Internship Report

Improving the latent harmonic space of SOMax with Variational Autoencoders

Submitted by
Benjamin Feldman
Centralesupélec

Under the guidance of
Gérard Assayag
Head of the Représentations musicales (RepMus) team

RepMus team at STMS lab
Institut de recherche et coordination acoustique/musique
1 place Igor Stravinsky, 75004 Paris

Summer 2021

Abstract

In this report, we explore how the RepMus team at IRCAM tackled the problem of automatic musical improvisation in the last few decades, with a particular focus on SOMax. Then, by training variational autoencoders on a dataset of chromas, we'll see a potential improvement to SOMax clustering model.

Remerciements

J'adresse tout d'abord mes remerciements à Gérard Assayag, qui m'a volontiers accueilli dans son équipe pour ce stage, et qui a su m'accorder une forte indépendance qui m'a permis d'aborder sereinement mes recherches. Je tiens également à remercier Tristan Carsault sans qui certains points cruciaux de ce rapport seraient restés incompris... enfin, je remercie tous ceux avec qui j'ai partagé mon bureau à l'IRCAM.

Contents

Introduction

1 Goal of the internship

The last decade has seen the rise of artificial intelligence, thanks to the increasing computing power and to progress made in the machine learning associated fields. AI techniques have been irrigating the sphere of *artificial creativity*, with applications to visual art, poetry or music. In music, a particular domain called *automatic improvisation* consists in creating musical interactions between humans and computers. Such devices have been developed at IRCAM (Institut de recherche et coordination acoustique/musique), a French scientific and artistic institution founded in 1970 by the composer and director Pierre Boulez. In this report, we will be focusing on SOMax, which is the third iteration of a long-going project (DYCI2) in the RepMus team at IRCAM, directed by Gérard Assayag. The main goal of SOMax is to create virtual agents that interact with other agents (i.e. real musicians or computers) in an improvisation context. Such an agent therefore needs some sort of latent musical knowledge, as well as being able to listen and react in real time to its context. The purpose of such systems is to create new musical experiences, by dialoguing with the musician, similarly to human-human improvisation.

The purpose of this internship is to improve SOMax's core model, which is, at the time, a little bit blurry. In the first part, we'll explore the state-of-the-art concerning automatic improvisation ; then in part 2, we'll give a detailed explanation of SOMax and of the methods used during the internship, and then we will present the works and results in part 3.

2 On improvisation

There is not a single definition for musical improvisation : it depends on the genre, the period... for instance, there are musical genres that imply a guideline or a basis material for improvisation, such as polyphonic improvisation over a *cantus firmus* in the Renaissance, a figured bass in the baroque era or a chord grids in jazz ; on the contrary, in free jazz or krautrock for instance, musicians can improvise without predefined constraints. The idea for an automatic improvisation software would be to combine those two types of improvisation : the machine should be able to work under contraints (e.g. follow a jazz grid, or play in the style of a composer) as well as having a free component in order to make the generation innovative and create happy accidents. This mode of functioning is certainly close to how most improvisers play.

While giving constraints to a computer is "easy", letting the computer wander freely in a musical world while paying attention to the constraints is much more difficult to tackle. This is the goal of SOMax : the computer generates music on its own (with self-consistency) but it is also paying attention to the player in real-time, while also having a corpus of reference of which it cannot escape.

State of the art

In this part we'll present what has been achieved at IRCAM concerning automatic improvisation, and then we'll see how Variational Autoencoders work.

1 OMax

OMax is a software developed at IRCAM by G. Assayag, M. Chemillier and G. Bloch aiming to generate improvisation based on a given material. The main idea is to go navigate a musical corpus in a non linear way, jumping from one chord/note to another that can be much further. One important feature of OMax is that the material is learned in real-time from the musician playing, thus creating a strong interaction between the machine and the human. This process is called "stylistic reinjection". The way OMax goes through the corpus is based on the Markovian properties of the corpus, and therefore recombines the corpus in a harmonically and melodically coherent way.

Figure 2.1: Source : [1]

2 SOMax 2

SOMax works by navigating in a non-linear way in a musical corpus, using a guidance provided by the playing of the human improviser. It is therefore a genuine human-machine interaction, where one reacts to the actions of the other in real time. SOMax works as following :

1. Building and modelling the corpus (offline)
2. Listening to the musician in real time (online)
3. Generation

2.1 Building and modelling the corpus (offline)

The user provides a midi file containing the corpus, which will serve as a training material.

Slicing This file is divided in slices, each slice corresponding to a midi event (see figure 2.2). For each slice, we compute "traits" (features). Those traits are related to

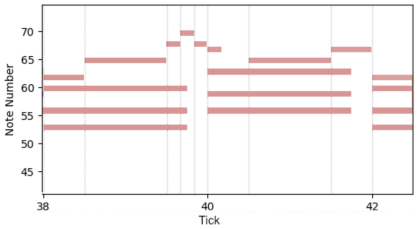


Figure 2.2: Slicing (source : [2])

the melodic/harmonic content (chromas, which notes are inside the slice... see [3]).

The slices are notated $S_u^{(\mathcal{C})}$ where u is the index and the exponent means that the slice belongs to the corpus $\mathcal{C}$. Formally, a slice is a set containing temporal, melodic and harmonic information :

$$S_u^{(\mathcal{C})} = \{t_u^{(\mathcal{C})}, d_u, \zeta_u, \tau_u, \delta_u, \theta_u^{(1)}, ..., \theta_u^{(Q)}\}$$

where :

- $t_u^{(\mathcal{C})}$ is the instant of apparition of the slice (in discretized time, where a unit corresponds to a beat)
- d_u is its length (idem)
- ζ_u is the tempo
- τ_u is the instant of apparition of the slice (in absolute time, i.e. in ms)
- δ_u is its length (idem)
- les $\theta_u^{(q)}$ are the different traits

Thus, we built a corpus $\mathcal{C}$ as a set of slices :

$$\mathcal{C} = \{S_1^{(\mathcal{C})}, ..., S_U^{(\mathcal{C})}\}$$

Clustering and classification We set Q as the number of traits. Once the corpus is built comes its modelling. We consider R "layers" (with $R < Q$), each containing a clustering with respect to a unique trait as well as a n-gram model.

Let's consider the layer $r \in [1, R]$. It is built with respect to the trait $\theta^{(q)}$ (we should write the layer r_q...). The clustering is written

$$\Theta^{(r)}(\theta^{(q)}|\mathcal{C})$$

which means that on this layer r, each slice will be clustered with respect to its q-th trait. Once this clustering is done (using a pretrained clustering algorithm : SOM, or VAE+kNN), we can classify each of those slices. We note

$$l_u^{(r)} = \Theta^{(r)}(\theta_u^{(q)}|\mathcal{C}) \in \mathbb{Z}$$

the label of slice u for trait q (r and q are closely related).

Modelling Finally, we build a model $\mathcal{M}^{(r)}$ as an application mapping a vector of labels $\boldsymbol{l}$ to a set containing all slices corresponding to $\boldsymbol{l}$.

2.2 Listening to the musician in real time (online)

Once this corpus is built, we aim to navigate through it in a non-linear way : the incoming audio flow has to be modelled in order to make it interact with the model of the corpus. This step is called the influence process.

The fundamental characteristic of what follows is the short-term memory of our model. Indeed, at instant t, the influences generated until passed time (up to $t - \delta t$) will have an impact on the output.

Slicing We begin (again) by slicing the input, almost identically as for the corpus. We create a flow of slices $\mathcal{K}$:

$$\mathcal{K} = \{S_1^{(\mathcal{K})}, ..., S_V^{(\mathcal{K})}\}$$

where

$$S_v^{(\mathcal{K})} = \{t_v^{(\mathcal{K})}, d_v, \zeta_v, \tau_v, \delta_v, \theta_v^{(1)}, ..., \theta_v^{(Q)}\}$$

Warning : where $t_u^{(\mathcal{C})}$ corresponds to an instant in the corpus, $t_v^{(\mathcal{K})}$ corresponds on the contrary to an instant in the current generation.

Clustering and classification As for the, R layers are created, each of them having a clustering according to a certain trait. The slices of $\mathcal{K}$ are therefore classified in the same way as those of $\mathcal{C}$. Each slice of influence $S_v^{(\mathcal{K})}$ thus has, for each layer r, a label $l_v^{(r)} = \Theta^{(r)}(\theta_v^{(q)}|\mathcal{K})$.

We now want to match those slices of nfluences to slices of the corpus.

Matching the corpus, "peaks" Let's consider a slice of influence $S_v^{(\mathcal{K})}$ at time $t_v^{(\mathcal{K})}$, on a layer r. Let $k \in \mathbb{Z}^*$ We have

$$\boldsymbol{l}_v^{(r)} := \begin{bmatrix} l_{v-k}^{(r)} & ... & l_v^{(r)} \end{bmatrix}$$

We pass this vector into the model $\mathcal{M}^{(r)}$ of the corpus, to obtain a set of slices of the corpus, noted $\Sigma_v^{(r)}$:

$$\Sigma_v^{(r)} = \mathcal{M}^{(r)}(\boldsymbol{l}_v^{(r)}) = \{S_{u_1}^{(\mathcal{C})}, ..., S_{u_j}^{(\mathcal{C})}\}$$

From $\Sigma_v^{(r)}$, we build a "peaks" matrix :

$$\boldsymbol{P}_v^{(r)} = \begin{bmatrix} t_{u_1}^{(\mathcal{C})} & ... & t_{u_j}^{(\mathcal{C})} \\ y_{u_1} & ... & y_{u_j} \end{bmatrix}$$

where y is a score given to the peak. We also write $\boldsymbol{p}_{u_m} = \begin{bmatrix} t_{u_m} \\ y_{u_m} \end{bmatrix}$

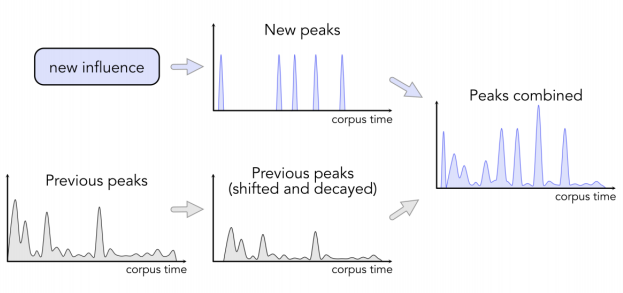


Figure 2.3: Short-term memory (source : [2])

Then comes the crucial step : the creation of the short-term memory. We add the peak $\boldsymbol{P}_{v-1}^{(r)}$ to the peak $\boldsymbol{P}_{v}^{(r)}$, while lowering their weight. More precisely, we offset the time of appearance of the peaks in the past, and exponentially decrease their score (see [2] and Figure 3.5).

2.3 Generation

A trigger signal asks the model to generate. For each trigger, the system fetches the peaks of each layer, which see their score multipilicated by a used-defined factor $\alpha^{(r)}$.

The final output is selected in the corpus as being the closest (time-wise) slice of the peak having the highest score (in the final peaks matrix).

3 Variational Auto-Encoders (VAEs)

An autoencoder is a machine learning model consisting in an encoder e which compresses the data x in a lower-dimensional space, and a decoder d that "reconstructs" the data such that $\hat{x} = d(e(x)) \approx x$.

A more sophisticated approach is to combine AEs with Variational Bayesian methods, thus resulting in VAEs[4]. In this framework, we assume that the set of data $X = \{x_i\}_{i=1}^{N}$ is generated using a random *latent* variable z:

1. z_i (latent data) is generated by a prior $p_{\theta*}(z)$
2. x_i is generated by a posterior $p_{\theta*}(x|z)$

We assume that the parametrized probability density functions are differentiable. Therefore we can retrieve the distribution of the data $p_{\theta*}(x)$ by approximating the

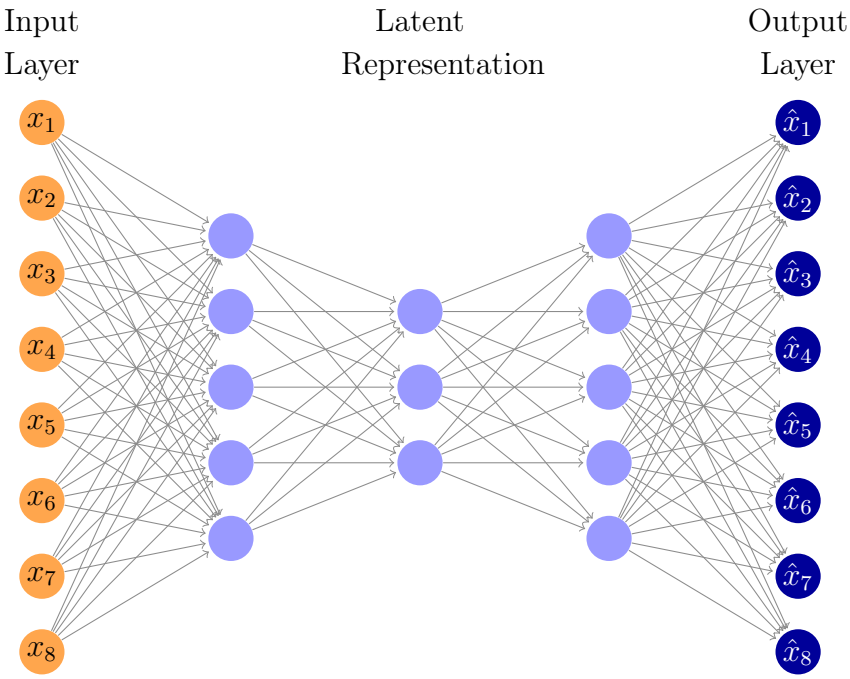


Figure 2.4: Architecture of an autoencoder (schematic from the *Tikz standalone* github)

following (intractable) integral :

$$p_{\theta*}(x) = \int p_{\theta*}(z)p_{\theta*}(x|z)dz$$

The principle of variational inference is to appriximate $p_{\theta*}(x|z)$ (which is intractable) by a simpler distribution $q_\phi(z|x)$ where ϕ is an auxiliary parameter. The goal is to learn θ and ϕ such that the discrepancy between p and q is minimal. $q_\phi(z|x)$ is the (probabilistic) encoder and $p_\theta(x|z)$ is the (probabilistic)decoder.

We aim to minimize the Kullback-Leibler divergence between the posterior $p_\theta(z|x)$ and its estimation $q_\phi(z|x)$.

$$D_{KL}(q_\phi(z|x)||p_\theta(z|x)) = \mathbb{E}_{q_\phi(z)}[\log q_\phi(z|x) - \log p_\theta(z|x)]$$

Using Bayes's formula :

$$D_{KL}(q_\phi(z|x)||p_\theta(z|x)) = \mathbb{E}_{q_\phi(z)}[\log q_\phi(z|x) - \log p_\theta(x|z) - \log p_\theta(z)) + \log p_\theta(x)]$$

($p_\theta(x)$ is independent from z)
Rearranging both sides gives us :

$$\log p_\theta(x) - D_{KL}(q_\phi(z|x)||p_\theta(z|x)) = \mathbb{E}_{q_\phi(z)}[\log p_\theta(x|z) - D_{KL}(q_\phi(z|x)||p_\theta(z))]$$

Therefore, assuming $p_\theta(x)$ is a constant (with respect to ϕ, minimizing the KL-divergence is equivalent to maximizing $\mathcal{L}(\theta, \phi, x)$ defined by

$$\mathcal{L}(\theta, \phi, x) = \underbrace{\mathbb{E}_{q_\phi(z)}[\log p_\theta(x|z)}_{reconstruction} - \underbrace{D_{KL}(q_\phi(z|x)||p_\theta(z))]}_{regularization}$$

We usually have an hyperparameter called β which alters the Kullback-Leibler part of the loss, in the following way :

$$\mathcal{L}(\theta, \phi, x) = \mathbb{E}_{q_\phi(z)}[\log p_\theta(x|z) - \beta D_{KL}(q_\phi(z|x)||p_\theta(z))]$$

The point of this is to give more important to the reconstruction during the early training epochs, and then progressively increase β to give the KL-loss importance.

In order to optimize $\mathcal{L}$, we have to specify the form of the distribution q. As in most cases, we will define $q_\phi(z|x) = \mathcal{N}(z|\mu(x;\phi), \Sigma(x;\phi))$ where Σ and μ are deterministic functions parametrized by ϕ. We also choose the prior to be $p(z) = \mathcal{N}(0, I)$. The KL-divergence between two Gaussian distributions is known as a closed form :

$$D_{KL}[\mathcal{N}(\mu(x), \sigma(x))||\mathcal{N}(0, I)] = \frac{1}{2}(tr(\Sigma(x)) + (\mu(x))^T(\mu(x)) - k - \log\det\Sigma(x))$$

where k is the dimensionality of the distributions.

In practice, $q_\phi(z|x)$ (encoder) and $p_\theta(x|z)$ (decoder) are implemented using neural networks, and μ and Σ are the outputs of the encoder. The latent vector z is then sampled using $\mathcal{N}(\mu(x), \sigma(x))$.

Our goal is to perform stochastic gradient descent on $\mathcal{L}$. However, this sampling step prevents us from doing backpropagation, hence the introduction of the *reparametrization trick*. It consists in replacing the sampling of $z \sim \mathcal{N}(\mu(x), \sigma(x))$ by the sampling of $\epsilon \sim \mathcal{N}(0, I)$, and then computing $z = \mu(x) + \Sigma^{\frac{1}{2}}(x) \times \epsilon$

Variational autoencoders have proven their strength in data generation as well as in clustering. We will be focusing on clustering in the following sections.

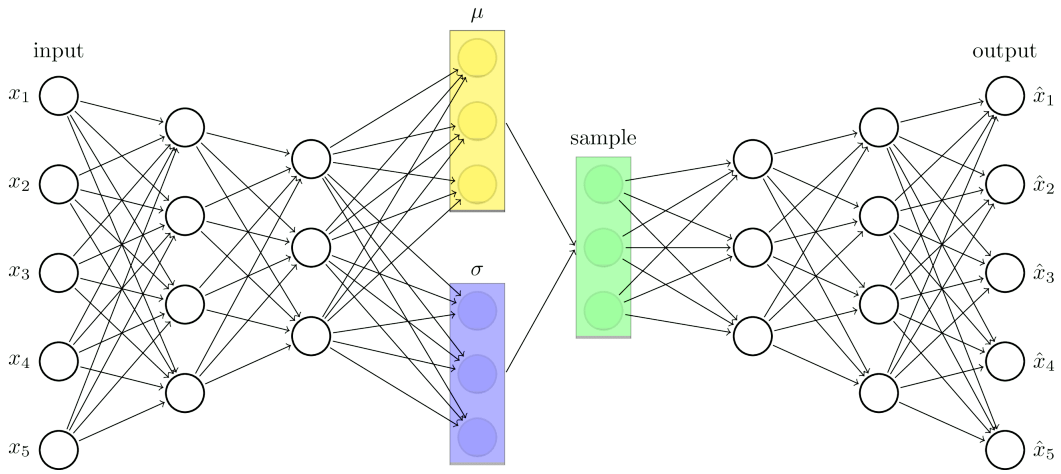


Figure 2.5: Architecture of a VAE

Improving the clustering model of SOMax

Code available on `https://github.com/benjamin-feldman/VAE-omax`.

In this section we will see how SOMax clustering layer can be improved by replacing the use of self-organizing maps (SOM) by a VAE.

1 Data and preprocessing

The data used for the experiments is the corpus of the J.S. Bach chorales, available in Music21.

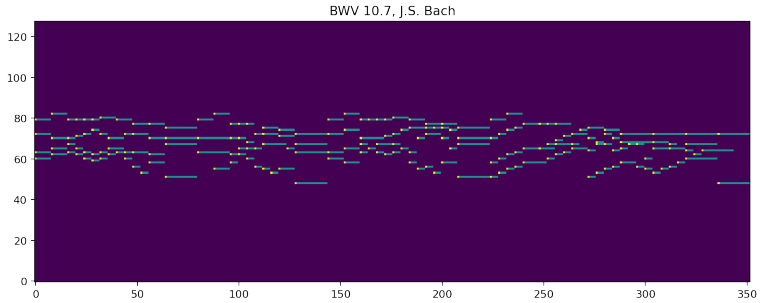


Figure 3.1: An example of a choral *Lob und Preis sei Gott, BWV 10*, in its pianoroll representation

Each piece is processed as follows:

1. Divide the piece at each beat in slices

2. Reduce each slice modulo 12 to have a 12-tone vector

3. Compute harmonics to create a fake chroma vector

The augmented harmonic content is computed by adding harmonics following the natural harmonic series. We end up with data points looking like fig. 3.2.

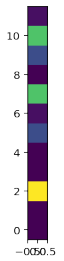


Figure 3.2: A preprocessed data point

2 Training

Then a VAE with 2D convolutional layers is trained on those data (the 12-tone vectors as seen as (12,1) matrixes, for implementation purposes), on Google Colab's GPUs. We used [5] as a starting point. All implementation is done using Python and Keras. The training is done using a β-VAE with a cold start, meaning that if we train for N epochs, then β will be set to 0 during the first $\frac{N}{10}$ epochs and will then increase linearly up to a value between 0 and 1 (in our experiment, the value 0.2 was found to be a good compromise). The training is done for 300 epochs, and lasts for about 45min. Going to further epochs "completes" the latent space, eliminating the "holes", but it does not change the overall structure of the latent space (as seen on following figures).

On fig. 2 we can see some latent spaces (all reduced in 2-D via principal component analysis) for different dimensions d.

3 Evaluation

At this point we should have a structured latent space of chromas. In order to evaluate the latent space, we create a synthetic labeled dataset of chromas. Each

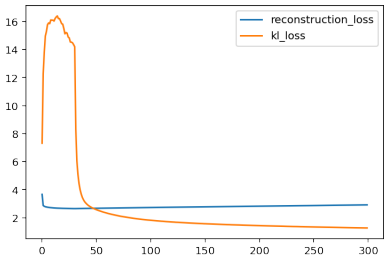


Figure 3.3: KL and reconstruction loss of the VAE. The KL starts diminishing around the 30th epoch, which is when β starts increasing.

chroma is built from a chord (restricted to major or minor triads) and then enriched harmonically.

We thus have a test dataset $Y = \{(s_i, l_i)\}_i$ where s_i is the slice and l_i the chord label (e.g. D:min).

We then project this dataset in the latent space (by feeding it into the encoder) and compute the centroid for each chord class. This is what we obtain (after 2-D PCA), with each chord class having a different color :

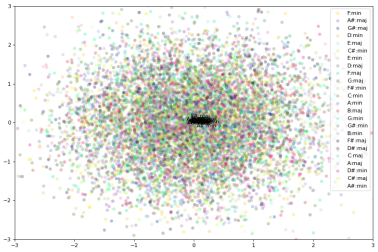
How to assess the quality of this space ? Let's assume we have a distance d defined on chords (more on this later), such that $d(c, c') \in [0, 1]$. We denote $V_K(c)$ the K nearest neighbors of c.

$$\forall c \in \text{Chords}, S_c = \frac{\sum_{c' \in V_K(c)} d(c, c')}{K}$$

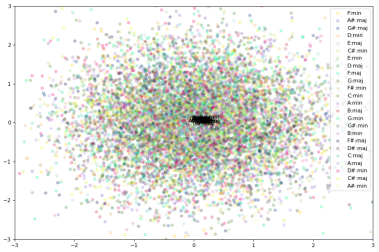
S_c is the "score" of c. The lower it is, the better the chord is placed in the space. The total score of the space is simply computed as $S = \frac{\sum_c S_c}{24}$. This will give us a tool to evaluate the performance of our model.

What distance do we use ? This was inspired by [6]. Let c_1 and c_2 be two chromas with chord labels l_1 and l_2. The first idea would be to use the euclidean distance between the two chromas. This approach is not well-suited since the distance between a tonic chord and its dominant would be higher than the distance between the tonic and its minor relative, which is not acceptable in the case of tonal music (where the dominant should be the "closest"). Therefore we use the following distance :

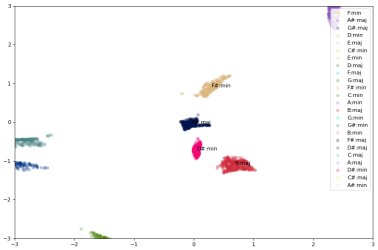
$$d(c_1, c_2) = \begin{cases} \min(1, ||c_1 - c_2||) & \text{if } c_1 \text{ is dominant to } c_2 \text{ or vice versa} \\ ||c_1 - c_2|| & \text{else} \end{cases}$$



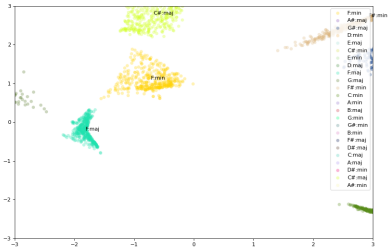
d=2, epoch = 0



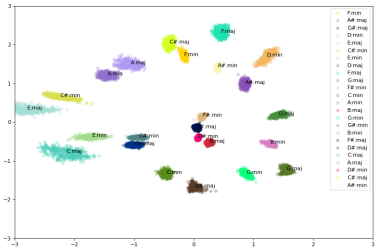
d=3, epoch = 0



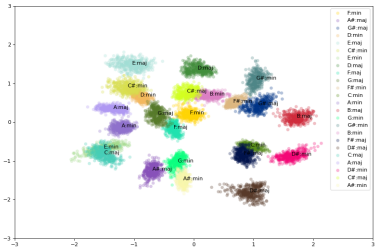
d=2, epoch = 20



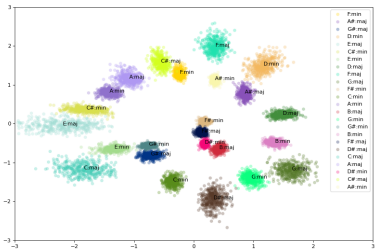
d=3, epoch = 20



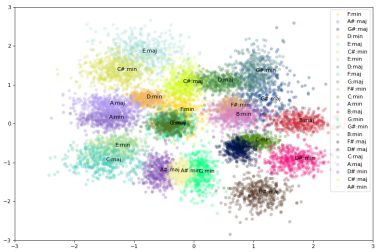
d=2, epoch = 100



d=3, epoch = 100



d=2, epoch = 299



d=3, epoch = 299

Figure 3.4: How latent spaces evolve with the training epoch

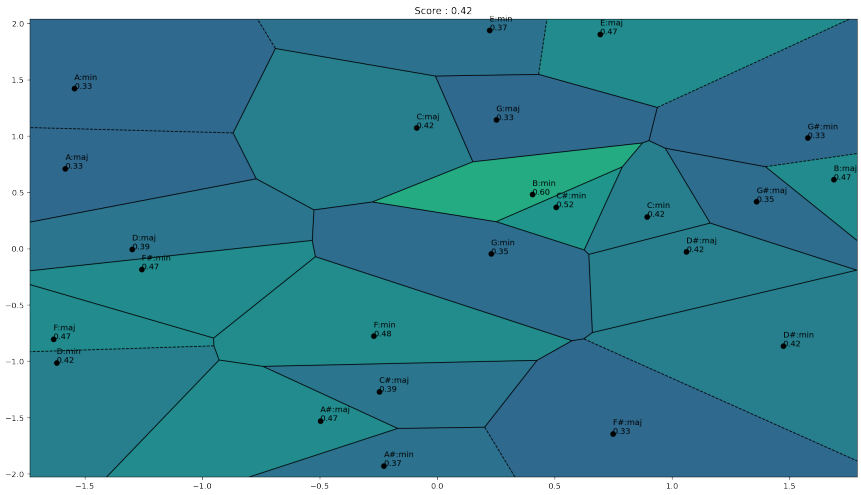


Figure 3.5: 3-D latent space reduced to 2-D via PCA ; each cluster has been reduced to one point (its centroid), and then a Voronoi tesselation has been computed

In order to have d between 0 and 1 as aforementioned, we simply need to normalize it by $\max_{c,c' \in \text{Chords}} d(c, c')$.

3.1 Comparison to the Tonnetz space

Considering such a distance, an ideal organization of the latent space would be a neo-riemannian Tonnetz space. Neo-riemannian considers that a transition from a chord to another should minimize the melodic movement (this rule was already known and applied for centuries). It defines three elementary operations to change from a chord to another :

- Operation R (relative), between a chord and its relative (e.g. Cmaj $\rightarrow$ Amin, or Cmin $\rightarrow$ Ebmaj)
- Operation L (Leittonwechsel), between a chord and its counter-relative (e.g. Cmaj $\rightarrow$ Emin or Cmin to Amaj)
- Operation P (parellel) between a chord to the other mode of the same chord (e.g. Cmaj $\rightarrow$ Cmin or Cmin $\rightarrow$ Cmaj)

These operations generate what we call the Tonnetz space :

Computing our score on the Tonnetz space gives us a score of 0.28 (with $K = 5$), see

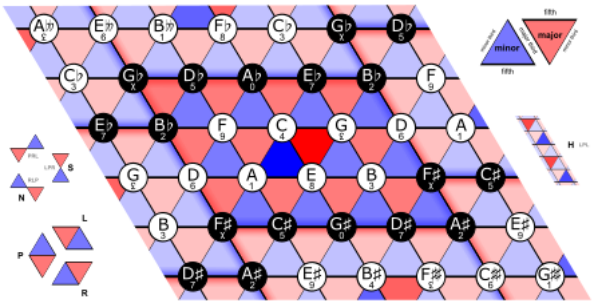


Figure 3.6: The Tonnetz space (source: Wikipedia)

fig. 3.8.

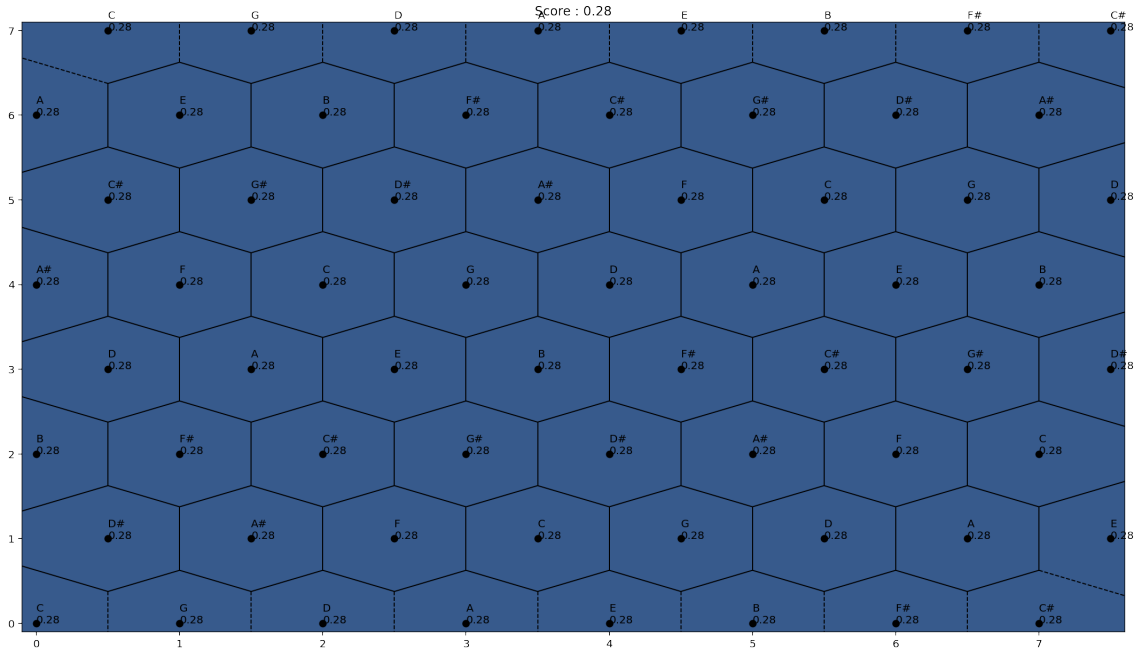


Figure 3.7: Evaluation of a 2D Tonnetz. $S = 0.28$

3.2 Comparison to the original SOM model

The current SOMax implementation uses a Self-Organizing Map (SOM) that was computed a while ago, in the form of 3600 chroma (12-D) vectors, as well as 3600 corresponding classes (121 classes in total). We first reduce the 3600 chroma vectors

to 121 (one by class, by averaging all the chromas of each class into one "centroid"). Then, we generate 24 chromas corresponding to the 24 minor and major chords. For each of those chromas, we find the closest chroma in the reduced SOM. Then, we obtain a correspondance between a chord label and a SOM chroma class, that enable us to visualize the SOM latent space (as always, reduced to 2 dimensions via PCA) :

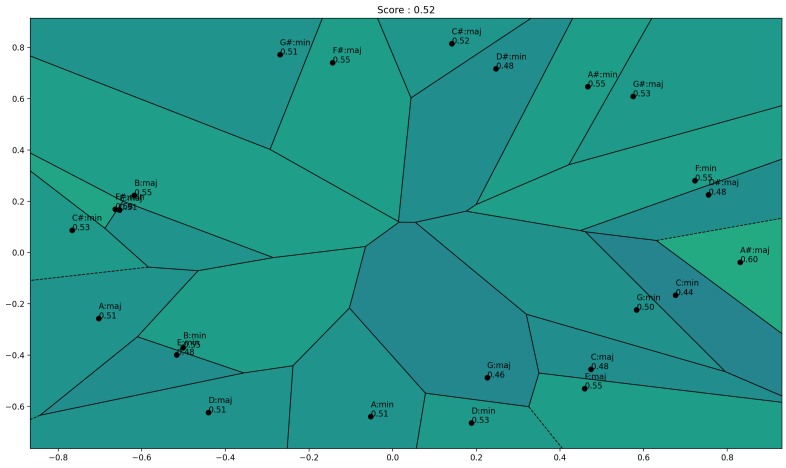


Figure 3.8: Evaluation of the SOM. $S = 0.52$

We first see from the score that the SOM is not performing as good as the VAE. Moreover, the apparent structure of the latent space is less convincing ; the fact that there is not space

3.3 Final results

The best results were obtained with a VAE set with $d = 3$, i.e. a 3-dimensional latent space.

d	**VAE**	**SOM**	**Tonnetz**
2	0.43	0.52	0.28
3	**0.42**		
4	0.44		
6	0.45		
8	0.44		
10	0.46		

Table 3.1: Scores of VAE with different latent-space dimensions, compared to the SOM score and Tonnetz score, using the aforementioned distance on chords

d	**VAE**	**SOM**	**Tonnetz**
2	0.49	0.63	0.28
3	**0.47**		
4	0.49		
6	0.51		
8	0.50		
10	0.50		

Table 3.2: Scores of VAE with different latent-space dimensions, compared to the SOM score and Tonnetz score, using the Euclidean distance between the chord vectors

Conclusion

During this internship I understood the gist of the research on automatic improvisation that was done at IRCAM in the last years, by spending the first month or so reviewing literature. My technical background in machine learning was helpful, although I had to deeply understand VAEs (which I had hardly touched), which was very instructive. After a review of SOMax 2, I proposed a new approach for the latent space of SOMax, using Variational Autoencoders. By defining a metric that evaluates the structure of the latent space on harmonic criterias, one can can conclude that this approach is superior. The next step would be to fully implement this model into the SOMax Core, but that was out of the reach of this internship. Doing so would enable us to fully test this new latent space, by listening to what SOMax generates using it as its clustering model, which is what SOMax is all about, in the end ; the metric used in this report is very simple and does not reflect, for instance, the distance between each cluster of the latent space, which is crucial (see fig. 2 to see how clusters are far more separated in the $d = 2$ space).

References

[1] Laurent Bonnasse-Gahot. An update on the somax project.

[2] Joakim Borg. The SOMax 2 Theoretical Model. `http://repmus.ircam.fr/`, 2021.

[3] Joakim Borg. The SOMax 2 Software Architecture. `http://repmus.ircam.fr/`, 2021.

[4] Carl Doersch. Tutorial on variational autoencoders, 2021.

[5] François Chollet. Variational autoencoder. `http://keras.io/examples/generative/vae/`.

[6] Tristan Carsault, Jérôme Nika, and Philippe Esling. Using musical relationships between chord labels in automatic chord extraction tasks. In *International Society for Music Information Retrieval Conference (ISMIR 2018)*, Paris, France, September 2018.

[7] Gérard Assayag and Shlomo Dubnov. Using factor oracles for machine improvisation. *Soft Computing*, 8(9):604–610, 2004.

[8] Ken Déguernel. *Apprentissage de structures musicales en contexte d'improvisation*. Theses, Université de Lorraine, March 2018.

[9] Jérôme Nika, Ken Déguernel, Axel Chemla-Romeu-Santos, Emmanuel Vincent, and Gérard Assayag. DYCI2 agents: merging the "free", "reactive", and "scenario-based" music generation paradigms. In *International Computer Music Conference*, Shangai, China, October 2017.

[10] Lilian Weng. From autoencoder to beta-vae. *lilianweng.github.io/lil-log*, 2018.

[11] Mathieu Prang. *Apprentissage pour la représentation musicale symbolique*. PhD thesis, IRCAM, 2021.