

Generating new music with deep probabilistic models

Valentin Vignal

NATIONAL UNIVERSITY OF SINGAPORE

2020

Generating new music with deep probabilistic models

Valentin Vignal

(BSc, CentraleSupélec)

**A THESIS SUBMITTED FOR THE DEGREE
OF MASTER OF COMPUTING
DEPARTMENT OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE**

2020

Supervisor:

Assistant Professor Harold Soh

Examiners:

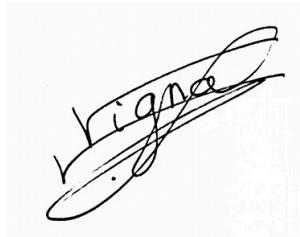
Assistant Professor Gim Hee Lee

Assistant Professor Jun Han

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.



Valentin Vignal

09 April 2020

ACKNOWLEDGEMENTS

I would like to express my special thanks and gratitude to my supervisor Harold Soh who gave me the opportunity to work on this project combining Artificial Intelligence and Music.

Secondly I would also like to thank all the members of the research team, and especially the PhD. students Abdul Fatir and Yaqi Xie who, despite their busy schedules, taught me, and help me in many scenarios.

I also would like to thank the Doctor Dorien Herremans who gave me precious advises at the beginning of my project.

TABLE OF CONTENTS

Summary	ix
LIST OF TABLES	x
LIST OF FIGURES	xi
1 Introduction	1
2 Background	3
2.1 Music representation	3
2.1.1 Musical stave	3
2.1.2 MIDI	5
2.1.3 Pianoroll	6
2.2 Music theory	7
2.2.1 Scale and Rhythm	7
2.2.1.1 Scale	7
2.2.1.2 Rhythm	8
2.2.2 Harmonics	8
2.2.2.1 Harmonics	8
2.2.2.2 Chords	9
2.2.2.3 Dissonance	9
2.2.2.4 Harmony	10
2.3 Music arrangement	11
2.4 Neural Network architectures	11
2.4.1 Convolutional neural network	12

2.4.1.1	Convolutional Layer	12
2.4.1.2	Pooling Layer	12
2.4.2	AutoEncoder	13
2.4.3	Variational AutoEncoder	14
2.4.3.1	Summary and vulgarization of VAE	16
2.4.4	Generative Adversarial Network	16
2.4.4.1	Discriminator cost function	19
2.4.4.2	Generator cost functions	19
2.4.4.2.1	Minimax	19
2.4.4.2.2	Heuristic, non-saturating game	19
2.4.4.2.3	Maximum likelihood game	19
2.4.4.3	Divergence choice	20
2.4.4.4	Summary and vulgarization of GANs	20
2.4.5	Recurrent Neural Networks	21
2.4.6	Transformers	21
2.4.7	Multimodel Variational AutoEncoder	24
2.4.8	Neural Autoregressive Distribution Estimation	26
2.4.9	Restricted Boltzmann Machine	27
3	Related works	29
3.1	Objectives	29
3.1.1	Generator	30
3.1.2	Accompaniment	30
3.2	Music Representation	30
3.2.1	Audio Signal	31
3.2.2	MIDI Signal	32
3.2.2.1	Pianoroll	32
3.2.2.2	Text	33
3.2.2.3	Chords	35

3.3	Encoding	35
3.3.1	Features	35
3.3.1.1	Metadata	35
3.3.2	Tensor encoding	36
3.4	Architectures	37
3.4.1	RBM	37
3.4.2	NADE	37
3.4.3	AE	38
3.4.4	VAE	38
3.4.5	RMVAE	38
3.4.6	GAN	39
3.4.7	Transformers	39
3.5	Generation Process	39
3.5.1	Sampling	40
3.5.2	Input Manipulation	40
3.5.3	Feed forward and RNN architecture	41
3.5.3.1	DeepBach	41
4	Contribution: Multi-Modal Music Generation	43
4.1	Objectives	43
4.2	Data representation	44
4.2.1	Polyphonic music	44
4.2.2	Monophonic Music	45
4.3	RMVAE Architecture	46
4.3.1	Global Architecture	46
4.3.2	Encoder	48
4.3.2.1	Convolutional encoder	48
4.3.2.2	Recurrent encoder	48
4.3.3	PoE	50

4.3.4	Recurrent layers	51
4.3.4.1	LSTM cells	51
4.3.4.2	RPoE	52
4.3.5	Decoder	52
4.3.5.1	Convolutional decoder	52
4.3.5.2	Recurrent decoder	53
4.3.6	Last layer	54
4.3.6.1	Polyphonic Music	54
4.3.6.2	Monophonic Music	55
4.4	Loss Function	55
4.4.1	Scale	56
4.4.2	Rhythm	58
4.4.3	Harmony	59
5	Results	61
5.1	Generated Samples	61
5.2	Activation function of the note _{continue} for monophonic music .	62
5.2.1	LSTM Encoder and Decoder	63
5.3	Tasks	63
5.3.1	Generate	64
5.3.2	Fill	65
5.3.3	Redo	66
6	Experiments	69
6.1	Transpose the data	69
6.2	Model size	71
6.3	Custom losses	73
6.3.1	Scale and Rhythm	73
6.3.2	Harmony	73

6.4	RPoE	74
7	Conclusion	78
Bibliography		79
Appendices		92
.1	Interpolation project	92
.2	Segment Sum	93
.3	Roll _n	93
.4	BandPlayer	94
.5	Coconet Process	95
.6	Transformer architecture	95
.7	Hyper-Parameters tuning	95
.8	Network details	98
.8.1	Hyper Parameters	98
.8.2	Masking	99
.9	Training details	99
.9.1	Noise	99
.9.2	KLD Annealing	100
.10	Implementation details	100

Summary

This paper introduces the work I have done for my dissertation. As a musician, I like to play music, improvise and create or arrange songs. The main goal of this dissertation is to create a neural network architecture able to handle several tasks a musician or a composer can do.

To this purpose, I created a new architecture that I call RMVAE (Recurrent Multimodal Variational AutoEncoder) (section 4.3) which combines the MVAE architecture [1] and LSTM cells (see section 2.4.5). Only one trained model can be used to create a melody, several musical parts at the same time, harmonize a melody or reconstruct missing parts in a song.

As a musician I also tried to integrate prior musical knowledge to the model by creating 3 cost functions (section 4.4).

For this project, I used the MIDI dataset which is Bach's Chorales dataset from the music21 corpus [2] and used their framework [3] to open and create MIDI files. The deep learning framework I used it Tensorflow/Keras [4, 5]. I made my python code for this dissertation available online: <https://github.com/ValentinVignal/midiGenerator>.

The results show that the extra losses are not helping the model which indicates that the neural network is able to understand those musical rules and tendencies by its own.

LIST OF TABLES

2.1	Note names and duration	5
2.2	Harmonics of $A4$	9
4.1	Correspondence between duration value and musical length . .	45
1	Neural Network Hyper-Parameters	99

LIST OF FIGURES

2.1	Musical Stave example	4
2.2	Notes on a musical stave	4
2.3	Notes on Piano	4
2.4	Pianoroll example from the software FL Studio [6]	6
2.5	C Major Pentatonic scale (or A Minor Pentatonic scale)	7
2.6	Resonance waveform	10
2.7	Lead voice and its harmony part	11
2.8	Max pooling operation	13
2.9	AutoEncoder	14
2.10	Reparameterization trick for VAE	15
2.11	The graphical model structure of GANs	17
2.12	GAN framework	18
2.13	Differences between the two direction of the KLD	20
2.14	Recurrent Neural Network	22
2.15	LSTM and GRU cell	22
2.16	Scaled Dot-Product Attention and Multi-Head Attention	23
2.17	Graphical model of the MVAE	24
2.18	MVAE architecture	25
2.19	NADE architectue	26
2.20	RBM architecture	27
3.1	Bach Doodle application	31
3.2	Example of an audio waveform	31
3.3	Example of audio spectrogram	31

3.4	Example of correspondence between a pianoroll representation and an array	32
3.5	(a) tonnetz and (b) the extended tonnetz matrix with pitch register	33
3.6	Example of correspondence between a pianoroll representation and a text	33
3.7	BachBot’s example encoding of three musical chords ending with a fermata (“pause”) chord	34
3.8	Example of correspondence between a pianoroll and its chords representation	35
3.9	Fermata symbol	36
3.10	Feed forward generation process	41
3.11	Graphical representation of DeepBach’s neural network architecture for the soprano prediction	42
4.1	Architecture of the RMVAE	47
4.2	RMVAE encoder architecture	49
4.3	Measure Encoder using bidirectional LSTM	50
4.4	RMVAE PoE Fully Connected layers	51
4.5	RMVAE LSTM layer	51
4.6	RPoE architecture	52
4.7	RMVAE decoder architecture	53
4.8	Measure Decoder using bidirectional LSTM	54
4.9	Scale Loss	57
4.10	Rhythm Loss	58
4.11	Harmony Circle for A	59
4.12	Harmony Loss	60
4.13	Harmony _{n} Loss	60

5.1	Generated pianoroll by the RMVAE	61
5.2	Generated pianoroll by the RMVAE with a softmax activation	62
5.3	Generated pianoroll by the RMVAE with recurrent architecture with softmax activation	63
5.4	Generated pianoroll by the RMVAE with recurrent architecture with softmax and sigmoid activation	64
5.5	Task <i>Generate</i>	64
5.6	Song given to the model for the <i>Fill</i> task	65
5.7	Output of the model for the <i>Fill</i> task of the first voice	65
5.8	Song given to the model for the <i>Redo</i> task	67
5.9	Iteration 1 of the <i>Redo</i> task	67
5.10	Iteration 2 of the <i>Redo</i> task	67
5.11	Iteration 3 of the <i>Redo</i> task	68
5.12	Iteration 4 of the <i>Redo</i> task	68
6.1	Loss comparison between transposed and non-transposed data	70
6.2	Accuracy comparison between transposed and non-transposed data	70
6.3	Global loss between for size of models	71
6.4	KLD value for different size of models	71
6.5	Loss of one voice for different size of models	71
6.6	Loss of one voice for different size of models	71
6.7	Pianoroll generated by the small model	72
6.8	Pianoroll generated by the medium model	73
6.9	Pianoroll generated by the large model	73
6.10	Loss comparison between models with and without Scale and Rhythm loss	74
6.11	Accuracy comparison between models with and without Scale and Rhythm loss	75

6.12	Loss comparison between models with and without Harmony loss	75
6.13	Accuracy comparison between models with and without Harmony loss	76
6.14	Loss comparison between models with and without RPoE layer	76
6.15	Accuracy comparison between models with and without RPoE layer	77
1	Segment sum operation	93
2	Roll ₂ example	94
3	COCONET process	96
4	Transformer architecture	97
5	β value through training for KLD annealing	101

LIST OF SYMBOLS

AE	Auto Encoder
AI	Artificial Intelligence
AMAE	ArgMax AutoEncoder
C-RBM	Convolutional Restricted Boltzmann Machine
CNN	Convolutional Neural Network
EDM	Electronic Dance Music
ELBO	Evidence Lower BOund
EQL	EQuation Learner
FC	Fully Connected
GAN	Generative Adversarial Network
GP	Gaussian Process
GPU	Graphic Processing Unit
GRU	Gated Recurrent Unit
KLD	Kullback-Leibler Divergence
LSTM	Long Short-Term Memory
MCMC	Markov Chain Monte Carlo
MDMM	Multimodal Deep Markov Model
MiB	MebiByte
MIDI	Musical Instrument Digital Interface
ML	Machine Learning
MVAE	Multimodal Variational AutoEncoder
NADE	Neural Autoregressive Distribution Estimation
NN	Neural Network
PoE	Product of Experts

RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RMVAE	Recurrent Multimodal Variational AutoEncoder
RNN	Recurrent Neural Network
RPoE	Recurrent Product of Experts
RTRBM	Recurrent Temporal Restricted Boltzmann Machine
SGD	Stochastic Gradient Descent
VAE	Variational AutoEncoder
VQ-VAE	Vector Quantisation - Variational AutoEncoder
VRAE	Variational Recurrent AutoEncoder
VST	Virtual Studio Technology

CHAPTER 1

Introduction

As a musician, I like to play music alone or within a band. I spend some time to create songs and arrange them. And finally, I enjoy jamming with a band and improvise.

On the deep learning part, music generation has recently considerably improved. Models like DeepBach [7], BachBot [8], BachDoodle [9] can generate, harmonize music in Bach's style. Other non-musical model came up recently. New architectures are being created and are able to generalize data in a more meaningful way, be more consistent through time [10], can handle more complex data with several modalities [11, 12].

Therefore, I try in this work to combine my understanding of music, the knowledge so far on music generation and the capability to handle complex data with several modalities. My goal is to create a unique model able to execute as many tasks as I could perform as a musician, create a song, harmonize a melody, jam with someone... I Instead of letting the model learn everything from scratch from the data, I also want to give it prior knowledge about music.

Thus, the first objective of my dissertation is to create a unique trained model able to generate a melody, generate several musical parts, create an accompaniment from a melody, create a melody from an accompaniment. The second objective is to insert some prior knowledge about music to the model.

The choices I have made for this project were mostly relying on my musical instinct. I tried to stay as close as possible to my musical thoughts in the way I constructed a model, process the data and so on.

The chapter 2 will explain some background knowledge to the reader. It will have a musical part and a deep learning part. The chapter 3 will summarize what has already been done about music generation in deep learning. The chapter 4 will present what are my contributions and what novel ideas I had for my Dissertation. Finally the chapters 5 and 6 will enumerate the results I got and what experiments I managed to conduct.

CHAPTER 2

Background

In this chapter, I will introduce some background knowledge that might be useful to the reader. Since this project is about music generation, I will explain and illustrate some basic concepts about music.

I will consider the Western music using equal temperament and don't consider the inharmonicity of stringed instruments. These are common assumptions in all the existing works about music generation.

The section 2.1 will explain how music is represented by musician and computers and how it is possible to use those representations to create tensors. The sections 2.2 and 2.3 will introduce the reader to basic musical rules, phenomena and definitions. Finally, the section 2.4 will enumerate several neural network architectures which are useful to understand this Dissertation.

2.1 Music representation

In this section I will explain how musicians represent the music on paper, and from it, how it is possible to represent the music in a abstract way in a computer without encoding any waveforms or actual *sounds*.

2.1.1 Musical stave

It is very useful for anyone to be able to write down their work to save it or share it with someone else. Musicians faced this issue too. They came up with the musical stave :

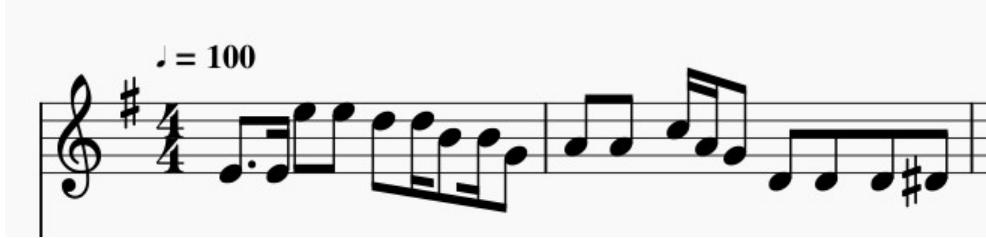


Figure 2.1: Musical Stave example

The vertical axis is the frequency axis and the horizontal axis corresponds to the time axis. In the figure 2.1, it is written that the tempo is 100bpm , the scale is *G major* (one \sharp) and the measure are divided with 4 beats ($4/4$ inscription). As said previously, the vertical position of a note indicates its frequency:

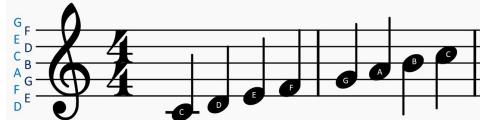


Figure 2.2: Notes on a musical stave

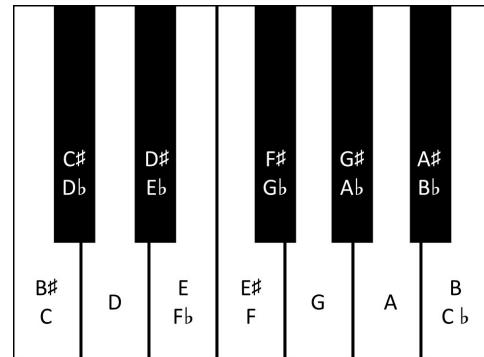


Figure 2.3: Notes on Piano

The figures 2.2 and 2.3 show the correspondence between the notes on a musical stave and on a piano keyboard. By default the notes correspond to the white keys of the piano (it is the *C major scale*).

One thing to know is that, between 2 followings notes (or piano keys), including white and black keys), the ratio between the fundamental frequencies of the 2 notes is always equal to $\sqrt{12}$. It means that if we consider the notes *A4* and *A5* (2 *A* notes in the octaves 4 and 5), the ratio between the fundamental frequencies is 2. This information will help to understand chords construction.

The length of a note is defined by its shape as shown in the Table 2.1. This

table is describing the relation between the length of a note, and its shape on the musical stave.

Name	Duration	Symbol
Whole note	4 beats	o
Half note	2 beats	♪
Quarter note	1 beat	♩
Eighth note	1/2 beat	♪
Sixteenth note	1/4 beat	♪

Table 2.1: Note names and duration

In this work, we won't consider shorter notes than Sixteenth notes because most of the songs are not using them. This is a common assumption in the existing work.

2.1.2 MIDI

The *MIDI* format (*.mid*) is a technical standard that describes a protocol. It was first used to carry musical messages between electronic instruments, software and devices. These messages are events about note information (for example, pitch, velocity, panning...) and some other parameters (for example, vibrato, volume...). The most important messages I will consider are:

- *Note on* is indicating that a note has to be played. It contains the channel information (which can be considered as an instrument), the pitch information (what note should be played) and the velocity. We could right an event as follow :

$$< \text{NoteOn}, 0, 50, 127 > \quad (2.1)$$

To describe the event *Start to play the note D3 with the maximum velocity (127) for the channel (instrument) 0*

- *Note off* is indicating to stop playing a note (for instance, release the keyboard key). The given parameters are the same as the ones given to the *Note On* event.

$$< NoteOff, 0, 50, 127 > \quad (2.2)$$

will stop the note started with the previous *Note On* event.

Each note is associated with a time value which can be expressed in number of ticks (time division). The header of file specifies how many ticks there are per quarter note.

2.1.3 Pianoroll

The pianoroll is a common representation of a musical stave in the music production softwares.



Figure 2.4: Pianoroll example from the software FL Studio [6]

The figure 2.4 shows a view of the pianoroll in the famous software *FL Studio* [6]. The composers of electronic music like EDM, Techno et caetera use this view to compose and arrange their songs.

2.2 Music theory

In this section, I will describe and explain some rules from music theory which are related to this work. The rules I am going to explain are the most common ones and most of the songs tend to follow them.

2.2.1 Scale and Rhythm

2.2.1.1 Scale

A *scale* is a set of notes. Because the human ear is now used to it, the notes of a scale will sound nice when they are played together. In traditional Western music, it generally consists of 7 notes. They are several types of scales. The most common is the *Major Scale* or the *Natural Minor Scale*. For example, the C Major scale uses all the white keys of the piano (Figure 2.3: A, B, C, D, E, F and G), as well as the A Natural Minor scale.

Another type of scales often used by the musicians to creates their *solo*s are the *Pentatonic* scales.

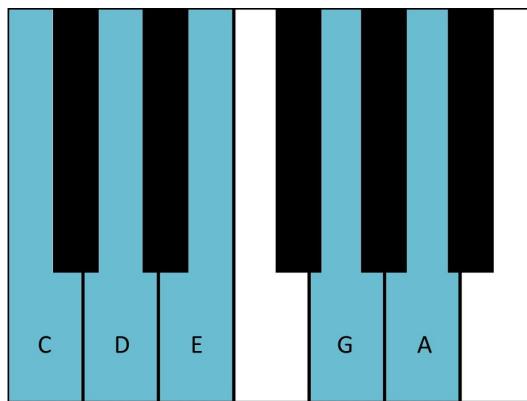


Figure 2.5: C Major Pentatonic scale (or A Minor Pentatonic scale)

As seen in the figure 2.5, the C Major pentatonic scale (which uses the same notes as the A Minor pentatonic scale) uses only five notes (A, C, D, E and G) which are contained in the C Major Scale. It is known that playing

in this scale will easily produce enjoyable and in-tune melodies or any musical parts.

2.2.1.2 Rhythm

The rhythm is an important part of the current music like *Pop Music*. It has the tendency to remain consistent through the song and to repeat some patterns. It helps the listener to easily follow the progression and allows him to listen what he's expecting to.

However, rhythm is more flexible than scale and harmonies, and there is no rhythm theory someone could follow. This is why classical music is not considered as a rhythmic music.

In this work, I considered and will consider only binary rhythm (each beat is divided into 2 smaller equal beats) and not ternary rhythm (each beat is divided into 3 smaller equal beats) because the binary rhythm is the most common one.

2.2.2 Harmonics

In this section, I will introduce some physical concept about musical sounds and timbre and then illustrate how it can explain some musical rules.

2.2.2.1 Harmonics

A sound is a sum of harmonics:

$$s(t) = \sum_{n=1}^{\infty} \alpha_n \sin(nft + \phi_n) \quad (2.3)$$

where f is the fundamental frequency and ϕ the phase.

Let us take an example with the *A4* note which has its fundamental frequency equals to $440Hz$. The 2^{nd} harmonic is *A5* $880Hz$. The consequence is

, when an instrument plays a *A4*, all the harmonics of a *A5* are also present. Let us take one step further, the *3rd* harmonic of *A4* is *E5* 1320Hz. It means it is possible to hear a *E5* from a played *A4*. The table 2.2 is referencing the firsts harmonics of the *A4* note.

Harmonic number	Frequency (Hz)	Note Name	Musical Interval
1	440	<i>A4</i>	Unison
2	880	<i>A5</i>	Octave
3	1320	<i>E5</i>	Fifth
4	1760	<i>A6</i>	Octave
5	2200	<i>C[#]6</i>	Major Third
6	2640	<i>E6</i>	Fifth

Table 2.2: Harmonics of *A4*

From the table 2.2, we can notice:

- The *A* and *E* notes are linked together (*Fifth* or reversed *Fourth* interval).
- The *A* and *C[#]* notes are linked together (*Major Third* interval).

2.2.2.2 Chords

The links between notes illustrated in the table 2.2 explains why a Major chord sound *nice* or *smooth*. A *A Major* chord is composed with 3 notes : *A*, *C[#]* and *E*. All the notes are already contained in the harmonics of a *A* sound.

A Minor chords (*A Minor* is *A*, *C*, *E*) will also sounds acceptable to the human ears because *A* and *C* share *E* in their harmonics (respectively the *Fifth* interval and *Third Major* interval)

2.2.2.3 Dissonance

When 2 frequencies are close to each other and added up, it is possible to observe a resonance phenomena.

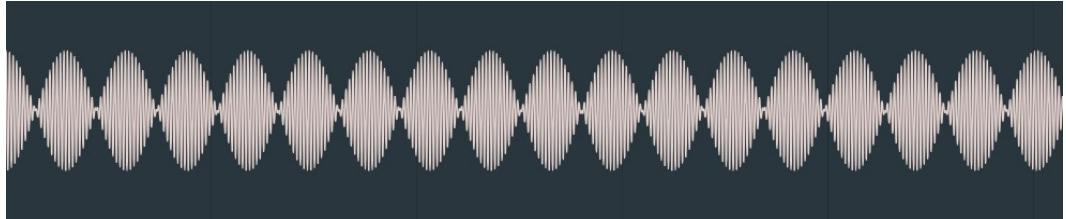


Figure 2.6: Resonance waveform

The figure 2.6 shows the waveform generated by a $B4$ ($494Hz$) and a $C5$ ($523Hz$) (*semitone* interval). This resonance is explained by the trigonometric identity:

$$\cos(f) + \cos(f + \delta f) = 2 \cos\left(\frac{2f + \delta f}{2}\right) \cos\left(\frac{\delta f}{2}\right) \quad (2.4)$$

This phenomena is unpleasant to hear and this is why, musician usually try to avoid to play notes that generate a resonance between their harmonics:

- *Semitone* interval (ex: A and $A\#$)
- *Tone* interval (ex: A and B)
- *Tritone* interval (ex: A and $D\#$)

2.2.2.4 Harmony

Either for classical music (Bach Chorales) or pop music (back singers), a common method to *fill a song* is to add harmony parts to the lead melody. The harmony parts usually follow the lead melody but on other notes. An example is provided in the figure 2.7.



Figure 2.7: Lead voice and its harmony part

The harmony parts will usually avoid unpleasant interval and mostly try to create the following ones:

- *Octave* and *unison* interval
- *Fifth* interval
- *Fourth* interval
- *Major Third* interval
- *Minor Third* interval

2.3 Music arrangement

Arranging a song is an entire musical field an a job. It is the art of giving an existing melody musical variety. To put it more simply, it is creating a accompaniment for an melody. It can includes chords, change the rhythm, add other musical parts.

To give an example, arranging a song could be create piano, guitar, drums and bass parts from voice melody with lyrics.

2.4 Neural Network architectures

In this section, I will briefly describe some neural network architectures.

2.4.1 Convolutional neural network

Convolutional networks are often used on images because they can preserve the spatial information. A CNN usually includes two types of layers :

- A convolutional layer
- A pooling layer

2.4.1.1 Convolutional Layer

For a 2D convolution, the convolutional layer takes as an input a tensor of shape (`height`, `width`, `channels`). The *filter* of the convolutional layer will have a shape (`h`, `w`, `channels`). Then the layer will do a 2D convolutional operation between the filter and the input through the axes corresponding to the `height` and the `width`.

$$y_\tau = \sum_{t=0}^{l-1} w_t \times x_{\tau+t} \quad (2.5)$$

The equation 2.5 shows the mathematical transformation for a 1D convolution with x as the input, w as the filter/kernel and y as the output.

2.4.1.2 Pooling Layer

A pooling layer is used to reduce the size of a tensor. It extracts a value from a region of the tensor. Two common poolings are:

- The *Average pooling* takes the average of the region
- The *Max pooling* takes the maximum of the region

The figure 2.8 illustrates how the max pooling operation works.

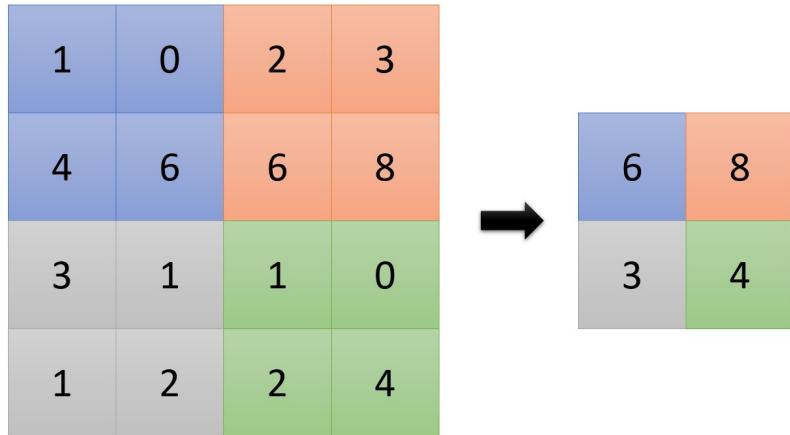


Figure 2.8: Max pooling operation

2.4.2 AutoEncoder

An AE [13, 14, 15] is composed of a *encoder* and a *decoder*. First, the input goes into the encoder. The output of the encoder is the latent space (the hidden layers), which is smaller than the input. The output of the encoder becomes the input of the decoder. The goal of the decoder is to reconstruct the input from the latent space. The hidden layers of the AE are smaller than the input which forces the network to compress the input and reduce its dimensions. Therefor, the AE has to learn "high level features" about the inputs. The figure 2.9 summarizes this architecture.

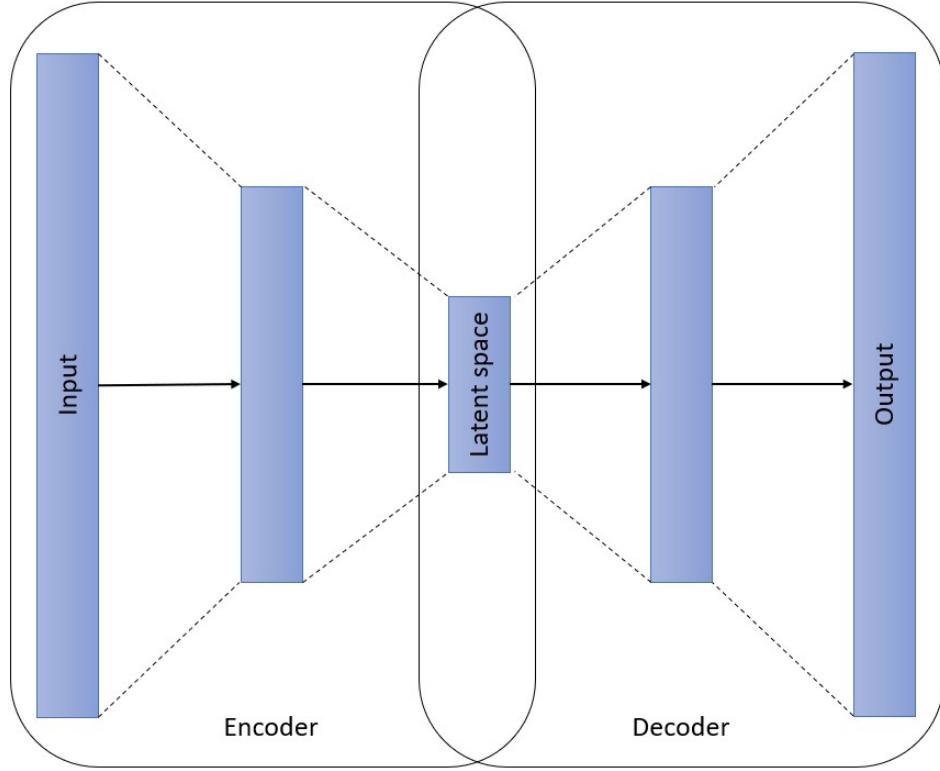


Figure 2.9: AutoEncoder
Source: Wikipedia

2.4.3 Variational AutoEncoder

The VAE [16, 17, 18, 19, 20] comes from the Bayesian Theorem (equation 2.6).

$$p(x|z) = \frac{p(z|x)p(x)}{p(z)} \quad (2.6)$$

It is a latent variable generative model of the form $p_\theta(x, z) = p(z)p_\theta(x|z)$ where $p(z)$ is the prior. z is usually a standard Gaussian distribution: $z \sim \mathcal{N}(0, I)$. The decoder $p_\theta(x|z)$ consists of a neural network, with the parameters θ composed with a simple likelihood (for example, a Bernoulli or Gaussian distribution). The objective of training is to maximize the marginal likelihood of the data (called the "*evidence*") (equation 2.7).

$$\text{argmax}_\theta(\log(p_\theta(x))) \quad (2.7)$$

However, since it is intractable (equation 2.8), the evidence lower bound is instead optimized. The ELBO is defined via an inference network $q_\phi(z|x)$, with parameters ϕ which serves as a intractable importance distribution (equation 2.9).

$$\log(p_\theta(X)) - \log \left(\int_Z p_\theta(X|Z)p(Z)dZ \right) \quad (2.8)$$

$$\text{ELBO}(x) = \mathbb{E}_{q_\phi(z|x)} [\log(p_\theta(x|z))] - \mathbb{D}_{KL} (q_\phi(z|x), p(z)) \quad (2.9)$$

Where $\mathbb{D}_{KL}(p, q)$ is the Killback-Leibler divergence between distributions p and q . The ELBO is usually optimize via stochastic gradient descent, using reparameterization trick to estimate the gradient [21].

The figure 2.10 shows the reparameterization trick. A training-time VAE implemented as a feedforward neural network, where $P(X|z)$ is Gaussian. Left is without the reparameterization trick, and right is with it Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior of these networks is identical, but the backpropagation can be applied only to the right network

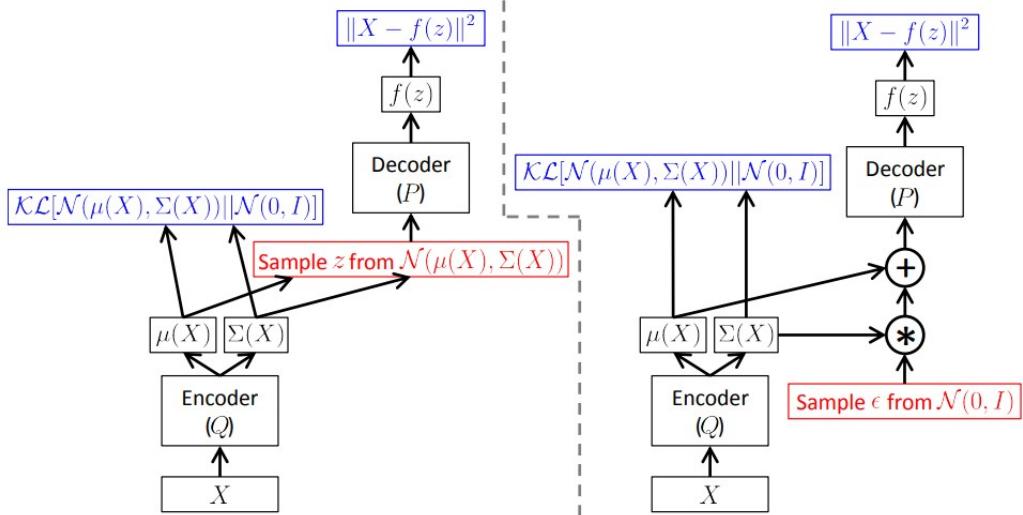


Figure 2.10: Reparameterization trick for VAE
Source: Tutorial on Variational Autoencoders [16]

2.4.3.1 Summary and vulgarization of VAE

A VAE is an autoencoder. The steps are the following:

1. The input goes first in the encoder which encodes it as a Gaussian distribution over the latent space.
2. A point is sample from this distribution.
3. This point is decoded by the decoder to reconstruct the input.

By encoding the normal distribution and not directly the latent space, it is possible to regularize the output of the encoder to avoid overfitting and ensure that the latent space has good properties that enable generative process.

To regularize the output of the decoder, an extra term is added to the loss function : Kulback-Leibler Divergence (KLD) between the encoded distribution and the centred and reduced normal distribution :

$$\mathbb{D}_{KL}(\mathcal{N}(\mu_{encoded}, \sigma_{encoded}), \mathcal{N}(0, 1)) \quad (2.10)$$

2.4.4 Generative Adversarial Network

The basic idea of GANs [22, 23, 24, 25] is to set up a game between two players. One of them is called the *generator* and creates samples that are intended to come from the same distribution as the training data. The other player is the *discriminator* and examines samples to determine whether they are real or fake. The discriminator learns using traditional supervised learning techniques, dividing inputs into two classes (real or fake). The generator is trained to fool the discriminator.

GANs are a structured probabilistic model containing latent variables z and observed variables x (figure 2.11).



Figure 2.11: The graphical model structure of GANs

The two players in the game are represented by two functions, each of which is differentiable both with respect to its inputs and with respect to its parameters. The discriminator is a function D that takes x as input and uses $\theta^{(D)}$ as parameters. The generator is defined by a function G that takes z as input and uses $\theta^{(G)}$ as parameters.

Both players have cost functions that are defined in terms of both players' parameters. The discriminator wishes to minimize $J^{(D)}(\theta^{(D)}, \theta^{(G)})$ by controlling $\theta^{(D)}$. The generator wishes to minimize $J^{(G)}(\theta^{(D)}, \theta^{(G)})$ by controlling $\theta^{(G)}$. Because each player's cost depends on the other player's parameters, but each player cannot control the other player's parameter, this scenario can be described as a game. The solution to a game is a Nash equilibrium. In this context, a Nash equilibrium is a tuple $(\theta^{(D)}, \theta^{(G)})$ that is a local minimum of $J^{(D)}$ with respect to $\theta^{(D)}$ and a local minimum of $J^{(G)}$ with respect to $\theta^{(G)}$.

The generator is simply a differentiable function G represented by a deep neural network. When z is sampled from some simple prior distribution, $G(z)$ yields a sample of x drawn from p_{model} .

The figure 2.12 illustrates the framework of GANs.

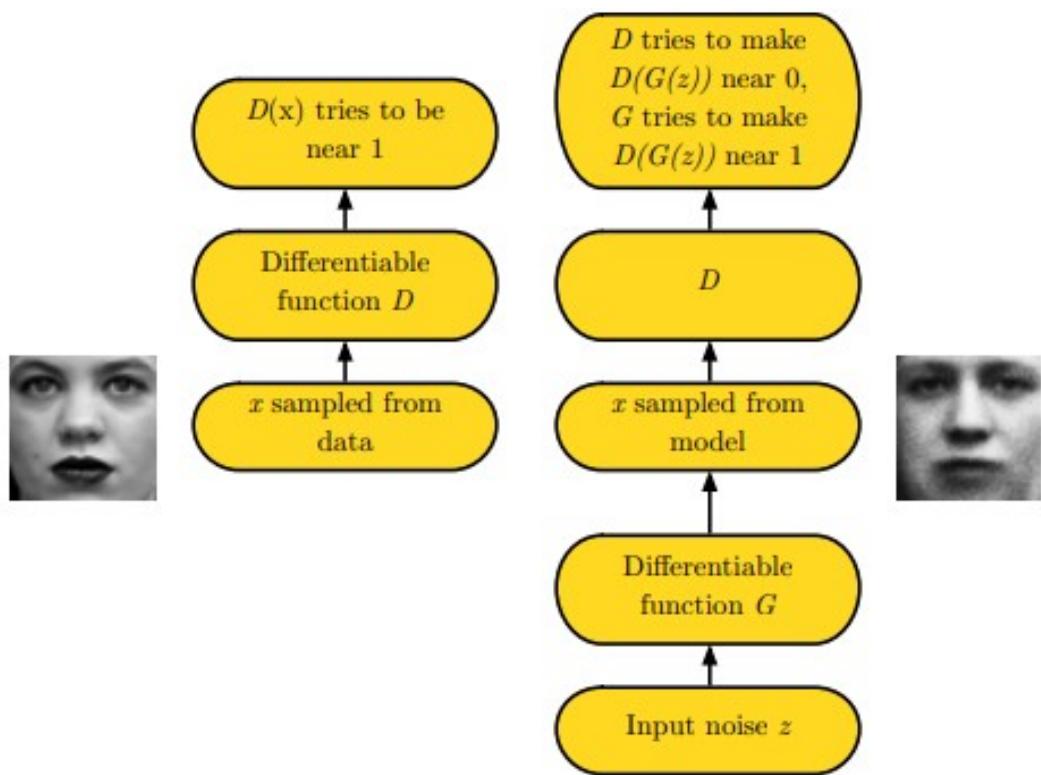


Figure 2.12: GAN framework
Source: Ian Goodfellow's paper [25]

2.4.4.1 Discriminator cost function

The cost function used for the discriminator is showed in the equation 2.11.

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{x \sim p_{data}}(\log(D(x))) - \frac{1}{2}\mathbb{E}_z(\log(1 - D(G(z)))) \quad (2.11)$$

2.4.4.2 Generator cost functions

There are several possibilities for the cost function of the generator.

2.4.4.2.1 Minimax The first one is the **Minimax** cost function where $J^{(G)} = -J^{(D)}$. This a *zero-sum game*. Zero-sum games are also called **minimax** games because of the form of the solution (equation 2.12).

$$\theta^{(G)*} = \operatorname{argmin}_{\theta^{(G)}} [\operatorname{argmax}_{\theta^{(D)}} [J^{(G)}(\theta^{(D)}, \theta^{(G)})]] \quad (2.12)$$

2.4.4.2.2 Heuristic, non-saturating game The second one is the **Heuristic, non-saturating game**. The minimax cost function doesn't perform well in practice. When the discriminator successfully rejects the generator samples with high confidence, the generator's gradient vanishes. The generator's cost becomes the equation 2.13

$$J^{(G)} == -\frac{1}{2}\mathbb{E}(\log(D(G(z)))) \quad (2.13)$$

The motivation of this cost function is to ensure that each player has a strong gradient it is "*losing*" the game.

2.4.4.2.3 Maximum likelihood game The third one is the **Maximum likelihood game**. This version maximizes the likelihood learning with GANs, which means minimizin the KLD between the data and the model. Goodfellow [24] showed that using the equation 2.14 where σ is the logistic function, is

equivalent to minimize the KLD between the data and the model.

$$G^{(G)} = -\frac{1}{2} \mathbb{E}_z [\exp (\sigma^{-1}(D(G(z))))] \quad (2.14)$$

2.4.4.3 Divergence choice

The choice of the divergence will influence the results of the training. The KLD is not symmetric. Minimizing $\mathbb{D}_{KL}(p_{data}, p_{model})$ is different from minimizing $\mathbb{D}_{KL}(p_{model}, p_{data})$. The figure 2.13 shows the differences between the two directions of the KLD. Minimizing the reverse KL might be expected

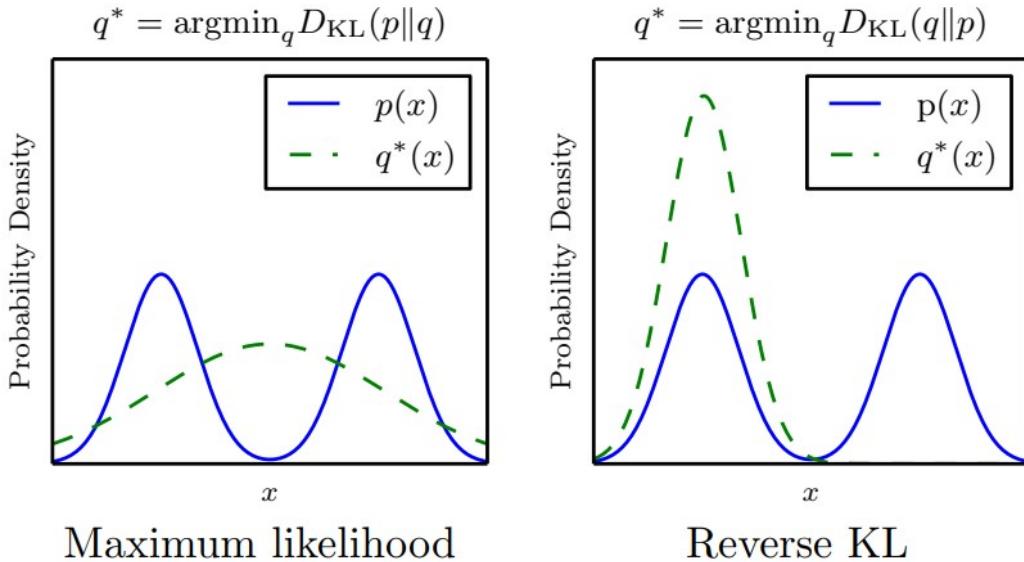


Figure 2.13: Differences between the two direction of the KLD
Source: Ian Goodfellow's paper [25]

to yield better samples because the model would prefer to generate samples that come only from modes in the training distribution. It would ignore some modes, rather than including all modes but generating some samples that do not belong from any training set mode.

2.4.4.4 Summary and vulgarization of GANs

GANs are composed of two models that are trained simultaneously:

- The *Generator* will learn how to create data that look real.
- The *Discriminator* will learn how to differentiate real and generated data.

The discriminator is trained with real data and data generated by the generator. The goal of the discriminator is to classify the real data as "*Real*" and the generated data as "*Fake*". On the other side, the generator takes some noise as an input to generate a datum. Its goal is to make the discriminator classify its generated data as "*Real*".

Through training, the generator will get better and create more consistent data so the discriminator classify them as "*Real*". It will then force the discriminator to become better and classify real and fake data more precisely. Thus, the generator is forced to create data which look more real. And so on...

2.4.5 Recurrent Neural Networks

A RNN [26] is a neural network where connections between nodes form a direct graph along a temporal sequence. The general architecture is showed in the figure 2.14.

The most used cells are the LSTM cell and the GRU cell. The architectures are showed in the figures 2.15.

2.4.6 Transformers

Transformers [10, 28, 29, 30, 31] have been introduced to the world by Ashish Vaswani et al. [32]. It is an attention mechanism illustrated in the figure 4. On this figure, the encoder is on the left and the decoder on the right.

The scaled dot-product attention and multi-head attention layers are drawn in the figures 2.16a and 2.16b.

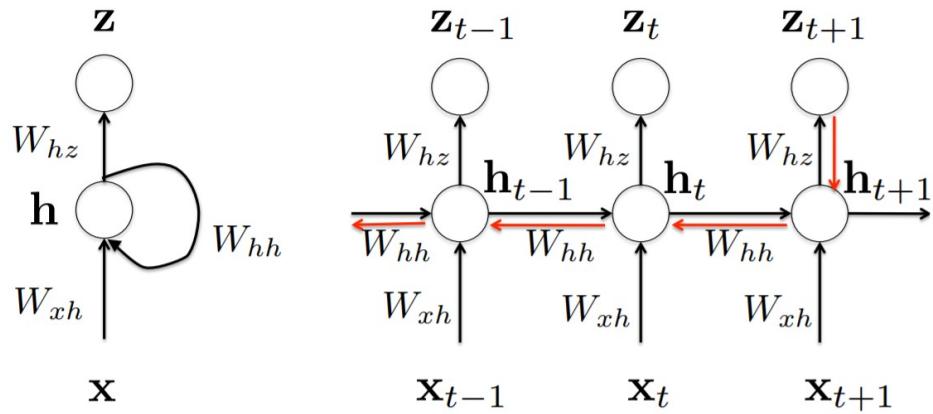


Figure 2.14: Recurrent Neural Network

Source: Gang Chen's paper [26]

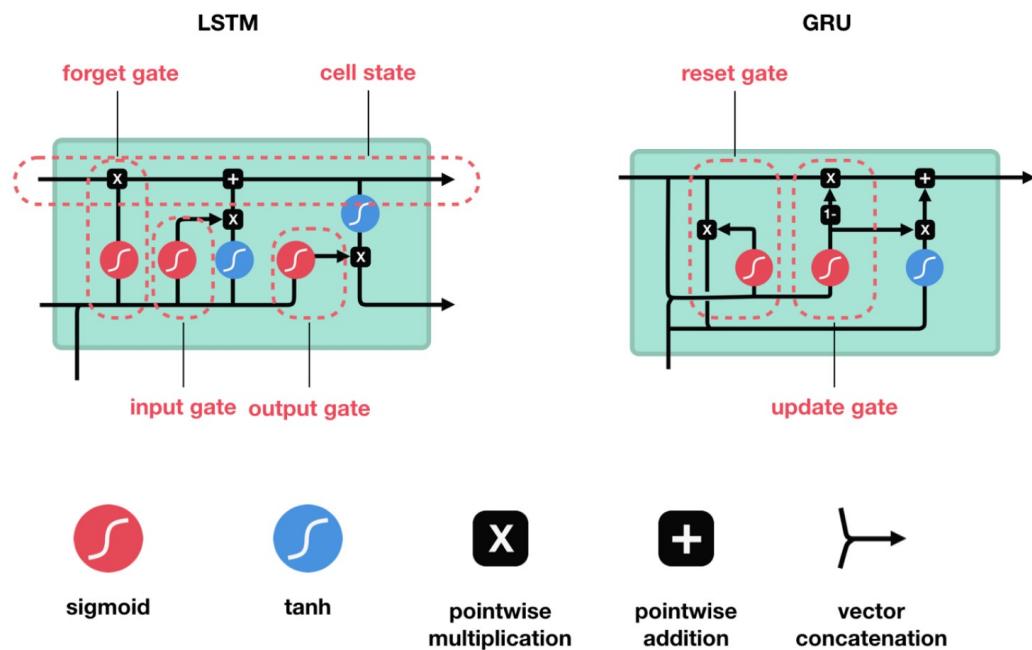


Figure 2.15: LSTM and GRU cell

Source: Michael Nguyen's tutorial [27]

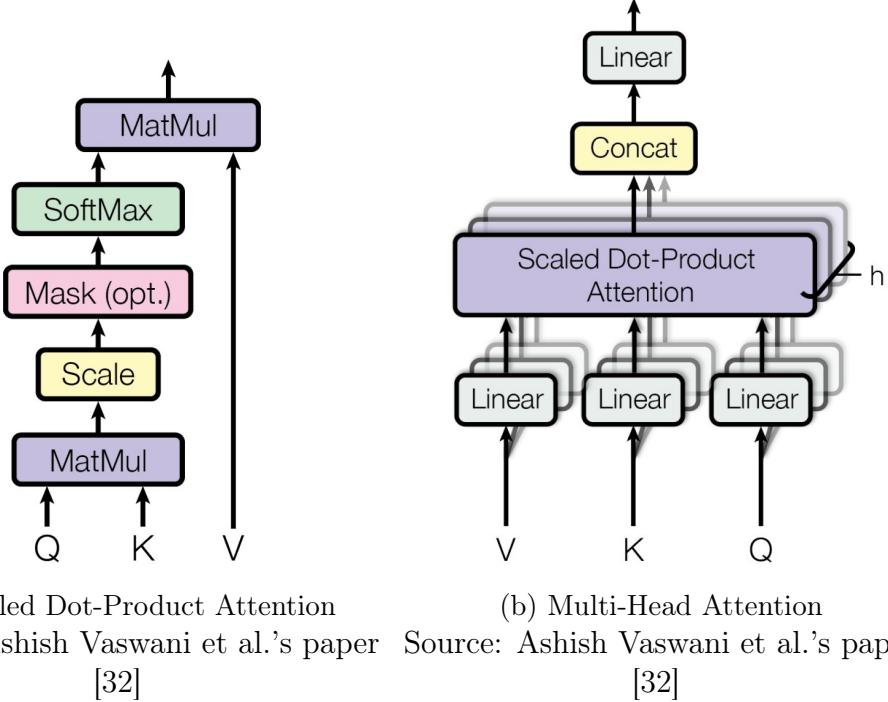


Figure 2.16: Scaled Dot-Product Attention and Multi-Head Attention
Source: Ashish Vaswani et al.'s paper [32]

The scaled dot-product attention operation follows the equation 2.15.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.15)$$

where d_k is the number of dimensions.

And the multi-head attention operation follows the the equation 2.16.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_d)W^O \quad (2.16)$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$.

To summarise the process, Q, K and V are respectively called the queries, the keys and the values. Their attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility

function of the query with the corresponding key.

The advantage of the transformer is that it is able to learn long-range dependencies. This is a challenge in many sequences tasks. However, a transformer doesn't take into account the position of the input from a sequence. Therefore, a positional encoding must be added.

2.4.7 Multimodel Variational AutoEncoder

The MVAE has been introduced by Mike Wu et al. [1]. The figure 2.17 represent the graphical model of the MVAE. The gray circles represent the observed variables. The MVAE uses a *Product of Experts* (PoE) inference network and a sub-sampled training paradigm to solve the multi-modal inference problem.

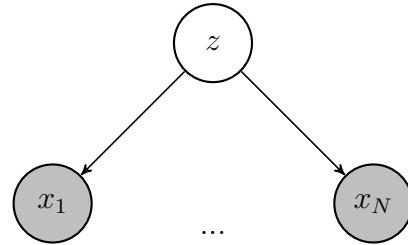


Figure 2.17: Graphical model of the MVAE

The conditional independence assumptions in the generative model (figure 2.17) imply a relation among joint- and single-modality posteriors :

$$\begin{aligned}
p(z|x_1, \dots, x_N) &= \frac{p(x_1, \dots, x_N|z)p(z)}{p(x_1, \dots, x_N)} \\
&= \frac{p(z)}{p(x_1, \dots, x_N)} \prod_{i=1}^N p(x_i|z) \\
&= \frac{p(z)}{p(x_1, \dots, x_N)} \prod_{i=1}^N \frac{p(z|x_i)p(x_i)}{p(z)} \\
&= \frac{\prod_{i=1}^N p(z|x_i)}{\prod_{i=1}^{N-1} p(z)} \frac{\prod_{i=1}^N p(x_i)}{p(x_1, \dots, x_N)} \\
&\propto \frac{\prod_{i=1}^N p(z|x_i)}{\prod_{i=1}^{N-1} p(z)} \approx \frac{\prod_{i=1}^N (\tilde{q}(z|x_i)p(z))}{\prod_{i=1}^{N-1} p(z)} = p(z) \prod_{i=1}^N \tilde{q}(z|x_i)
\end{aligned} \tag{2.17}$$

With $\tilde{q}(z|x_i)$ the model approximation of $\frac{p(z|x_i)}{p(z)}$

In other words, we can use a product of experts (PoE), including a “*prior expert*”, as the approximating distribution for the joint-posterior.

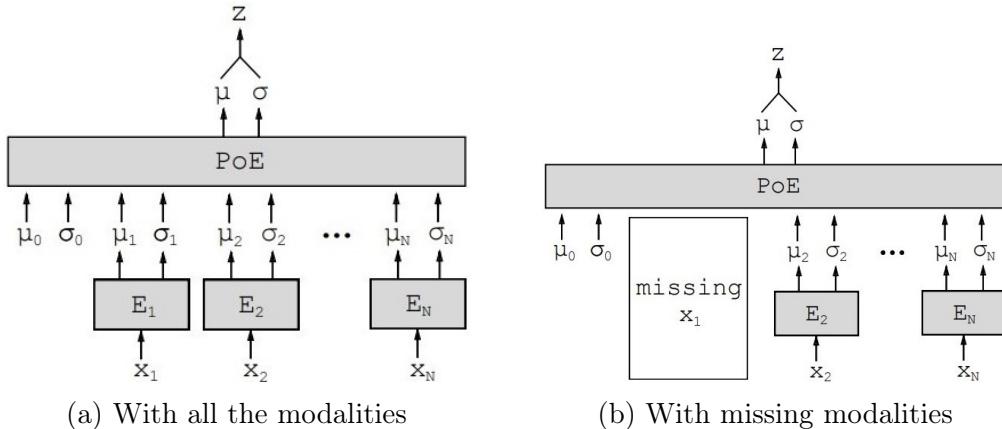


Figure 2.18: MVAE architecture
Source: Mike Wu’s paper [1].

The figure 2.18 and equation 2.17 show how the PoE can handle missing modalities by simply ignoring them.

2.4.8 Neural Autoregressive Distribution Estimation

A NADE [33, 34] is a neural network made to support a D -dimensional distribution $p(x)$ which verify:

$$p(x) = \prod_{d=1}^D p(x_{o_d} | x_{o_{<d}}) \quad (2.18)$$

It is a feed forward network parameterized as follow:

$$p(x_{o_d} = 1 | x_{o_{<d}}) = \text{sigm}(V_{o_d, \cdot} h_d + b_{o_d}) \quad (2.19)$$

$$h_d = \text{sigm}(W_{\cdot, o_{<d}} x_{o_{<d}} + c) \quad (2.20)$$

where, with H as the number of hidden units, $V \in \mathbb{R}^{D \times R}$, $b \in \mathbb{R}^D$, $W \in \mathbb{R}^{H \times D}$, $c \in \mathbb{R}^H$.

The hidden matrix W and bias c are shared by each hidden layer h_d . The figure 2.19 illustrates the Nade model. There is no path of connections

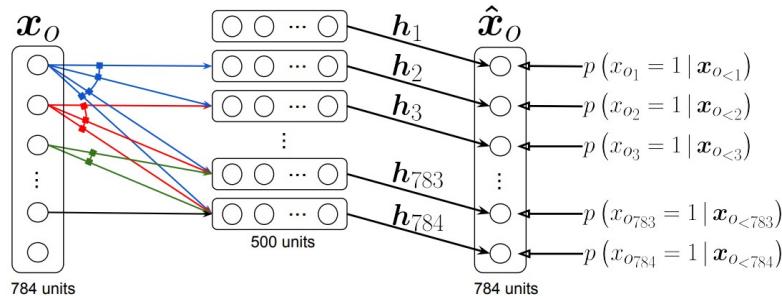


Figure 2.19: NADE architectue
Source: Benigno Uria et al.'s paper [33]

between an output and the value being predicted, or element x_O later in the ordering. Arrows connected together correspond to connections with shared (tied) parameters.

2.4.9 Restricted Boltzmann Machine

A RBM is an undirected graphical model [35, 36, 37, 38] showed in the figure 2.20. As an AE, it encodes the input x in a latent space h where:

$$h_i = \text{sigmoid}(x^T W_i + a) \quad (2.21)$$

where x is the input, h is the vector corresponding to the hidden layer, W are the weights, a is the hidden layer bias vector. This is the encoding phase.

For the decoding, or reconstruction phase, the predicted output by the model is:

$$x_i^* = \text{sigmoid}(h W^T + b) \quad (2.22)$$

where x^* is the reconstructed input, W are the same weights as the forward pass, and b are the observed layer bias vector.

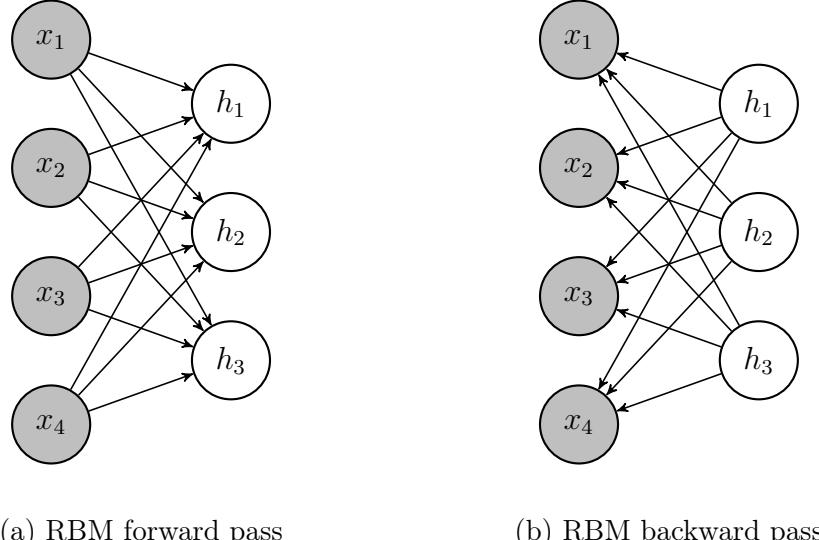


Figure 2.20: RBM architecture

RBM are Energy-based models and a joint configuration (x, h) has an en-

ergy given by:

$$E(x, h) = - \sum_i a_i x_i - \sum_j b_j h_j - \sum_{(i,j)} x_i h_j w_{ij} \quad (2.23)$$

The probability that the network outputs x^* is given by summing over all possible hidden vectors:

$$p(x^*) = \frac{1}{Z} \sum_h e^{-E(x^*, h)} \quad (2.24)$$

where Z is the partition function:

$$Z = \sum_{(x,h)} e^{-E(x,h)} \quad (2.25)$$

Training a RBM consist of implementing a gradient descent of the log-likelihood to increase the value of $\log(p(x))$ for x in the dataset.

$$\frac{\partial \log(p(x))}{\partial w_{ij}}$$

CHAPTER 3

Related works

I will expose in this chapter the works that have already been done about music generation with deep neural networks. In the section 3.1, I will expose the different objectives of some implementations. In the sections 3.2 and 3.3, I will introduce how the music is represented. I will then enumerate in the section 3.4 what architectures have already been used. Finally, in the section 3.5, I will illustrate what are the generation processes used.

3.1 Objectives

In this section, I will describe some examples of what it is possible to do with neural networks to generate music.

As explained in the section 4.1, my Dissertation's goal is to create a single model able to do as many task as possible a musician or a composer could do. For instance:

- Generate music with several part (for instance several instruments)
- Create a accompaniment from a melody
- Create a melody from an accompaniment
- Create a musical part from other musical parts

3.1.1 Generator

First it is possible to create a music melody. It can be either monophonic (only one note played at one time) or polyphonic (several notes can be played at the same time).

Secondly, it is possible to create several musical parts at the same time. Each one of them can be either monophonic or polyphonic. A musical part can be considered as an instrument or voice for a Chorale. The challenge is to create musical parts that work together.

Generation is the most common objective choice among the existing works [8, 39, 40, 41, 42].

3.1.2 Accompaniment

Given a melody, or some musical parts, the goal is to create new musical parts which can be combined with the input and be played together [7, 9].

This is for example the goal of DeepBach from Gaëtan Hadjeres et al.'s paper [7].

Bach Doodle [9] is an online tool developed by Google. Users can create their own melody and have it harmonized by a model in the style of Bach. The figure 3.1 shows the view of this application. The black melody is the melody entered by the user (me) and Bach Doodle created the accompaniment (red, green and blue melodies).

3.2 Music Representation

In this section, I will present different types of inputs the related works use.

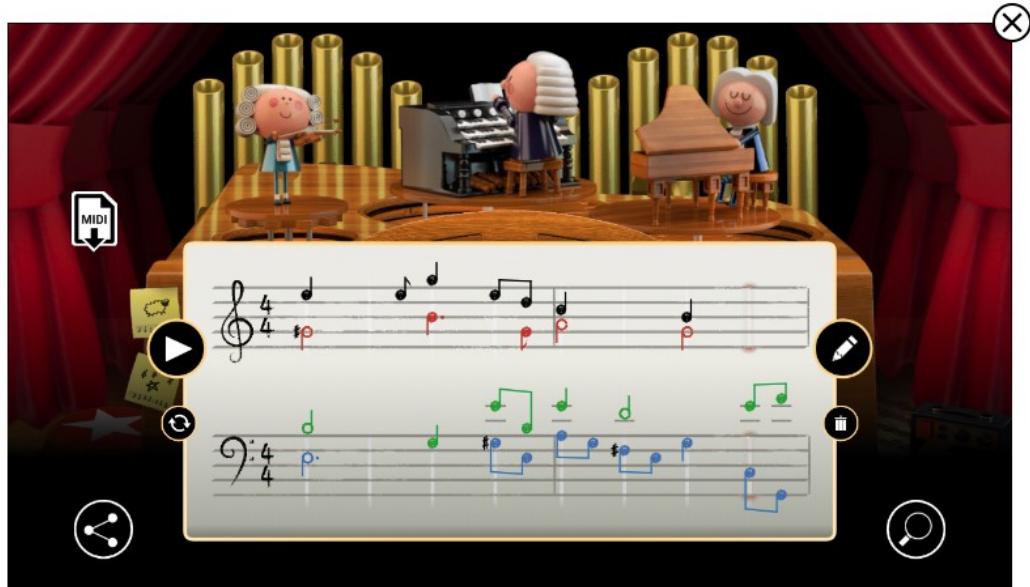


Figure 3.1: Bach Doodle application

Source: BachDoodle

3.2.1 Audio Signal

The first data used to create music is the audio signal. Some works [43, 44, 45, 46, 47, 48] have been done to generate music audio signal. This signal can be represented either by its waveform [43], its Fourier Transform, or its spectrogram (figures 3.2, 3.3).

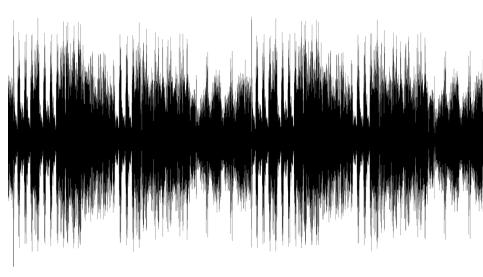


Figure 3.2: Example of an audio wave-
form

Source: Needpix.com

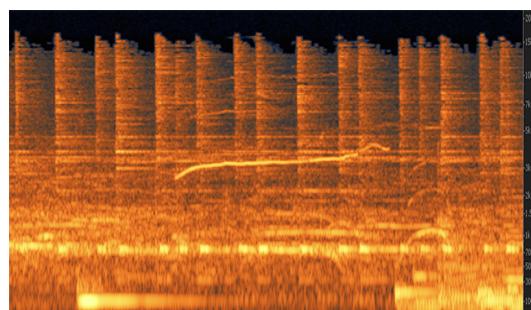


Figure 3.3: Example of audio spectro-
gram

Source: iZotope's tutorial [49]

Using the audio signal allows the model to handle every aspects of the song at the same time (instrument, timber, emotion...), and every song in the same

way. The biggest issue of this method is that the number of points needed to create a waveform is incredibly huge. The usual sampling frequency is $48kHz$. Therefore, the main difficulty is to stay consistent through time.

3.2.2 MIDI Signal

Most of the works done on music generation use the *MIDI* representation (or any other translation of midi like pianoroll or text). [39, 7, 40, 8, 50, 51, 52, 41, 42, 53, 54, 55]

3.2.2.1 Pianoroll

The pianoroll representation can be considered as an image. Thus, usual deep learning methods on images can be applied [40, 39, 41, 42, 45]. The figure 3.4 shows a very naive example of how to convert a pianoroll view to an array.



Figure 3.4: Example of correspondence between a pianoroll representation and an array

Chin-Hua Chuan et al. [39] introduced another representation to help the model to understand relations between notes. They use *Tonnetz* matrix [56] to represent polyphonic music. As explained in their paper, "*Tonnetz is a graphical representation used by music theorists and musicologists in order to study tonality and tonal spaces*".

Instead of encoding notes in a one-dimensional tensor (figure 3.5a), they encode them in a 2-dimensional tensor where the relative positions between two notes is meaningful.

The figure 3.5a illustrates a common form of tonnetz. Each node in the tonnetz network represents one of the 12 pitch classes. The nodes on the same horizontal line follow the circle-of-fifth ordering: the adjacent right neighbor is the perfect-fifth and the adjacent left is the perfect-fourth. Three nodes connected as a triangle in the network form a triad, and the two triangles connected in the vertical direction by sharing a baseline are the parallel major and minor triads. For example, the upside-down triangle filled with diagonal lines in the figure 3.5a is C major triad, and the solid triangle on the top is C minor triad. Note that the size of the network can be expanded boundlessly; therefore, a pitch class can appear in multiple places throughout the network.

In Chuan's paper, they extended this matrix. The figure 3.5b shows an extended tonnetz matrix example. They include in this extended matrix the pitch (octave number) which was missing in the first one.

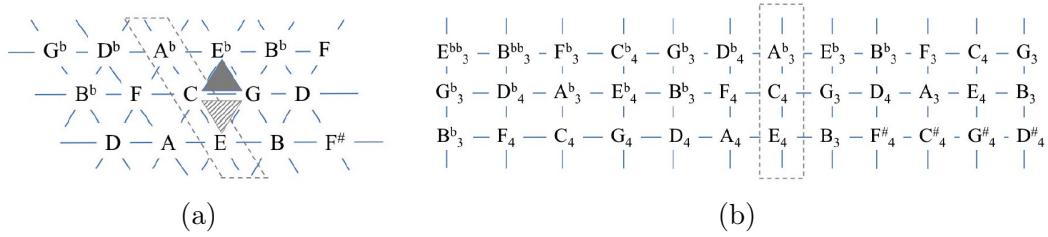


Figure 3.5: (a) tonnetz and (b) the extended tonnetz matrix with pitch register
Source: Ching-Hua Chuan's paper [39].

3.2.2.2 Text

Another approach is to convert the MIDI format into text. [7]



Figure 3.6: Example of correspondence between a pianoroll representation and a text

The figure 3.6 shows a simple example of how it is possible to convert an pianoroll view to a text. However, using text to represent a pianoroll usually forces the framework to work only in a monophonic way. A common choice of the existing works that working with text is to embed the text into a fix-sized vector using a `word2vec` [57, 58, 59, 60, 61] model [8, 52].

Despite the text constraint, Feynman Liang and al. [8] managed to encode polyphonic music in text and use it to correctly train their model: "BachBot".

They quantize time into sixteenth notes (♪). Consecutive frames are separated by a unique delimiter ("|||"). Within each frame, they represent individual notes rather than entire chords. Each frame consists of four (Soprano, Alto, Tenor, and Bass) $\langle \text{Pitch}; \text{Tie} \rangle$ tuples where $\text{Pitch} \in \{0; 1; \dots; 127\}$ represents the MIDI pitch of a note and $\text{Tie} \in \{\text{True}; \text{False}\}$ distinguishes whether a note is tied with a note at the same pitch from the previous frame or is articulated at the current timestep. They *order notes within a frame in descending MIDI pitch*. The figure 3.7 shows the encoding process.

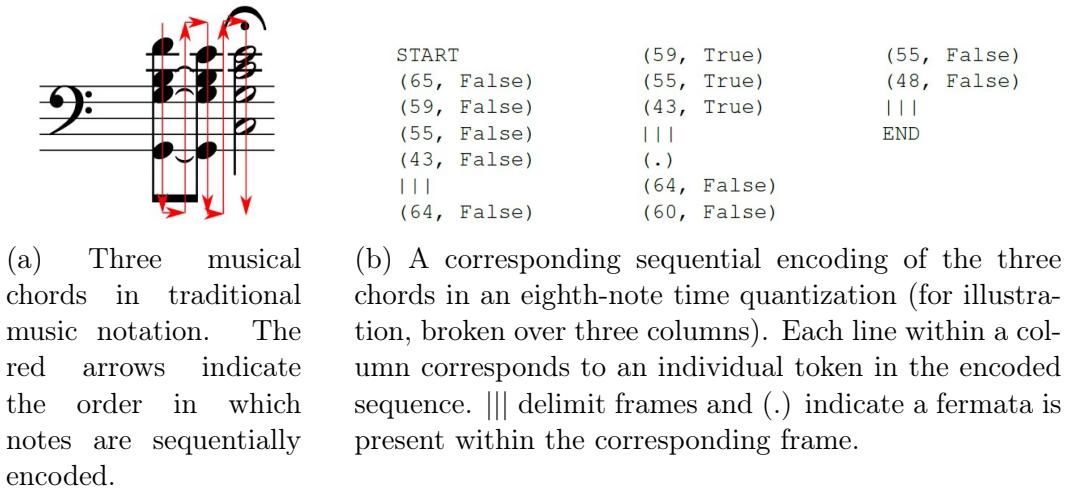


Figure 3.7: BachBot's example encoding of three musical chords ending with a fermata ("pause") chord

Source: Feynman Lian's paper [8].

3.2.2.3 Chords

This is the last approach I will describe in this paper. As the same way as Jazz music, it is possible to simplify the description of a musical piece by writing down the chords.

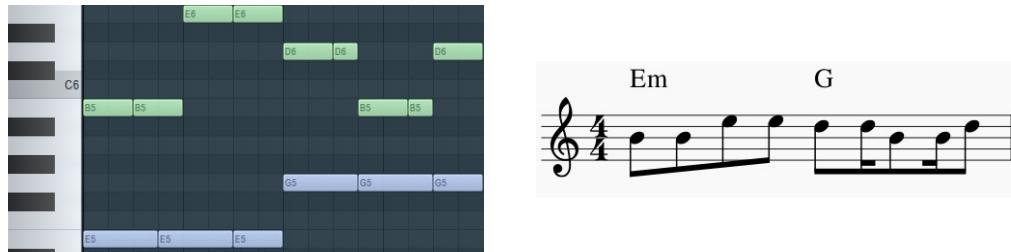


Figure 3.8: Example of correspondence between a pianoroll and its chords representation

For instance, in the figure 3.8, the bass line can be simplified by considering the chords. The chords are then written and be considered as notes.

3.3 Encoding

3.3.1 Features

It is possible to extract features from the data to help the neural network to understand how music works, like the chords, the current key signature...

- The first option is to do it manually and give the features as an input of the model
- The second option is to let the model learn it by itself. This is the choice I have made for this project.

3.3.1.1 Metadata

The features that can be passed as an input to the model can also be metadata like the time signature or the key of the music...

Deep Bach from Gaëtan Hadjeres et al.’s paper [7] consider the fermata symbol (figure 3.9) and the beat subdivision number (an integer between 1 and 4).



Figure 3.9: Fermata symbol
Source: publicdomainvector

In this project I chose to not include any metadata in the model because of the difficulty to find or reconstruct them for the available dataset. However, I gives to the model the information about the beats number and subdivisions by considering the entire measure as one input.

Another way to help the model by giving it information without actually passing metadata is to normalize the data. For example, BachBot [8] and other works [39, 41] transpose all their dataset in either C major or A minor key.

For the same reason as not passing metadata, I didn’t normalize the data before training my models. However, because I was having poor results, I decided afterward to transpose all the song in either the C major scale or the A minor scale.

3.3.2 Tensor encoding

It is possible to extract two different methods on how to encode the tensor which will be given to the neural network

The first method is the *value encoding*. It means that the values in the input and/or output tensor are actually meaningful. For example, it can be an integer which is the pitch of a note.

The second method is called *item encoding*. Instead of having the number of the pitch in the tensor (let’s say 60), it will be a very sparse tensor with a bunch

on 0 and a 1 at the position 60 in the pitch dimension. This values stored in the tensors can be considered as *activation* values. This is the retained method in my Dissertation.

3.4 Architectures

In this section, I will present different architectures that have already been used for music generation. It will expose the architecture of my project in the section 4.3.

3.4.1 RBM

The RBM architecture is explained in the section 2.4.9.

Nicolas Boulanger-Lewandowski et al. [41] use an RBM based model RTRBM [62, 63] and created their own architecture called RNN-RBM. They applied their model on different MIDI dataset to create polyphonic music.

Stefan Lattner et al. [42] use another RBM based model called C-RBM [64, 65]. They apply this model on pianoroll images. They also implement a different constraints like their self-similarity constraint to specify a repetition structure (e.g. AABA) in the general music piece.

3.4.2 NADE

The NADE architecture is explained in the section 2.4.8.

And as an example, Cheng-Zhi Anna Huang et al. [40] use an orderless NADE [34] to generate music. They introduce COCONET, a deep convolutional model to reconstruct partial scores. It consists of a corruption process that masks out a subset $x_{\neg C}$ of variables, followed by a process that independently resamples variables x_i (with $i \notin C$) according to the distribution $p_\theta(x_i|x_C)$ emitted by the model with parameters θ .

The figure 3 illustrates how the process works. At each step, a random subset of notes is removed, and the model is asked to infer their values. New values are sampled from the probability distribution put out by the model, and the process is repeated.

Google also created Bach Doodle [9], an online tool re-implementing CONET in Tensorflow.js [66] which harmonizes a melody given by the user.

3.4.3 AE

The AutoEncoder architecture is explained in the section 2.4.2.

This is the most common architecture used for music generation at the time I am writing this report. The AutoEncoder can be recurrent [8, 39, 7, 46, 47] as Feynman Liang et al.’s BachBot [8] or not [44].

For instance, BachBot [8] and DeepBach [7] use an encoder/decoder architecture with LSMT layers on the latent space. [39]

Soroush Mehri et al. [47] or Nal Kalchbrenner et al. [46] have created different AE architecture to synthesize audio waveform and can be applied to music sounds.

3.4.4 VAE

The Variational AutoEncoder architecture is explained in the section 2.4.3.

Sander Dieleman et al. [44] use a VAE (or more precisely a VQ-VAE [67] or a AMAE [44]) to encode audio waveform and reconstruct it with a stylistic consistency across tens of seconds.

3.4.5 RMVAE

Since I created this new architecture, there is no existing work using this one. My project is, for now, the only work done using a Recurrent Multimodal

Variational AutoEncoder to generate music.

The RMVAE's architecture is explained in the section 4.3

3.4.6 GAN

The Generative Adversarial Network Architecture is explained in the section 2.4.4

For instance, the model WaveGan from Chris Donahue et al. [45] use a GAN architecture to synthesize audio waveform in several domains including drums and piano.

Gino Brunner et al. [54] use CycleGAN [68] to create a model able to apply a style transfer on musical pieces (between jazz, pop and classical style).

3.4.7 Transformers

The Transformer's architecture is explained in the section 2.4.6. Some works have already been on music with Transformers as, for instance, Cheng-Zhi Anna Huang et al. [69] and Kristy Choi et al. [70].

Huang et al. introduce the Music Transformer, a model able to generate consistent samples longer than samples generated by a RNN architecture. They achieve this performance by using a Transformer with relative attention (equation 3.1).

$$\text{RelativeAttention} = \text{Softmax}\left(\frac{QK^T + S^{rel}}{\sqrt{D_h}}\right)V \quad (3.1)$$

3.5 Generation Process

In this section, I will enumerate different techniques to generate music from different model architectures.

3.5.1 Sampling

The sampling process can either be done with a VAE, or an GAN architecture [45].

A VAE keeps its latent space distribution as close as the standard normal distribution $\mathcal{N}(0, 1)$. Thus, by giving a random vector sampled from a standard normal distribution $\mathcal{N}(0, 1)$, the decoder will construct an output similar to the dataset the model trained on.

The generator of a GAN has been trained to generated consistent data from noise. Then, the process generation for a GAN is to give a random input from a standard normal distribution to generator.

3.5.2 Input Manipulation

Generating music by manipulating the input is typically how style transfer algorithms work [71, 72, 73]. The input is considered as a variable which is updated to minimize a loss function constructed from a content target and a style target. The content similarities between the input and the content target is reduced through the iterations. In the same way, the style similarities between the style target and the input will be minimized.

This process works well for images to create a new image combining content and style from 2 different images. People tried to apply the same methods on music [74, 75, 54, 48], but as Shuqi et al. [76] say, music is more complex and has several level of style (timber, performance and composition). Therefor, creating a music style transfer algorithm is more complicated than for images and the results are currently not as good.

3.5.3 Feed forward and RNN architecture

This is the easiest and most common generation process through all the works [8, 39, 40, 55], and this is the one I chose for this project. An input is given to the model and the model returns an output. The figure 3.10 illustrates this process. This output can be the next frame/beat/measure (in my case, a measure) of the music.

In my case, the model returns the next measure of the music.

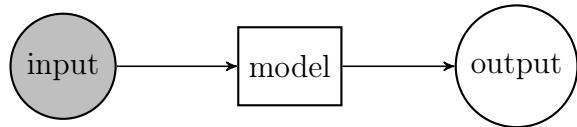


Figure 3.10: Feed forward generation process

3.5.3.1 DeepBach

To give an illustration, DeepBach [7] uses a Markov Chain Monte Carlo algorithm to re-harmonize a song and transform it. From the existing song, it re-predicts every notes one by one. The algorithm is describe in the figure 3.11.

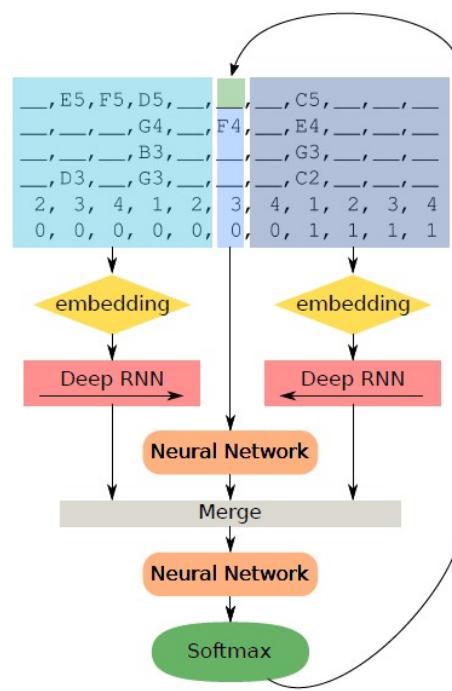


Figure 3.11: Graphical representation of DeepBach’s neural network architecture for the soprano prediction

Source: DeepBach paper [7]

CHAPTER 4

Contribution: Multi-Modal Music Generation

I will explain in this chapter what have been my original works for my Dissertation. In the section 4.1, I will explain what I have been aiming to do. In the section 4.2, I will explain how I decided to represent the music and how I decided to work with it. In the section 4.3, I will describe my novel architecture (RMVAE) that I have created for my Dissertation. And finally, in the section 4.4, I'll illustrate how I gave some prior knowledge to the network with custom loss functions.

4.1 Objectives

As a musician, I wanted to create a model architecture who could copy the way I think about music. The challenge is to use only one trained model to be able to:

- Generate music with several musical parts (meaning several instruments)
- Create a accompaniment from a melody
- Create a melody from a accompaniment
- Create a musical part from other musical parts

The created model can also handle missing data (for example a missing measure from an instrument).

To summarize, the objective is to create a unique trained model which can generate or arrange a musical piece whatever is already present or missing.

4.2 Data representation

This project works with MIDI data. These music representation is explained in the section 2.1.2.

The shortest beat division is a quarter of a beat (sixteenth note: ♩)

The considered music are binary and with 4 beats per measure.

Because I think that a measure can be consider as an entire object (with its rhythm, its chords ...), I chose to divide music by measure. Thus, a time step for the neural network is the representation of an entire measure. The shape of a tensor representing a measure will be (16, 128, channels):

- 16 is the number of sixteenth notes (♩) is a measure.
- 128 is the number of different MIDI notes possible (from 0 to 127).
- `channels` $\in \{1, 2\}$ is explained in the sections 4.2.1 and 4.2.2.

The number of instruments is fixed and are different inputs of the neural network. Hence, for one instrument, its associated tensor has a shape of (`nb_measures`, 16, 128, `channels`). `nb_measures` is the number of measures considered.

4.2.1 Polyphonic music

For polyphonic music, the number of `channels` it 2.

The first channel is called the *activation* channel. The value is either 1 for a played note or 0 for a non-played note.

The second channel is called the *duration* channel. The value is an integer corresponding of "*how many sixteenth notes (♩) it lasts*". The table 4.1 shows the correspondence between the value of the duration channel and the musical length representation.

Duration value	Musical length
1	♪
2	♪
3	♪.
4	♩
5	$\text{♩} \text{♪}$
6	♩.
7	$\text{♩} \text{♪.}$
8	♩
9	$\text{♩} \text{♪}$
10	$\text{♩} \text{♪}$
11	$\text{♩} \text{♪.}$
12	♩.
13	$\text{♩} \text{♪}$
14	$\text{♩} \text{♪}$
15	$\text{♩} \text{♪.}$
16	♩

Table 4.1: Correspondence between duration value and musical length

If a note is not activated (activation channel = 0), then the duration channel is set to 0 too.

$$\text{channel}_{\text{activation}} = 0 \iff \text{channel}_{\text{duration}} = 0 \quad (4.1)$$

4.2.2 Monophonic Music

The goal was to reduce the memory space and simplify the model for monophonic music. Since monophonic music means there is a maximum of one note played simultaneously at one time t , I chose to not consider the rests and consider that every notes last until the next note is played.

Thus, I deleted the duration channel. Now, the number of channels = 1.

An *additional note* is inserted which is the $note_{\text{continue}}$. The tensor shape of a step is now (16, 128 + 1, 1). When $note_{\text{continue}}$ is set to 1, it means there is no new note to be played. And when $note_{\text{continue}}$ is set to 0, it means a new notes has to be played.

$$note_{continue} = 1 \implies \forall note \in notes_{[0:128]}, note = 0 \quad (4.2)$$

$$note_{continue} = 0 \implies \exists! note \in notes_{[0:128]}, note = 1 \quad (4.3)$$

And since I consider in this part only monophonic music, there is only one note (included $note_{continue}$ per each time division)

$$\exists! note \in notes_{[0:129]}, note = 1 \quad (4.4)$$

4.3 RMVAE Architecture

I have created a new architecture which is able to handle all the objectives defined in the section 4.1.

To do so, I continued the work of Mike We et al. [1] and their Multi-modal Variational AutoEncoder (MVAE) (see section 2.4.7) and I simplified the work of Tan Zhi-Xuan et al. [12] and their Multimodal Deep Markov Models (MDMM).

4.3.1 Global Architecture

I use the capability of the MVAE to reconstruct data and handle several modalities at the same time with the help of the product of expert operation. But because the MVAE doesn't handle time across the data, it had to be transformed. And that is basically what does the MDMM. However, the training process of a MDMM is quite complicated and slow. Because of this, I came up with a novel architecture that I call RMVAE (Recurrent Multimodal Variational AutoEncoder). This architecture is described in the figure 4.1.

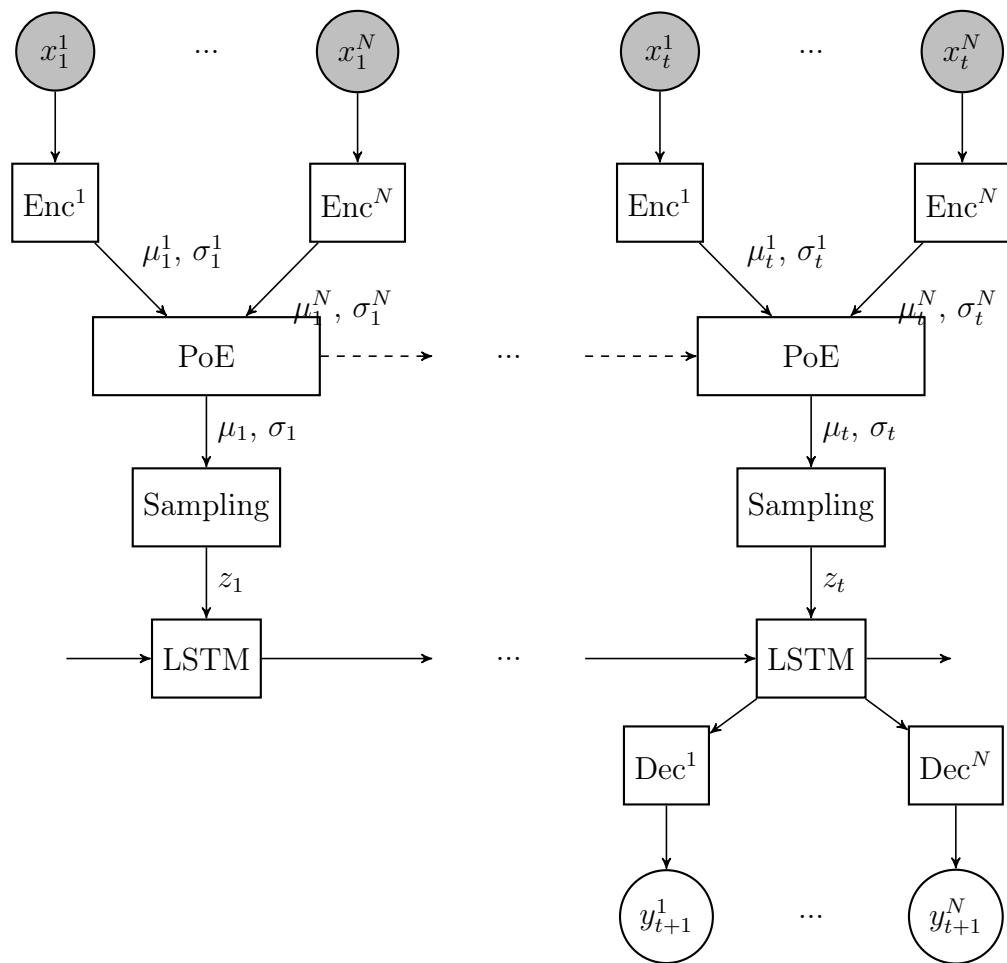


Figure 4.1: Architecture of the RMVAE

4.3.2 Encoder

Each instruments (each modalities) has its own encoder. As explained in the section 4.2, the input tensor for one instrument has the following shape: `(nb_steps, 16, 128, channels)`.

I tried 2 different architectures of encoders and decoder. The first architecture (section 4.3.2.1) considers a measure as an image and apply image operations on it with convolution layers. The second architecture (section 4.3.2.2) considers a measure as a time series of notes and use an RNN architecture to encode it.

4.3.2.1 Convolutional encoder

For a given instrument i , and a given step t , the tensor (with a shape of `(16, 128, channels)`) can be considered as a pianoroll image, hence usual imaging operations can be applied.

The encoder Enc^i process each steps of a the instrument i . And encoder is composed of:

1. Convolutional layers and pooling layers
2. Fully Connected layers

The general architecture of the encoder is showed in the figure 4.2.

The filter sizes of the convolutional layers are `(5, 5)`. I chose this dimension so the receptive field of the first layer is a third major interval and an entire beat.

4.3.2.2 Recurrent encoder

For a given instrument i , and a given measure m , the tensor has a shape of `(16, 128, channels)`. The first dimensions (`(16,)`) correspond of a

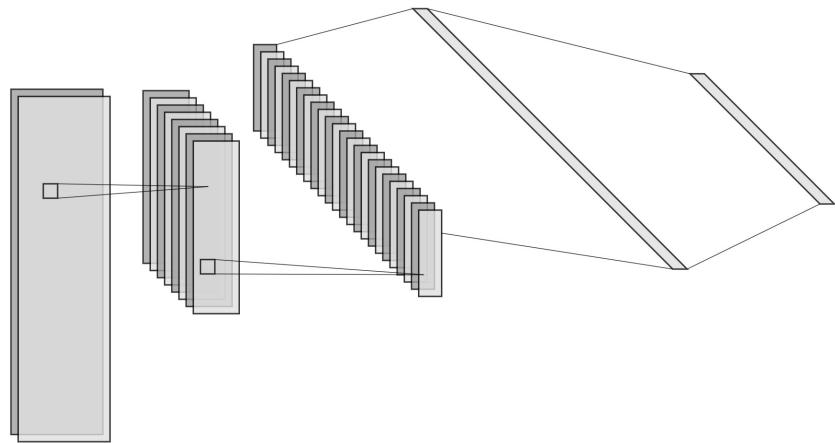


Figure 4.2: RMVAE encoder architecture

sixteenth note (♪) in time. Therefore, I created an encoder using bidirectional LSTM cells. This architecture is shown in the figure 4.3.

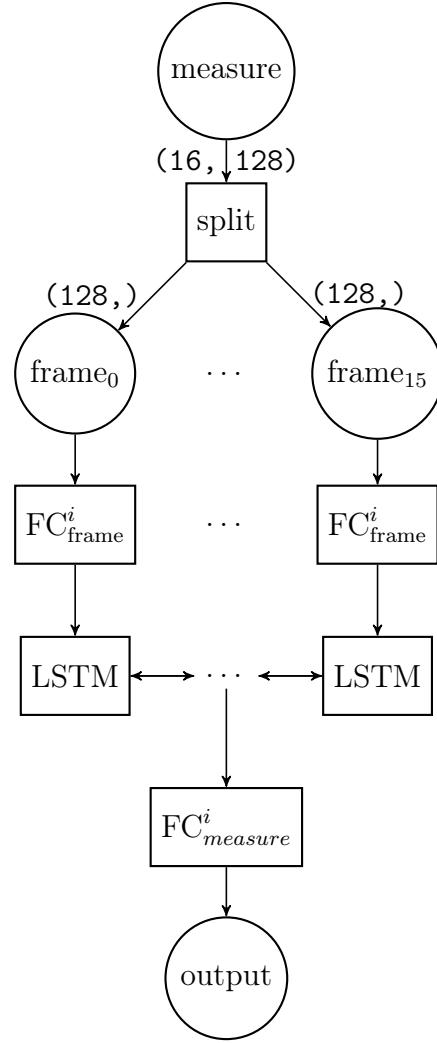


Figure 4.3: Measure Encoder using bidirectional LSTM
Encoder for a measure of an instrument i . A measure has a shape of $(16, 128)$ and a frame as a shape of $(128,)$.

4.3.3 PoE

For a given instrument i and a given step t , the output of the encoder Enc^i will then go through 2 different FC layers fc_μ^i and fc_σ^i which will return the mean and variance of the encoded normal distribution (figure 4.4).

Then, for a given step t , all the distribution representation go through the PoE layer to be combined and sampled as shown in the figure 2.18.

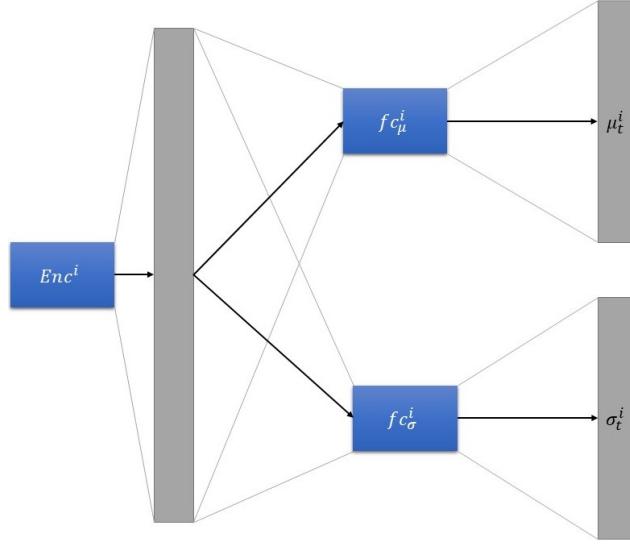


Figure 4.4: RMVAE PoE Fully Connected layers

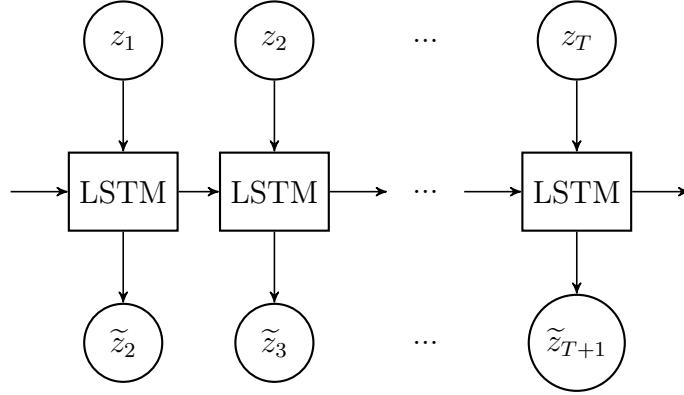


Figure 4.5: RMVAE LSTM layer

4.3.4 Recurrent layers

I have now all the latent representations for every steps : $\{z_1, \dots, z_T\}$.

4.3.4.1 LSTM cells

To learn, how this latent space evolves through time, RMVAE uses recurrent layers with LSTM cells (figures 2.14 , ??). This is shown in the figure 4.5.

4.3.4.2 RPoE

To try to catch the time dependency as good as possible, I implemented a new layer architecture. I call it Recurrent Product of Experts (RPoE).

The result of the Product of Experts at the step t is given as a modality for the Product of Experts at the step $t + 1$.

The architecture is displayed in the figure 4.6.

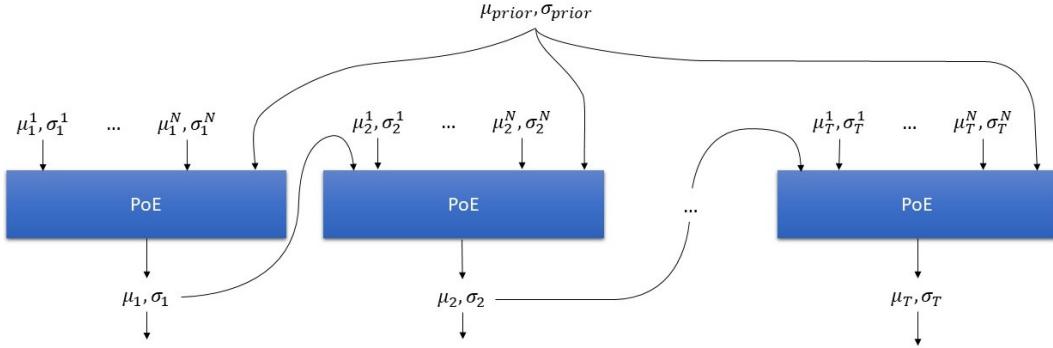


Figure 4.6: RPoE architecture

4.3.5 Decoder

Each instruments (each modalities) has its own decoder. The decoder Dec^i associated to the instrument i will, for a given step t , reconstruct back the output with the same shape as the input ((16, 128, channels)).

4.3.5.1 Convolutional decoder

A decoder is composed of:

1. Fully Connected layers
2. Transposed Convolutional layers

The dimensions of the layers of the encoder Enc^i are the same as the dimensions of the decoder Dec^i .

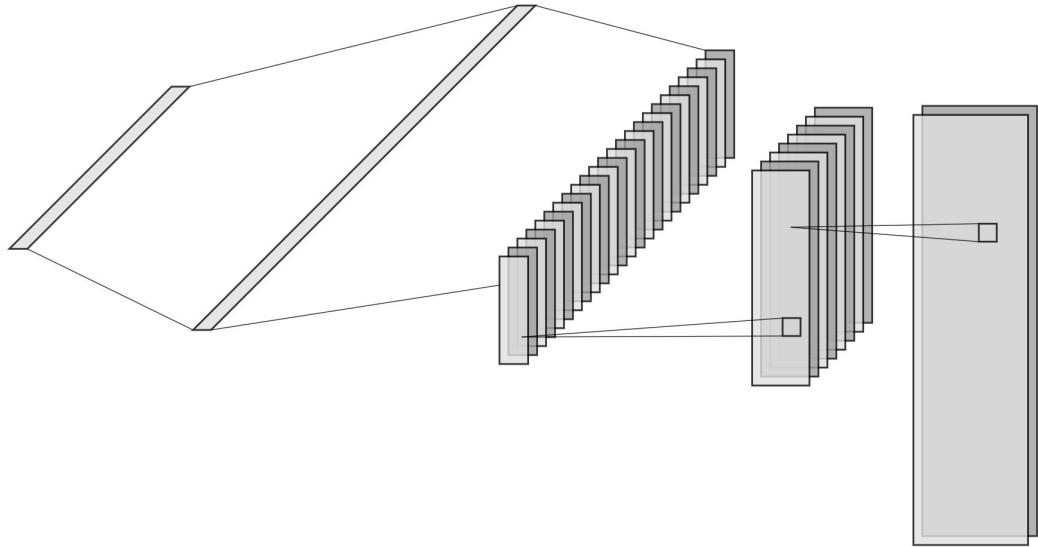


Figure 4.7: RMVAE decoder architecture

The decoder global architecture is showed in the figure 4.7.

This decoder was associated with the convolutional encoder form the section 4.3.2.1.

4.3.5.2 Recurrent decoder

To decode the latent space encoded with the recurrent encoder (from the section 4.3.2.2), I use a recurrent architecture composed with bidirectional LSTM cells. This architecture is illustrated in the figure 4.8

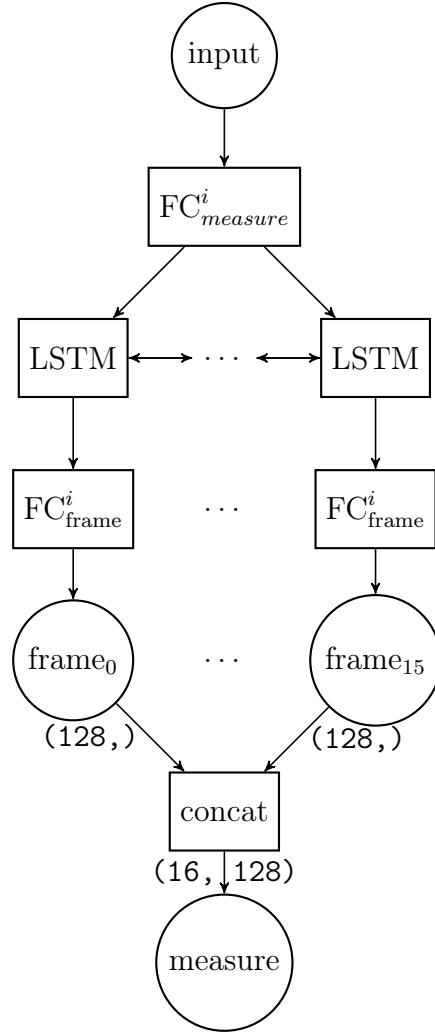


Figure 4.8: Measure Decoder using bidirectional LSTM
 Decoder for a measure of an instrument i . A measure has a shape of $(16, 128)$ and a frame as a shape of $(128,)$.

4.3.6 Last layer

The last layer of the NN is a Fully Connected layer with a well chosen activation function.

4.3.6.1 Polyphonic Music

For polyphonic music (section 4.2.1), the activation function is a *sigmoid* function for the channel 1 which represents the activation of the notes. The

activation function for the channel 2 is a *ReLU* function.

4.3.6.2 Monophonic Music

For monophonic music (section 4.2.2), the activation function is a *sigmoid* function for the *additional note* (index 128: $note_{continue}$) because it is an activation note indicating if it wants to continue the previous note or play a new one. And for the other 128 notes, the activation function is a *softmax* function since only one note can be played at the same time.

At the beginning of my project, I was considering the $note_{continue}$ as a *normal note* and simply applying a softmax across all the notes. But the result was disappointing: the frequency of $note_{continue}$ in the dataset was too high. The network wasn't generating anything because the $note_{continue}$ had the highest probability.

4.4 Loss Function

For this project, I created new loss functions to help the network to understand how music works. As a musician, I asked myself how I would proceed if I was given the same task as the NN (generation of music, accompaniment...). This is how I got the idea to create those new loss functions.

In a song, when a musician is playing, it is not possible for him play every one of the 12 notes whenever he wants. A music usually follows a pattern, a chord progression, and he has to follow it. From the chords played, the scale, some notes sound better to the human ears than others. But there is no "*ground truth*" for a solo part for example. Every solo can be considered as a "*ground truth*" if it sounds nice. And I wanted my model to be able to *improvise*, generate new music.

I got the idea of helping the model to do *acceptable mistakes* and restraint

it do to *unacceptable mistakes*. In other word, during the training part, if the model doesn't hit the note it should have hit, I don't want to *punish* it too much with a high loss value if the note sounds actually nice with the music. On the other side, I want to *punish* it if the note it hits is completely wrong with a higher loss value.

To summarize what I previously just said, during the training part, if the model hits a note which sounds nice, a reward is given by adding a negative number to the loss. If the hit note doesn't sound nice, then a penalty is given by adding a positive number to the loss. The algorithm 1 describes the process.

Algorithm 1 Add a Reward or a penalty to the generated note

Input: *noteTruth, notePredict*

Output: *loss*

```

1: reward > 0, penalty > 0
2: function LOSS(noteTruth, notePredict)
3:   loss  $\leftarrow$  commonLoss(noteTruth, notePredict)
4:   if soundsGood(notePredict) then
5:     loss  $\leftarrow$  loss - reward
6:   else
7:     loss  $\leftarrow$  loss + penalty
8:   end if
9:   return loss
10: end function

```

The difficulty is to create the function `soundsGood` from the algorithm 1. The following sections describe the idea I have had to know whether a note sounds nice or not.

4.4.1 Scale

I call this loss function *Scale* because the idea is to reconstruct the *local scale*.

This loss function takes as inputs all the instruments output:

$$\text{Scale}(\text{truth}, \text{predict})$$

where the shape of the input tensors is `(nb_instruments, 16, 128)` for both which represent the activated notes (where there is a 1) of every instruments for the next measure. The figure 4.9 shows the operations of this function.

The idea is to take all the played notes for all the instruments in the truth tensor. Because the same note from different octaves should be considered as the same note, I apply a segment sum to get in the end a tensor of shape `(12,)` which corresponds to the 12 notes. The details of the segment sum operation are explained in the appendix .2.

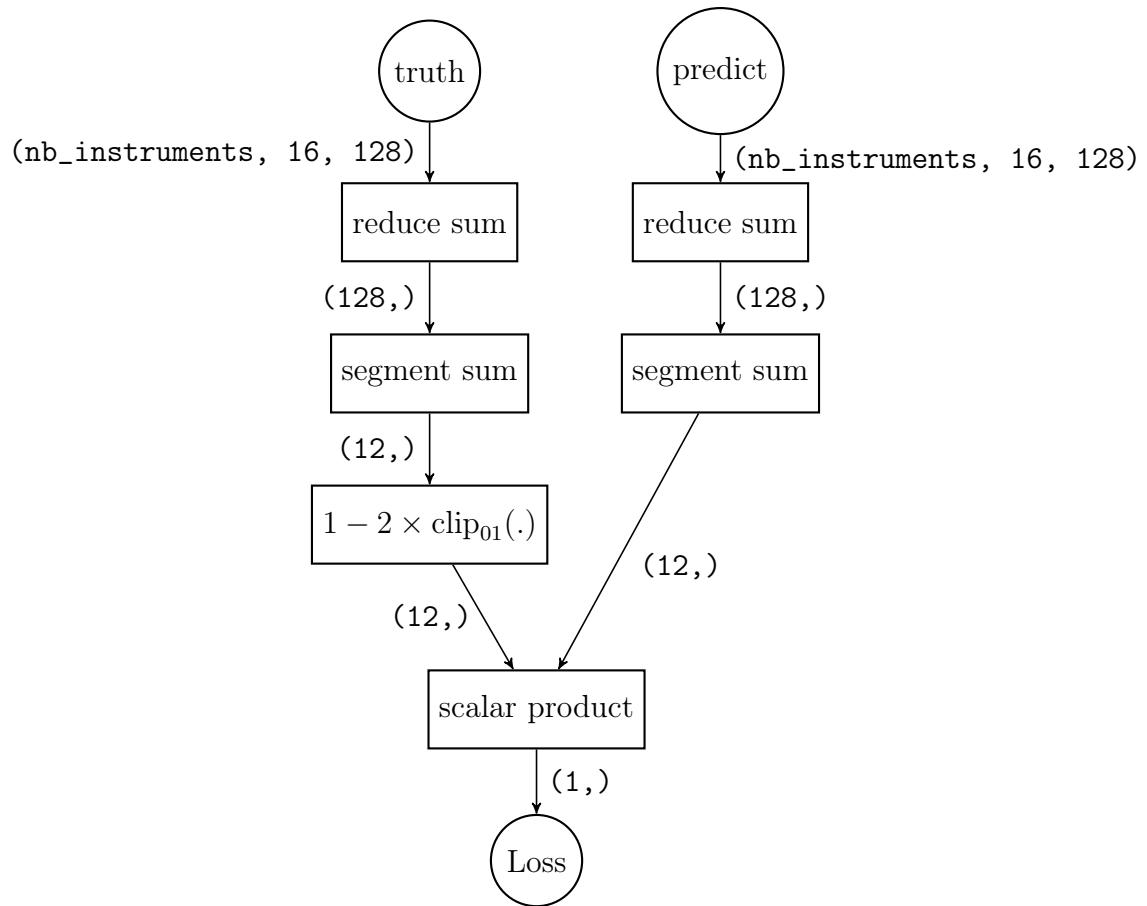


Figure 4.9: Scale Loss

4.4.2 Rhythm

The idea behind the Rhythm loss is that we want to preserve the rhythm of the music. Indeed, the rhythm is sometimes important and consistence is need through the musical piece. Thus, to preserve it, every time the model plays a note that is not played at the same time as an instrument in the ground truth, it gets a penalty. Conversely, every time the model plays a note at the same time as another one in the ground truth, it gets a reward.

The implementation showed in the figure 4.10 is very similar to the Scale implementation.

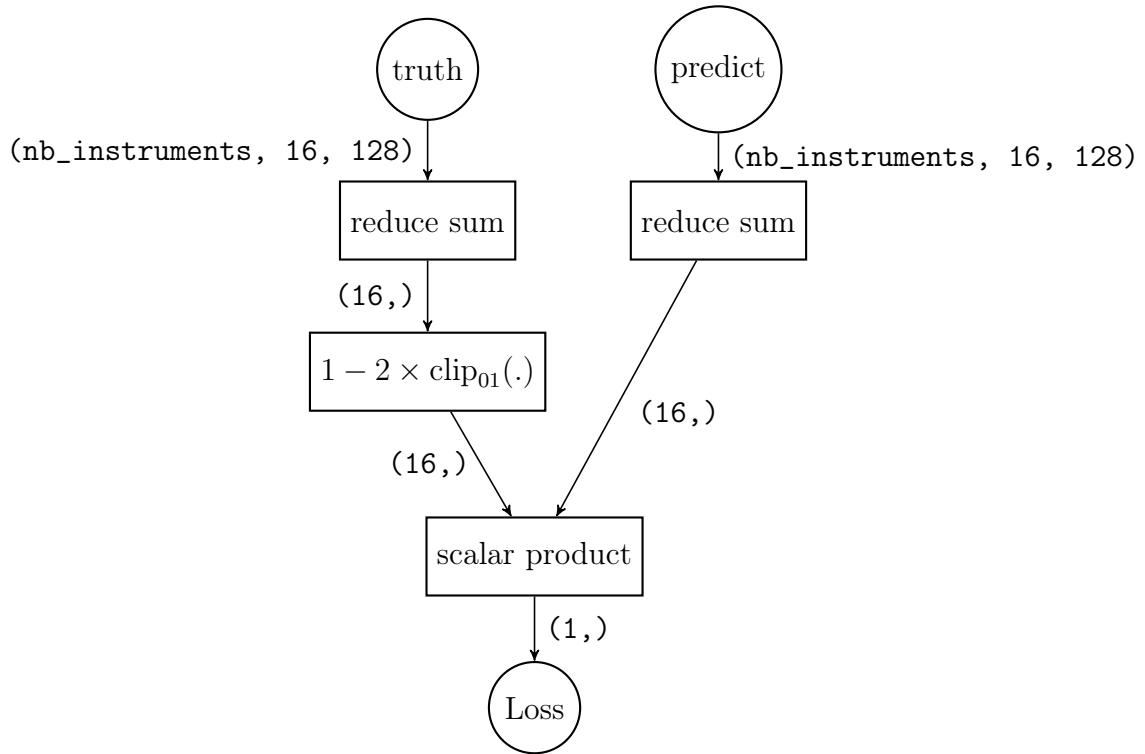


Figure 4.10: Rhythm Loss

4.4.3 Harmony

I explained in the section 2.2.2 that some notes will sound smooth together because of their common harmonics. On the other side, some notes won't be elegant when played together because of the resonance problem between some of their harmonics.

This is something every composer has in mind when he creates a second musical part which will harmonize the first one. He will keep the second voice in the scale and he will keep an acceptable musical interval between the notes of the 2 melodic parts.

The figure 4.11 shows the *acceptable* and *non-acceptable* musical intervals for the note *A*.

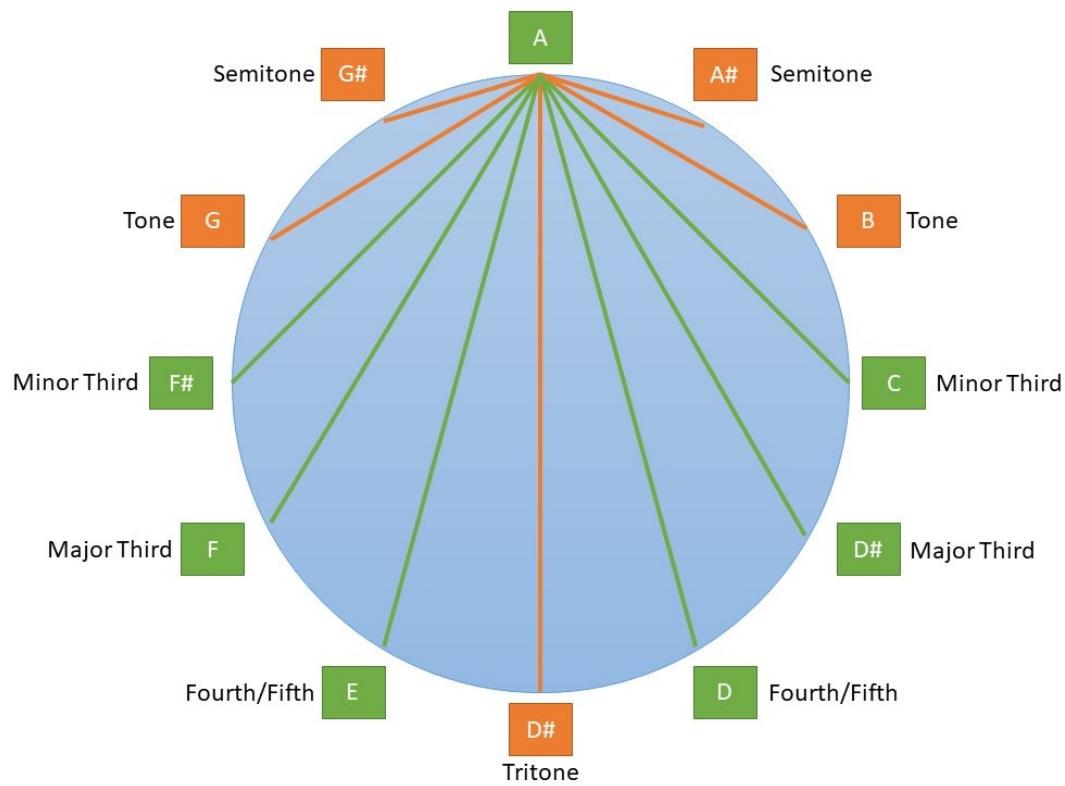


Figure 4.11: Harmony Circle for *A*

From this figure, I realized there are actually 3 musical intervals to avoid:

- The *semitone* interval

- The *tone* interval
- The *tritone* interval. Actually, this interval was named "*Diabolus in musica*" ("Devil in the music") and was forbidden by the church.

To prevent the model to generate such intervals, I created a loss function which gives a penalty every time there is one of this intervals. To do so, I created a sub-function harmony_n which penalizes the presence of the n^{th} interval (counted in semitone).

The operations of the cost function harmony and harmony_n are showed in the figures 4.12 and 4.13. The roll_n operation is explained in the appendix .3.

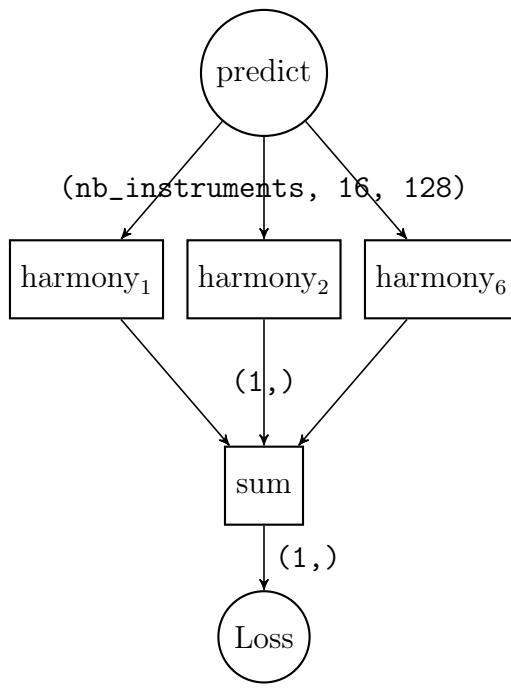


Figure 4.12: Harmony Loss

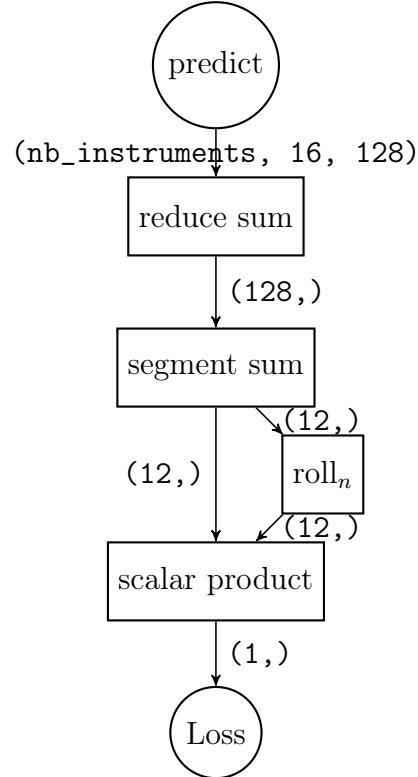


Figure 4.13: harmony_n Loss

CHAPTER 5

Results

In this chapter I will share the results I got. In the section ??, I will expose the results I obtained with a convolutional encoder and decoder architecture. In the section ??, I will share the results I obtained with a recurrent encoder and decoder architecture. To be fair, the results are quite poor and disappointing.

5.1 Generated Samples

For a CNN encoder and decoder, the results weren't matching my expectation. To illustrate it with an image, the figure 5.1 shows a generated sample by a RMVAE.

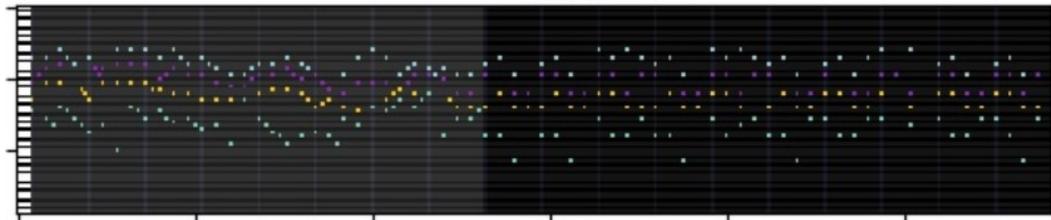


Figure 5.1: Generated pianoroll by the RMVAE
The grey part is the seed given to the model and the black part is the generated part

As we can see on the figure 5.1, the global shape of the the generated part is very different from the seed part. The main reason is a lack of rhythm and variation. Most of the generated notes are quarter notes (♩), and the model generates always the same notes.

However, even though the generated sample have huge lack of rhythm and variation, it is not wrong. It is one tune, consistent with the seed, and the four voices work well together.

5.2 Activation function of the $note_{continue}$ for monophonic music

As explained in the section 4.3.6.2, for monophonic music, I consider an extra note ($note_{continue}$) which indicates that the previous note is sustained. The shape for a tensor describing a measure is (16, 128 + 1).

At the beginning, I considered the $note_{continue}$ as a *normal* note, and apply a softmax activation through the 128 + 1 notes for each of the 16 frames. Unfortunately, it resulted with a too high probability for the $note_{continue}$ and the model wasn't generated anything. The figure 5.2 shows a generated sample. As explained, the generated part (grey part) contains only few notes.

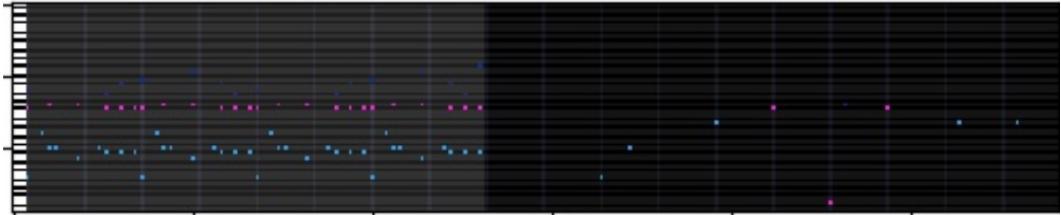


Figure 5.2: Generated pianoroll by the RMVAE with a softmax activation
The grey part is the seed given to the model and the black part is the generated part

To counter this silence problem I changed the activation function of the notes:

- The $note_{continue}$ activation is a sigmoid activation. If the value is ≥ 0.5 , then no new note is played and the previous note is sustained. If the value is < 0.5 , then a new note is played

- The 128 remaining notes are still going through a softmax activation to choose what is the note to be played (if $note_{continue} < 0.5$).

By doing so, I helped the model to generate new notes and get results like the figure 5.1. However, the lack of rhythm is obvious and all the notes have length of one beat (\bullet).

5.2.1 LSTM Encoder and Decoder

With the recurrent architecture (explained in sections 4.3.2.2 and 4.3.5.2), I obtained the same behavior as with convolutional architecture (sections 4.3.2.1 and 4.3.5.1).

The figures 5.3 and 5.4 illustrate it. We can see on the pianoroll views that the generation is more consistent when the activation function on the note $note_{continue}$ is the sigmoid function.

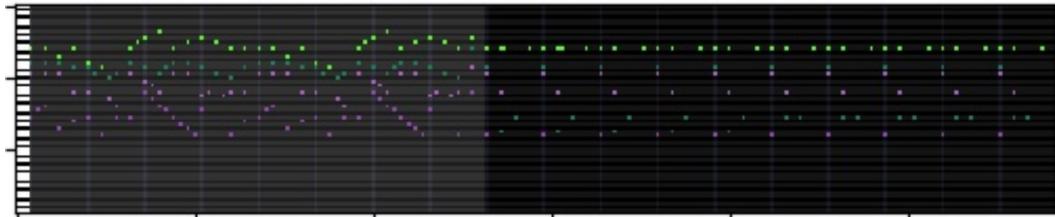


Figure 5.3: Generated pianoroll by the RMVAE with recurrent architecture with softmax activation

The grey part is the seed given to the model and the black part is the generated part

However, the issues stay the same. There is absolutely no rhythm, all the notes are quarter notes (\bullet) and there is no variation. For instance, in the figure 5.4, the first voice (the pink one at the top) is only playing 3 different notes.

5.3 Tasks

I created several scripts to use a single trained model for several tasks. Those tasks are:

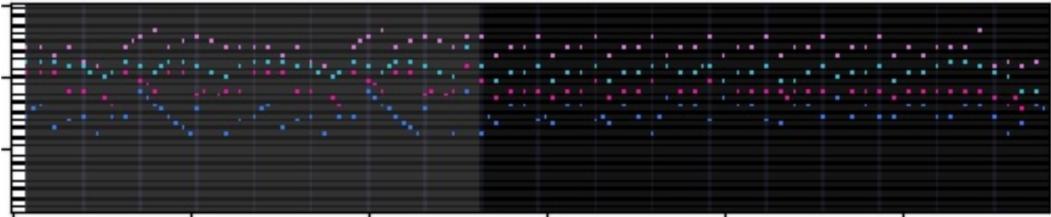


Figure 5.4: Generated pianoroll by the RMVAE with recurrent architecture with softmax and sigmoid activation

The grey part is the seed given to the model and the black part is the generated part

- Generate (section 5.3.1)
- Fill (section 5.3.2)
- Redo (section 5.3.3)

5.3.1 Generate

This is the simplest task I have created for the model. The figure 5.5 is the result of it.

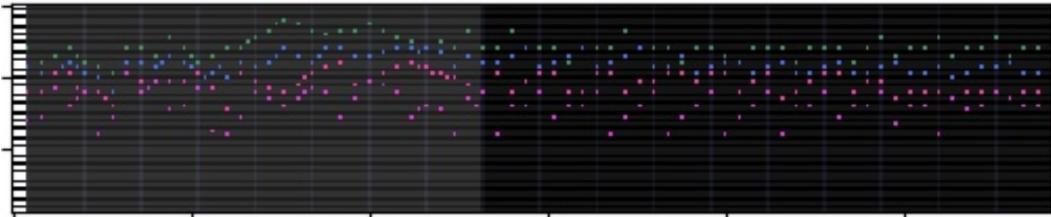


Figure 5.5: Task *Generate*

The model is given a seed (grey part of the figure 5.5). From this seed it generates the next measure. It will then consider the seed and the previously generated measure to generate the next one. And so one...

The algorithm 2 describes this task.

The RMVAE takes a mask as an input with a shape of `(nb_instruments, nb_measure)`. Therefore, to ignore one or several instrument(s), the only thing

Algorithm 2 Generate function

Input: $seed$ (list of measures), n (number of generated measures)
Output: $generation$ (list of measures)

```
1: function GENERATE( $seed, n$ )
2:    $seed\_length \leftarrow \text{len}(seed)$ 
3:    $generation \leftarrow seed$ 
4:   for  $k \leftarrow 1$  to  $n$  do
5:      $input \leftarrow generation[-seed\_length :]$ 
6:      $output \leftarrow \text{model.predict}(input)$ 
7:      $generation.append(output)$ 
8:   end for
9:   return  $generation$ 
10: end function
```

to do is to put 0 at the positions $mask[i, :]$ for every instruments i we want to ignore.

However, as you can see in the figures 5.5, the generated sample is poor in variation and complexity.

5.3.2 Fill

The *Fill* task creates a musical part (an instrument voice) from the others. As an example, the figures 5.6 and 5.7 illustrate this task.

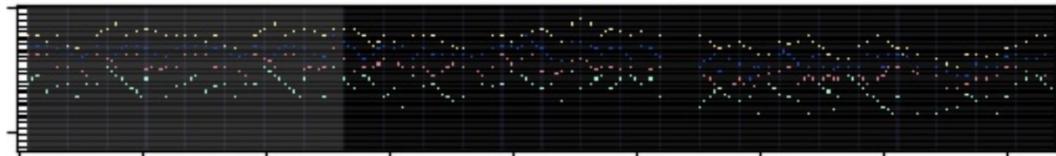


Figure 5.6: Song given to the model for the *Fill* task

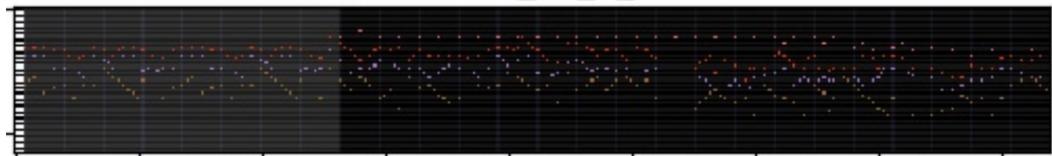


Figure 5.7: Output of the model for the *Fill* task of the first voice

I gave to the model the figure 5.6 and apply the fill algorithm on the voice

0 (the one at the top, yellow in the figure 5.6 and pink in the figure 5.7). To do so, the model delete the voice to fill and use the remaining ones as a seed to generate the next measure of the missing voice. This is described in the algorithm 3.

Algorithm 3 Fill function

Input: *song* (list of measures), *instrument* (instrument to replace)
Output: *filled* (list of measures)

```

1: function FILL(song, instrument)
2:   song.deleteInstrument(instrument)
3:   filled  $\leftarrow$  song
4:   for k  $\leftarrow$  0 to len(song) – seedLength – 1 do
5:     input  $\leftarrow$  song[k : k + seedLength]
6:     output  $\leftarrow$  model.predict(input)
7:     filled[k + seedLength, instrument]  $\leftarrow$  outputs[instrument]
8:   end for
9:   return filled
10: end function
```

As you can see in the figure 5.7, the filled voice has a big lack of complexity. Most of the notes are quarter notes (J) and *E*. However, this note doesn't sound bad with the song and at the end, the model plays other notes than *E* which also aren't completely wrong.

5.3.3 Redo

The *Redo* task is used to recreate a song. The model will replace one by one all the different voices from a given song. To do so, it will iterate the *Fill* task over all the voices. The algorithm 4 and the figures 5.8, 5.9, 5.10, 5.11, 5.12 illustrate this task.

The song from the figure 5.8 as been given to the model. It has 4 voices (4 different instruments). Here are the for steps:

1. (figure 5.9) The model apply the *Fill* task on the voice 3 (from the top)

Algorithm 4 Redo function

Input: $song$ (list of measures)

Output: $redone$ (list of measures)

```
1: function REDO( $song$ )
2:    $redone \leftarrow song$ 
3:   for  $k \leftarrow 0$  to  $nb\_instruments - 1$  do
4:      $redone \leftarrow Fill(redone, k)$ 
5:   end for
6:   return  $redone$ 
7: end function
```

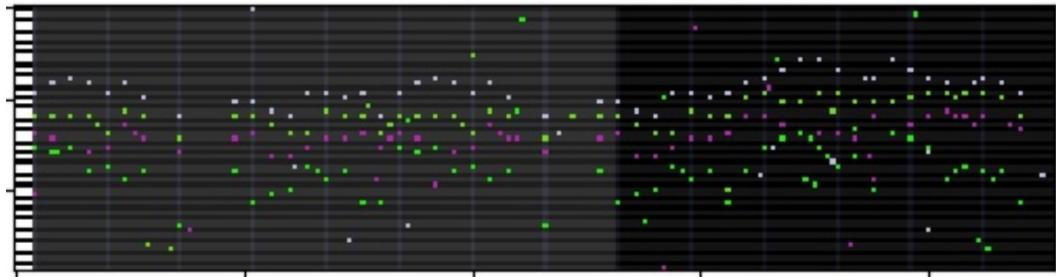


Figure 5.8: Song given to the model for the *Redo* task

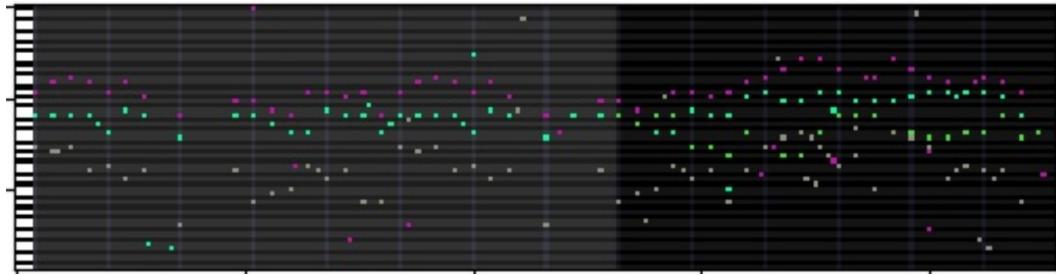


Figure 5.9: Iteration 1 of the *Redo* task

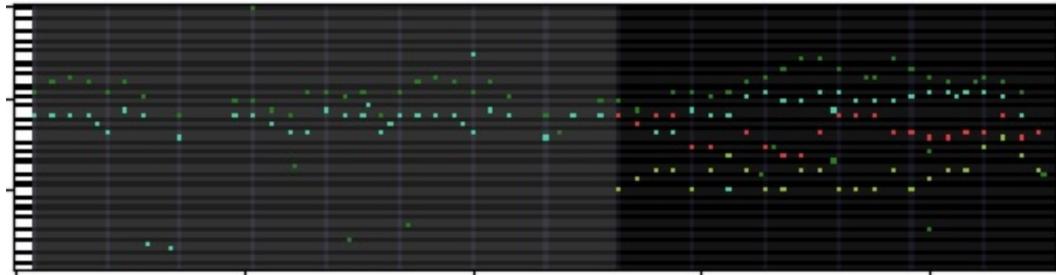


Figure 5.10: Iteration 2 of the *Redo* task

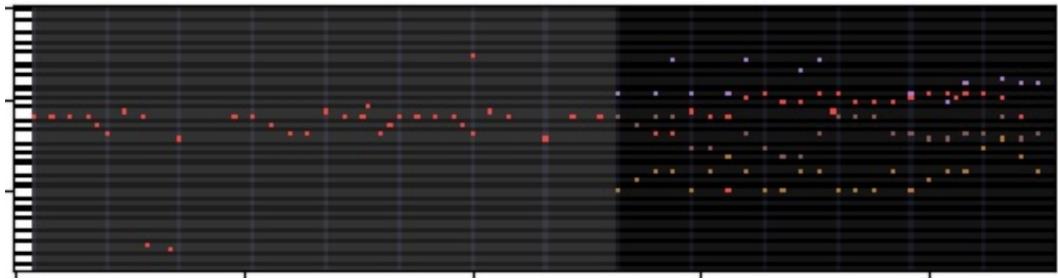


Figure 5.11: Iteration 3 of the *Redo* task

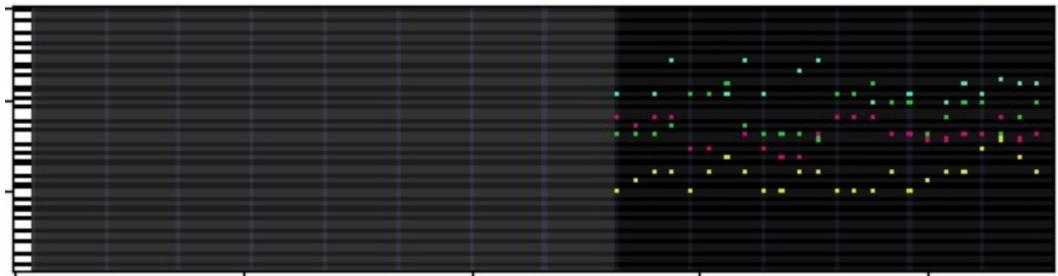


Figure 5.12: Iteration 4 of the *Redo* task

2. (figure 5.10) The model apply the *Fill* task on the voice 4 (from the top)
3. (figure 5.11) The model apply the *Fill* task on the voice 1 (from the top)
4. (figure 5.12) The model apply the *Fill* task on the voice 2 (from the top)

As you can see, the song loses a lot in complexity. The notes are mostly quarter notes (♩). Each voices mostly stays on few notes. This is very repetitive.

However, the generated sample doesn't contain much tension and is not dissonant.

CHAPTER 6

Experiments

In this chapter I expose the experiments I have conduct for this project. Evaluating a generated musical piece is a complex task. Despite the existence of the some musical rules, judging and evaluating a musical piece is a subjective task. Most of the paper working on music use the population to evaluate whether their model is qualitative enough. [40, 7, 69, 8, 9]. They create a poll and let the people decide what sample they prefer between several of them.

By lack of time and result, I haven't been able to create an online poll to evaluate the musical samples I generated.

6.1 Transpose the data

To help the model and get sharper distributions, I created a script to transpose all the music in C major or A minor scale. As I suspected, it didn't help the model much.

- The losses stayed the same.
- The accuracy didn't increase.
- The generation is as poor as before.

Therefor, the model is able to internally transpose the input to analyse it. This capacity might come from the convolutional layers, able to detect features without considering its position in the input. The figures 6.1 and 6.2 show how close are the loss and the accuracy between 2 models.

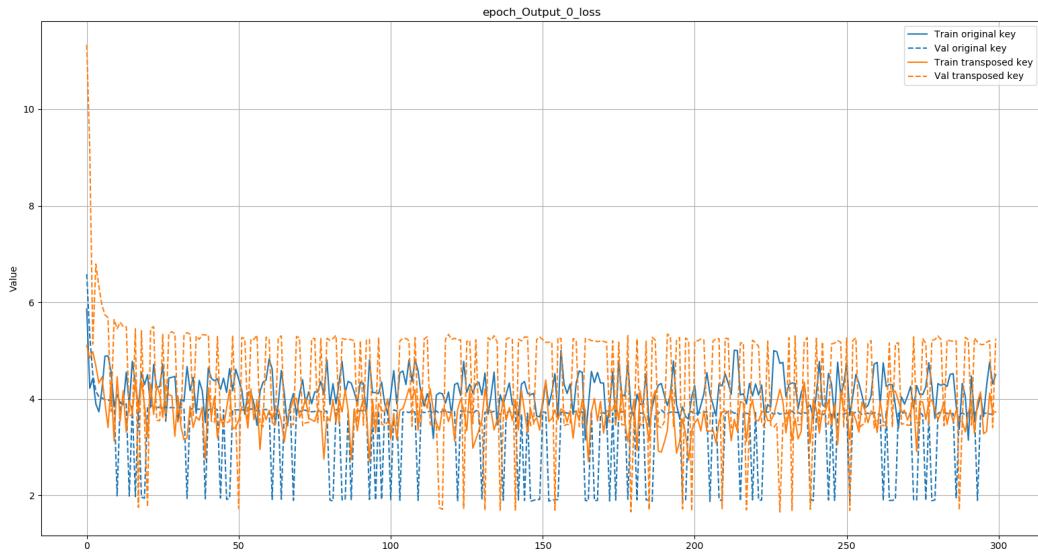


Figure 6.1: Loss comparison between transposed and non-transposed data

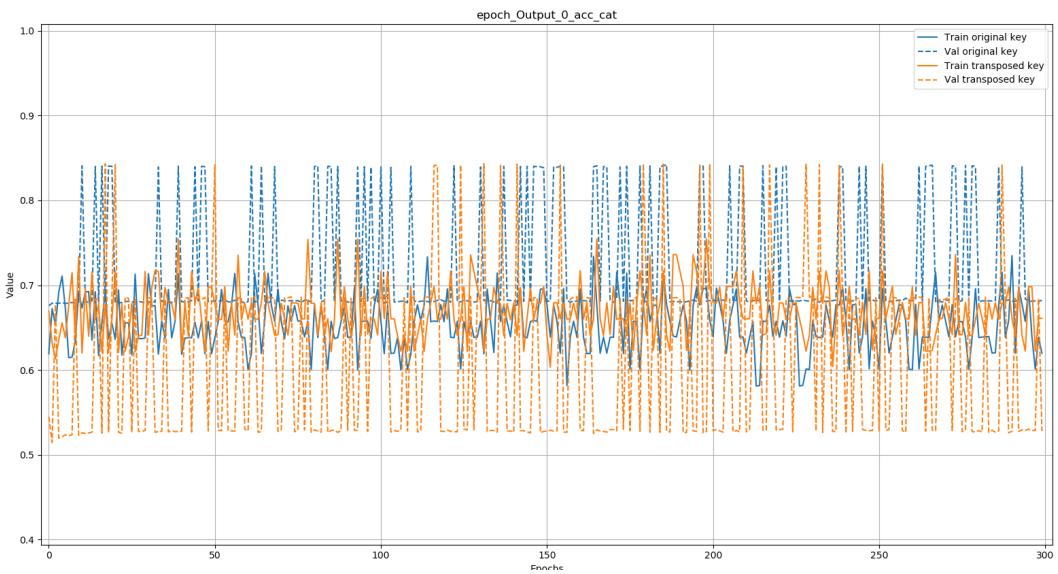


Figure 6.2: Accuracy comparison between transposed and non-transposed data

The first one has been trained on non-transposed songs and the second one has been trained on transposed songs.

6.2 Model size

I tried to modify the model size to check if a bigger or smaller model could converge better. I tried several combinations with some Grid-Search algorithm, Bayesian-Search algorithm (see appendix .7) and by manual tries.

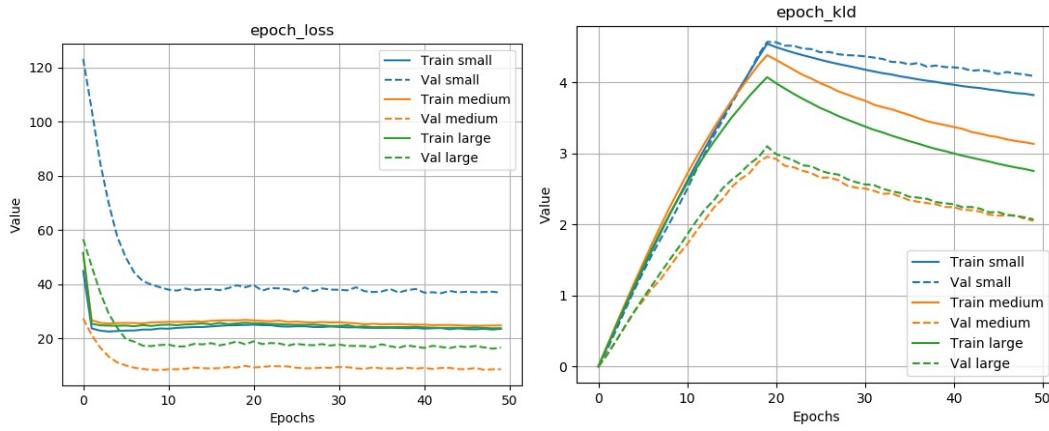


Figure 6.3: Global loss between for size of models

Figure 6.4: KLD value for different size of models

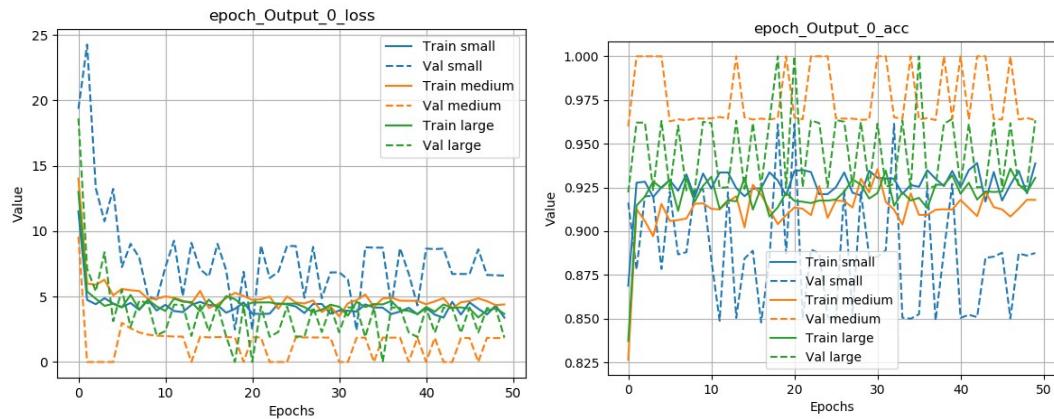


Figure 6.5: Loss of one voice for different size of models

Figure 6.6: Loss of one voice for different size of models

The figures 6.3, 6.4, 6.5 and 6.6 shows respectively the value of the Global

loss, the KLD, the loss of one output, the accuracy of one output through 50 epochs of training for different size of models. There are 3 size of models:

- *Small*: with a size of about $3000Mib$ on GPU
- *Medium*: with a size of about $8500Mib$ on GPU
- *Large*: with a size of about $10500Mib$ on GPU

I couldn't test with bigger model because of the GPU size limitation.

From these different figures, we can see there is no concrete difference between the 3 models on the training data. The values of the global loss, the accuracy and the loss for one output are basically the same. However, the KLD value is slightly better when the model is bigger.

For the validation data, the medium model seems to perform better for every metrics. However.

However despite the difference of value between the different metrics, the generation is not very different, the notes are mostly quarter notes (♩) and there is no variation in the generated music.

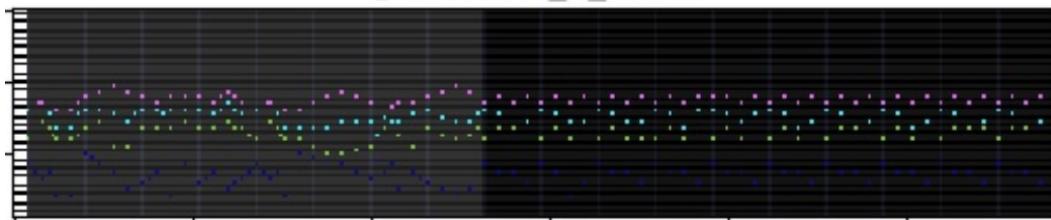


Figure 6.7: Pianoroll generated by the small model
[Link](#)

As shown in the figures 6.7, 6.8, 6.9, the generation is quite similar through all the models. The generated part is looping over the same notes.

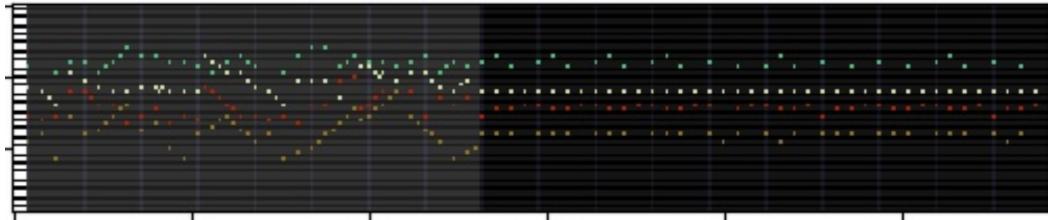


Figure 6.8: Pianoroll generated by the medium model
[Link](#)

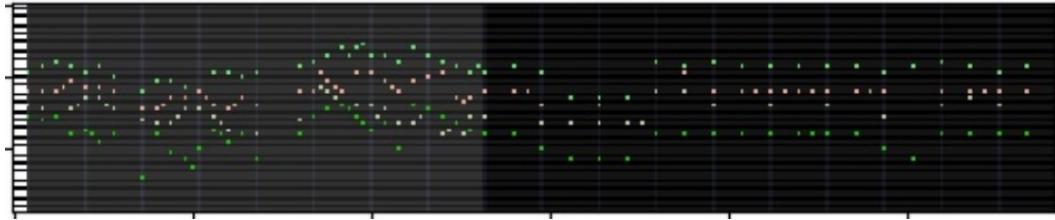


Figure 6.9: Pianoroll generated by the large model
[Link](#)

6.3 Custom losses

6.3.1 Scale and Rhythm

The figures 6.10 and 6.11 show the loss and the accuracy of an output through the training of two models. The first one doesn't use the custom loss functions *Scale* (section 4.4.1) and *Rhythm* (section 4.4.2), but the second one uses them. As we can see, there is no difference between the 2 graphs.

6.3.2 Harmony

The figures 6.12 and 6.13 show the loss and accuracy of an output through the training of two models. The first one doesn't use the *Harmony* function (section 4.4.3), but the second one uses it. As we can see, there is no difference between the 2 graphs

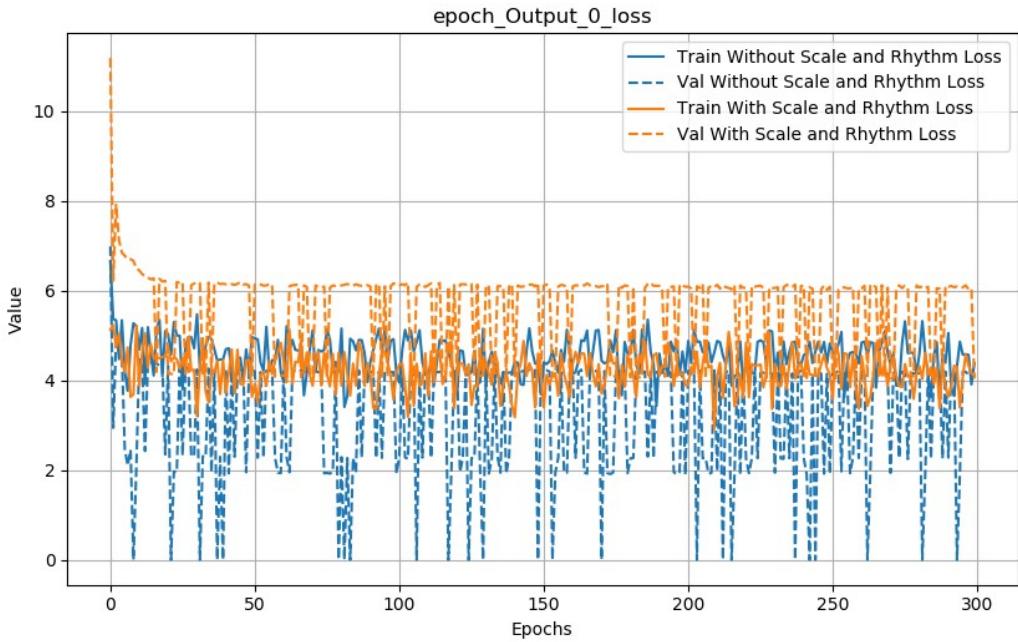


Figure 6.10: Loss comparison between models with and without Scale and Rhythm loss

6.4 RPoE

It appeared that the RPoE layer (section 4.3.4.2) doesn't add anything meaningful to the model. As we can see in the figures 6.14 and 6.15, the loss values and accuracy values between a model using a RPoE layer and a model not using it are not different.

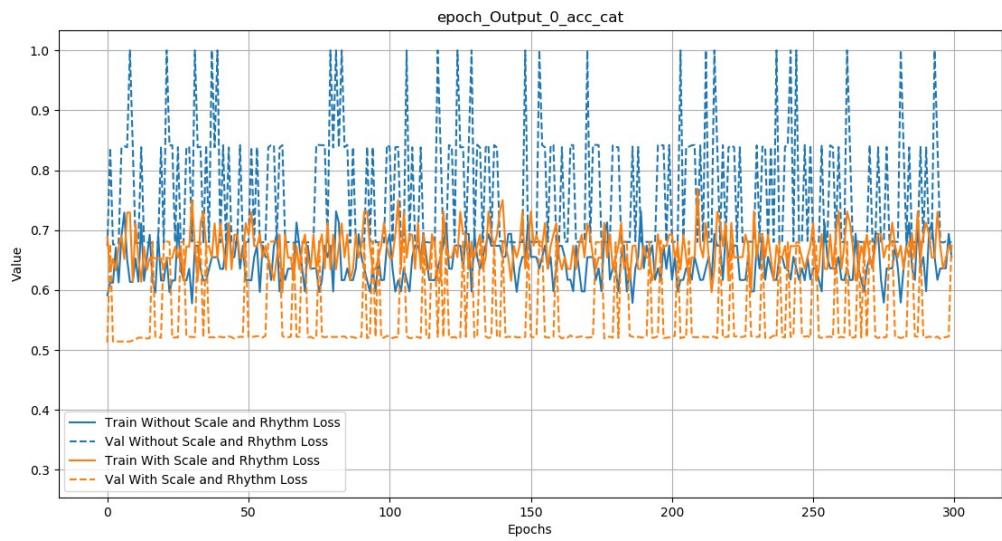


Figure 6.11: Accuracy comparison between models with and without Scale and Rhythm loss

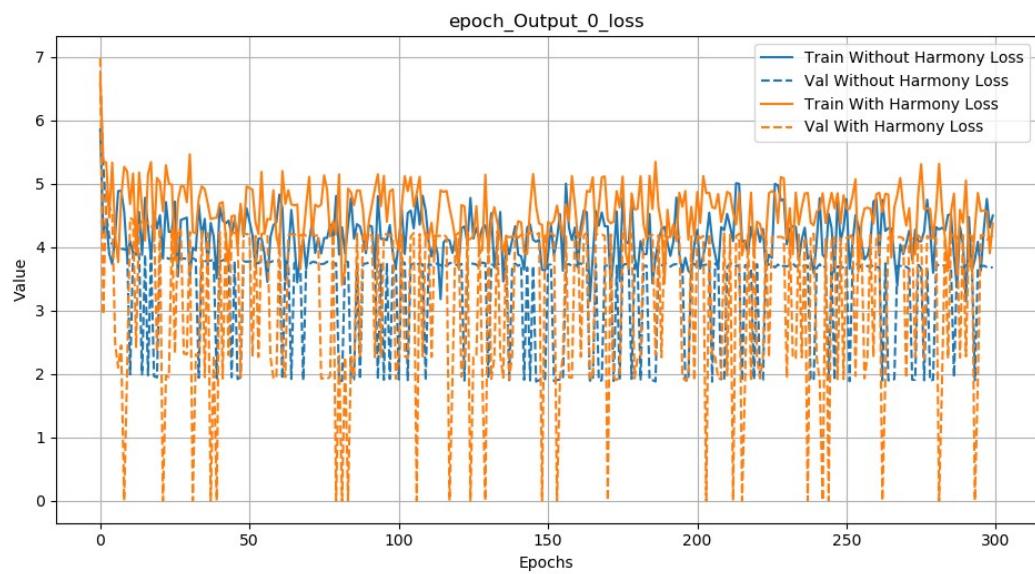


Figure 6.12: Loss comparison between models with and without Harmony loss

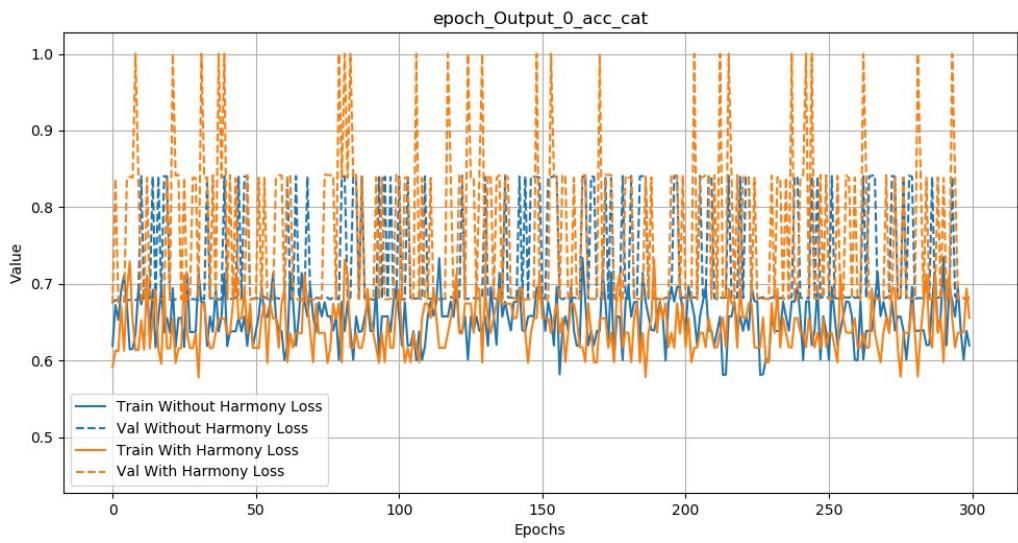


Figure 6.13: Accuracy comparison between models with and without Harmony loss

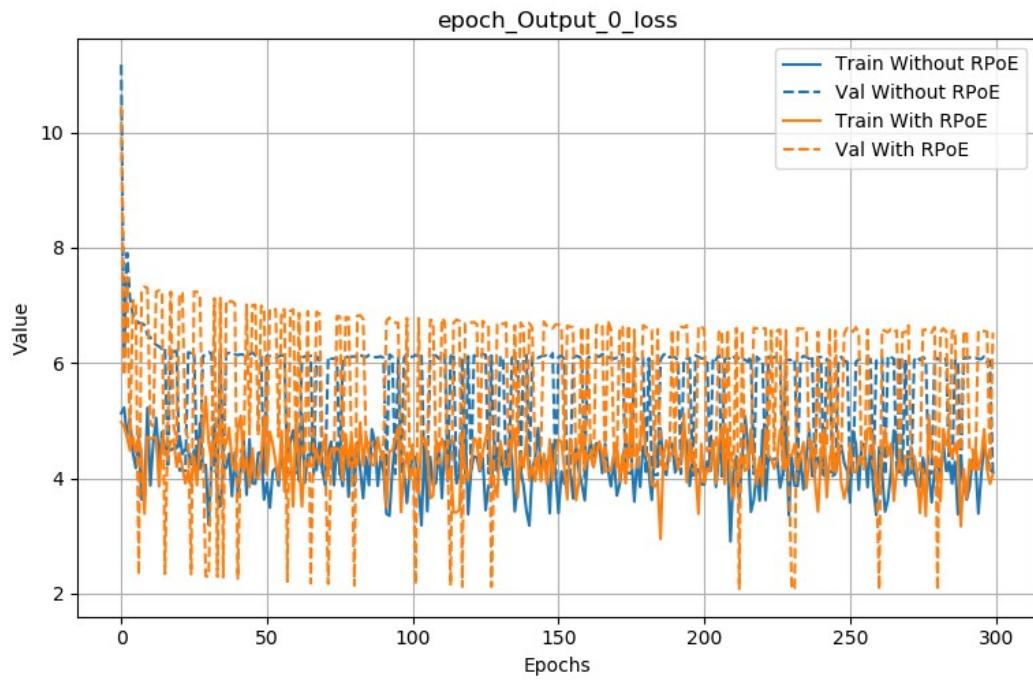


Figure 6.14: Loss comparison between models with and without RPoE layer

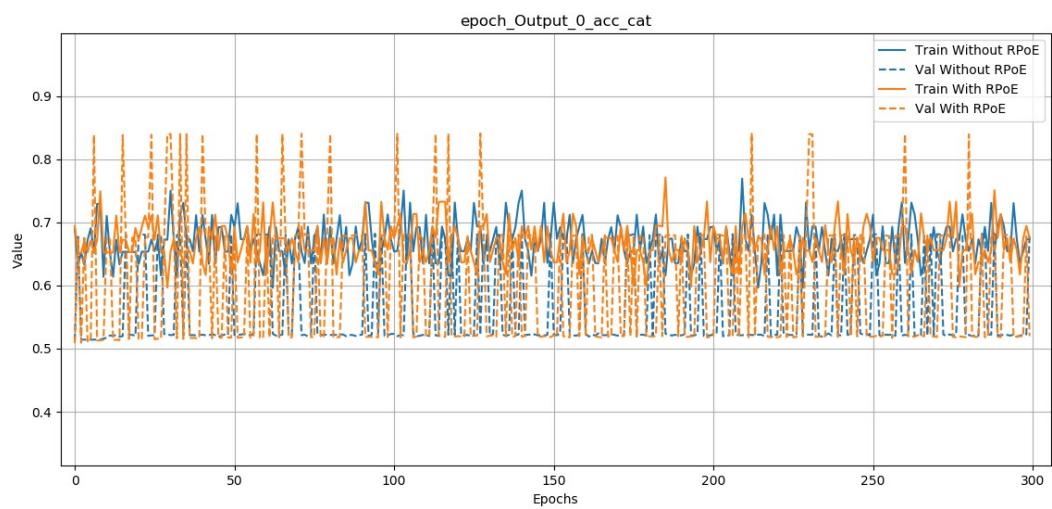


Figure 6.15: Accuracy comparison between models with and without RPoE layer

CHAPTER 7

Conclusion

This dissertation project provides a new neural network architecture (RM-VAE) and a new layer to handle time dependencies between data (RPOE).

I have showed that the RMVAE architecture is able to learn musical rules from a pianoroll view and adding extra losses (Scale, Rhythm and Harmony losses) doesn't help the training.

Future work

Find the scale with some already known scale template ?

Try with different encoding : Text BachBot or DeepBach says the way you write music with text is important Maybe convolution and image is not good enough

I constructed my own pianoroll view. Maybe try to use Pypianoroll will give a better representation.

I didn't want to change the dataset to C major so it can handle change of key. But seems too hard for the neural network

Bibliography

- [1] M. Wu and N. Goodman, “Multimodal Generative Models for Scalable Weakly-Supervised Learning,” in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 5575–5585. [Online]. Available: <http://papers.nips.cc/paper/7801-multimodal-generative-models-for-scalable-weakly-supervised-learning.pdf>
- [2] “music21.corpus.chorales — music21 Documentation.” [Online]. Available: <https://web.mit.edu/music21/doc/moduleReference/moduleCorpusChorales.html>
- [3] “music21: a Toolkit for Computer-Aided Musicology.” [Online]. Available: <https://web.mit.edu/music21/>
- [4] “TensorFlow,” library Catalog: www.tensorflow.org. [Online]. Available: <https://www.tensorflow.org/?hl=fr>
- [5] “Keras | TensorFlow Core,” library Catalog: www.tensorflow.org. [Online]. Available: <https://www.tensorflow.org/guide/keras?hl=fr>
- [6] “FL Studio.” [Online]. Available: <https://www.image-line.com/flstudio/>
- [7] G. Hadjeres, F. Pachet, and F. Nielsen, “DeepBach: a Steerable Model for Bach Chorales Generation,” *arXiv:1612.01010 [cs]*, Dec. 2016. [Online]. Available: <http://arxiv.org/abs/1612.01010>
- [8] F. T. Liang, M. Gotham, M. Johnson, and J. Shotton, “Automatic Stylistic Composition of Bach Chorales with Deep LSTM,” in *ISMIR*, 2017.

- [9] C.-Z. A. Huang, C. Hawthorne, A. Roberts, M. Dinculescu, J. Wexler, L. Hong, and J. Howcroft, “The Bach Doodle: Approachable music composition with machine learning at scale,” *arXiv:1907.06637 [cs, eess, stat]*, Jul. 2019, arXiv: 1907.06637. [Online]. Available: <http://arxiv.org/abs/1907.06637>
- [10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin, “Attention is All you Need,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5998–6008. [Online]. Available: <http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf>
- [11] M. Wu and N. Goodman, “Multimodal Generative Models for Scalable Weakly-Supervised Learning,” in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 5575–5585. [Online]. Available: <http://papers.nips.cc/paper/7801-multimodal-generative-models-for-scalable-weakly-supervised-learning.pdf>
- [12] Z.-X. Tan, H. Soh, and D. C. Ong, “Factorized Inference in Deep Markov Models for Incomplete Multimodal Time Series,” *arXiv:1905.13570 [cs, stat]*, May 2019, arXiv: 1905.13570. [Online]. Available: <http://arxiv.org/abs/1905.13570>
- [13] M. Moor, M. Horn, B. Rieck, and K. Borgwardt, “Topological Autoencoders,” *arXiv:1906.00722 [cs, math, stat]*, Feb. 2020, arXiv: 1906.00722. [Online]. Available: <http://arxiv.org/abs/1906.00722>

- [14] M. Tschannen, O. Bachem, and M. Lucic, “Recent Advances in Autoencoder-Based Representation Learning,” *arXiv:1812.05069 [cs, stat]*, Dec. 2018, arXiv: 1812.05069. [Online]. Available: <http://arxiv.org/abs/1812.05069>
- [15] M. Rudolph, B. Wandt, and B. Rosenhahn, “Structuring Autoencoders,” *arXiv:1908.02626 [cs, stat]*, Aug. 2019, arXiv: 1908.02626. [Online]. Available: <http://arxiv.org/abs/1908.02626>
- [16] C. Doersch, “Tutorial on Variational Autoencoders,” *arXiv:1606.05908 [cs, stat]*, Jun. 2016. [Online]. Available: <http://arxiv.org/abs/1606.05908>
- [17] “The variational auto-encoder.” [Online]. Available: <https://ermongroup.github.io/cs228-notes/extras/vae/>
- [18] “Tutorial - What is a variational autoencoder?” library Catalog: jaan.io. [Online]. Available: </what-is-variational-autoencoder-vae-tutorial/>
- [19] H. Akrami, A. A. Joshi, J. Li, S. Aydore, and R. M. Leahy, “Robust Variational Autoencoder,” *arXiv:1905.09961 [cs, eess, stat]*, Dec. 2019, arXiv: 1905.09961. [Online]. Available: <http://arxiv.org/abs/1905.09961>
- [20] W. Liu, R. Li, M. Zheng, S. Karanam, Z. Wu, B. Bhanu, R. J. Radke, and O. Camps, “Towards Visually Explaining Variational Autoencoders,” *arXiv:1911.07389 [cs]*, Mar. 2020, arXiv: 1911.07389. [Online]. Available: <http://arxiv.org/abs/1911.07389>
- [21] D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” *arXiv:1312.6114 [cs, stat]*, May 2014, arXiv: 1312.6114. [Online]. Available: <http://arxiv.org/abs/1312.6114>
- [22] *GAN Deep Learning Architectures - review*, Sep. 2017. [Online]. Available: <https://sigmoidal.io/beginners-review-of-gan-architectures/>

- [23] I. Goodfellow, “Generative Adversarial Networks (GANs),” p. 86, 2016.
- [24] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative Adversarial Networks,” *arXiv:1406.2661 [cs, stat]*, Jun. 2014, arXiv: 1406.2661. [Online]. Available: <http://arxiv.org/abs/1406.2661>
- [25] I. Goodfellow, “NIPS 2016 Tutorial: Generative Adversarial Networks,” *arXiv:1701.00160 [cs]*, Apr. 2017, arXiv: 1701.00160. [Online]. Available: <http://arxiv.org/abs/1701.00160>
- [26] G. Chen, “A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation,” *arXiv:1610.02583 [cs]*, Jan. 2018, arXiv: 1610.02583. [Online]. Available: <http://arxiv.org/abs/1610.02583>
- [27] M. Nguyen, “Illustrated Guide to LSTM’s and GRU’s: A step by step explanation,” Jul. 2019, library Catalog: towardsdatascience.com. [Online]. Available: <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>
- [28] “Transformer model for language understanding | TensorFlow Core,” library Catalog: www.tensorflow.org. [Online]. Available: <https://www.tensorflow.org/tutorials/text/transformer?hl=fr>
- [29] G. Giacaglia, “Transformers,” Dec. 2019, library Catalog: towardsdatascience.com. [Online]. Available: <https://towardsdatascience.com/transformers-141e32e69591>
- [30] M. Allard, “What is a Transformer?” Mar. 2020, library Catalog: medium.com. [Online]. Available: <https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04>

- [31] J. Alammar, “The Illustrated Transformer,” library Catalog: jalammar.github.io/illustrated-transformer/ [Online]. Available: <http://jalammar.github.io/illustrated-transformer/>
- [32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention Is All You Need,” *arXiv:1706.03762 [cs]*, Dec. 2017, arXiv: 1706.03762. [Online]. Available: <http://arxiv.org/abs/1706.03762>
- [33] B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle, “Neural Autoregressive Distribution Estimation,” *arXiv:1605.02226 [cs]*, May 2016, arXiv: 1605.02226. [Online]. Available: <http://arxiv.org/abs/1605.02226>
- [34] B. Uria, I. Murray, and H. Larochelle, “A Deep and Tractable Density Estimator,” *arXiv:1310.1757 [cs, stat]*, Jan. 2014, arXiv: 1310.1757. [Online]. Available: <http://arxiv.org/abs/1310.1757>
- [35] “Restricted Boltzmann Machine Tutorial | Deep Learning Concepts,” Nov. 2018, library Catalog: [www.edureka.co](https://www.edureka.co/blog/restricted-boltzmann-machine-tutorial/) Section: Artificial Intelligence. [Online]. Available: <https://www.edureka.co/blog/restricted-boltzmann-machine-tutorial/>
- [36] G. Montufar, “Restricted Boltzmann Machines: Introduction and Review,” *arXiv:1806.07066 [cs, math, stat]*, Jun. 2018, arXiv: 1806.07066. [Online]. Available: <http://arxiv.org/abs/1806.07066>
- [37] R. Salakhutdinov, A. Mnih, and G. Hinton, “Restricted Boltzmann machines for collaborative filtering,” in *Proceedings of the 24th international conference on Machine learning - ICML '07*. Corvalis, Oregon: ACM Press, 2007, pp. 791–798. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1273496.1273596>

- [38] A. Fischer and C. Igel, “An Introduction to Restricted Boltzmann Machines,” Jan. 2012, pp. 14–36.
- [39] C.-H. Chuan and D. Herremans, “Modeling Temporal Tonal Relations in Polyphonic Music through Deep Networks with a Novel Image-Based Representation,” p. 8.
- [40] C.-Z. A. Huang, T. Cooijmans, A. Roberts, A. Courville, and D. Eck, “COUNTERPOINT BY CONVOLUTION,” p. 8, 2017.
- [41] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, “Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription,” *arXiv:1206.6392 [cs, stat]*, Jun. 2012, arXiv: 1206.6392. [Online]. Available: <http://arxiv.org/abs/1206.6392>
- [42] S. Lattner, M. Grachten, and G. Widmer, “Imposing higher-level Structure in Polyphonic Music Generation using Convolutional Restricted Boltzmann Machines and Constraints,” *Journal of Creative Music Systems*, vol. 2, no. 2, Mar. 2018, arXiv: 1612.04742. [Online]. Available: <http://arxiv.org/abs/1612.04742>
- [43] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “WaveNet: A Generative Model for Raw Audio,” *arXiv:1609.03499 [cs]*, Sep. 2016. [Online]. Available: <http://arxiv.org/abs/1609.03499>
- [44] S. Dieleman, A. van den Oord, and K. Simonyan, “The challenge of realistic music generation: modelling raw audio at scale,” in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc.,

- 2018, pp. 7989–7999. [Online]. Available: <http://papers.nips.cc/paper/8023-the-challenge-of-realistic-music-generation-modelling-raw-audio-at-scale.pdf>
- [45] C. Donahue, J. McAuley, and M. Puckette, “Adversarial Audio Synthesis,” *arXiv:1802.04208 [cs]*, Feb. 2019, arXiv: 1802.04208. [Online]. Available: <http://arxiv.org/abs/1802.04208>
- [46] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. v. d. Oord, S. Dieleman, and K. Kavukcuoglu, “Efficient Neural Audio Synthesis,” *arXiv:1802.08435 [cs, eess]*, Jun. 2018, arXiv: 1802.08435. [Online]. Available: <http://arxiv.org/abs/1802.08435>
- [47] S. Mehri, K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. Courville, and Y. Bengio, “SampleRNN: An Unconditional End-to-End Neural Audio Generation Model,” *arXiv:1612.07837 [cs]*, Feb. 2017, arXiv: 1612.07837. [Online]. Available: <http://arxiv.org/abs/1612.07837>
- [48] C.-Y. Lu, M.-X. Xue, C.-C. Chang, C.-R. Lee, and L. Su, “Play as You Like: Timbre-enhanced Multi-modal Music Style Transfer,” *arXiv:1811.12214 [cs, eess]*, Nov. 2018, arXiv: 1811.12214. [Online]. Available: <http://arxiv.org/abs/1811.12214>
- [49] “Understanding Spectrograms,” library Catalog: www.izotope.com. [Online]. Available: <https://www.izotope.com/en/learn/understanding-spectrograms.html>
- [50] K. Adiloglu and F. Alpaslan, “A machine learning approach to two-voice counterpoint composition,” *Knowledge-Based Systems*, vol. 20, pp. 300–309, Apr. 2007.

- [51] D. Herremans and K. Sörensen, “Composing fifth species counterpoint music with a variable neighborhood search algorithm,” *Expert Systems with Applications*, vol. 40, no. 16, pp. 6427–6437, Nov. 2013. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0957417413003692>
- [52] D. Herremans, “Modeling Musical Context Using Word2vec,” p. 8, 2017.
- [53] F. Colombo, J. Brea, and W. Gerstner, “Learning to Generate Music with BachProp,” *arXiv:1812.06669 [cs, eess, stat]*, Jun. 2019, arXiv: 1812.06669. [Online]. Available: <http://arxiv.org/abs/1812.06669>
- [54] G. Brunner, Y. Wang, R. Wattenhofer, and S. Zhao, “Symbolic Music Genre Transfer with CycleGAN,” *arXiv:1809.07575 [cs, eess, stat]*, Sep. 2018, arXiv: 1809.07575. [Online]. Available: <http://arxiv.org/abs/1809.07575>
- [55] J. Wu, C. Hu, Y. Wang, X. Hu, and J. Zhu, “A Hierarchical Recurrent Neural Network for Symbolic Melody Generation,” *arXiv:1712.05274 [cs]*, Sep. 2018, arXiv: 1712.05274. [Online]. Available: <http://arxiv.org/abs/1712.05274>
- [56] L. F. Mason, “Essential Neo-Riemannian Theory for Today’s Musician,” p. 98.
- [57] Y. Goldberg and O. Levy, “word2vec Explained: deriving Mikolov et al.’s negative-sampling word-embedding method,” *arXiv:1402.3722 [cs, stat]*, Feb. 2014, arXiv: 1402.3722. [Online]. Available: <http://arxiv.org/abs/1402.3722>
- [58] D. Karani, “Introduction to Word Embedding and Word2Vec,” Sep. 2018, library Catalog: towardsdata-

- science.com. [Online]. Available: <https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa>
- [59] X. Rong, “word2vec Parameter Learning Explained,” *arXiv:1411.2738 [cs]*, Jun. 2016, arXiv: 1411.2738. [Online]. Available: <http://arxiv.org/abs/1411.2738>
- [60] “A Beginner’s Guide to Word2Vec and Neural Word Embeddings,” library Catalog: pathmind.com. [Online]. Available: <http://pathmind.com/wiki/word2vec>
- [61] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality,” in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 3111–3119. [Online]. Available: <http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf>
- [62] I. Sutskever, G. Hinton, and G. Taylor, “The Recurrent Temporal Restricted Boltzmann Machine,” p. 8.
- [63] R. Mittelman, B. Kuipers, S. Savarese, and H. Lee, “Structured Recurrent Temporal Restricted Boltzmann Machines,” p. 9.
- [64] M. Norouzi, “CONVOLUTIONAL RESTRICTED BOLTZMANN MACHINES FOR FEATURE LEARNING,” p. 61.
- [65] M. Norouzi, M. Ranjbar, and G. Mori, “Stacks of Convolutional Restricted Boltzmann Machines for Shift-Invariant Feature Learning,” p. 8.

- [66] “TensorFlow.js | Machine Learning for Javascript Developers,” library Catalog: www.tensorflow.org. [Online]. Available: <https://www.tensorflow.org/js?hl=fr>
- [67] A. van den Oord, O. Vinyals, and k. kavukcuoglu, “Neural Discrete Representation Learning,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 6306–6315. [Online]. Available: <http://papers.nips.cc/paper/7210-neural-discrete-representation-learning.pdf>
- [68] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks,” *arXiv:1703.10593 [cs]*, Nov. 2018, arXiv: 1703.10593. [Online]. Available: <http://arxiv.org/abs/1703.10593>
- [69] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, “Music Transformer,” *arXiv:1809.04281 [cs, eess, stat]*, Sep. 2018. [Online]. Available: <http://arxiv.org/abs/1809.04281>
- [70] K. Choi, C. Hawthorne, I. Simon, M. Dinculescu, and J. Engel, “Encoding Musical Style with Transformer Autoencoders,” *arXiv:1912.05537 [cs, eess, stat]*, Dec. 2019, arXiv: 1912.05537. [Online]. Available: <http://arxiv.org/abs/1912.05537>
- [71] V. Shetty, “Neural Style Transfer Tutorial -Part 1,” Mar. 2019, library Catalog: towardsdatascience.com. [Online]. Available: <https://towardsdatascience.com/neural-style-transfer-tutorial-part-1-f5cd3315fa7f>

- [72] L. A. Gatys, A. S. Ecker, and M. Bethge, “A Neural Algorithm of Artistic Style,” *arXiv:1508.06576 [cs, q-bio]*, Sep. 2015, arXiv: 1508.06576. [Online]. Available: <http://arxiv.org/abs/1508.06576>
- [73] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, “Universal Style Transfer via Feature Transforms,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 386–396. [Online]. Available: <http://papers.nips.cc/paper/6642-universal-style-transfer-via-feature-transforms.pdf>
- [74] M. Kaliakatsos-Papakostas, M. Queiroz, C. Tsougras, and E. Cambouropoulos, “Conceptual Blending of Harmonic Spaces for Creative Melodic Harmonisation,” *Journal of New Music Research*, vol. 46, no. 4, pp. 305–328, Oct. 2017, publisher: Routledge _eprint: <https://doi.org/10.1080/09298215.2017.1355393>. [Online]. Available: <https://doi.org/10.1080/09298215.2017.1355393>
- [75] Y.-N. Hung, I.-T. Chiang, Y.-A. Chen, and Y.-H. Yang, “Musical Composition Style Transfer via Disentangled Timbre Representations,” *arXiv:1905.13567 [cs, eess]*, May 2019, arXiv: 1905.13567. [Online]. Available: <http://arxiv.org/abs/1905.13567>
- [76] S. Dai, Z. Zhang, and G. G. Xia, “Music Style Transfer: A Position Paper,” *arXiv:1803.06841 [cs, eess]*, Jul. 2018, arXiv: 1803.06841. [Online]. Available: <http://arxiv.org/abs/1803.06841>
- [77] S. S. Sahoo, C. H. Lampert, and G. Martius, “Learning Equations for Extrapolation and Control,” *arXiv:1806.07259 [cs, stat]*, Jun. 2018. [Online]. Available: <http://arxiv.org/abs/1806.07259>

- [78] P. I. Frazier, “A Tutorial on Bayesian Optimization,” *arXiv:1807.02811 [cs, math, stat]*, Jul. 2018, arXiv: 1807.02811. [Online]. Available: <http://arxiv.org/abs/1807.02811>
- [79] R. P. Adams, “A Tutorial on! Bayesian Optimization! for Machine Learning,” p. 45.
- [80] “scikit-optimize: sequential model-based optimization in Python — scikit-optimize 0.7.4 documentation.” [Online]. Available: <https://scikit-optimize.github.io/stable/>
- [81] J. Brownlee, “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning,” Jul. 2017, library Catalog: machinelearningmastery.com Section: Deep Learning Performance. [Online]. Available: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>
- [82] V. Bushaev, “Adam — latest trends in deep learning optimization.” Oct. 2018, library Catalog: towardsdatascience.com. [Online]. Available: <https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c>
- [83] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv:1412.6980 [cs]*, Jan. 2017, arXiv: 1412.6980. [Online]. Available: <http://arxiv.org/abs/1412.6980>
- [84] C.-W. Huang, S. Tan, A. Lacoste, and A. C. Courville, “Improving Explorability in Variational Inference with Annealed Variational Objectives,” in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 9701–9711. [Online]. Available: <http://papers.nips.cc/paper/>

8178-improving-explorability-in-variational-inference-with-annealed-variational-objectives.
pdf

Appendices

.1 Interpolation project

At the start of my dissertation, I tried another project which was to create music with a waveform representation. Subham S. Sahoo et al. [77] present their improved network for equation learning EQL[÷], that overcomes the limitation of the earlier works. In particular, their main contribution are:

- They propose a network architecture that can handle divisions as well as techniques to keep training stable
- They improve model/instance selection to be more effective in identifying the right network/equation
- They demonstrate how to reliably control a dynamical robotic system by learning its forward dynamics equations from very few random try-outs/tails.

More precisely, their model is a shallow network able to learn complex function containing multiplication, sine and cosine functions, and it is able to extrapolate them. Those are very usual operations processed by most of the VST to create musical sounds.

Thereby, I tried to interpolate musical waveform with their architecture. My idea was to construct a transition between 2 parts (for example a verse and a chorus) or reconstruct some missing data from a song.

Unfortunately, the results were poor... Most of the time, the model was converging to the mean of the functions (which is 0 for musical waveform).

I think their model is working well for their applications because they use it to extrapolate robotic dynamic equations which usually don't include many sine and cosine components. Contrariwise, a musical waveform contains an enormous number of sine and cosine components.

.2 Segment Sum

The segment sum operation allows me to sum the values of a vector of shape $(128,)$ (corresponding of all the different notes in the different octaves) in a vector of shape $(12,)$ (corresponding to the different 12 notes).

The figure 1 describes what the segment sum operation would perform if there were only 4 different notes (A, B, C, D) and 3 different octaves ($1, 2, 3$) for a total of 12 notes.

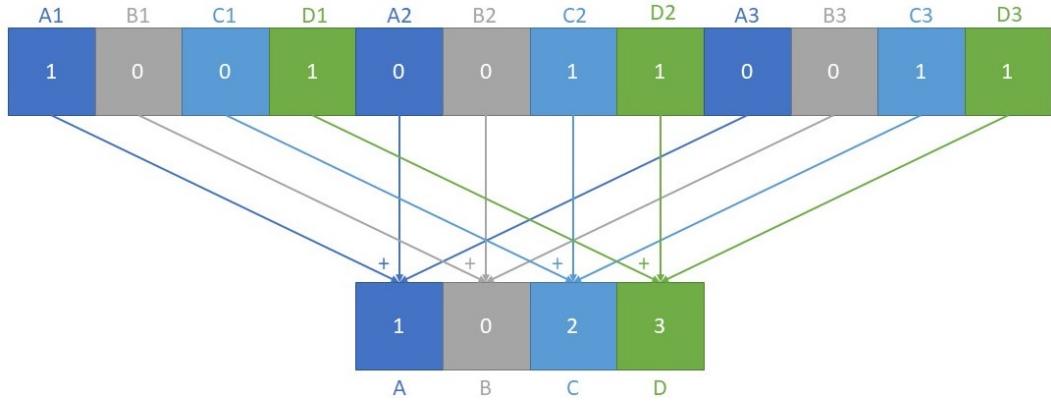


Figure 1: Segment sum operation

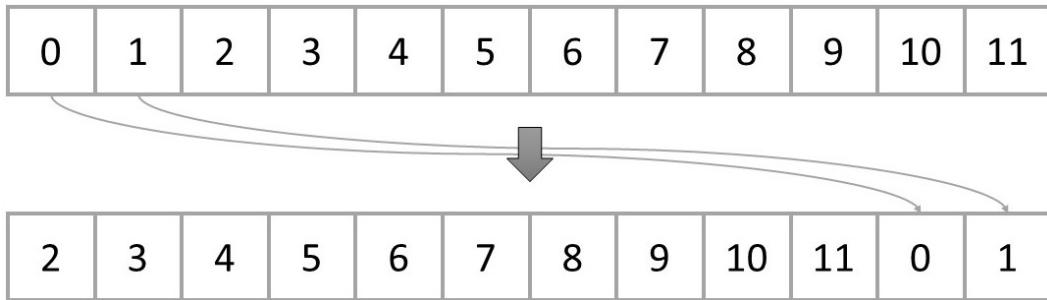
.3 Roll_n

The Roll_n operation takes a one dimensional tensor, takes the first n values and put append them at the end. The algorithm 5 explains how it works and the figure 2 shows an example with $n = 2$.

Algorithm 5 Roll_n function

Input: tensor, n**Output:** tensor (Rolled tensor)

```
1: function ROLL(tensor, n)
2:   for k  $\leftarrow$  1 to n do
3:     firstElement  $\leftarrow$  tensor.pop(0)
4:     tensor.append(firstElement)
5:   end for
6:   return tensor
7: end function
```

Figure 2: Roll₂ example

.4 BandPlayer

Training a model and generating music is interesting. But as musician, I wasn't fully satisfied with this application. To fulfil my need, I created a python script to be able to play with the trained model in real time. I call this application **BandPlayer** because the user can play with and jam with a virtual band.

The application I have created

- Allows the user to play music with the instrument of his choice
- Can load a trained model
- Allows the user to choose what part he wants to play in the "*band*"
- Allows the user to choose what parts from the "*band*" he wants to play with

- Feeds the model with the notes played
- Play the generated notes by the model with the user
- Display the notes played by the model and the user in a pianoroll view

Calling the model to make the notes prediction while the user is playing is computationally feasible because my model is working on an entire measure at one time. Therefor the model is called only one time per measure which is not to frequent to allow the application to be real time.

.5 Coconet Process

The figure 3 illustrates the coconet process. This process is explained in the section 3.4.2.

.6 Transformer architecture

The figure 4 illustrates the transformer's architecture. This architecture is explained in the section 2.4.6.

.7 Hyper-Parameters tuning

Tuning hyper-parameters to train the best possible model is a difficult task in deep learning. The reason is because every number or setting can be considered as a hyper-parameter and, most of time, there is no proof or rules to guide us to choose the best value a parameter should have. The only help the researcher can have is his intuition and his understanding on how a specific parameter will influence a specific process.

A common doing is to implement a grid-search algorithm. For each hyper-parameters, a set of possible value is specified and the algorithm is simply to

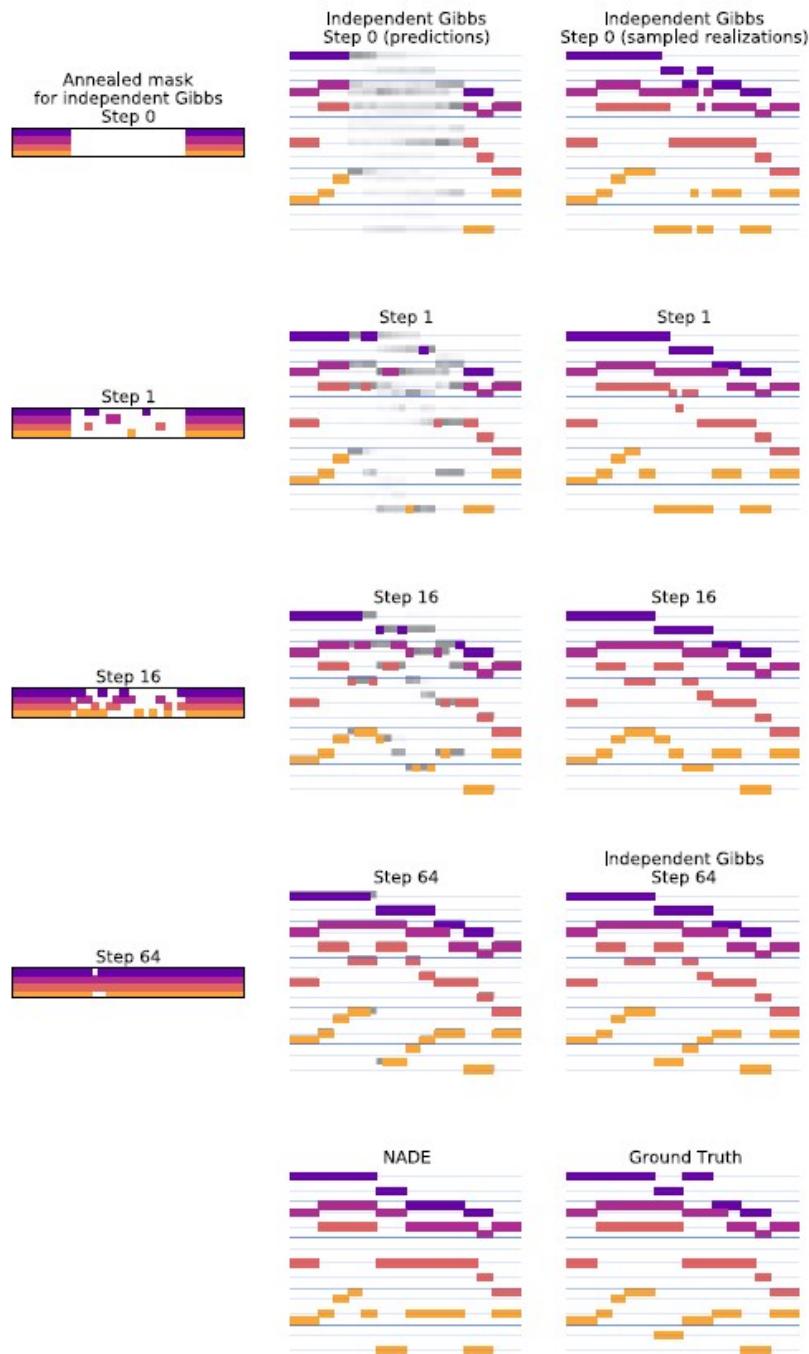


Figure 3: COCONET process
Source: Cheng-Zhi Anna Huang et al.'s paper [40]

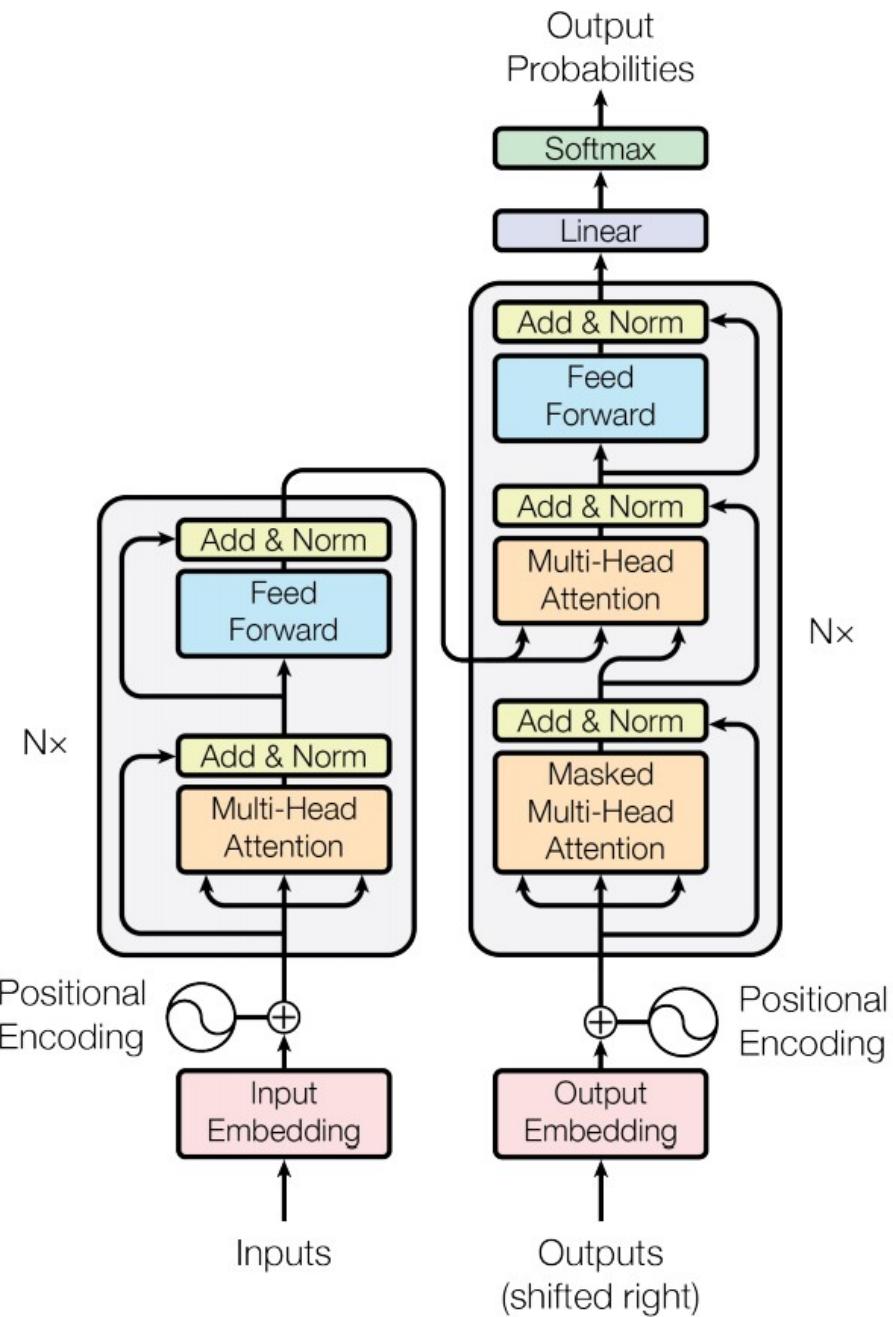


Figure 4: Transformer architecture
Source: Ashish Vaswani et al.'s paper [32]

train the model with all the possible combinations of parameters. However, the number of iteration grow exponentially with the number of hyper-parameters.

It has been showed that a random-search algorithm is in average faster and better than a grid search.

Another possibility is to implement a Bayesian optimization [78, 79]. It is a sequential design strategy for global optimization of black-box functions that doesn't require derivatives. It uses GP to learn a unknown function. Since the number of hyper-parameters in my project is very high, (at some point, I was trying to tune over 20 parameters), I implemented a script to do a Bayesian Search over them. To do so I used the library scikit-optimize [80]. The main advantages of a Bayesian Search over a Grid or a Random Search are:

- It requires less iterations.
- The domain of search for a specific parameter value is not discrete but continuous.

.8 Network details

To create and train my models I used the python frameworks Tensorflow [4] and Keras [5].

.8.1 Hyper Parameters

The table 1 is listing the principal value of the hyper-parameters used.

Through all the manual training I manually launched, the Grid Search and Bayesian Search algorithm I ran, I have continuously been updated the value of these hyper-parameters.

Hyper Parameter	Value
optimizer	Adam optimizer [81, 82, 83]
learning rate	10^{-3}
fully connected layer dropout	2×10^{-2}
convolutional layer dropout	10^{-3}
lstm dropout	10^{-2}
decay	10^{-1}
λ_{scale}	10^{-1}
λ_{rhythm}	10^{-2}
$\lambda_{\text{semitone}}$	10^{-2}
λ_{tone}	10^{-2}
λ_{tritone}	10^{-2}

Table 1: Neural Network Hyper-Parameters

.8.2 Masking

To handle missing data, modalities (instruments), the model takes as input a mask. The shape of a mask is `(nb_instruments, nb_measures)` and composed of 0 and 1. If there is a 0 at the position (i, m) of the mask, it means that the measure m of the instrument i is missing.

.9 Training details

.9.1 Noise

During the training, I insert some noise in the input data. For monophonic music, an activation value will be changed (from 0 to 1 or from 1 to 0) with a probability p . I usually have put $p \in [10^{-4}, 10^{-3}]$. The chosen range of p seems low but a higher noise can sometimes alter the input too much, even for a human ear.

For example, let's assume a noise with $p = 10^{-2}$. A shape of a tensor representing a measure is `(16, 128)`. For every frame (shape of `(128,)`), the expectation of altered note is 1.28. It is an added or deleted note for every

frame (Sixteenth ♫) which completely deform the music.

.9.2 KLD Annealing

It is usually difficult to train a VAE because of the KLD which adds a big constraint on the latent space. To help the model to converge, I implemented a *KLD annealing* [84].

$$loss = loss_{output} + \beta \mathbb{D}_{KL} \quad (1)$$

I linearly increases β from 0 to 1 through the epochs from $epoch_{kld\ start}$ to $epoch_{kld\ stop}$. Before $epoch_{kld\ start}$ $\beta = 0$, and after $epoch_{kld\ stop}$, $\beta = 1$.

The equation 2 and the figure 5 show the evolution of β through the training.

$$\beta(epoch) = \begin{cases} 0 & \text{if } \frac{epoch}{epoch_{total}} \leq epoch_{kld\ start} \\ \frac{(epoch/epoch_{total}) - epoch_{kld\ start}}{epoch_{kld\ stop} - epoch_{kld\ start}} & \text{if } epoch_{kld\ start} \leq \frac{epoch}{epoch_{total}} \leq epoch_{kld\ stop} \\ 1 & \text{if } \frac{epoch}{epoch_{total}} \geq epoch_{kld\ stop} \end{cases} \quad (2)$$

.10 Implementation details

My project is coded in Python language. I have made my GitHub repository accessible at <https://github.com/ValentinVignal/midiGenerator>.

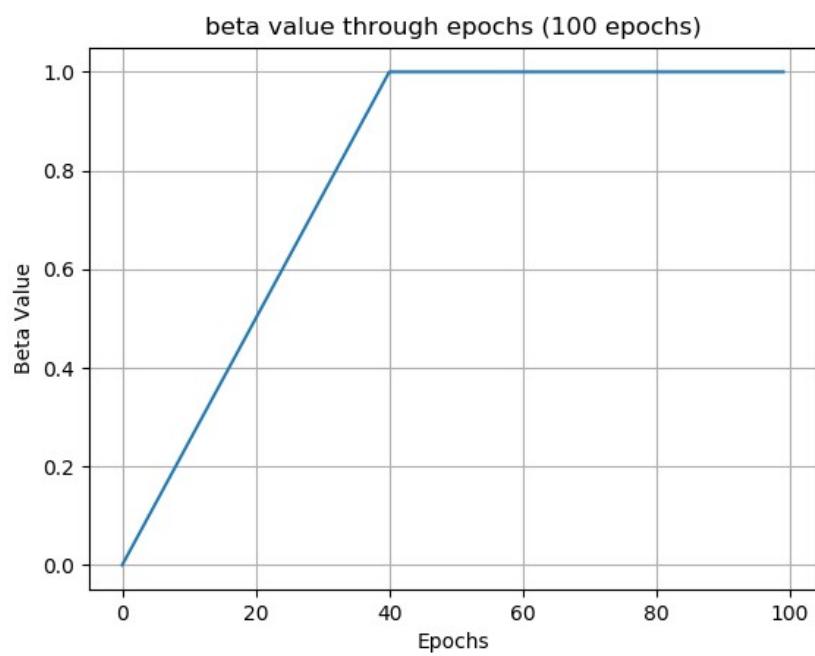


Figure 5: β value through training for KLD annealing