Latent sequencing for dynamic musical patterns

Axel Chemla–Romeu-Santos[1,4], Constance Douwes[1,2,3], Théophile Dupre[1,2,3], Hadrien Marquez[1,2,3], Robin Malzac[1,2,3], Yann Teytaut[1,2,3]

[1] Institut de Recherche et Coordination Acoustique Musique (IRCAM)
UMPC - CNRS UMR 9912 - 1, Place Igor Stravinsky, F-75004 Paris
[2] Télécom ParisTech - 46, Rue Barrault, F-75013 Paris
[3] Sorbonne Université - 4, Place Jussieu, F-75005 Paris
[4] Università Degli Studie di Milano, Laboratorio d'Informatica Musicale (LIM)

January 20, 2019

Abstract

Generative systems are a burgeoning class of machine-learning models aimed at generating new data thanks to observation and analysis of already existing instances. They perform what could be referred to as synthesis by learning. The training of such systems is based on two simultaneous optimization tasks. On the one hand, data are represented into a latent space in a relevant, compressed manner. On the other hand, any sample from the latent space must be converted back into data with similar properties as the original ones while allowing generalization. In this work, we want to apply a specific class of generative systems, that is the variational models based on variational auto-encoders, to percussive sounds in both symbolic and audio worlds. In the symbolic approach, a "rhythm" space filled with drum patterns is extracted out of the percussive songs. In the audio approach, one-shot recordings associated with classical drums are learnt by the variational model. All in all, by connecting both of them and making them interact, our goal is to create a completely controllable instrument.

1 Introduction

One of the main goal of research in computer music is to design state-of-the-art tools allowing to manipulate new approaches and a whole world of possibilities for creating music. Out of all the techniques aimed at generating music that have been developed throughout the 20th century [1], the fluidity and variety of control along with the freedom of use tended to become key issues. Today, generative models are flourishing and very promising candidates for this task as they perform what could be called audio synthesis by learning while keeping in mind the idea of user experience. Many frameworks are based on generative models. However, we will only consider a specific framework relying on variational auto-encoders [2].

Our proposal here is to develop a twofold procedure thanks to this tool. First, this framework will be used in the symbolic world in order to create a rhythmic pattern generator, one that will allows us to generate stochastic patterns from the direct interaction with an intermediate "rhythm" space. Then, this framework will be used in the audio world, in order to create a triggered sounds generator, one that will be compatible with the rhythm decomposition and generation of our variational pattern generator.

This document is organized in several sections. In section 2, general knowledge on variational auto-encoders is presented as it constitutes the starting point of our further investigations. In section 3, our pattern generator is introduced. By using variational auto-encoders and a symbolic representation of drums as activation matrices, one can retrieve the rhythm and thus generate the associated pattern. In section 4, we detail our sound generator which consists in learning several one-shot percussive drum recordings in order to eventually extrapolate audio data out of them.

$$\mathbf{x} \rightarrow \boxed{\text{Encoder}} \rightarrow \mathbf{z} \rightarrow \boxed{\text{Decoder}} \rightarrow \hat{\mathbf{x}}$$

Inputs *Latent representation* *Outputs*

Figure 1: Architecture of a "classic" auto-encoder. Data $\mathbf{x}$ are encoded into a latent code $\mathbf{z}$, which can be decoded to reconstruct an estimation of data $\hat{\mathbf{x}}$.

2 Variational auto-encoders

Variational auto-encoders, that will be referred to as VAEs in the following of this document, are one of the most famous generative systems that can be found in the literature [2, 3] and use the paradigm of variational inference [4]. Both our pattern and audio generators will rely on such an architecture hence introducing here its concepts seems essential.

2.1 General presentation

The variational auto-encoder model inherits from the same architecture than the "classic" auto-encoder but formulates strong statistical hypothesis inherited from variational inference principles.

2.1.1 Auto-encoders

Auto-encoders result in the association of two neural networks, namely the *encoder* and the *decoder*. The training of the encoder is aimed at representing the input data $\mathbf{x}$ in an abstract, dimensionality reduced code $\mathbf{z}$. All of these compressed representations $\mathbf{z}$ are contained within the *latent space*. Then the decoder is used to produce an output $\hat{\mathbf{x}}$ seen as an estimation of the original data $\mathbf{x}$ (see Figure 1). The objective is to have $\hat{\mathbf{x}}$ close to $\mathbf{x}$.

2.1.2 Variational inference

A traditionnal auto-encoder is however inadequate for the purpose we pursue, that is a generative goal, as it lacks the essential property of enabling generalization.

Indeed, a latent code $\mathbf{z}$ can somehow be seen as a set of parameters used to synthesize an output with similar characteristics than the inputs $\mathbf{x}$. But since auto-encoders are deterministic, a given input $\mathbf{x}$ will be associated with a very specific $\mathbf{z}$ in the latent space, thus there is no assurance that any random $\mathbf{z}$ would be decoded in a significant, interesting output.

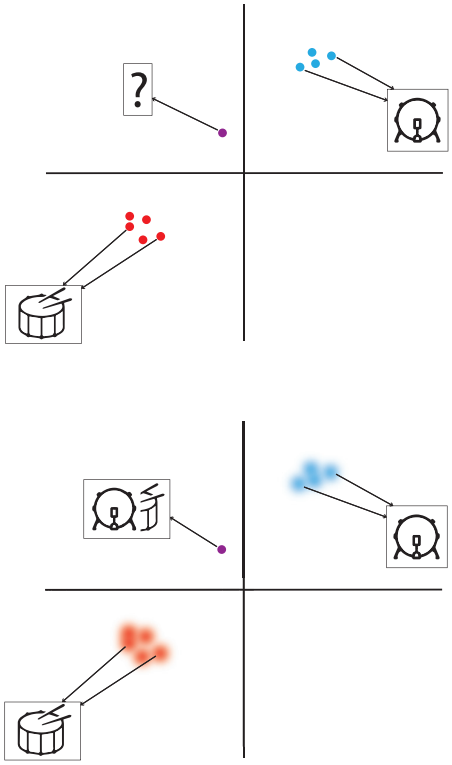

Figure 2: Latent spaces of an auto-encoder (above) and a variational auto-encoder (below). As the classic auto-encoder is deterministic, it has no generalization properties : not all random latent positions will lead to a relevant reconstruction. Thanks to the theory of variational inference, the VAE encodes data no more in a deterministic way but according to statistical distributions, allowing generalization properties.

In order to tackle this issue down, we must rely on variational inference principles : a given input $\mathbf{x}$ will be associated, in the latent space, with a statistical distribution of latent variables $\mathbf{z}$, not only a specific $\mathbf{z}$. This solves the problem and allows us to generate new data by analyzing and learning existing ones. This is summarized on Figure 2.

For now, we do not explain how to formalize the introduction of variational inference principles since it will be done in the following of this section.

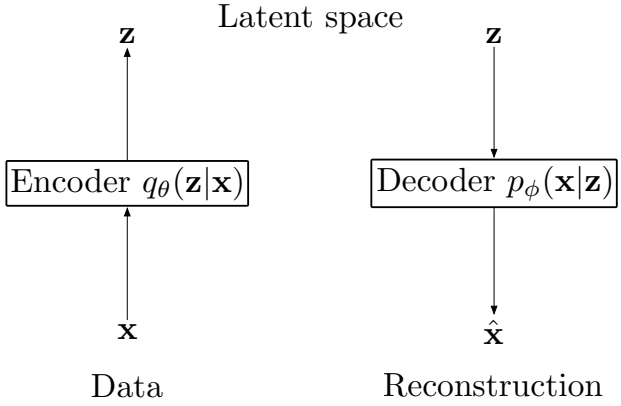

Figure 3: A variational auto-encoder (VAE) is a two-step procedure : data $\mathbf{x}$ are converted into latent variables $\mathbf{z}$, that parameter to $q_\theta(\mathbf{z}|\mathbf{x})$, and then converted back into an estimation $\hat{\mathbf{x}}$, that parameter to $p_\phi(\mathbf{x}|\mathbf{z})$.

2.1.3 Variational auto-encoders

In short, variational auto-encoders (VAEs) can be seen as a statistical extension to traditionnal auto-encoders. They are based on a two-step architecture and their main purpose is the creation of a *latent space* in which the original data are subjected to a dimensionality reduction by being represented through abstract latent variables.

The first procedure consists in *encoding* the data in the latent space, in which one is free to navigate, in a over-compressed, high-level representation. Each entry $\mathbf{x}$ will correspond to latent codes $\mathbf{z}$ that parameter to a statistical distribution. Then, the second procedure consists in *decoding* estimated data $\hat{\mathbf{x}}$, that parameter to another statistical distribution, from any latent position $\mathbf{z}$ (see Figure 3).

2.2 Encoder and decoder

The *encoder* is a neural network whose inputs are our data $\mathbf{x}$ and its outputs are a hidden representation $\mathbf{z}$ of the former. If we denote its parameters, that is weights and biases, θ then our encoder can be referred to as $q_\theta(\mathbf{z}|\mathbf{x})$, meaning that our latent parameters will fit the stochastic law $q_\theta(\mathbf{z}|\mathbf{x})$ which is usually a Gaussian probability density of mean μ and variance σ^2 : $\mathcal{N}(\mu, \sigma)$. Now, this Gaussian distribution can be sampled to get random values of latent representations $\mathbf{z}$.

The *decoder* is also a neural network whose inputs are the representations $\mathbf{z}$ and its outputs are an "estimation" of the data $\hat{\mathbf{x}}$ given in the form of parameters of the probability distribution of the data. If we denote its parameters, that is weights and biases, ϕ then our decoder can be referred to as $p_\phi(\mathbf{x}|\mathbf{z})$. During this reconstruction, we go from a smaller to a larger dimension, hence we naturally have loss of information and it is necessary to quantify it.

2.3 Loss function

In order to characterize completely a VAE, we need to define a *loss function*, one that will be minimized throughout the training. The usual loss function used with variation auto-encoders is the negative log-likelihood combined with a regularizer. However, as it is explicitly demonstrated in [2], minimizing this loss function is strictly equivalent to maximizing the ELBO function, defined as follows:

$$\begin{aligned} \text{ELBO}(\theta, \phi, \mathbf{x}, \mathbf{z}) &= \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[\log p_\phi(\mathbf{x}|\mathbf{z})] \\ &- \mathcal{D}_{\text{KL}}[q_\theta(\mathbf{z}|\mathbf{x})||p(\mathbf{z})]. \end{aligned} \tag{1}$$

The acronym ELBO stands for *Evidence Lower BOund*, meaning that this function acts as a lower bound for our evidence, i.e. the probability $p(\mathbf{x})$, since one can show that $p(\mathbf{x}) \geq \text{ELBO}(\theta, \phi, \mathbf{x}, \mathbf{z})$ [2].

The first term in (1) equates to the opposite of the reconstruction loss measured as a log-likelihood, and encourages the decoder to reconstruct data as similar as possible to the original ones. The second term in (1) is a regularizer calculated as the Kullback-Leibler divergence between $q_\phi(\mathbf{x}|\mathbf{z})$ and $p(\mathbf{z})$, and is used for measuring the "proximity" between q and p. The standard normal distribution is usually chosen for the prior, that is $p(\mathbf{z}) = \mathcal{N}(0, 1)$, as this the most general hypothesis one can make. All in all, this term helps during the learning to keep the representations $\mathbf{z}$ of similar data close in the latent space (see Figure 4).

According to [5], we will slightly modify the ELBO function by adding a hyperparameter β before the KL-divergence[1] :

$$\begin{aligned} \text{ELBO}(\theta, \phi, \mathbf{x}, \mathbf{z}) &= \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[\log p_\phi(\mathbf{x}|\mathbf{z})] \\ &- \beta \times \mathcal{D}_{\text{KL}}[q_\theta(\mathbf{z}|\mathbf{x})||p(\mathbf{z})]. \end{aligned} \tag{2}$$

[1] Note that $\beta = 1$ corresponds to the general VAE enhanced in [4].

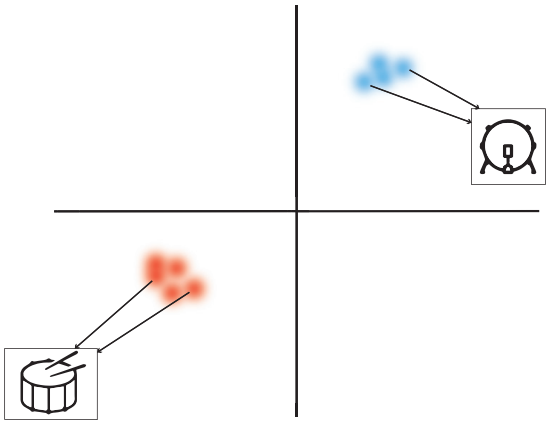

Figure 4: Latent codes $\mathbf{z}$ proximity in the latent space. Suppose the data $\mathbf{x}$ are percussive sounds recordings, represented as distributions $\mathbf{z}$ in the latent space. Latent codes $\mathbf{z}$ associated with the same instrument should be close to one another : red distributions are all linked to snares recordings while blue distributions are all linked to kicks recordings.

β acts like a regularization constraining the capacity of the latent variables $\mathbf{z}$ and will encourage, during the training, to learn the most relevant representation of the data. In the following, we will use $\beta = 4$.

2.4 Learning with VAEs

The learning step relies on gradient descent in order to optimize the loss function, or maximize ELBO function since it is equivalent, with respect to parameters of both encoder and decoder, that is θ and ϕ. These parameters are updated at each epoch with a given learning rate [6].

Another important remark is that any used dataset should be divided in a train part (approximately 80%), that will be used for training the VAE, and a test part (approximately 20%). At the end of every epoch, the loss function should be calculated for these both subsets to supervise the training. Indeed, the loss must keep decreasing for the *train* part but must not overfit these data, which happens when the loss function for the *test* part stops decreasing with the number of epochs.

3 Pattern generation

The goal of this section is to present how to create a latent space associated with several categories of drum patterns. By using VAE architecture, we will perform a training aimed at extracting a rhythm space in which each pattern will be located at a specific position. Then, by navigating freely into this space, it will be possible to interpolate between several patterns.

3.1 Symbolic drums dataset

First and foremost, we need to create our own dataset for pattern generation. In order to do so, we chose to gather some MIDI files and convert them into activation matrices.

3.1.1 Activation matrices

An activation matrix M is binary, its values are either 0 or 1, and is of size $N_i \times N_s$ where N_i is the number of instruments and N_s is the number of steps used for dividing the rhythm. They allow us to know, at a given step, if an instrument is played or not.

Indeed, let $i \in \{1 \dots N_i\}$ and $j \in \{1 \dots N_s\}$ be two integers, then $M_{ij} = 1$ would mean that the i^{th} drum is played at the j^{th} step. On the contrary, $M_{ij} = 0$ would mean that it is not played.

It is necessary to classify all types of drums into a limited number of categories. It has been chosen to arrange them using the classes represented on Figure 5, which are *snares*, *kicks*, *hi-hats opened*, *hi-hats closed*, *cymbals*, *claps*, *toms* and *others* drums. Thus, we can fix the parameter $N_i = 8$.

Figure 5: Classification of drums in $N_i = 8$ labels.

As for the parameter N_s, it has been decided to divide the bar into 64 steps for all MIDI files, which leads to $N_s = 64$.

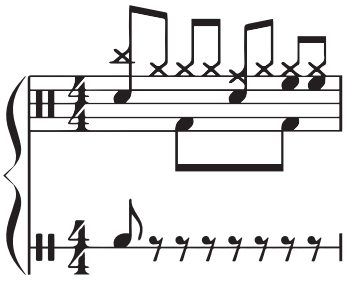

Figure 6: Single-bar sheet of a drumset (above) playing with a timbale (below).

Steps	1	2	3	4	5	6	7	8
Snare	1	0	0	0	1	0	0	0
Kick	0	0	1	0	0	0	1	0
Hi-Hat Closed	0	1	1	1	0	1	1	1
Hi-Hat Opened	0	0	0	0	1	0	0	0
Cymbals	1	0	0	0	0	0	0	0
Toms	0	0	0	0	0	0	1	1
Other	1	0	0	0	0	0	0	0
Claps	0	0	0	0	0	0	0	0

Table 1: Activation matrix for pattern decomposition of the musical sheet given Figure 6.

3.1.2 Example on a single-bar sheet

Let us take a simple example explaining what we want to achieve. We consider a musical sheet composed of a single bar in which a drumset and a timbale play together. This sheet is represented on Figure 6.

Now, let us imagine that we do have the MIDI file associated with this sheet and that $N_s = 8$ (hence we consider the quaver as the minimal sub-division of the rhythm). Then, the corresponding activation matrix that we want to obtain is represented on Table 1 where the *cymbal* and the *other* on the first step correspond respectively to the crash and the timbale. Plus, as there is no *clap*, the associated line is filled with 0.

3.1.3 Symbolic dataset

First, we uploaded a midi dataset composed of 9000 drum MIDI files from the internet[2]. However, some of them were drum fills (to be eliminated because they were not drum grooves), repeated or instrumentless informed.

[2]One can find the original midi dataset on this website : https://www.fruityclub.net/ressources-presets/9000-drums-midi-avec-differents-styles/

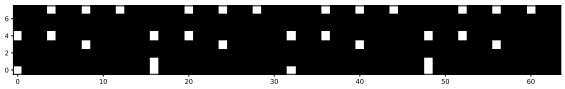

Figure 7: Activation matrix 8×64 manipulated in Python and used during the training. Black points mean that the instrument (associated line) is not played at the given step (associated row). White points, on the contrary, correspond to activations.

After putting these files into order, we finally arrived to 1095 MIDI files corresponding to drum grooves from many genres such as rock, salsa, dance, jazz, etc. Then, we transformed them into activation matrices thank to the use of `Music21` library. We took the first bar of each midi file and converted it into an *array* format (namely `npy`), as it is represented on Figure 7.

3.2 Pattern generation training

It is now possible to pursue the training of our VAE on the symbolic dataset we made. Here are presented the several parameters needed for this task.

First and foremost, we must note that the outputs have to parameter to a Bernoulli distribution $\mathcal{B}(p)$ with $p \in [0, 1]$ as our inputs are exclusively binary. In order to work with 1D inputs, the activation matrices must be flattened before entering the VAE and will be deflattened at the output of the VAE.

Plus, the parameters of the several VAE modules were chosen as follows

$$\begin{aligned} \text{Batch size} &= 8 \\ \text{Input dimension} &= 512 \\ \text{Encoder hidden layer dimension} &= 800 \\ \text{Latent space dimension} &= 2 \\ \text{Decoder hidden layer dimension} &= 800 \\ \text{Ouput dimension} &= 512 \end{aligned}$$

Finally, the training loop is run on 200 epochs and the warm-up coefficient β from (2) is taken equal to 4. It is reassuring to observe that the loss function (see Figure 8 keep decreasing for both test and train sets over the number of epoch.

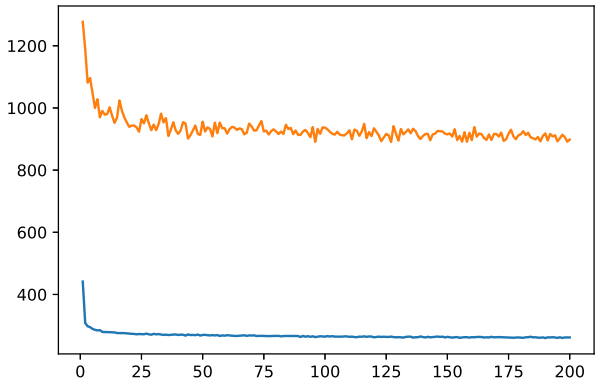

Figure 8: Plot of the loss as a function of the number of epoch for the train part (orange tracing) and the test part (blue tracing).

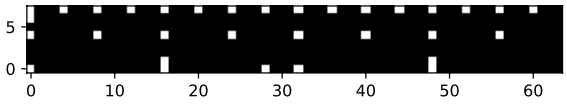

Figure 9: Generated activation matrix 8×64 created by the decoder from a sample in the latent space.

3.3 Results

Once our encoder and decoder have been trained and tested simultaneously, we will be able to generate new outputs.

A way to do this is to take a sample from our latent space that is part of the $\mathcal{N}(0, 1)$ distribution, so that the decoder can output the corresponding activation matrix as in Figure 9, which can be compared to the one highlighted in Figure 7.

In order to hear what this activation matrix corresponds to, a conversion similar as the one used in 3.1.3 is necessary and will allow us to obtain a waveform (WAV files) as in Figure 10. At this point of the project, we haven't generated our proper sounds yet, therefore eight samples from our audio dataset (see section 4.1) have been picked up.

The results are very convincing since we can observe that the latent space is well organized: samples that are close in the latent space creates rhythm pattern that also sound close. We finally denote that the latent space is settled by musical genre. A representation of the latent space is available on Figure 11.

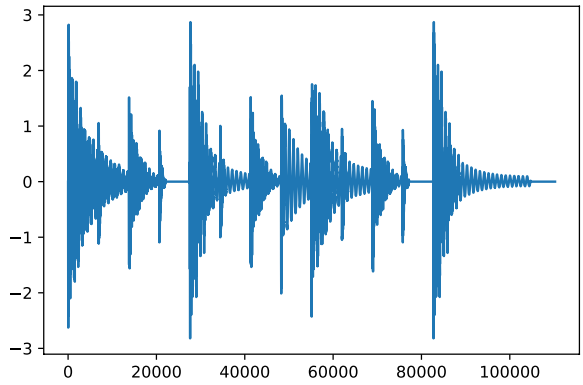

Figure 10: Waveform of the generated pattern from the activation matrix.

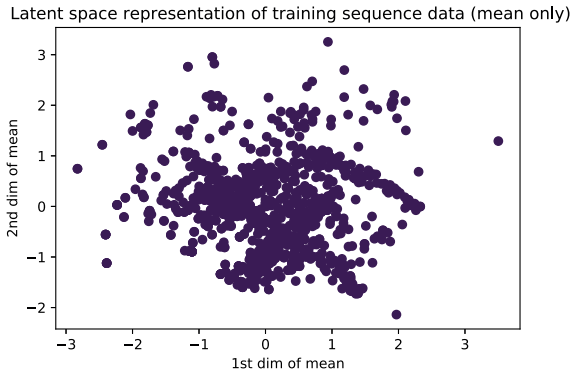

Figure 11: Representation of the latent space. Here are plotted the means of its both dimensions. Data are well represented because they are centered and burst in the range $[-4; 4]$.

As it is desired to achieve a controllable instrument, a GUI window associated with this latent space has been created thanks to the `Tkinter` library. Once the user produces a left mouse click on the window, a pattern is triggered, one that is generated with respect to the position of the click in the latent space.

4 Sound generation

This section focuses on the latent rhythm generator that is dedicated to sound generation. First, we will need to construct a specific audio dataset, which will be labelled by types in accordance with the categories of drums isolated for our sequencer.

4.1 Audio drums dataset

4.1.1 Gathering audio data

The audio dataset contains exclusively drums created by drum machines. It is composed of approximately 200 drum machines and, for each of them, contains one-shot recordings that have been classified according to the classes of our pattern generator: claps, cymbals, hi-hat closed, hi-hat opened, kicks, snares, toms and other drums. All sounds are mono WAV audio files sampled at 44 100 Hz. All in all, 6300 one-shot audio samples are to be learnt with this dataset.

The original version of this dataset has been found online[3], on a website listing many audio datasets that are relevant to music information retrieval. Then, the exploration and labellisation steps were ours.

4.1.2 Data pre-processing

During the training, only a representation of each audio file will be learnt, like its spectrogram one could obtain with the *Short-Time Fourier Transform* (STFT).

However, we will use here the *Non-Stationary Gabor Transform* (NSGT)[4] instead of the STFT as it presents several advantages with comparison to it. First, the NSGT allows to custom the scale. Plus, its dimensions are smaller than STFT, leading to a faster training. At last, NSGT is invertible, which is an essantial property since the VAE will learn how to encode and decode Gabor transforms but our desire is to decode audio data, hence the VAE output must be subjected to an inverse NGST.

[3]One can find the original audio dataset on this website: https://www.audiocontentanalysis.org/data-sets/

[4]From the library of sir Grill available here : https://github.com/grrr/nsgt

4.2 Sound generation training

In order to compute the sound generation training, we can no longer rely on VAEs as simple as for the pattern generator. Indeed, due to both dimension and nature of the learnt data (2D spectrograms of audio files), the networks to be used have to be much bigger.

Among the numerous categories of existing neural networks, the convolutional neural networks (CNNs) have been proven worthy candidates for task such as image recognition. Since our goal here is to learn from spectrograms, which can be seen as images, we will naturally rely on convolutional encoders and decoders to perform the learning.

First and foremost, the inputs are not binary this time hence the outputs should parameter to a Gaussian distribution $\mathcal{N}(\mu, \sigma)$ of mean μ and variance σ^2.

The parameters of the several convolutional VAE modules, including three convolutional and deconvolutional layers, were mostly inspired from the work of [7] and are summarized in the following.

General parameters

Batch size	=	200
Input dimension	=	410×157
Latent space dimension	=	128

Convolutional layers

		Layer 1	Layer 2	Layer 3
Input channels	=	1	8	16
Output channels	=	8	16	32
Kernel size	=	11×5	11×5	11×5
Stride	=	3×2	3×2	3×2
Padding	=	2×2	2×2	2×2

Deconvolutional layers

		Layer 1	Layer 2	Layer 3
Input channels	=	32	16	8
Output channels	=	16	8	1
Kernel size	=	11×5	11×5	11×5
Stride	=	3×2	3×2	3×2
Padding	=	2×2	2×2	2×2

Finally, the training loop is run on 500 epochs and the warm-up coefficient β from (2) is taken equal to 4.

4.3 Results

Unfortunately, the training of sound generation didn't go well. The decoder currently outputs `NaN` that are not expected because we should obtain something that corresponds to spectrograms.

However, even if the training would have gone well, the spectrograms would not have been auto-sufficient to retrieve the sound as they contain no information regarding the phase. That is why we made up a functional Griffin Lim algorithm in order to reconstruct the phases from the spectrograms [8]. This algorithm is iterative and finds the signal having the magnitude part of its Gabor transform as close as possible to the modified spectrogram.

5 Conclusion

Achieved work

Throughout this document, we try to implement step by step two generators linked to percussive sounds thanks to variational auto-encoders. When it comes to manipulate percussive sounds, different approaches have to be enhanced.

- On the one hand, our investigations focused on the symbolic aspect related to the rhythm. After "translating" relevant MIDI files into activation matrices, we were able to train a variational auto-encoder. The latent space of the latter allowed us to create a full pattern generator organized in music genres, and which is controllable thanks to a decidated GUI.

- On the other hand, the audio aspect needed was the second fundamental part of our study. After gathering one-shot percussive recordings produced by drum machines and process them thanks to the Gabor transform, we started to set their training up by settling the "convolutional" variational auto-encoder needed for the learning, although the results were not pertinent yet and the source of the problem have to be examined.

Perspectives

This work can be pursued and improved in several axes.

- Fix the issue and throw again the training for the audio dataset.
- When this is done, a GUI decidated to audio should be implemented and shoud allow to choose how each instrument "sound".
- Connect the symbolic and the audio worlds to complete the instrument.
- Use an (semi-)annotated dataset from sequences to condition the network and be able to generate patterns associated with a given style.

References

[1] *Viewpoints on the history of digital synthesis*, J.O. Smith, pp. 1-1, International Computer Music Association, 1991

[2] *Variational inference: A review for statisticians*, Blei and al., Journal of the American Statistical Association, volume 112 issue 518, 2017

[3] *Stochastic backpropagation and approximate inference in deep generative models*, Rezende and al., Daan,arXiv:1401.4082,2014

[4] *Auto-encoding variational bayes*, Kingma and al., arXiv:1312.6114, 2013

[5] *beta-vae: Learning basic visual concepts with a constrained variational framework*, Higgins and al., Alexander, 2016

[6] *How to train deep variational autoencoders and probabilistic ladder networks*, Sønderby, Casper Kaae and Raiko, Tapani and Maaløe, Lars and Sønderby, Søren Kaae and Winther, Ole, arXiv:1602.02282, 2016.

[7] *Learning latent spaces for real-time synthesis of audio waveforms*, C. Aouameur, ATIAM Master's internship report, 2018.

[8] Griffin D. and Lim J. (1984). *Signal Estimation from Modified Short-Time Fourier Transform.* IEEE Transactions on Acoustics, Speech and Signal Processing. 32 (2): 236–243.