Machine Learning Project
Latent sequencing for dynamic musical patterns

Axel Chemla Romeu Santos[1,4] - *Ninon Devis*[1,2,3]
Constance Douwes[1] - *Cyril Lavrat*[1,2,3] - *Emmanouil Plitsis*[1,2,3]
Alice Rixte[1,2,3] - *Lydia Rodriguez de la Nava*[1,2,3]

[1] Institut de Recherche et Coordination Acoustique Musique (IRCAM)

[2] Télécom Bretagne [3] Sorbonne Université

[4] Universita Degli Studie di Milano, Laboratorio d'Informatica Musicale (LIM)

January 20, 2020

"A computer can be programmed to play instrumental music,
to aid the composer, or to compose unaided"
M. V. Mathews [1]

Abstract

Computational music generation is a current field of research that is still growing alongside with the burst of machine learning's potential. Especially when it comes to the domain of generating systems which seems to be particularly suitable for this purpose as it aims to synthesise new data from the analysis of already existing instances. The *Variational Auto-encoder* (VAE) is a major subclass of generative systems. It builds a low dimensional latent space by encoding the input distribution with proper regularisations learned during the learning phase and constrains. It then rebuilds some new data by sampling from it. Those reconstructions stemming from the latent space are then semantically meaningful samples with similar properties compared to the inputs. One of the branches of sound synthesis is granular synthesis. It generates sounds by dividing up samples into little sound grains that can then be manipulated and played at different speeds, frequencies, levels and phases. The main goal of this project is to use the principle of the VAE in order to recreate a fully controllable, practical and user-friendly instrument: a granular synthesiser.

1 Introduction

Computer music remains one of the major fields embodying the crossroad between sciences and arts. In accordance with its multidisciplinarity, it takes advantage of all the state-of-the-art technologies in order to serve the creativity of the artists and support them through the process of composition. Machine learning is becoming more and more present and efficient in many fields, and computer music is no exception. Indeed, generative models have shown an incredible ability to produce highly realistic sounds, images and texts and thus can be used as a tool for music synthesis. In the deep generative models' world, variational auto-encoders (VAE) are among the most widely used ones, and therefore the one that will be relied on for this present work.

As a first part of this project, the theory will be presented, as well as the general knowledge that is required to understand the mechanisms of the VAE. The latter will first be applied to the MNIST database. It will then be adapted for audio processing in order to implement a granular synthesiser. Afterwards, the implementation and design of the VAE to make it into an instrument for music composition will be discussed. The results will finally be presented along with the limits of the tool and the drawn conclusions.

"What I cannot create, I do not understand."
Richard Feynman

2 Variational auto-encoders

2.1 Generative models

Generative systems are machine learning models described as a class of statistical models that can generate new data instances. A generative system will model how data are placed throughout a space and then reconstruct new data with properties similar to the original [2].

2.2 General presentation

Variational auto-encoders belong to the models of generative systems. They inherit the same architecture as the classic auto-encoder but they are built on Bayesian inference which is a method of statistical inference in which the probabilities of various postulated causes are calculated from the observation of known occurrences.

2.2.1 Auto-encoders

The auto-encoder is an unsupervised artificial neural network whose training is built upon two simultaneous optimization tasks. As shown on figure 1, a first neural network named *the encoder* constructs *the latent space* in order to provide a low-dimensional representation of the data. A second neural network designated as *the decoder* generates back outputs that are as close to the original inputs as possible. Therefore, the latent space can be seen as a compressed abstract space, or images of the learned database, and may be used as an intermediate space for any processing.

However the auto-encoder is based on deterministic encoding, meaning that not all the latent positions can be leveraged to produce relevant reconstructions.

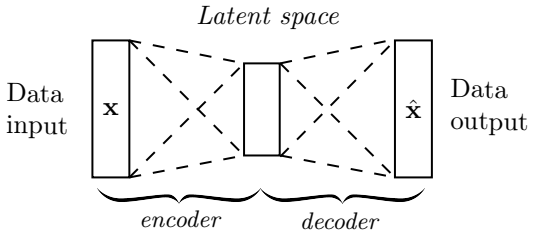


Figure 1: Basic auto-encoder architecture
$\mathbf{x}$ is encoded into a latent space.
Navigation into the latent space
lead to the construction of an estimation, $\hat{\mathbf{x}}$

2.2.2 Variational inference

Thanks to variational inference the auto-encoder can organise data in an optimal space but it can also generate new consistent data traveling through this space (see figure 2). To achieve this, variational inference approximates a conditional density of latent variables given observed ones by optimizing a family of densities over the latent variables [3].

In our case let $\mathbf{x} \in \mathbf{R^D}$ be the observed variables (input of the VAE) and $\mathbf{z} \in \mathbf{R^E}$ the set of latent variables with joint density $p(\mathbf{z}, \mathbf{x})$. The approximate inference problem is to compute the conditional density $p(\mathbf{z}|\mathbf{x})$ of latent variables as a function of observations. This can be written as follows, where $p(\mathbf{x})$ is the marginal density of observation.

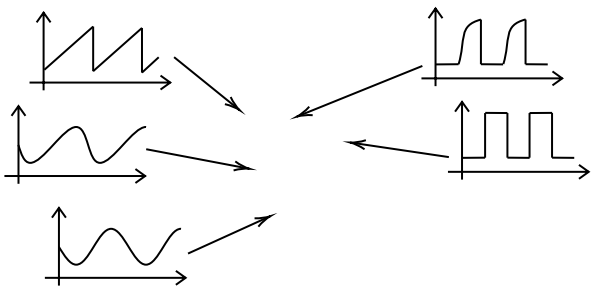


Figure 2: Regularisation tends to create a "gradient" over latent space's encoded information

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z},\mathbf{x})}{p(\mathbf{x})}$$

At the end of the process, an input $\mathbf{x}$ will be associated to a statistical distribution of latent variables $\mathbf{z}$. This may be seen as a blur filter on our latent space making connections between areas in the space partially meshed by $\mathbf{x}$ in $\mathbf{z}$.

2.2.3 Variational Auto-Encoders : VAE

Variational Auto-Encoders (VAE) can be seen as an auto-encoder using variational inference to build a continuous latent space. The target of the VAE is not to reconstruct $\hat{\mathbf{x}}$ closest to $\mathbf{x}$ but to generate a space in which it is possible to produce new $\hat{\mathbf{x}}$, similar to the inputs. Reconstructions of these new $\hat{\mathbf{x}}$ is done by the decoder part.

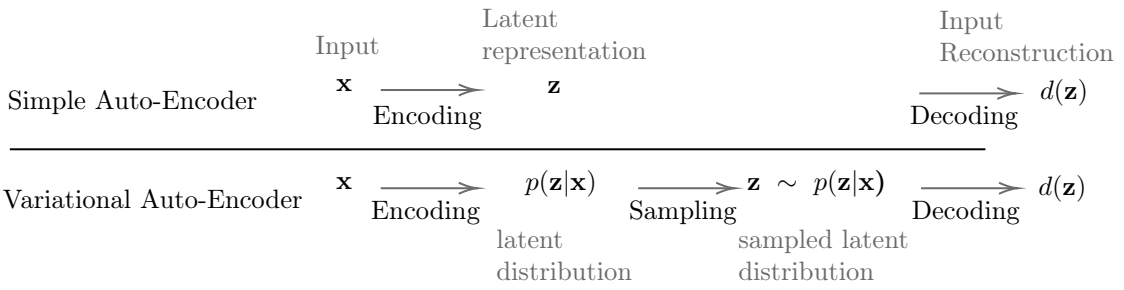


Figure 3: Auto-Encoder and Variational Auto-Encoder differences

2.3 Encoder and Decoder

In order to compress $\mathbf{x}$ into a small dimensional space, the encoder used in a VAE is a neural network where inputs are the raw data $\mathbf{x}$ and output are a latent representation $\mathbf{z}$ with a smaller dimension. As a conditional filter of parameters θ the neural network's encoder can be represented by a conditional function: $q_\theta(\mathbf{z}|\mathbf{x})$ which is likely a stochastic distribution like a Gaussian or a Bernoulli one.

In another hand, the decoder expands the data representation from the latent space to the original space format with a conditional network of parameter ϕ, $p_\phi(\mathbf{x}|\mathbf{z})$.

2.4 Learning

2.4.1 Activation Functions

An activation function defines the output of a node or a set of nodes given an input or a set of inputs. Indeed, a neuron has no boundary by construction and is able to output infinity. To avoid that behaviour, activation function operates as a comparison mask to set the output value of node. Moreover, the activation function must be non-linear to allow the system to model non-linear behaviour.

2.4.2 Feed Forward

Given inputs from the database or from the previous layer, each unit computes transformations $\mathbf{h}$ and then applies an activation function $g(\mathbf{h})$. During this process, each variable is stored in cache and is further used in the backward propagation.

2.4.3 Loss function

Machine learning is based on the optimization of a loss function. It evaluates the gap between the algorithm model and the input data given an objective. By minimizing the "cost" associated with the event, neural networks tend to encode the objective in its structure.

2.4.4 Backward propagation

The goal of backward propagation is to lead random initial settings of each neuron to converge to a value that minimizes the loss function. When a neural network propagates an input signal $\mathbf{x}$ through its parameters it then back-propagates information about error, in reverse through the network. First the network makes a guess about the data using its parameters set randomly at the beginning, then the network distance from the objective is measured with a loss function. Finally the error is back-propagated to adjust the wrong parameters.

2.4.5 Adam Optimizer

During the update step in the backward propagation, a value has to be set at a local minimum using the loss function's cost. To do so the conventional method is to use a gradient descent method. Adam is an efficient method that can be used to update network weights in a neuronal network [4].

2.5 VAE and MNIST characters drawing

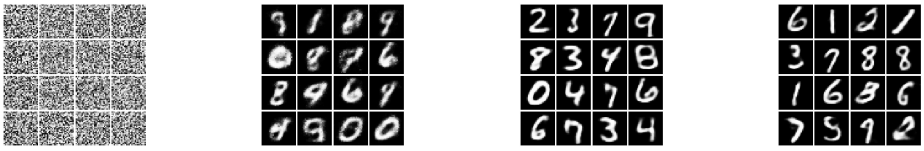
MNIST database is a large database of handwritten digits normalized into a 28x28 pixels bounded box with only black and white pixels. Like all VAEs, the goal here is to construct a latent space that describes variations between different inputs, digits in this case.

Here, the latent space is built by linear layers. There are 2 layers for the encoder, the first one is a classic linear layer that maps the size of images. The two next layers are put in parallel in order to compute μ and log_var. The second part, the decoder is also made of two layers but this time in sequence. Each layer is made of a Linear network with ReLU activation function except for the last one which contains a sigmoïd function to smooth out images.

After a couple of Feed Forward and Backward propagations for each batch of data, the encoder and decoder can reconstruct data and even create new ones by setting value of μ and log_var at the input of the decoder. The neural network converges very fast and can reproduce quite faithfully numbers from the second iteration.

Predictably, the first iteration is barely noise with a high loss. The second iteration gives a better result and it is already possible to guess that it is digits. The eighth iteration reconstructs the best image possible as the loss stagnates around 104 from this point to the end.

The command of our generative model, is a position in the latent space given to the decoder. Traveling into this latent space, produce a continuum of output from decoder. (as shown on figure 2)



(a) Iteration 1 Loss = 150.6 (b) Iteration 2 Loss = 135 (c) Iteration 8 Loss = 103.2 (d) Iteration 42 Loss = 104.4

Figure 4: Results pictures of MNIST training for several epochs

3 VAE as a Granular synthesizer

3.1 Granular synthesis

Granular synthesis is a very well known additive synthesis technique first used by Iannis Xenakis [5] for music composition and sound designs. It involves the generation of thousands of very short sonic *grains* in order to create a combination of them for a larger acoustic event. A grain of sound is a tiny part of a signal, every grain having a specific waveform and peak amplitude [6]. The grains are then concatenated in order to produce a musical and rhythmical event.

3.2 Database

The construction of the database is an essential part of the work as it conditions the training of the VAE. The goal is to generate sonic grains, hence the database must be sliced into small grains with the same size as the output. The `loader.py` is the main script that deals with the construction of the database. It uses the librosa library, a python package for music and audio analysis. The `get_data_loader` function loads the raw database which is then sliced and shuffled into an array of proper size grains thanks to the `Grains` class. This database is then sent to the `train_loader` and the `test_loader`. In all circumstances, in order to ensure that the training and the testing are done on a coherent dataset, some of the grains are plotted.

3.3 Train and Test

The `vae_main.py` gets the useful arguments and computes the main code. The database is split in order to obtain a training dataset and a test dataset. The training set, 80% of the database, is used for the VAE training whereas the testing dataset, which has to be independent from the training one but with the same probability distribution, is used to point out any over fitting.

3.4 Model of VAE's architecture

As we have seen, the VAE structure is based on two neural networks: one for the encoder and one for the decoder. The VAE has to learn from a grain and then will reconstruct audio wave forms with similar properties as the inputs. Those grains can be seen as images, hence, the model of the VAE is rooted on convolutional neural networks (CNN). Basically, the role of the CNN is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good inference. The encoder and the decoder are then made respectfully with four layers of convolutions with judicious choices for the parameters:

	Convolutional layers				Deconvolutional layers			
	layer 1	layer 2	layer 3	layer 4	layer 1	layer 2	layer 3	layer 4
Input Channels	1	64	32	16	1	16	32	64
Output Channels	64	32	16	1	16	32	64	1
Kernel Size	5	8	10	13	13	10	8	5
Stride	1	2	4	4	4	4	2	1
Padding	3	4	5	7	6	5	4	1

The latent space has a forced size of 16 hidden dimensions and the padding values are chosen so that the dimensions at the entry of the encoder is the same at the exit of the decoder, generally trying to follow: $padding = \lceil (kernel/2) \rceil$

For the training, batches of 62 are sent and according to the building of the database, with grains with a size of 512. Every layer of convolution and deconvolution is accompanied by one batch normalization (BN on the figure 5) that raises the independence of each layer and slightly reduces over-fitting by adding some noise to each hidden layer's activation [7]. After every batch normalisation one ReLu function is added to avoid non-linearity. The last step of the architecture of the VAE is a Tanh function which replaces the Sigmoïd of the MNIST VAE as it is more appropriate for Gaussian parameters and has a stronger gradient. The global architecture of the VAE is presented figure 5:

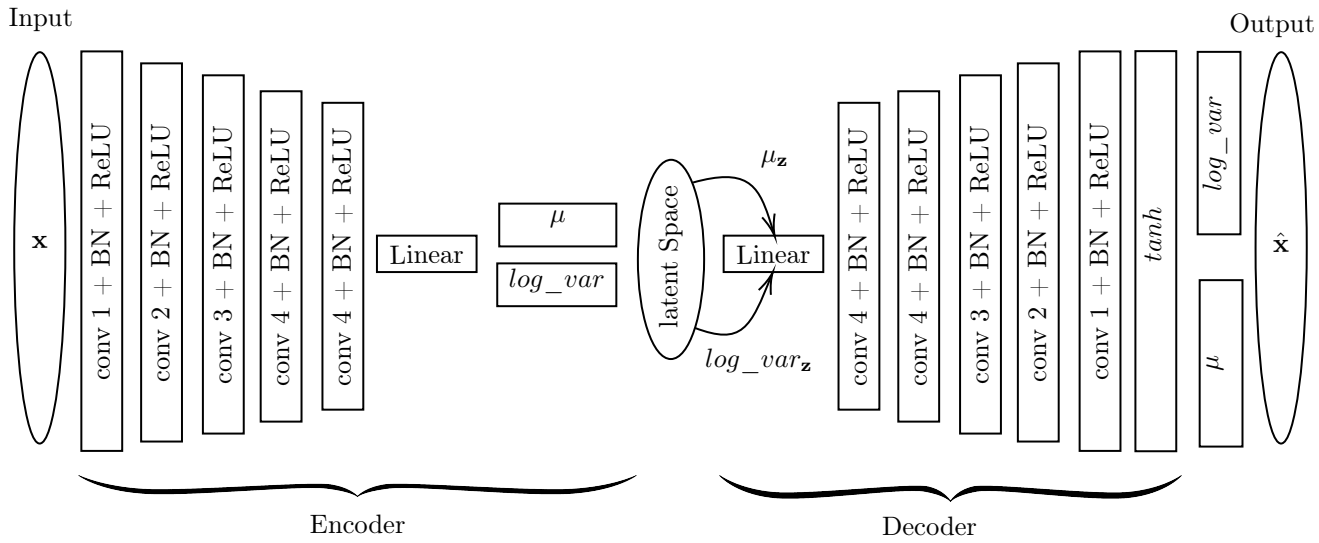


Figure 5: VAE Model's architecture

3.5 Loss Function

As VAEs are based on variational bayesian method [8], the statistical inference problems can be seen as optimization problems. The next step is to define a loss function that will decrease during the training process. First and foremost, as the inputs are not binary anymore the loss function must take Gaussian parameters as variables.

The reparameterization trick [9] is used and thus the loss function is divided in two parts: on the one hand, the *reconstruction loss* compares the outputs with the inputs, on the other hand the *variational loss* compares the latent vector with a zero mean of the Gaussian distribution.

The decrease of the loss function is equivalent to a maximisation of the Evidence Lower BOund (ELBO) function [3]:

$$ELBO(\theta, \phi, \mathbf{x}, \mathbf{z}) = \mathbb{E}_{q\theta}[logp_\phi(\mathbf{x}|\mathbf{z})] - \mathcal{D}_{KL}[q_\theta(\mathbf{z}|\mathbf{x})||p(\mathbf{z})] \tag{1}$$

3.5.1 Reconstruction loss

The first part in equation (1), $\mathbb{E}_{q\theta}[logp_\phi(\mathbf{x}|\mathbf{z})]$, stands for the reconstruction loss. As the `loss_train_test.py` script displays, the `recon_loss` coerces the decoder to rebuild outputs that are as close as possible to the inputs.

3.5.2 Variational loss

The Kullback-Leibler divergence, second part of equation (1), is a measure of the dissimilarity between two probability distributions, here between $q_\phi(\mathbf{x}|\mathbf{z})$ and p($\mathbf{z}$). This loss term penalises the VAE if it starts to produce latent vectors that are unlikely from the desired distribution.

$p(\mathbf{z})$ is the latent variable distribution that will be sampled. The simplest way to do so is hence to choose a Gaussian distribution as $p(\mathbf{z})$ equal to $\mathcal{N}(0, 1)$ and the aim is to have $q(\mathbf{z}|\mathbf{x})$, the projection of our data $\mathbf{x}$ into latent variable space, as close as possible to a Gaussian with parameters $\mu(\mathbf{x})$ and $\Sigma(\mathbf{x})$, the mean and the variance given $\mathbf{x}$. The KL divergence between those two distributions can be simplified in accordance with the obtained distributions and can be written as [10]:

$$D_{KL}[Q(z|X)||P(z|X)] = D_{KL}[\mathcal{N}(\mu(\mathbf{x}), \Sigma(\mathbf{x})||\mathcal{N}(0, 1))] = \frac{1}{2}\sum_k (exp(\Sigma(\mathbf{x})) + \mu^2(\mathbf{x}) - 1 - \Sigma(\mathbf{x}))$$

With q($\mathbf{z}|\mathbf{x}$) the projection of our data $\mathbf{x}$ into the latent variable space and p($\mathbf{x}|\mathbf{z}$) that generates data given the latent variable. In other words, q($\mathbf{z}|\mathbf{x}$) is the encoder, $\mathbf{z}$ the latent space and p($\mathbf{x}|\mathbf{z}$) is the decoder.

The global loss is finally calculated as $ELBO(\theta, \phi, \mathbf{x}, \mathbf{z}) = \mathbb{E}_{q\theta}[logp_\phi(\mathbf{x}|\mathbf{z})] - \beta\mathcal{D}_{KL}[q_\theta(\mathbf{z}|\mathbf{x})||p(\mathbf{z})]$

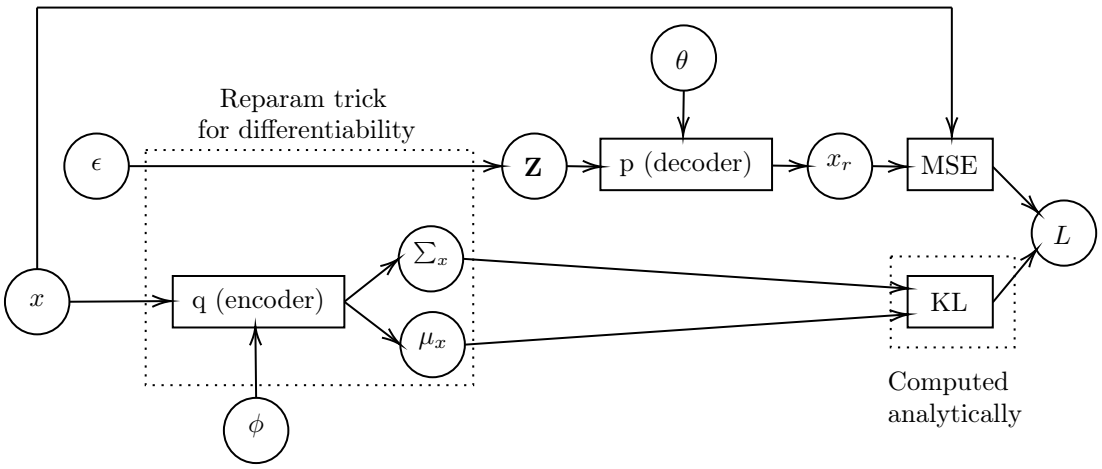


Figure 6: Loss function building path
ϵ stands for a sample noise

3.5.3 The β-VAE

β is an additional hyper parameter placed as a regulator of the KL divergence. The β-VAE consistently discovers more factors of variation in the data, and it learns a representation that covers a wider range of factor values. The β factor increases the lower bond formulation and regulates the qualitative nature of the representations learnt by the model which will learn the most relevant representation of the data.[11]

The last part of the construction of VAE is to save the trained model so as to reconstruct samples from the latent space. The first part, the serialization, saves the model in the `saved_model` file. Then, the deserialization restores the data in order to store the trained model for a future utilisation without the waste of training time. The `comparison.py` script plots a comparision of some random inputs facing their reconstruction, and then synthesises an audio file from all the reconstructions in order to check the training process.

4 Implementation and design of the synthesizer.

After implementing and training the VAE with the database, an efficient implementation of the actual granular synthesizer is still needed. Two different and complementary approaches were explored in order to maximize the odds of success. Both approaches link the VAE to Pure Data/Max, which provides a better suited environment for designing an interface. Finally, the choice of interface is discussed, as it is of great importance in granular synthesis.

4.1 VAE as a Max external

One of the most suitable lead to connect the VAE to Max is to use a Max external. The aim is to get the control of the VAE via Torchscript to make use of the Python code in C. The external reads the grains. For that, it has a cross-fade option, which consists of cross-fading the grains to generate new sounds. The external receives three parameters : a signal consisting of frequency values, another signal consisting of the starting sample of the grain and an integer input for the length of the cross-fade. It outputs the signal corresponding to those parameters.

4.2 Controlling the VAE using Pure Data/Max via OSC

Our second approach is to use Python as the synthesizer's sound engine, and only use Pure Data as a controller, sending control signals by OSC, using the UDP protocol. In fact, every kind of software (or hardware) that can communicate via OSC can be used as an interface in this way.

In Python, the pyo signal processing library is being used. We create a simple granular synthesizer in pyo, using the Granulator object, which loads the (flattened) output of the decoder to a pyo Data Table, and performs simple manipulations, like playback at varying pitch of many simultaneous grains. Pyo also comes with it's own GUI, which can also be used instead of external software (although it can be buggy when used with an IDE, so beware).

In Pure Data, using the mrpeach external, we are able to send vectors of size 16 to Python via OSC (using the UDP protocol) in real time, with coordinates chosen according to our interface of choice.

The first implementation simply takes batches of floating point numbers in the range $(0, 1)$ as inputs and outputs a batch of looping grains, which are stored in a DataTable and manipulated by a Granulator object. The change of coordinates changes the grain in real time, and also more of the parameters can be changed, like the pitch, the number of simultaneous grains etc.

Another simple implementation is selecting a continuous trajectory in the latent space and sampling enough points from there, which corresponds to the creation of a 1D sound table of consecutive sound grains (using cross-fading to avoid artifacts due to amplitude jumps), which can be played back and manipulated like a regular sound file by a classical granular synthesizer. This approach has the benefit of reducing the latency caused by the decoder, which is considerable when choosing grains in real time. The choice of trajectory can be complex enough to have enough variation in all dimensions. In the prototype, only linear trajectories centered at the origin were chosen, which can be controlled by choosing an end point in the latent space.

5 Conclusions and perspectives

5.1 Conclusions

The principle of the VAE can be used in order to synthesize sound. After building a suitable database made of little sound grains, the VAE processes them through convolutional networks in order to create a latent space that can be sampled to build new sounds with similar properties as the inputs. The trained model is then plugged to Max MSP in order to create a generator of sound, comparable to an instrument, for music composition.

5.2 Perspectives

This project could have been pursued in several ways:

- A data pre-processing could have been apply for a faster training. The first idea would have to perform a Short-Term Fourier Transform (STFT) analysis with a sliding Hamming window on the input audio signal. In this method [12], the magnitude spectra are sent to the encoder whereas the phase spectra are stored for the synthesis. The output audio signal is then reconstructed by combining the decoded magnitude spectra with the phase spectra, and finally applying an inverse STFT.
- An other pre-processing would have been to use the Non-Stationary Gabor Transform (NSGT) instead of STFT as it allows to tailor the scale and according to its small dimension, enables a faster training.
- Liking the latent space to a variational synthesizer with a handy and user-firendly user interface would have been the ultimate achievement.

References

[1] M. V. Mathews, "The digital computer as a musical instrument," *Science*, vol. 142, no. 3592, pp. 553–557, 1963.

[2] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," *arXiv preprint arXiv:1401.4082*, 2014.

[3] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[5] I. Xenakis, *Formalized music: thought and mathematics in composition*. No. 6, Pendragon Press, 1992.

[6] C. Roads, "Introduction to granular synthesis," *Computer Music Journal*, vol. 12, no. 2, pp. 11–13, 1988.

[7] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, ACM, 2008.

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

[9] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," in *Advances in Neural Information Processing Systems*, pp. 2575–2583, 2015.

[10] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther, "Ladder variational autoencoders," in *Advances in neural information processing systems*, pp. 3738–3746, 2016.

[11] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework.," *ICLR*, vol. 2, no. 5, p. 6, 2017.

[12] F. Roche, T. Hueber, S. Limier, and L. Girin, "Autoencoders for music sound modeling: a comparison of linear, shallow, deep, recurrent and variational models," *arXiv preprint arXiv:1806.04096*, 2018.