in Section III we will introduce the (actually very simple) concept of autoencoder, the way to represent music and to train it on a corpus of Celtic melodies. Section IV will introduce a straightforward way of using an autoencoder to generate new melodies, illustrated by various examples. Section V will discuss some approaches for controlling the generation of the melodies. Section VI will discuss some limitations and further developements, before concluding this article.

III. Autoencoder

An *autoencoder* is a feedforward[3] neural network with exactly one hidden layer and with a structural constraint: the number of nodes of the output layer is equal to the number of nodes of the input layer. The output layer actually *mirrors* the input layer, creating its peculiar symmetric diabolo (or sand-timer) shape aspect, as shown in Figure 1.

[2] See a classification and analysis, e.g., in [Bri20].

[3] *Feedforward* neural network, also named *multilayer Perceptron (MLP)*, is the most basic and common type of artificial neural network. It is composed of an arbitrary sequence of layers composed of artificial neurons.

Figure 2: *Compressed cars by César (left) and by Ichwan Noor (right)*

i. Training

An autoencoder is trained with each of the examples as the input and as the output, trying to minimize the difference between the reconstructed data and the original input data. As the hidden layer usually has fewer nodes than the input layer, the *encoder* component must *compress* information[4], while the *decoder* component has to *reconstruct*, as accurately as possible, the initial information. This forces the autoencoder to *discover* significant (discriminating) *features* to encode useful information into the hidden layer nodes, considered as a vector of *latent variables*.

ii. Generation

Indeed the motivation for an autoencoder is neither in just learning the identity function and nor in the direct compressing of data, as opposed to some experiments in using compression for creating art, e.g., the compressed cars by the sculptor César in the 1960s and more recently by the sculptor Ichwan Noor (see Figure 2). The *latent vector* of an autoencoder constitute a compact representation (some kind of label [Sun17]) of the common features of the learnt examples. By *instantiating* this latent vector and *decoding* it (by feedforwarding it into the decoder), we can generate a new musical content corresponding to the values of the latent variables and in the same format as the training examples.

iii. Representation and Encoding

In order to use an autoencoder with music, we need to define a way to represent that music. As in this article we focus on *algorithmic music composition*, we will consider a *symbolic* representation (of notes and durations), as opposed to some *audio* representation (waveform signal or spectrum). We choose a *piano roll* representation, for its simplicity. Piano roll (and its name) is inspired from automated mechanical pianos with a continuous roll of paper with perforations (holes) punched into it. In practice, it is a two dimensional table with the x axis representing the successive time steps and the y axis the pitch, as shown in Figure 3.

[4]Compared to traditional dimension reduction algorithms, such as principal component analysis (PCA), feature extraction by an autoencoder is nonlinear, thus more general, but it does not ensure orthogonality of the dimensions, as we will see in Section VI.ii.

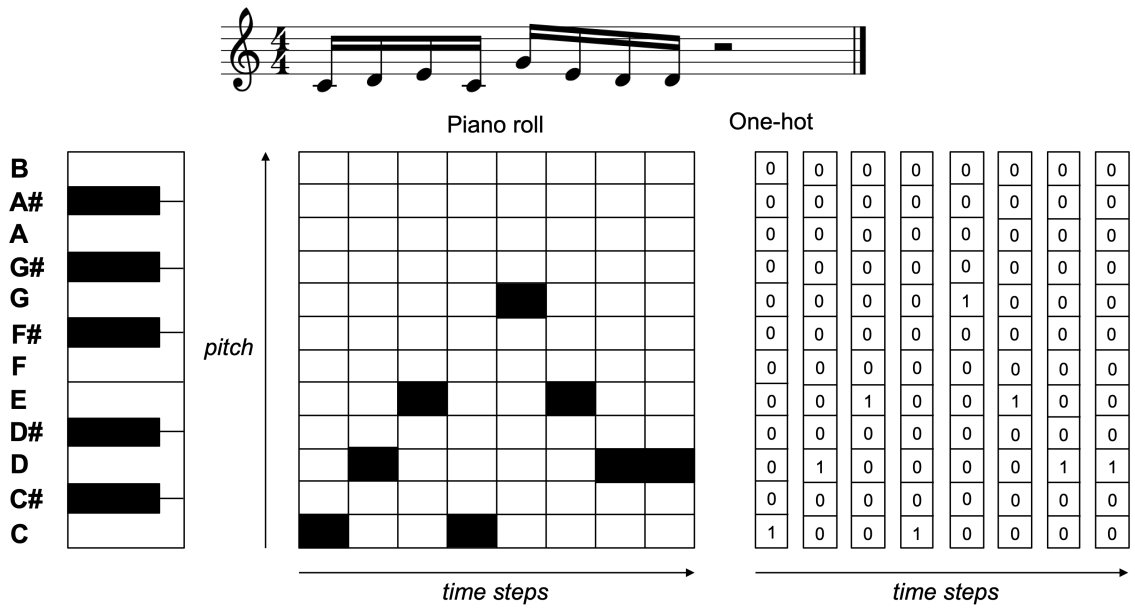

Figure 3: *Example of: (top) score; (middle) piano roll; (right) one-hot encoding – with a time step of a sixteenth note*

There is still an additional step from the representation to the artificial network input, this is the *encoding* of a representation (of a musical content). It consists in the *mapping* of the representation (composed of a set of *variables*, e.g., pitch or dynamics) into a set of *inputs* (also named *input nodes* or *input variables*) for the neural network architecture. The most frequent type of encoding is *one-hot-encoding*[5], where a discrete or a categorical variable is encoded as a *categorical variable*, through a vector with the number of all possible elements as its length. Then, to represent a given element, the corresponding element of the *one-hot vector*[6] is set to 1 and all other elements to 0. For instance, the pitch of a note is represented as shown in the right part of Figure 3[7].

Note that a *global time step* has to be fixed and usually corresponds, as stated by Todd in [Tod89], to the greatest common factor of the durations of all the notes to be learned. In the case of the corpus that we will consider, it is a sixteenth note. Also note that, there is no way to distinguish between a long note and a repeated short note[8]. Therefore, we will use the solution proposed in [HPN17] to consider holding curent note (a *hold*) as a special kind of note (pitch). This solution is simple, but its main limitation is that it only applies to the case of monophonic melodies[9]. We will also consider silence (rest) as a special kind of note[10]. These two special cases will be added to the set of possible note pitches for the one-hot vector.

[5]The advantage of *one-hot encoding* over *value encoding* (direct encoding of a variable as a scalar) is its robustness against numerical operations approximations (discrete versus analog), at the cost of a high cardinality and therefore a potentially large number of nodes for the architecture.

[6]The name comes from digital circuits, *one-hot* referring to a group of bits among which the only legal (possible) combinations of values are those with a single *high* (hot!) (1) bit, all the others being *low* (0).

[7]The Figure also illustrates that a piano roll could be straightforwardly encoded as a sequence of one-hot vectors to construct the input representation of an architecture, as we will see in Figure 6.

[8]Actually, in the original mechanical paper piano roll, the distinction is made: two holes are different from a longer single hole. The end of the hole is the encoding of the end of the note.

[9]Which is the case for Celtic melodies. Polyphonic music would need to be represented as different voices/tracks.

[10]For the reason discussed in [BHP19, Section 4.11.7].

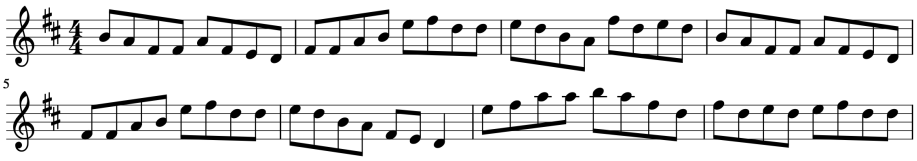

Figure 4: *"The Green Mountain" (first 8 measures)*

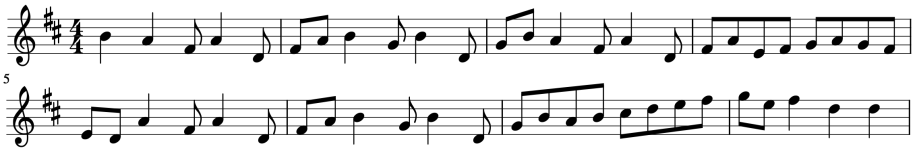

Figure 5: *"Willa Fjord" (first 8 measures)*

iv. Learning Celtic Melodies

In this article, we will use as corpus a set of Celtic melodies, selected from the folk music repository "The Session" [Kei16]. In practice, we selected 75 melodies, all in 4/4, in D Major key, and tagged as "Reel" (a type of Celtic dances). Two examples, "The Green Mountain" and "Willa Fjord", are shown[11] in Figures 4 and 5, respectively.

The shortest melody in the corpus is 8 measures long and the shortest note duration is a sixteenth note. The lowest note pitch is G_3 and the highest note pitch is B_5. Thus, the number of possible notes within the [G_3, B_5] interval is 29. The size of the final one-hot vector is thus 31 (after adding the hold and rest cases). The size of the the input representation is therefore: 8 (measures) $\times$ 16 (sixteenth notes per measure) $\times$ 31 (size of the one-hot vector) = 8 $\times$ 16 $\times$ 31 = 3,968.

v. Architecture

Successive melody time slices are encoded into successive one-hot vectors which are concatenated and directly mapped to the input nodes of the neural network autoencoder architecture. In Figure 6, each blackened vector element, as well as each corresponding blackened input node element, illustrate the specific encoding (one-hot vector index) of a specific note time slice, depending on its actual pitch (or a note hold in the case of a longer note). The dual process happens at the output. Each grey output node element illustrates the chosen note (the one with the highest probability), leading to a corresponding one-hot index, leading ultimately to a sequence of notes.

The input layer and the ouput layer of the autoencoder architecture have 3,968 nodes. The hidden layer has an arbitrary number of nodes[12], e.g., 1500 nodes. The output layer activation function (as well as the hidden layer activation function) is sigmoid and the loss (reconstruction error) function is binary cross entropy[13]. The optimizer algorithm is ADAM and training hyperparameters are: number of epochs = 100 and minibatch size = 20. We use the Keras framework as front end, Theano platform as the back end and our own made representation library[14]. Purposively, we do not use any additional optimization, as to keep it simple and generic.

[11] Actually, only their 8 first measures, which is the actual length of the melodies that will be considered, as explained just below.

[12] Which will be varied, as we will see in Section IV.i.

[13] See, e.g., [BHP19, Sections 5.5.3 and 5.5.4] for details about the reasons of these choices.

[14] It transforms a musical score into data for the architecture and vice-versa. We designed it for our course at UNIRIO which is available at `http://www-desir.lip6.fr/~briot/cours/unirio3/`. It uses the Music21 symbolic music representation library as pivot and also for reading MIDI and ABC music formats.

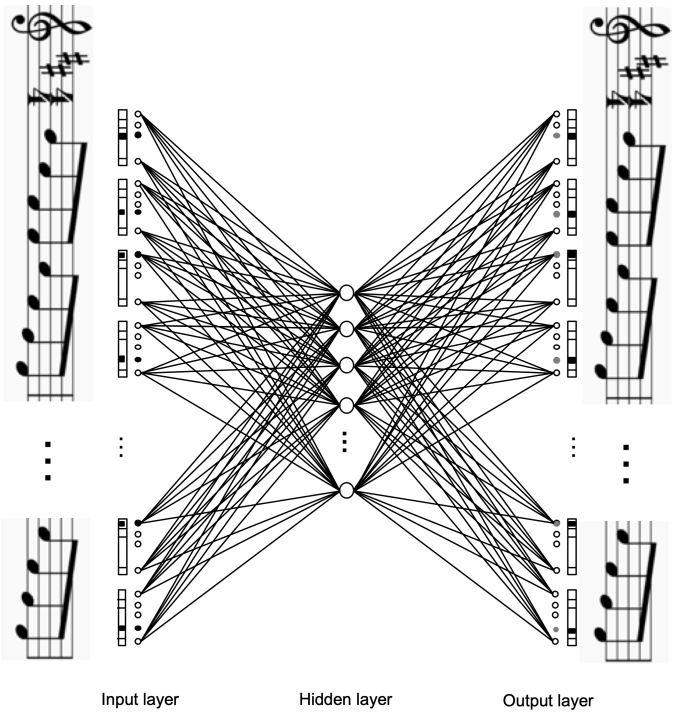

Figure 6: *Autoencoder architecture for learning melodies*

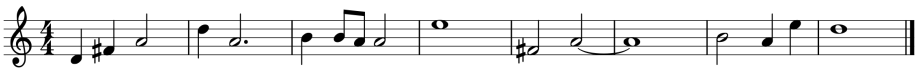

Figure 7: *Example of melody generated from a random latent vector by the decoder component of the autoencoder ($h = 1500$) trained on the Celtic melodies corpus*

Training the architecture proceeds by presenting an example of melody at the input layer and at the output layer (as the target for the reconstruction)[15]. The training procedure will incrementally adjust the connexion weights between neurons in order to minimize the reconstruction error.

IV. Generation

The model having being trained, it may be used for generation. As explained in Section III.ii, we have to instantiate the latent vector, usually denoted as z, and feedforward it into the decoder component. This will reconstruct some melody corresponding to the compressed version. Figure 7 shows an example of melody generated, from values randomly generated[16].

i. Size of the Hidden Layer

For simplification, we will name h the size of the hidden layer (which is also the size of the latent vector z). Setting the value of h is an important decision. If h is large, close to the input (and

[15] Actually, a mini batch of examples, randomly selected from the training set, is used for each epoch.

[16] The ranges of the possible values for each latent variable may be determined by computing the lowest and the highest values of latent variables for each training example, see Figure 8.

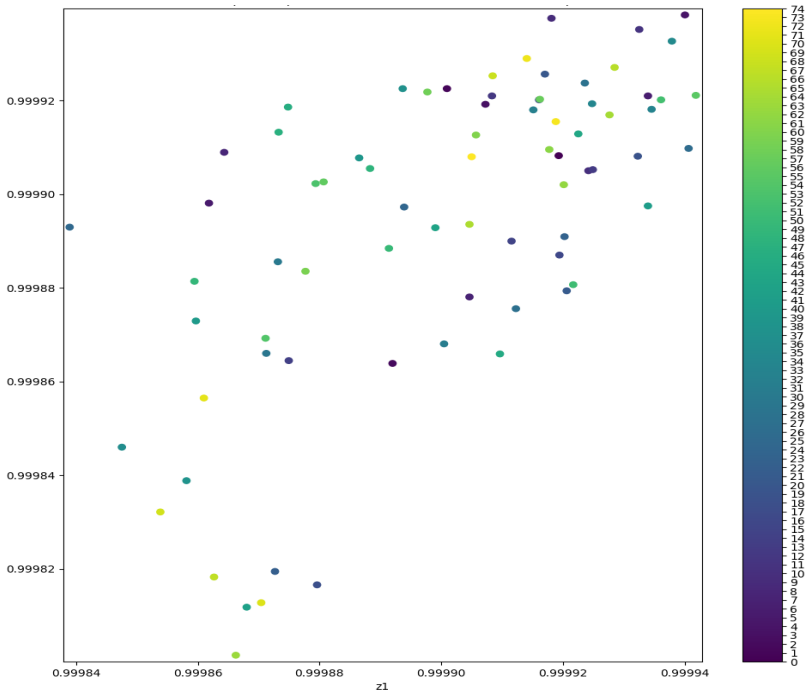

Figure 8: *Values of z generated by the encoder component of the autoencoder (h = 2) for each of the 75 examples of Celtic melodies. Each dot and its color (from blue to yellow) corresponds to each melody of the corpus (from #0 to #74)*

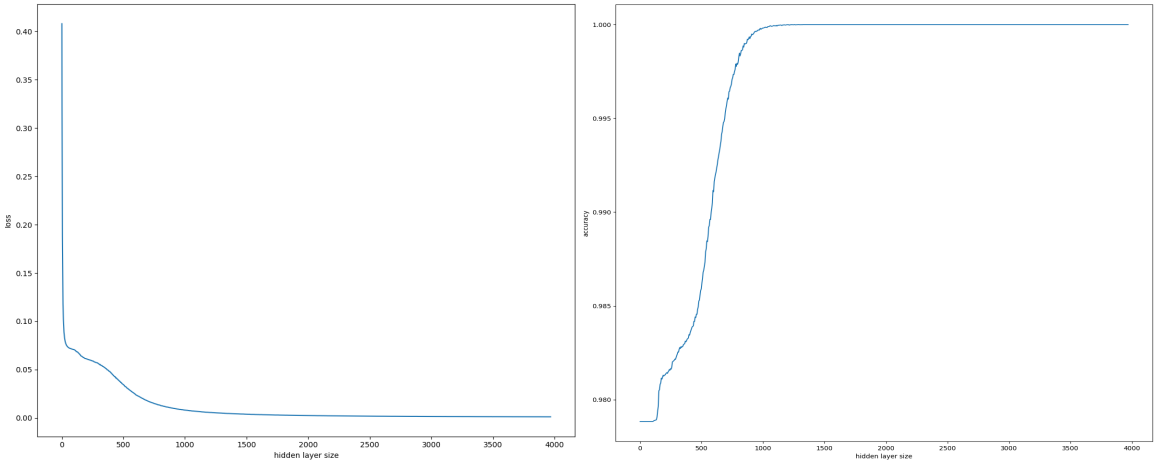

Figure 9: *(left) Loss and (right) Accuracy of the reconstruction by the autoencoder, in function of h, the size of the hidden layer*

output) layer size, reconstruction will be almost perfect or even perfect, but there will be many latent variables to be instantiated in order to generate melodies[17]. If h is small, the control of the generation will be easier to understand, specially in the case of $h = 2$ where the latent vector can

[17] And moreover, the autoencoder may not be enough forced to extract interesting features.

be visualized in a 2D-figure[18], as shown for each example of the corpus[19] in Figure 8, but the reconstruction will not be optimal.

Figure 9 shows the loss (the error of the reconstruction) and the accuracy (the precision of the reconstruction), in function of h. Our experiment shows that $h = 1.258$ is the lowest value for which the accuracy is equal to 1, that is, for which the reconstruction of all training examples is perfect.

In Figure 10, we show the reconstruction by the autoencoder of "The Green Mountain" melody, in function of the value of h. We could see that with $h = 1000$, the reconstruction is still perfect[20]. With $h = 750$, reconstruction has only a minor error: the last note of the 6th measure, a D quarter note, has been substituted by D and B 16ths notes. With $h = 500$, some further errors and simplifications appear, although most of melodic motives are still preserved. It is from $h = 150$ that the melody motives disappear and from $h = 100$ the melody gets over simplified and remains quasi stable.

It may seem surprising that, even with a little size of the hidden layer, the autoencoder could reconstruct partially an initial melody. What does the autoencoder is in fact to map the various dimensions corresponding to the various latent variables to some variational characteristics which vary among examples, e.g., main melodic motif, average duration of notes, etc.[21] What is common to all examples is stored into the connexion weights of the network, for it to be able to reconstruct a melody. What is specific to examples is stored into the latent variables. Therefore, changing the values of the latent variables will allow to change the melodies generated.

V. Approaches for Generation

i. Exploration

The latent space (the space of variation for z the latent vector of latent variables) may be explored with various operations to *control/vary* the generation of content, e.g., as summarized in [RER+18]:

- *translation*;
- *interpolation*;
- *averaging*;
- *attribute vector arithmetics*.

ii. Interpolation

We may for instance do interpolation (morphing) between two existing melodies, "The Green Mountain" (see Figure 4) and "Willa Fjord" (Figure 5) with, e.g., 5 steps of linear interpolation. In practice, we compute the value of z resulting from feedforwarding "The Green Mountain" into the encoder component of the autoencoder, as well as the value of z resulting from feedforwarding "Willa Fjord"; we compute the various values/steps for interpolating z; and we feedforward each value into the decoder component of the autoencoder to reconstruct the corresponding successive

[18] Let us think of the analogy of approximating a 3D location on earth onto a 2D map and even onto a 1D road on this map.

[19] To compute the value of z corresponding to a given melody, we just need to feedforward the melody data into the encoder component of the autoencoder and retrieve the values of the hidden layer nodes, i.e. the latent variables.

[20] The reconstruction of "The Green Mountain" is perfect but it is not the case for all the training examples, otherwise the accuracy (for all examples) would have been equal to 1. Also note that, as opposed to the initial score of "The Green Mountain" in Figure 4 which has a key signature (with two ♯, i.e. D Major) because the actual key was part of the specification, the score of the reconstruction has no key signature but the notes are indeed equivalent.

[21] As will be discussed in Section VI.ii.

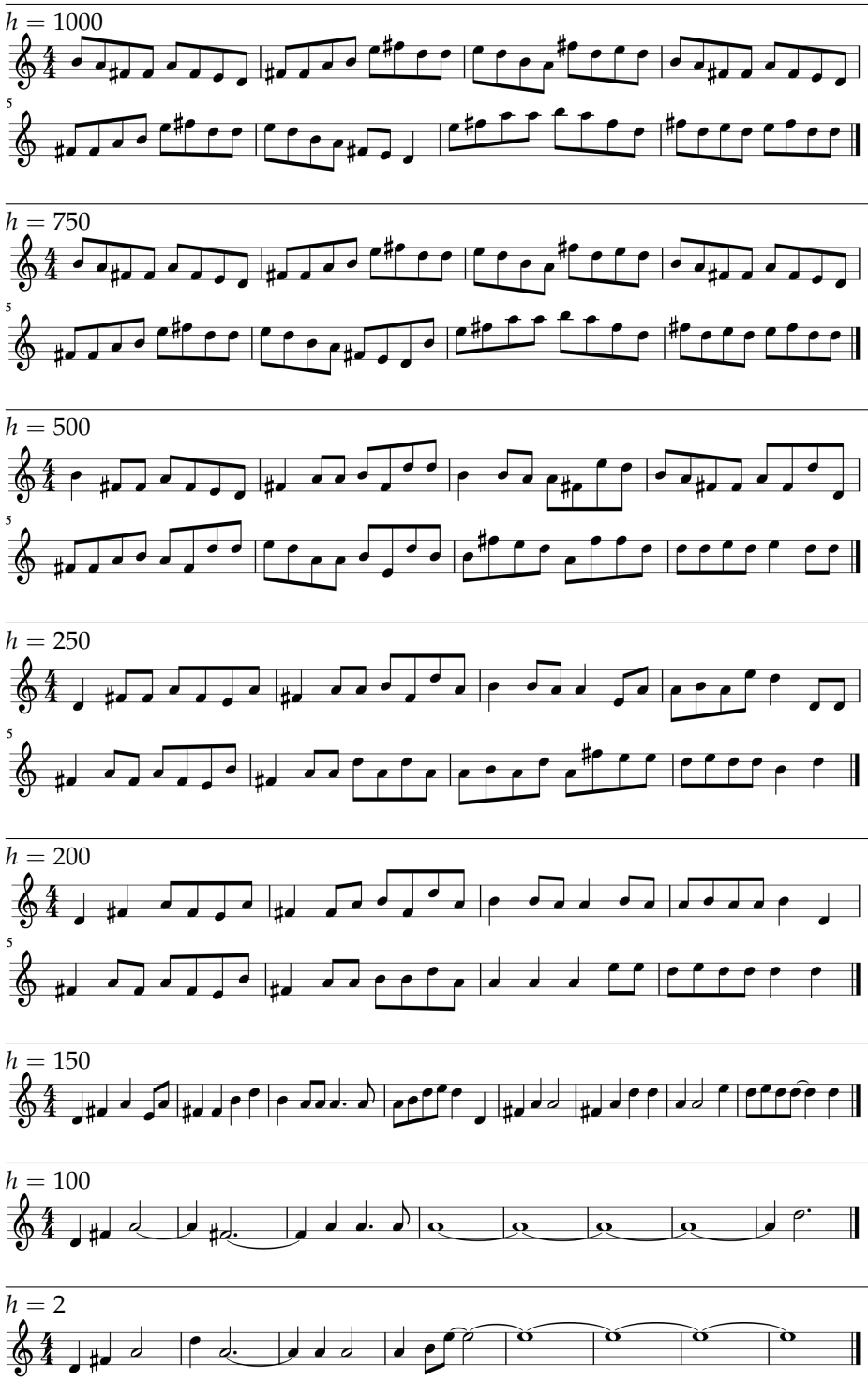

Figure 10: *(from top to bottom) Reconstruction by the autoencoder of "The Green Mountain" for h = 1500, 750, 500, 250, 200, 150, 10, 2*

melodies. In the case of $h = 1500$, the resulting melodies are shown in Figure 11. We could see that the interpolation, although correct (it maps the start and the target), is not uniform. Step 1 is equal to start and step 4 is equal to target. This discontinuity limitation will be analyzed and addressed in Section VI.i.

We can also do interpolation between arbitrary values of z. With $h = 2$, we interpolate the value of z_1 between its minimum value and its maximum value[22], while z_2 stays constant to its mean value, as shown in Figure 12. Unfortunately, the resulting melody, shown in Figure 13, is actually constant for all steps. The same happens, with a different generated melody shown in Figure 14, when interpolating the value of z_2. This is actually another illustration of some limitation of the ways the autoencoder dispatches the melodies in the latent space, as will be analyzed in Section VI.i.

iii. Atribute Vector Arithmetics

The idea is to specify some attribute vector capturing a given characteristic (e.g., long notes, high pitch range, etc.) and to apply it to an existing example in order to influence it. An *attribute vector* is computed as the mean of the latent vectors of all examples sharing that characteristic. As an example, let us augment the corpus with a set of (80) soprano melodies from Bach chorales (BWV 347 is shown at Figure 15) and train the autoencoder accordingly[23]. Let us compute the mean of latent vectors of all Bach chorales soprano voices examples. Then, let us consider the Celtic melody "The Green Mountain" (shown in Figure 4), compute its latent vector, add the "Bach chorales" attribute latent vector and create the corresponding melody, shown in Figure 16. We can see that the original melody has been simplified, with longer notes, such as in the Bach chorales corpus, while keeping the basic melodic motif. Figure 17 shows[24] the position of this chimera melody regarding the Celtic and Bach corpus. We can see that the original "The Green Mountain" melody (circled blue spot) is on the left, within the main part of the Celtic corpus (dark purple blue spots), while its "Bach-ized" version (circled yellow spot) is right at the center of the Bach corpus (green spots).

iv. Recursion

In [KCK16], Kazakçi *et al.* proposed an original way of content generation from autoencoders. The idea is to feedforward a random initial content (random melody representation) into the autoencoder and recursively feedforward the generated output as the input until the output gets to a fixed point. They have experimented with a dataset of handwritten numerical digits (the MNIST dataset for handwritten digits recognition) and generated new types of visual patterns that they name "digits that are not". We have applied their approach on our autoencoder trained on Celtic melodies. Figure 18 shows an example of progressive refinement of a melody. Note that each melody generated corresponds to some attractor of the network and their number is finite[25].

[22] These values, as well as mean values, for all latent variables, are computed from the latent vectors for all training examples.

[23] We transpose Bach melodies into the D Major key in order to be aligned with the Celtic corpus. Also, as some Bach melodies are outside of the Celtic pitch range, the pitch range must be adjusted and as a consequence the one-hot vector size (and the autoencoder input layer size) is changed.

[24] It uses the T-distributed Stochastic Neighbor Embedding (t-SNE) nonlinear dimensionality reduction machine learning algorithm for visualization [vdMH08]. t-SNE models each high-dimensional object by a two (or three) dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points. The difference with an autoencoder is that t-SNE does not try to to minimize a reconstruction error but instead tries to preserve the neighborhood distances.

[25] This is some kind of *mode collapse*, as for generation by generative adversarial networks (GAN) [KAHK17].

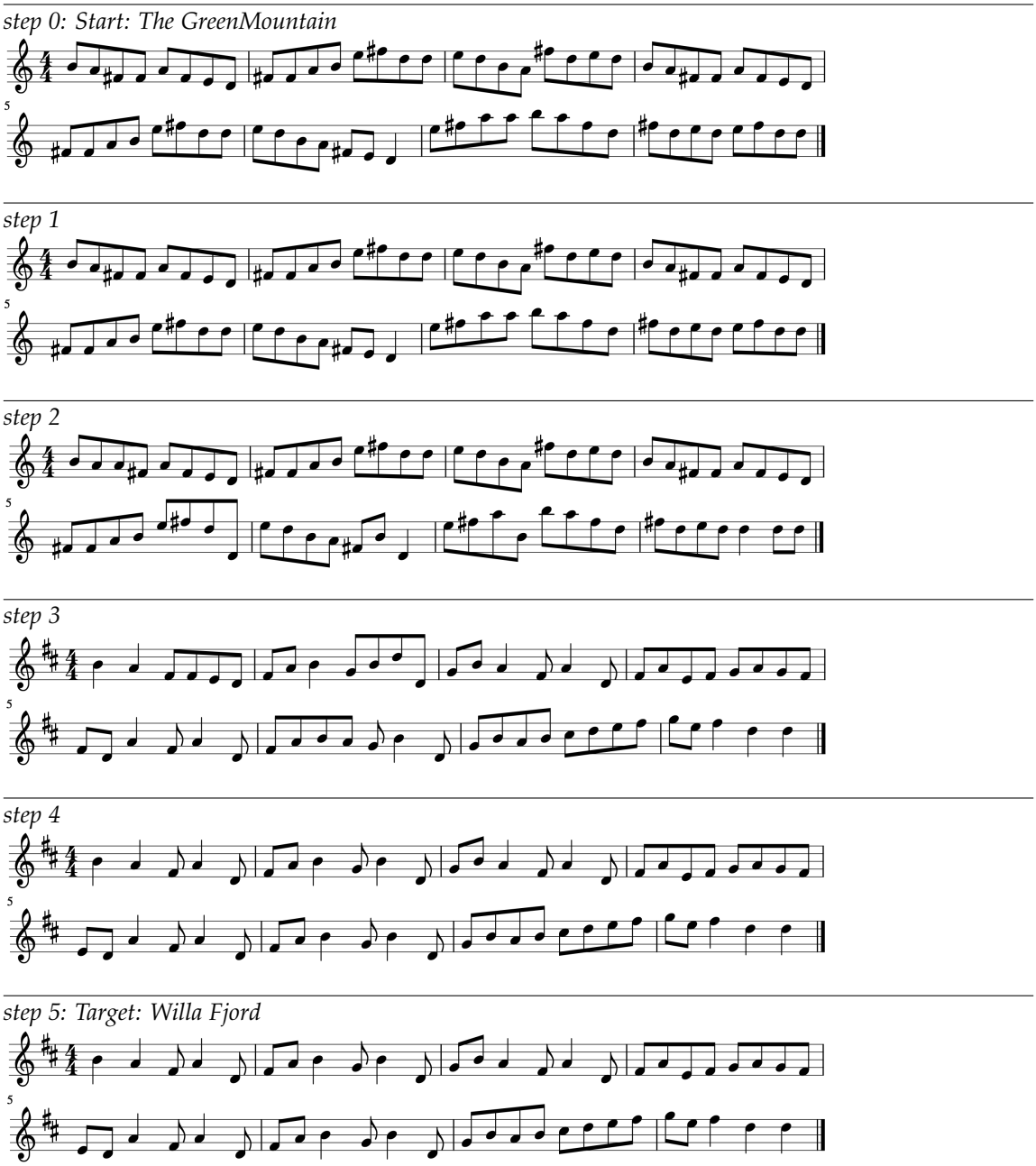

Figure 11: *(from top to bottom) Melodies resulting from the interpolation (5 steps) by the autoencoder (h = 1500), from "The Green Mountain" to "Willa Fjord"*

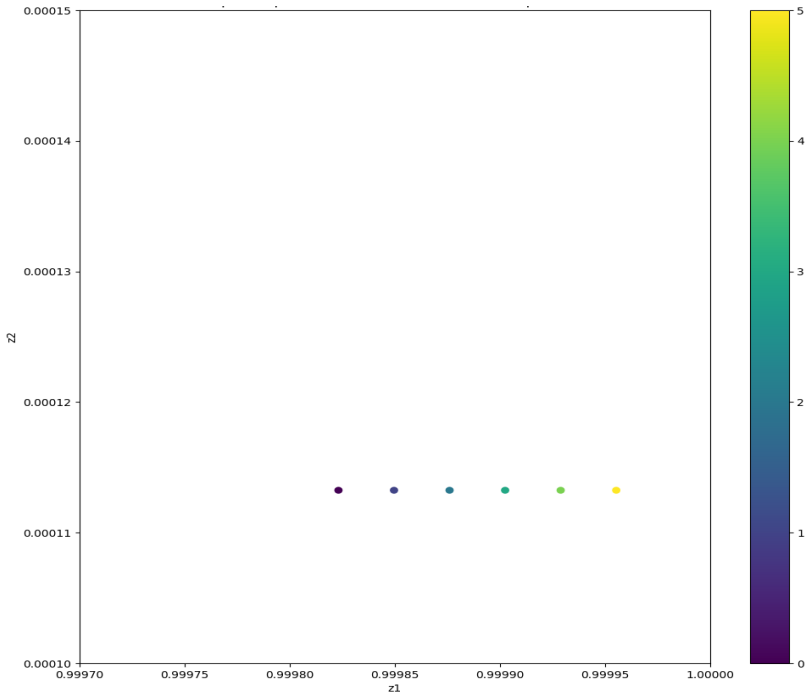

Figure 12: *Values of z for the interpolation (5 steps) of z_1 (from its min value (dark purple blue spot) to its max value (yellow spot)), while z_2 is constantly equal to its mean value*

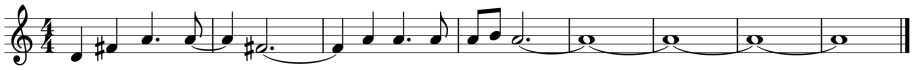

Figure 13: *Melody resulting from the interpolation (5 steps) by the autoencoder (h = 2) of the value of z_1 (from its min value to its max value), while z_2 is constantly equal to its mean value*

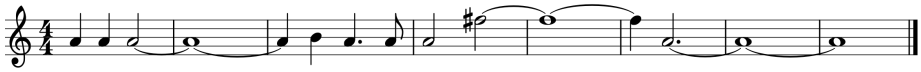

Figure 14: *Melody resulting from the interpolation (5 steps) by the autoencoder (h = 2) of the value of z_2 (from its min value to its max value), while z_1 is constantly equal to its mean value*

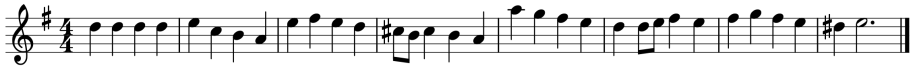

Figure 15: *Bach Chorale BWV 347 soprano voice (transposed into D Major key) first 8 measures*

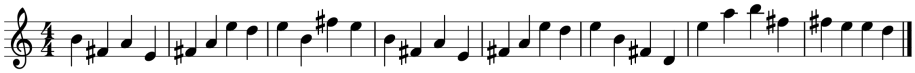

Figure 16: *"The Green Mountain" transformed into a Bach chorales-like melody by the autoencoder (h = 1500)*

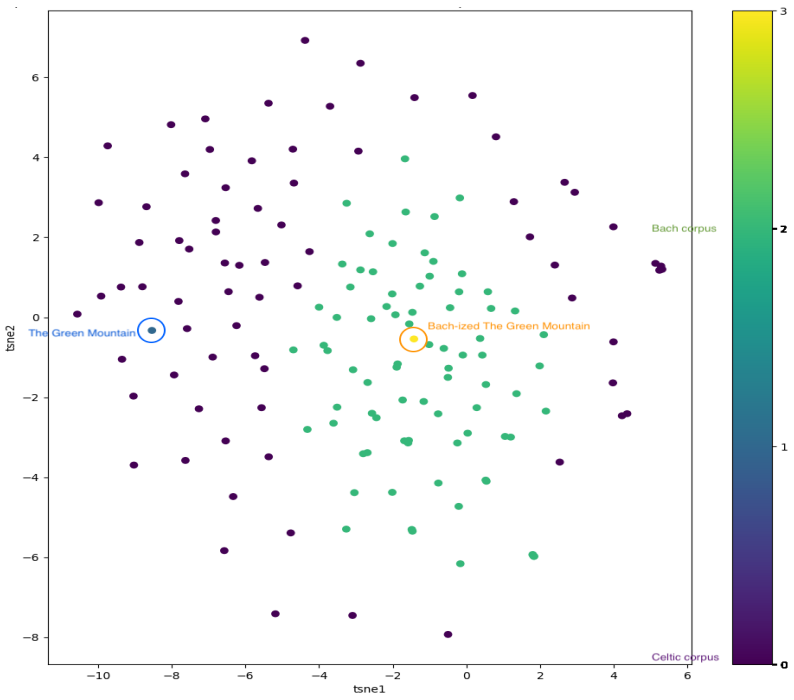

Figure 17: *TSNe representation of the following melodies: joint Celtic corpus (dark purple blue spots) – including "The Green Mountain" (circled blue spot) –; Bach corpus (green spots); and "Bach-ized The Green Mountain" (circled yellow spot) generated by the autoencoder (h = 1500)*

v. Objective Optimization

Another approach is by controlling the exploration of the latent space (the input of the decoder) by optimizing some property. This idea has been initially proposed in 1988 by Lewis and named by him *creation by refinement* [Lew88]. The idea is to "revert" the standard way of using gradient descent for standard task – adjust the connexion *weights* in order to *minimize* the *classification error* –, into a very different task – adjust the *input* in order to *maximize* an expected *classification result*[26]. This approach can be seen as the precursor of various approaches for controlling the creation of a content by maximizing some target property[27], such as Deep Dream [MOT15] and style transfer [GEB15], see more details in [Bri20].

We apply this approach to optimize the value of the latent vector of the autoencoder to match some objective[28]. In practice, a vector of random values is produced, as initial values of the latent vector. Then, an optimization algorithm[29] is applied on the latent vector to maximize the objective. We experimented with three different objectives for the melody:

[26] In Lewis' initial proposal, the neural network which is a feedforward binary classifier is at first trained with positive and negative examples of what he names "well formed" melodies, defined as follows: 1) using only the unison, 3rd and 5th intervals between notes and 2) following some scale degree stepwise motion. Then, a vector of random values is used as the initial input of the network and refined as to obtain a positive classification (i.e. a well formed melody). See more details in [Lew88].

[27] The target property may be of any kind as long as it may be measured and thus optimized.

[28] Sun [Sun17] may have been the first author to propose this approach for autoencoders. In his experiments, the target property is to generate a melody consonant to an existant melody. Note that he used *stacked autoencoders*, i.e. nested autoencoders with a decreasing number of hidden layers.

[29] Gradient-based or even simple random generate-and-test.

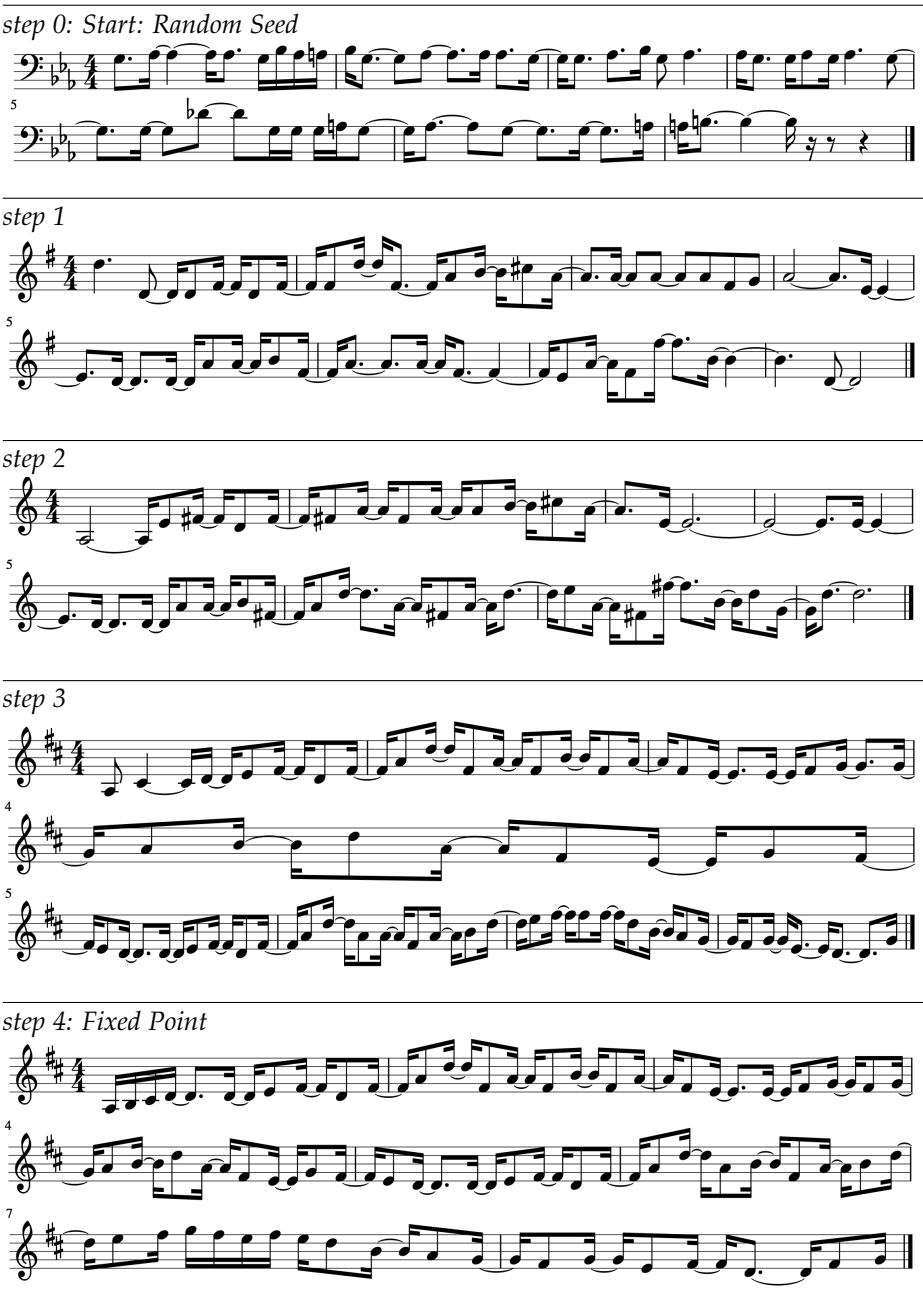

Figure 18: *(from top to bottom) Example of successive melodies generated by the autoencoder (h = 1500) for each step of the recursion*

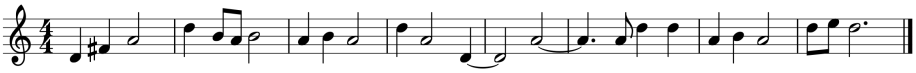

Figure 19: *Example of melody generated by the autoencoder (h = 1500) with the objective of its first note being a* C_4

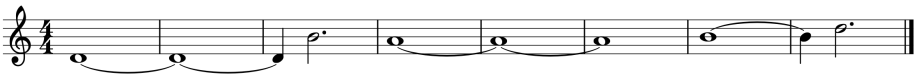

Figure 20: *Example of melody generated by the autoencoder (h = 1500) with the objective of maximizing the number of hold*

- first note to be as close as possible to a C_4[30], shown in Figure 19;
- maximize the number of hold (i.e. having notes as long as possible), shown in Figure 20; and
- minimize the number of hold (i.e. having notes as short as possible), shown in Figure 21.

VI. Further Developments

i. Variational Autoencoder

As noted in Section V.ii, although producing interesting results, the autoencoder suffers from some discontinuity limitation. We could see that the interpolation, although correct (it maps the start and the target), is not continuous and thus creates discontinuities in the generation when exploring the latent space. The reason, as discussed in [Roc19], is that the autoencoder is solely trained to encode and decode with a minimal loss, no matter how the latent space is organized. The approach is then to regulate the latent space to ensure that the latent space has better continuity properties for the generation.

A *variational autoencoder* (VAE) [KW14] is a refinement of an autoencoder with the added constraint that the encoded representation, i.e. the latent variables, follows some prior probability distribution[31], usually a Gaussian distribution. This regularization ensures two main properties: *continuity* (two close points in the latent space should not give two completely different contents once decoded) and *completeness* (for a chosen distribution, a point sampled from the latent space should provide a "meaningful" content once decoded) [Roc19]. The price to pay is some larger reconstruction error, but the tradeoff between reconstruction and regularity can be adjusted depending on the priorities (as we will see in Section VI. ii).

[30] Note that the objective is not completely fulfilled. The first note of the generated melody is a D_4. This makes sense because the corpus of melodies is in the key of D Major and many of them start with a D. This is important to remember that we can optimize some objective but within the boundaries of the representation that the autoencoder has learnt. Opening up this structural restriction is possible but with another generation (and architectural) model, see, e.g., structure imposition with a restricted Boltzmann machine (RBM) [LGW16].

[31] This constraint is implemented by adding a specific term to the cost function which computes the cross-entropy between the distribution for each latent variable and the prior distribution. The model and implementation is actually more sophisticated, instead of an encoder encoding an input as a single point, a variational autoencoder encodes it as a *distribution* over the latent space, from which the latent variables are sampled, as explained in [Roc19].

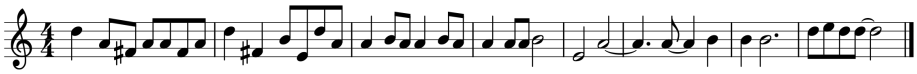

Figure 21: *Example of melody generated by the autoencoder (h = 1500) with the objective of minimizing the number of hold*

Figure 22 shows the melodies generated by the variational autoencoder with $h = 1500$ by interpolation between "The Green Mountain" and "Willa Fjord". By comparing it with the generation by the autoencoder (Figure 11), we could see that the interpolation is more continuous, at the price of some imperfect reconstruction of the two original melodies.

In the case of $h = 2$, we may compare the melodies generated by the variational autoencoder for the interpolation of values of z_1 and of z_2 (shown in Figures 23 and 24, respectively), to the melodies generated by the autoencoder (Figures 13 and 14).

Interestingly, our preliminary experiments show that in the case of attribute arithmetics (Section V.iii), recursion (Section V.iv) and objective optimization (Section V.v) generation strategies, the results are better when using the autoencoder than when using the variational autoencoder. This could suggest that variational autoencoders are not necessarily an improvement for all kind of generation strategies[32].

ii. Interpretation and Disentanglement

Another issue is that the characteristics (meaning, e.g., note duration range, note pitch range, motif, etc.) of the dimensions captured by the latent variables are automatically "chosen" by the autoencoder architecture (variational or not), in function of the training examples and the configuration. Thus, they can only be interpreted *a posteriori*. For instance, in Figures 23 and 24, we can observe that z_1 seems to capture the range as well as the average of note durations, while z_2 captures refinements of the melody motif.

Furthermore, as for the mapping between a genotype and a phenotype, there may be a many-to-many mapping between latent variables and characteristics. In fact, the dimensions captured by the latent variables are not independent (orthogonal), as in the case of Principal component analysis (PCA). However, various techniques[33] have being recently proposed to improve the disentanglement of the dimensions (see, e.g., [MRST19]). Some recent approaches also propose to "force" the meaning of latent variables, by splitting the decoder into various components and training them onto a specific dimension (e.g., rhythm or pitch melody) [YWW+19].

iii. RNN Encoder-Decoder and Variational Recurrent Autoencoder (VRAE)

A practical limitation of an autoencoder is that the size of the input (and output) layer is fixed and as a result also the length of the music generated. The solution is to combine: the *generative* property of the autoencoder with the *variable length* property of a recurrent neural network (RNN) architecture (see [BHP19, Section 6.5]). The idea is to embed a recurrent network (RNN) within the encoder and a similar RNN within the decoder (thus, named an *RNN Encoder-Decoder* [CvMG+14]), as shown in Figure 25.

A natural further step is to combine this with the variational characteristic of a variational autoencoder, resulting in what is named a *variational recurrent autoencoder* (VRAE) [FvA15], an example being the MusicVAE architecture [RER+18]. We will not further detail VAE and VRAE architectures here because of space limitation. Please see, e.g., [BHP19] for various examples of systems and results.

[32] Indeed, as pointed out in [GBC16, Section 20.10.3], there are still some troubling issues about variational autoencoders.

[33] Two examples of approaches are: 1) increasing the weight of the prior distribution conformance (the β-VAE approach) [HMP+17]; 2) ensuring that for a given dimension no other dimension will be present by using a classifier to check the equiprobability among the classes along other dimensions (the antagonist approach) [RTC19].

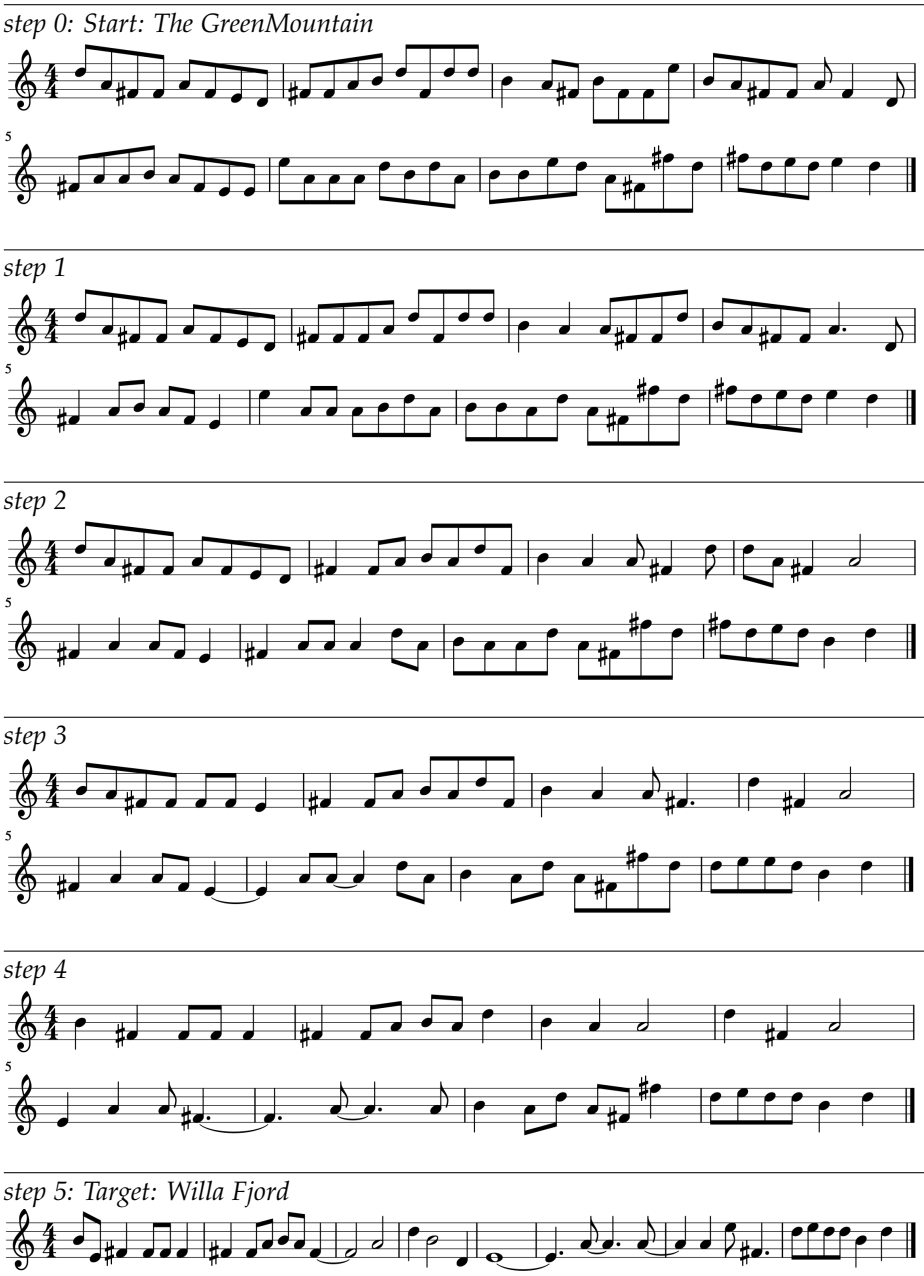

Figure 22: *(from top to bottom) Melodies resulting from the interpolation (5 steps) by the variational autoencoder (h = 1500), from "The Green Mountain" to "Willa Fjord"*

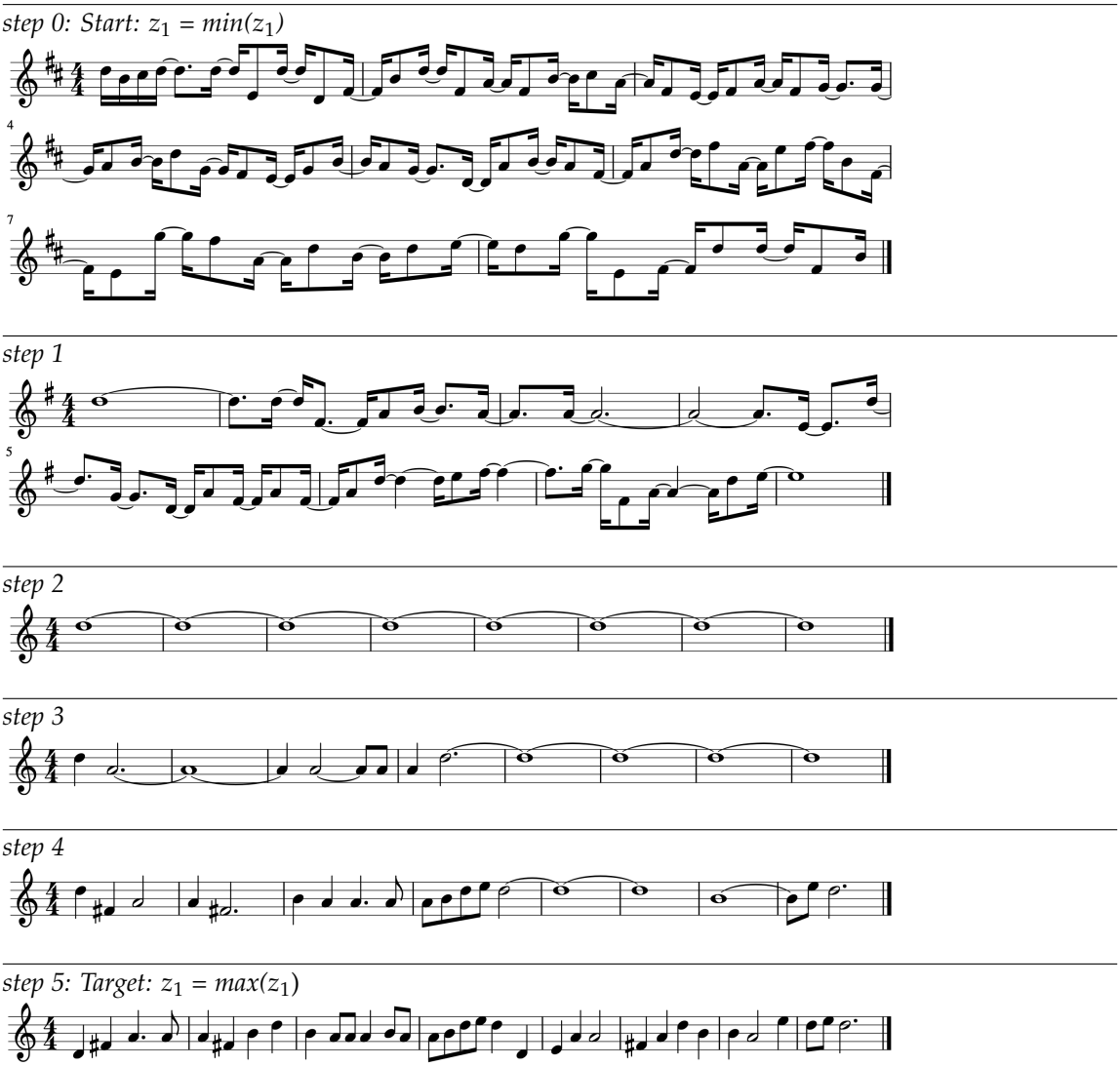

Figure 23: *(from top to bottom) Melodies resulting from the interpolation (5 steps) by the variational autoencoder (h = 2) of the value of z_1 (from its min value to its max value), while z_2 is constantly equal to its mean value*

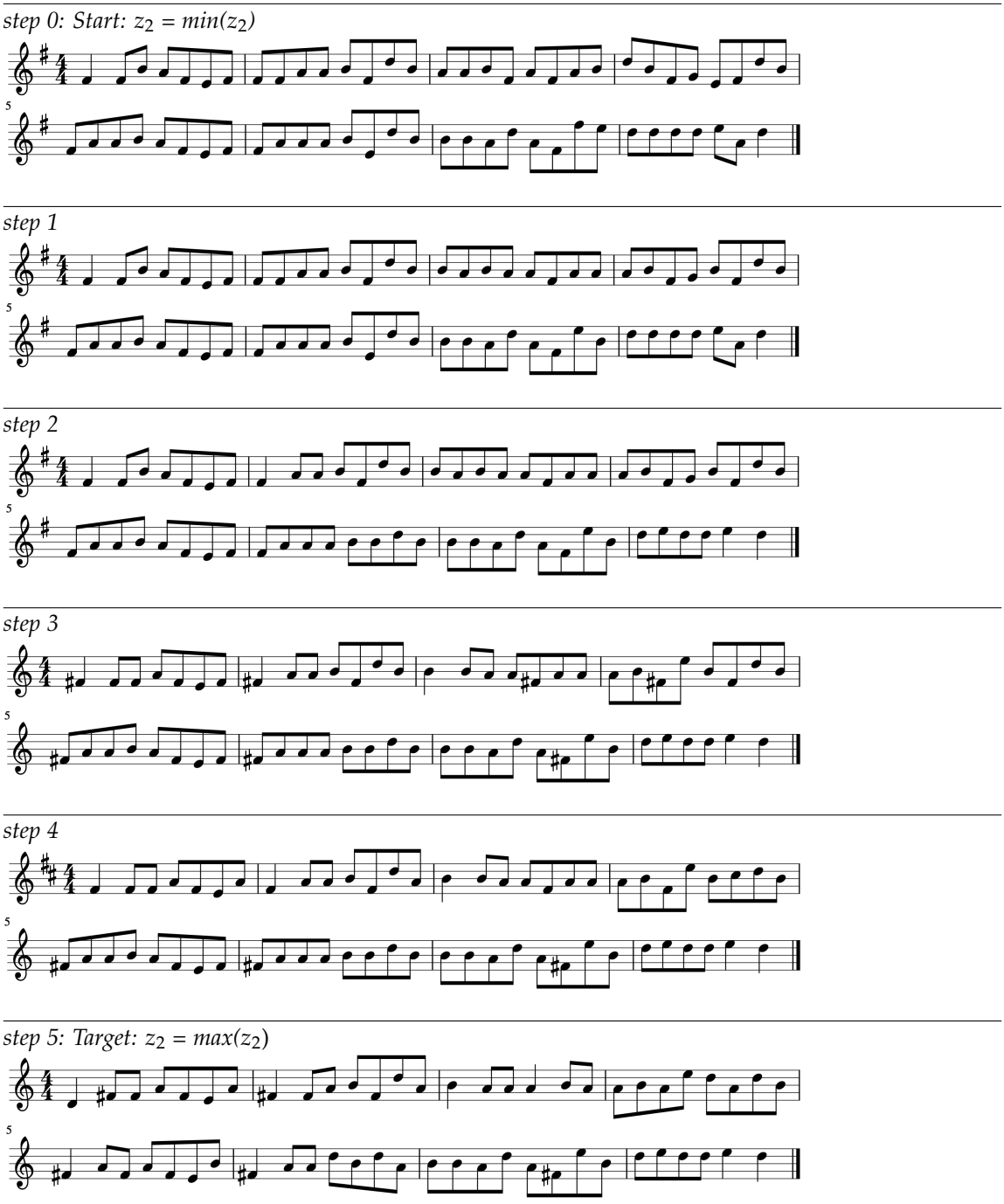

Figure 24: *(from top to bottom) Melodies resulting from the interpolation (5 steps) by the variational autoencoder (h = 2) of the value of z_2 (from its min value to its max value), while z_1 is constantly equal to its mean value*

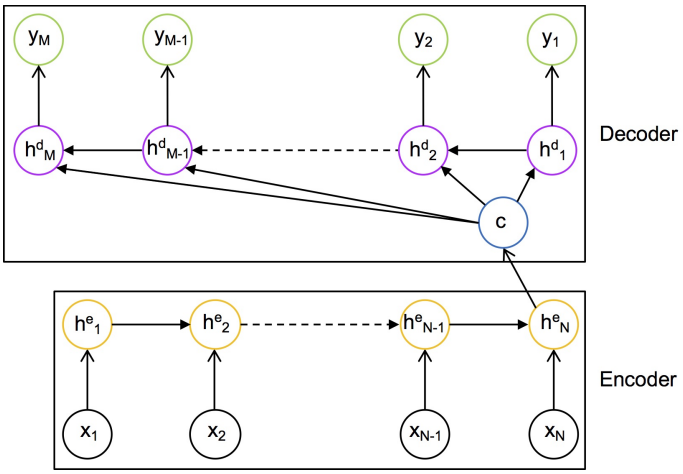

Figure 25: *RNN Encoder-Decoder architecture. Inspired from [CvMG+14]. The hidden layer h^e_t of the RNN encoder acts as a memory which iteratively accumulates information about the successive elements x_t of an input sequence read by the RNN encoder; resulting in a final state h^e_N; which is passed to the RNN decoder as the summary c of the whole input sequence; which then iteratively generates the output sequence by predicting the next item y_t given its hidden state h^d_t and the summary c as a conditioning additional input*

VII. Conclusion

The use of artificial neural networks and deep learning architectures and techniques for the generation of music (as well as other artistic contents) is a very active area of research. In this paper, we have introduced and illustrated the use of the notion of autoencoder to generate music. Various approaches have also been discussed to control the generation. Some potential output of our preliminary experiments shows that for some of the generation approaches, an autoencoder may seem to generate more interesting results than its refined version, a variational autoencoder. We hope that this article will help at showing the potential as well as open questions of using deep learning, and more specially autoencoders, for music generation.

Acknowledgements

We thank all participants to the course on this topic that we gave at UNIRIO in 2018–2019. We thank the editors of this journal for inviting us to write this article.

References

[BHP19] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. *Deep Learning Techniques for Music Generation*. Computational Synthesis and Creative Systems. Springer International Publishing, 2019.

[Bri20] Jean-Pierre Briot. From artificial neural networks to deep learning for music generation – History, concepts and trends, April 2020. arXiv:2004.03586.

[Cop00] David Cope. *The Algorithmic Composer*. A-R Editions, 2000.

[CvMG+14] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations

using RNN Encoder-Decoder for statistical machine translation, September 2014. arXiv:1406.1078v3.

[FV13] Jose David Fernández and Francisco Vico. AI methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research (JAIR)*, (48):513–582, 2013.

[FvA15] Otto Fabius and Joost R. van Amersfoort. Variational Recurrent Auto-Encoders, June 2015. arXiv:1412.6581v6.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning. MIT Press, 2016.

[GEB15] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, September 2015. arXiv:1508.06576v2.

[HI59] Lejaren A. Hiller and Leonard M. Isaacson. *Experimental Music: Composition with an Electronic Computer*. McGraw-Hill, 1959.

[HMP+17] Irina Higgins, Loïc Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework, April 2017.

[HPN17] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. DeepBach: a steerable model for Bach chorales generation, June 2017. arXiv:1612.01010v2.

[KAHK17] Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira. On convergence and stability of GANs, December 2017. arXiv:1705.07215v5.

[KCK16] Akin Kazakçi, Mehdi Cherti, and Balázs Kégl. Digits that are not: Generating new types through deep neural nets. In François Pachet, Amilcar Cardoso, Vincent Corruble, and Fiammetta Ghedini, editors, *Proceedings of the 7th International Conference on Computational Creativity (ICCC 2016)*, pages 188–195, Paris, France, June 2016. https://arxiv.org/abs/1606.04345v1.

[Kei16] Jeremy Keith. The Session, Accessed on 21/12/2016. https://thesession.org.

[KW14] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes, May 2014. arXiv:1312.6114v10.

[Lew88] John Peter Lewis. Creation by refinement: A creativity paradigm for gradient descent learning networks. In *IEEE International Conference on Neural Networks*, volume II, pages 229–233, San Diego, CA, USA, July 1988.

[LGW16] Stefan Lattner, Maarten Grachten, and Gerhard Widmer. Imposing higher-level structure in polyphonic music generation using convolutional restricted Boltzmann machines and constraints, December 2016. arXiv:1612.04742v2.

[MOT15] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Deep Dream, 2015. https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

[MRST19] Emile Mathieu, Tom Rainforth, N. Siddharth, and Yee Whye Teh. Disentangling disentanglement in variational autoencoders, June 2019. arXiv:1812.02833v3.

[Nie09] Gerhard Nierhaus. *Algorithmic Composition: Paradigms of Automated Music Generation*. Springer Nature, 2009.

[PW99] George Papadopoulos and Geraint Wiggins. AI methods for algorithmic composition: A survey, a critical view and future prospects. In *AISB 1999 Symposium on Musical Creativity*, pages 110–117, April 1999.

[RER+18] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, volume 80, pages 4364–4373. ACM, Montréal, PQ, Canada, July 2018.

[Roc19] Joseph Rocca. Understanding variational autoencoders (VAEs) – Building, step by step, the reasoning that leads to VAEs, September 2019. https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73.

[RTC19] Thomas Robert, Nicolas Thome, and Matthieu Cord. DualDis: Dual-branch disentangling with adversarial learning, June 2019. arXiv:1906.00804v1.

[Sun17] Felix Sun. DeepHear – Composing and harmonizing music with neural networks, Accessed on 21/12/2017. https://fephsun.github.io/2015/09/01/neural-music.html.

[Tod89] Peter M. Todd. A connectionist approach to algorithmic composition. *Computer Music Journal (CMJ)*, 13(4):27–43, 1989.

[vdMH08] Laurens van der Maaten and Geoffrey H. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research (JMLR)*, 9:2579–2605, 2008.

[YWW+19] Ruihan Yang, Dingsu Wang, Ziyu Wang, Tianyao Chen, Junyan Jiang, and Gus Xia. Deep music analogy via latent representation disentanglement, October 2019. arXiv:1906.03626v4.