

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
DEPARTEMENT OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE

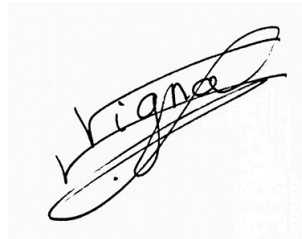
2020

Advisor:
Examiners:

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.

A handwritten signature in black ink, appearing to read 'Vignal', is written over a light gray rectangular background. The signature is stylized with loops and a long horizontal stroke extending to the right.

Valentin Vignal
21 March 2020

ACKNOWLEDGEMENTS

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

Secondly I would also like to thanks all the member 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 yould like to thanks the Doctor Dorien Herremans who gave me some precious advices at the beginning of my project.

TABLE OF CONTENTS

1	Introduction	1
2	Background	2
2.1	Music representation	2
2.1.1	Musical stave	2
2.1.2	MIDI	4
2.1.3	Pianoroll	5
2.2	Music theory	5
2.2.1	Scale and Rhythm	6
2.2.2	Rhythm	7
2.2.3	Harmonics	7
2.3	Music arrangement	10
2.4	Neural Network architectures	10
2.4.1	Convolutional neural network	10
2.4.2	AutoEncoder	11
2.4.3	Variational AutoEncoder	12
2.4.4	Generative Adversial Network	13
2.4.5	Recurrent Neural Networks	13
2.4.6	Transformers	14
2.4.7	Multimodel Variational AutoEncoder	14
2.4.8	Muldimodal Deep Markov Model	16
2.4.9	Neural Autoregressive Distribution Estimation	16
2.4.10	Restricted Boltzmann Machine	17

3	Related works	19
3.1	Objectives	19
3.1.1	Generator	19
3.1.2	Accompaniment	20
3.2	Music Representation	20
3.2.1	Audio Signal	20
3.2.2	MIDI Signal	21
3.3	Encoding	24
3.3.1	Features	24
3.3.2	Tensor encoding	25
3.4	Architectures	26
3.4.1	RBM	26
3.4.2	NADE	26
3.4.3	AE	27
3.4.4	VAE	27
3.4.5	RMVAE	27
3.4.6	GAN	28
3.4.7	Reinforcement learning	28
3.5	Generation Process	28
3.5.1	Sampling	28
3.5.2	Input Manipulation	29
3.5.3	Feed forward and RNN architecture	29
4	Contribution	31
4.1	Objectives	31
4.2	Data representation	31
4.2.1	Polyphonic music	32
4.2.2	Monophonic Music	33
4.3	RMVAE Architecture	34

4.3.1	Global Architecture	34
4.3.2	Encoder	34
4.3.3	PoE	36
4.3.4	Recurrent layers	36
4.3.5	Decoder	38
4.3.6	Last layer	38
4.4	Loss Function	40
4.4.1	Scale	41
4.4.2	Rhythm	41
4.4.3	Harmony	42
5	Experiments	47
6	Conclusion	48
7	Appendices	54
.1	Interpolation project	54
.2	Segment Sum	54
.3	Roll _n	55
.4	BandPlayer	55
.5	Coconet Process	55

SUMMARY

My asbtract text

<https://github.com/ValentinVignal/midiGenerator>

LIST OF TABLES

2.1	Note names and duration	4
2.2	Harmonics of A_4	8
4.1	Correspondence between duration value and musical length . .	33

LIST OF FIGURES

2.1	Musical Stave example	2
2.2	Notes on a musical stave	3
2.3	Notes on Piano	3
2.4	Pianoroll example from the software FL Studio	5
2.5	C Major Pentatonic scale (or A Minor Pentatonic scale) . . .	6
2.6	Resonance waveform	8
2.7	Lead voice and its harmony part	9
2.8	Max pooling operation	11
2.9	AutoEncoder	12
2.10	Recurrent Neural Network	13
2.13	Graphical model of the MVAE	14
2.11	LSTM cell	14
2.12	GRU cell	14
2.14	MVAE architecture	15
2.15	NADE architectue	16
2.16	RBM architecture	17
3.1	Example of an audio waveform	20
3.2	Example of audio spectrogram	20
3.3	Example of correspondence between a pianoroll representation and an array	21
3.4	(a) tonnetz and (b) the extended tonnetz matrix with pitch register	22

3.5	Example of correspondence between a pianoroll representation and a text	22
3.6	BachBot's example encoding of three musical chords ending with a fermata ("pause") chord	23
3.7	Example of correspondence between a pianoroll and its chords representation	24
3.8	Fermata symbol	25
3.9	Feed forward generation process	29
3.10	Graphical representation of DeepBach's neural network archi- tecture for the soprano prediction	30
4.1	Architecture of the RMVAE	35
4.2	RMVAE encoder architecture	36
4.3	RMVAE PoE Fully Connected layers	37
4.4	RMVAE LSTM layer	37
4.5	RPOE architecture	38
4.6	RMVAE decoder architecture	39
4.7	Scale Loss	42
4.8	Rhythm Loss	43
4.9	Harmony Circle for A	44
4.10	Harmony Loss	45
4.11	Harmony _{n} Loss	46
1	Segment sum operation	54
2	Roll ₂ example	55
3	COCONET process	56

LIST OF SYMBOLS

AE	Auto Encoder
AI	Artificial Intelligence
AMAE	ArgMax AutoEncoder
C-RBM	Convolutional Restricted Boltzmann Machine
CNN	Convolutional Neural Network
FC	Fully Connected
GAN	Generative Adversial Network
GRU	Gated Recurrent Unit
KLD	Kullback-Leibler Divergence
LSTM	Long Short-Term Memory
MCMC	Markov Chain Monte Carlo
MDMM	Multimodal Deep Markov Model
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	Vectore Quantisation - Variational AutoEncoder
VRAE	Variational Recurrent AutoEncoder

CHAPTER 1

Introduction

My Introduction

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.

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 :

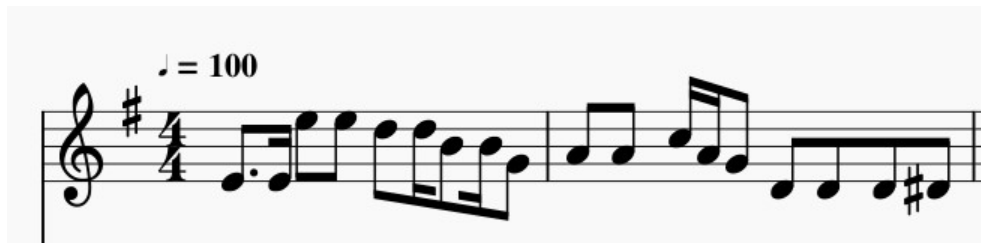
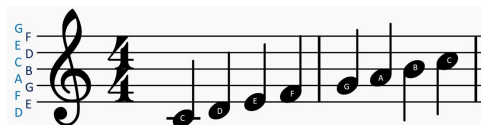


Figure 2.1: Musical Stave example

The vertical axis is the frequency axis and the horizontal axis corresponds to the time axis. In the figure 2.1, we know that the tempo is $100BPM$, that the scale is *G major* and the measure are divided in 4 beats (4/4 inscription). As said previously, the vertical position of a note indicates its frequency :



Name	Duration	Symbol
Whole note	4 beats	♩
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

2.1.2 MIDI

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 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 write an event as follows :

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

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 (like release the keyboard key). The given parameters should be the same as the ones given to the *Note On* event.

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

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 should specify how many ticks there are per quarter note.

2.1.3 Pianoroll

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

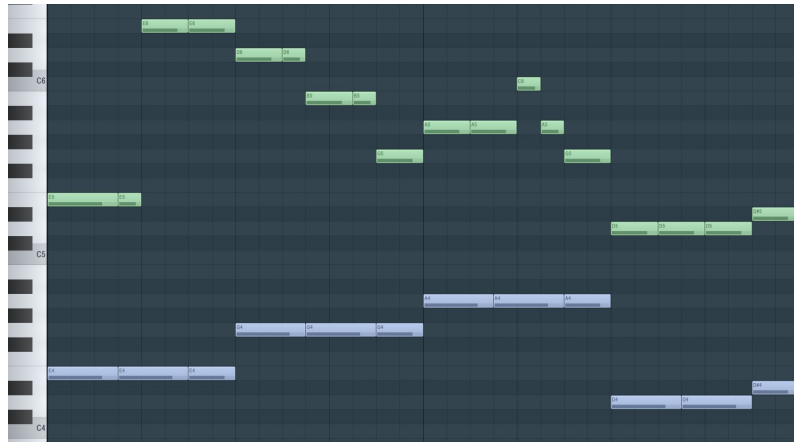


Figure 2.4: Pianoroll example from the software FL Studio

The figure 2.4 shows a view of the pianoroll in the famous software *FL Studio*. 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 the music theory which are related to this work. The rules I am going to talk about are the more common ones and most of the songs are following them.

2.2.1 Scale and Rhythm

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 *solos* are the *Pentatonic* scales.

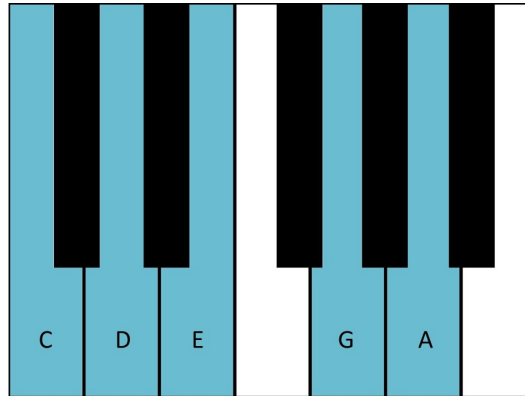


Figure 2.5: C Major Pentatonic scale (or A Minor Pentatonic scale)

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.

2.2.2 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 repeat some patterns. It helps the listener to easily follow the progression and allows him to listen what he's expecting.

However, there are no such rules for rhythm as there are for scales and harmonies. 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 rhythm.

2.2.3 Harmonics

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

Harmonics

A sound is a sum of harmonics:

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

With f the fundamental frequency and ϕ the phase difference value.

Let's take an example with the *A4* note which has its fundamental frequency equals to $440Hz$ for a common tuning. Then the 2^{nd} harmonic is *A5* $880Hz$. So, when an instrument plays a *A4*, all the harmonics of a *A5* are also present. Let's take one step further, the 3^{rd} 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 first harmonics of the note *A4*.

Harmonic number	Frequency (Hz)	Note Name	Musical Interval
1	440	$A4$	Unison
2	880	$A5$	Octave
3	1320	$E5$	Fifth
4	1760	$A6$	Octave
5	2200	$C\#6$	Major Third
6	2640	$E6$	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).

Chords

And this is why a Major chord sound nice to the ears. 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 (*Fifth* interval and *Third Major* interval respectively)

Dissonance

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

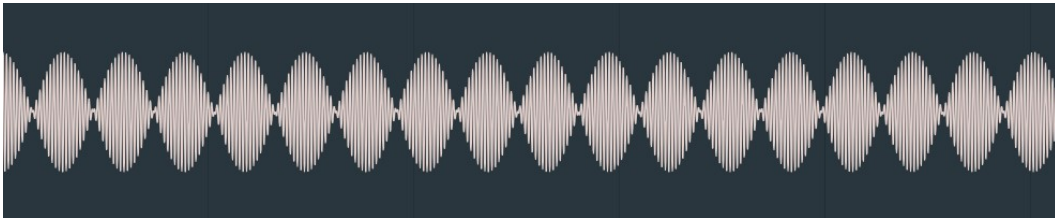


Figure 2.6: Resonance waveform

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

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

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

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

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 using all the previous observations and rules to create the harmony parts of a song given the lead melody. It

2.4 Neural Network architectures

2.4.1 Convolutional neural network

The convolutional networks are often used on images because they can preserve the spatial information. A CNN can include two types of layers :

- A convolutional layer
- A pooling layer

Convolutional Layer

For a 2D convolution, the convolutional layer which takes as 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 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.

Pooling Layer

A pooling layer is used to reduce the size of a tensor. It extracts a value from a region of the tensor. It can be:

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

The figure 2.8 shows how the max pooling operation works.

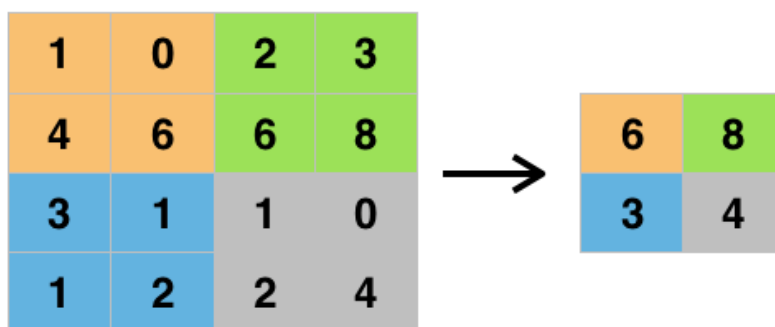


Figure 2.8: Max pooling operation
Source: Wikipedia

2.4.2 AutoEncoder

An AE [31, 40, 36] 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 is supposed to force the network to compress the input and reduce its dimensions. The AE is then forced to learn "high level features" about the inputs. The figure 2.9 illustrates this architecture.

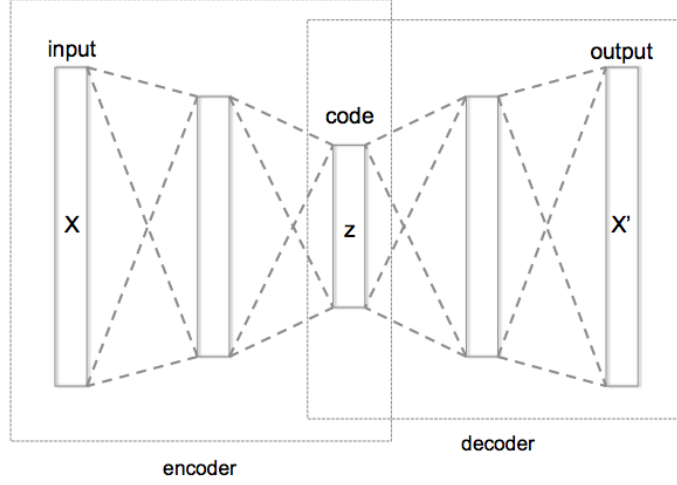


Figure 2.9: AutoEncoder
Source: Wikipedia

2.4.3 Variational AutoEncoder

A VAE [11, 5, 4, 7, 25] 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 decoder 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 as 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 :

$$\mathbb{D}_{KL}(\mathcal{N}(\mu_{encoded}, \sigma_{encoded}), \mathcal{N}(0, 1)) \quad (2.6)$$

2.4.4 Generative Adversial Network

GAN [2, 15, 16] are composed of two models that are trained simultaneously:

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

The discriminator can be trained with real images and images generated by the generator. The goal of the discriminator is to classify the real images as "*Real*" and the generated images as "*Fake*". On the other side, the generator takes some noise as an input to generate an image. And his goal is to make the discriminator classify his generated images as "*Real*".

2.4.5 Recurrent Neural Networks

A RNN is a neural network where connections between nodes form a direct graph along a temporal sequence. The general architecture is showed in the figure 2.10.

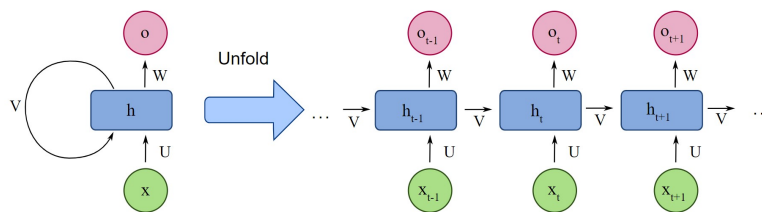


Figure 2.10: Recurrent Neural Network
Source: Wikipedia

The most used cells used are the LSTM cell and the GRU cell. The architectures are showed in the figures 2.11 and 2.12.

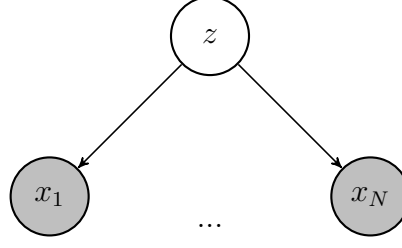


Figure 2.13: Graphical model of the MVAE

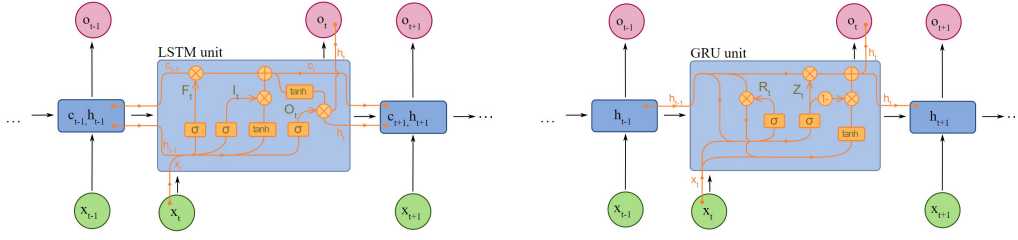


Figure 2.11: LSTM cell
Source: Wikipedia

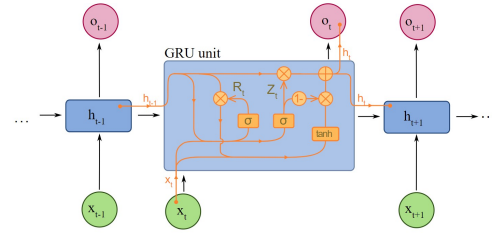


Figure 2.12: GRU cell
Source: Wikipedia

2.4.6 Transformers

2.4.7 Multimodal Variational AutoEncoder

The MVAE has been introduced by Mike Wu et al. [44]. The figure 2.13 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.

The conditional independence assumptions in the generative model (figure 2.13) imply a relation among joint- and single-modality posteriors :

$$\begin{aligned}
p(z|x_1, \dots, x_N) &= \frac{p(x_1, \dots, x_N|z)p(z)}{p(x_1, \dots, x_N)} \\
&= \frac{p(z)}{p(x_1, \dots, x_N)} \prod_{i=1}^N p(x_i|z) \\
&= \frac{p(z)}{p(x_1, \dots, x_N)} \prod_{i=1}^N \frac{p(z|x_i)p(x_i)}{p(z)} \\
&= \frac{\prod_{i=1}^N p(z|x_i)}{\prod_{i=1}^{N-1} p(z)} \frac{\prod_{i=1}^N p(x_i)}{p(x_1, \dots, x_N)} \\
&\propto \frac{\prod_{i=1}^N p(z|x_i)}{\prod_{i=1}^{N-1} p(z)} \approx \frac{\prod_{i=1}^N (\tilde{q}(z|x_i)p(z))}{\prod_{i=1}^{N-1} p(z)} = p(z) \prod_{i=1}^N \tilde{q}(z|x_i)
\end{aligned} \tag{2.7}$$

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

The PoE can be used , including a “*prior expert*”, as the approximating distribution for the joint-posterior.

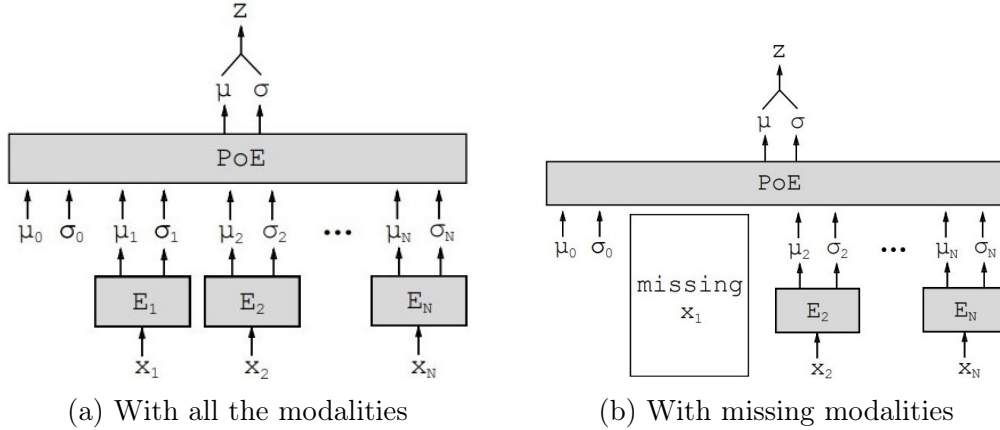


Figure 2.14: MVAE architecture
Source: Mike Wu’s paper [44].

The figure 2.14 shows how the PoE can handle missing modalities by simply ignoring them.

2.4.8 Muldimodal Deep Markov Model

2.4.9 Neural Autoregressive Distribution Estimation

A NADE [41, 42] is a neural network made to support a D -dimensional distribution $p(x)$ which verify:

$$p(x) = \prod_{d=1}^D p(x_{o_d} | x_{o_{<d}}) \quad (2.8)$$

It is a feed forward network parameterized as follow:

$$p(x_{o_d} = 1 | x_{o_{<d}}) = \text{sigm}(V_{o_d}, h_d + b_{o_d}) \quad (2.9)$$

$$h_d = \text{sigm}(W_{.,o_{<d}} x_{o_{<d}} + c) \quad (2.10)$$

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 2.15 illustrates the Nade model. There is no path of connections

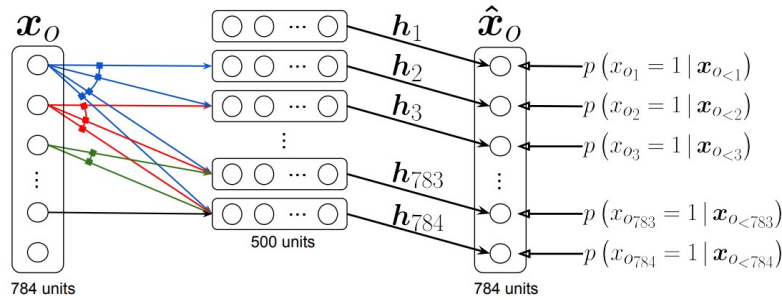


Figure 2.15: NADE architectue
Source: Benigno Uria et al.'s paper [41]

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.

2.4.10 Restricted Boltzmann Machine

A RBM is an undirected graphical model [3, 30, 37, 13] showed in the figure 2.16. As an AE, it encodes the input x in a latent space h where:

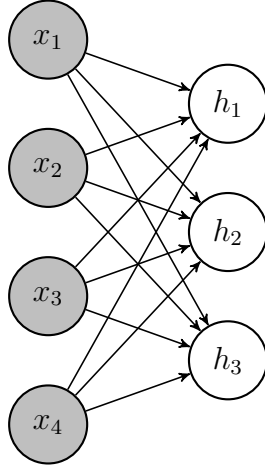
$$h_i = \text{sigmoid}(x^T W_i + a) \quad (2.11)$$

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.

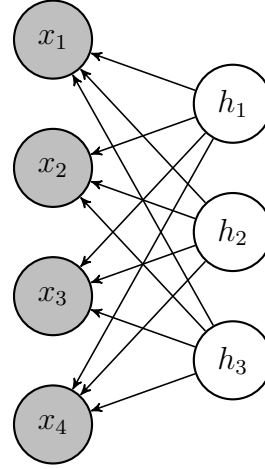
For the decoding, or reconstruction phase, the predicted output by the model is:

$$x_i^* = \text{sigmoid}(h W^T + b) \quad (2.12)$$

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



(a) RBM forward pass



(b) RBM backward pass

Figure 2.16: RBM architecture

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

ergy given by:

$$E(x, h) = - \sum_i a_i x_i - \sum_j b_j h_j - \sum_{(i,j)} x_i h_j w_{ij} \quad (2.13)$$

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

$$p(x^*) = \frac{1}{Z} \sum_h e^{-E(x^*h)} \quad (2.14)$$

where Z is the partition function:

$$Z = \sum_{(x,h)} e^{-E(x,h)} \quad (2.15)$$

Training a RBM consist by 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 work that has already been done about music generation with deep neural networks. In a first time, I will expose the different objectives of some implementations. Then I will introduce

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, the goal of this project is to create a single model able to do everything.

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).

It is also possible to create several musical parts at the same time. Each of them can be either monophonic or polyphonic. The musical parts can be considered as different instruments or voices for a Chorale. The challenge is to create musical parts that work together. This is the most common objective choice among the existing works [24, 9, 20, 8, 23].

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.

This is for example the goal of DeepBach from Gaëtan Hadjeres et al.'s paper [17].

3.2 Music Representation

In this section, I will present different types of inputs the recent works are using.

3.2.1 Audio Signal

The first of data used to create music is the audio signal. Some works [34, 10, 12, 21, 27] have been done to generate some an music audio signal. This signal can be represented either by its waveform [34], its Fourier Transform, or its spectrogram (figures 3.1, 3.2).

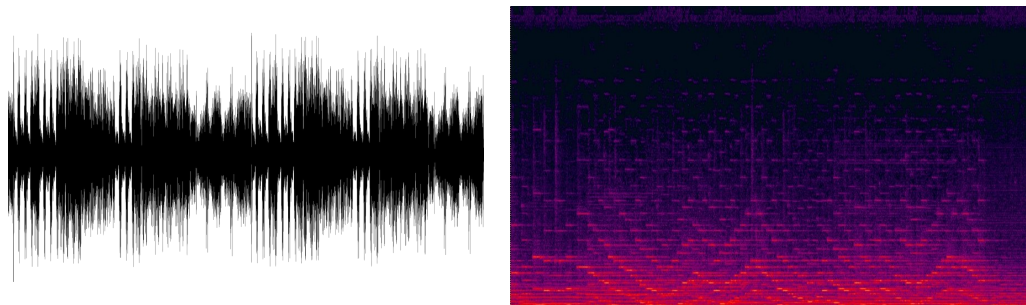


Figure 3.1: Example of an audio wave-

Source: Needpix.com

Figure 3.2: Example of audio spectro-

gram

Source: Wikipedia

Using the audio signal allows the model to handle every aspect of the song at the same time, and every song in the same way. The biggest issue of this method is that the number of points in time is incredibly huge. The usual

sampling frequency is $48kHz$. Therefore, the challenge is to stay consistent through time.

3.2.2 MIDI Signal

Most of the work done on music generation is using the *MIDI* representation (or any other translation of midi like pianoroll or text). [9, 17, 20, 24, 6, 19, 18, 8, 23]

Pianoroll

The pianoroll representation can be considerate as an image so usual deep learning methods on images can be applied [20, 9, 8, 23]. The figure 3.3 shows a very naive example of how to convert a pianoroll view to an array.



Figure 3.3: Example of correspondence between a pianoroll representation and an array

Chin-Hua Chuan et al. [9] introduced another representation which is supposed to help the model to understand relations between notes. They use *Tonnetz* matrix [26] 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 the notes in a one-dimensional vector (figure 3.4a), they encode them in a 2-dimensional vector where the relative positions between two notes is meaningful.

The figure 3.4a 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 3.4a 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 3.4b shows an extended tonnetz matrix example. They include in this extended matrix the pitch (octave number) which was missing in the first one.

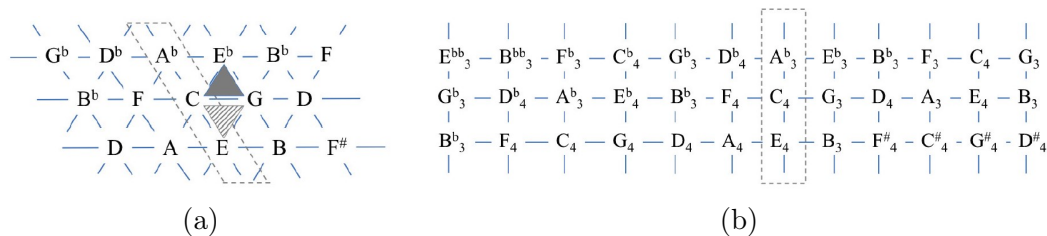


Figure 3.4: (a) tonnetz and (b) the extended tonnetz matrix with pitch register
Source: Ching-Hua Chuan's paper [9].

Text

Another approach is to convert the MIDI format into text. [17]



Figure 3.5: Example of correspondence between a pianoroll representation and a text

The figure 3.5 shows as simple example of how it is possible to convert an pianoroll view to a text. However, using text to represent image usually force the framework to work only in a monophonic way. A common choice of the existing works that are working with is text is to embed the text into a fix-sized vector using a `word2vec` [14, 22, 35, 1, 28] model [24, 18].

Despite the text constraint, Feynman Liang and al. [24] managed to encode polyphonic music in text and use it to correctly train their model: "BachBot".

They quantizer 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 3.6 shows the encoding process.

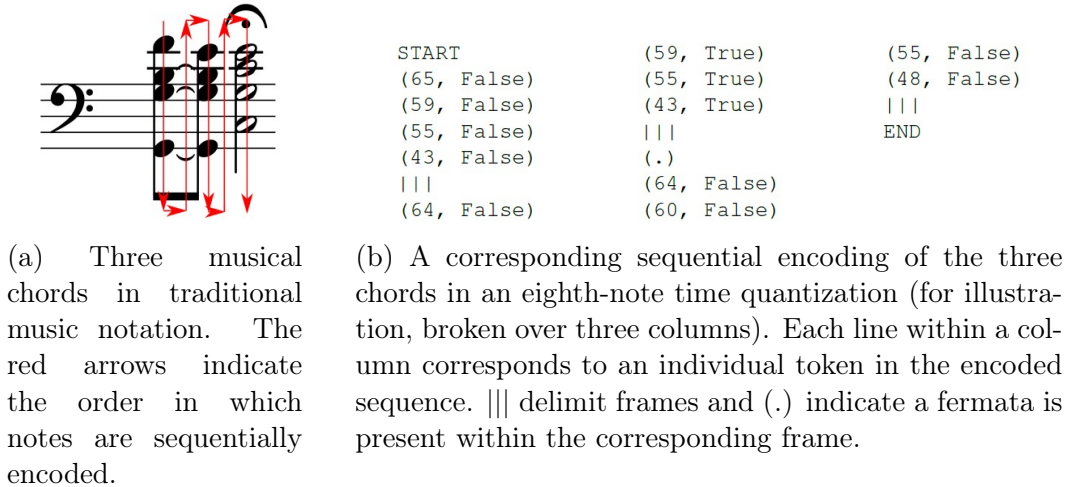


Figure 3.6: BachBot's example encoding of three musical chords ending with a fermata ("pause") chord

Source: Feynman Lian's paper [24].

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.

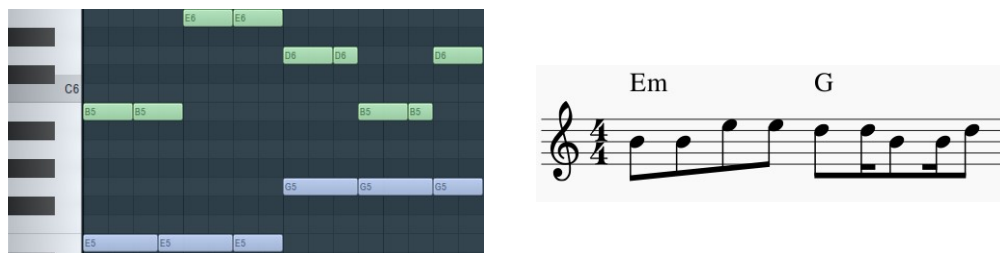


Figure 3.7: Example of correspondence between a pianoroll and its chords representation

For instance, in the figure 3.7, 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.

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 [17] consider the fermata symbol (figure 3.8) and the beat subdivision number (an integer between 1 and 4).



Figure 3.8: 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 not intentionally 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 [24] and other works [9, 8] transpose all their dataset in either C major or A major key.

For the same reason as not passing metadata, I don’t normalize the data before training my models.

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 is means that the values in the input and/or output tensor are actually meaning full. 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 this work.

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 2.4.10. Nicolas Boulanger-Lewandowski et al. [8] use an RBM based model RTRBM [38, 29] and created their own architecture called RNN-RBM. They applied their model on different MIDI dataset to create polyphonic music.

Stefan Lattner et al. [23] use another RBM based model called C-RBM [32, 33]. 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 2.4.9.

And as an example, Cheng-Zhi Anna Huang et al. [20] use an orderless NADE [42] to generate music. They introduce COCONET, a deep convolutional model to reconstruct partial scores. The process consists of a corruption process that masks out a subset x_{-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.

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 [24, 9, 17, 21, 27] as Feynman Liang et al.'s BachBot [24] or not [10].

For instance, BachBot [24] and DeepBach [17] use an encoder/decoder architecture with LSMT layers on the latent space. [9]

Soroush Mehri et al. [27] or Nal Kalchbrenner et al. [21] 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. [10] use a VAE (or more precisely a VQ-VAE [43] or a AMAE [10]) to encode audio waveform and reconstruct it with a stylistic consistency accross tens of seconds.

3.4.5 RMVAE

Since I created this new architecture, there is no existing work using this architecture. 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. [12] use a GAN architecture to synthesize audio waveform in several domains including drums and piano.

3.4.7 Reinforcement learning

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 [12].

A VAE keeps its latent space distribution 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

3.5.3 Feed forward and RNN architecture

This is the easiest and most common generation process through all the work. [24, 9, 20] And input is given to the model. The model returns an output. The figure 3.9 illustrates this process. This output can be the next frame/beat-/measure of the music.

I also chose this generation process for my project. In my case, the model returns the next measure of the music.

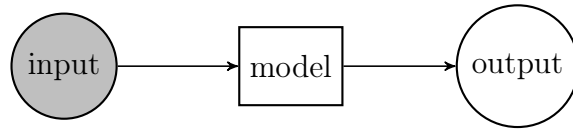


Figure 3.9: Feed forward generation process

DeepBach

DeepBach [17] uses a Markov Chain Monte Carlo algorithm to re-harmonize a song and transform it. From the existing song, it re-predicting every notes one by one. The algorithm is describe in the figure

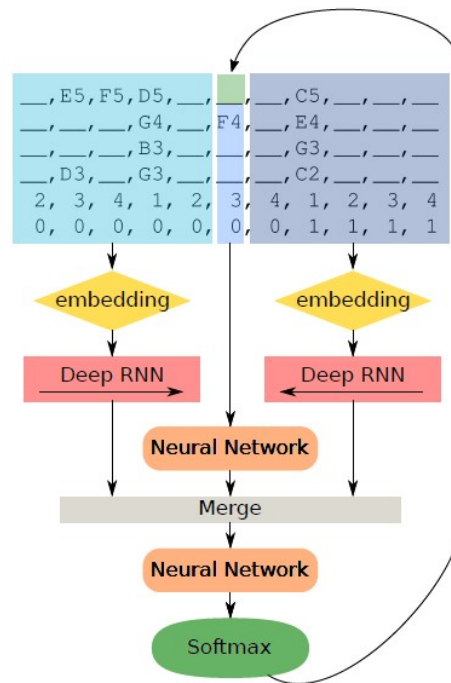


Figure 3.10: Graphical representation of DeepBach’s neural network architecture for the soprano prediction

Source: DeepBach paper [17]

CHAPTER 4

Contribution

4.1 Objectives

As a musician, I wanted to create an 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 music part from other musical parts

The created model can also handle missing data (for example a missing measure from a 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 shape of a tensor representing a measure will be (16, 128, `channels`):

- 16 is the number of sixteenth notes (♪) in 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` is 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.

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

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

















Duration value	Musical length
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	

Table 4.1: Correspondence between duration value and musical length

4.2.2 Monophonic Music

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

Thus, I get read of duration channel and the number of **channels** = 1.

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

To do so, 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. [44] and their Multimodal Variational AutoEncoder (MVAE) and I simplified the work of Tan Zhi-Xuan et al. [39] 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 reason, I came up with a novel architecture that I call RMVAE (Recurrent Multimodal Variational AutoEncoder). This architecture is described in the figure 4.1.

4.3.2 Encoder

Each instrument (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)`. 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.

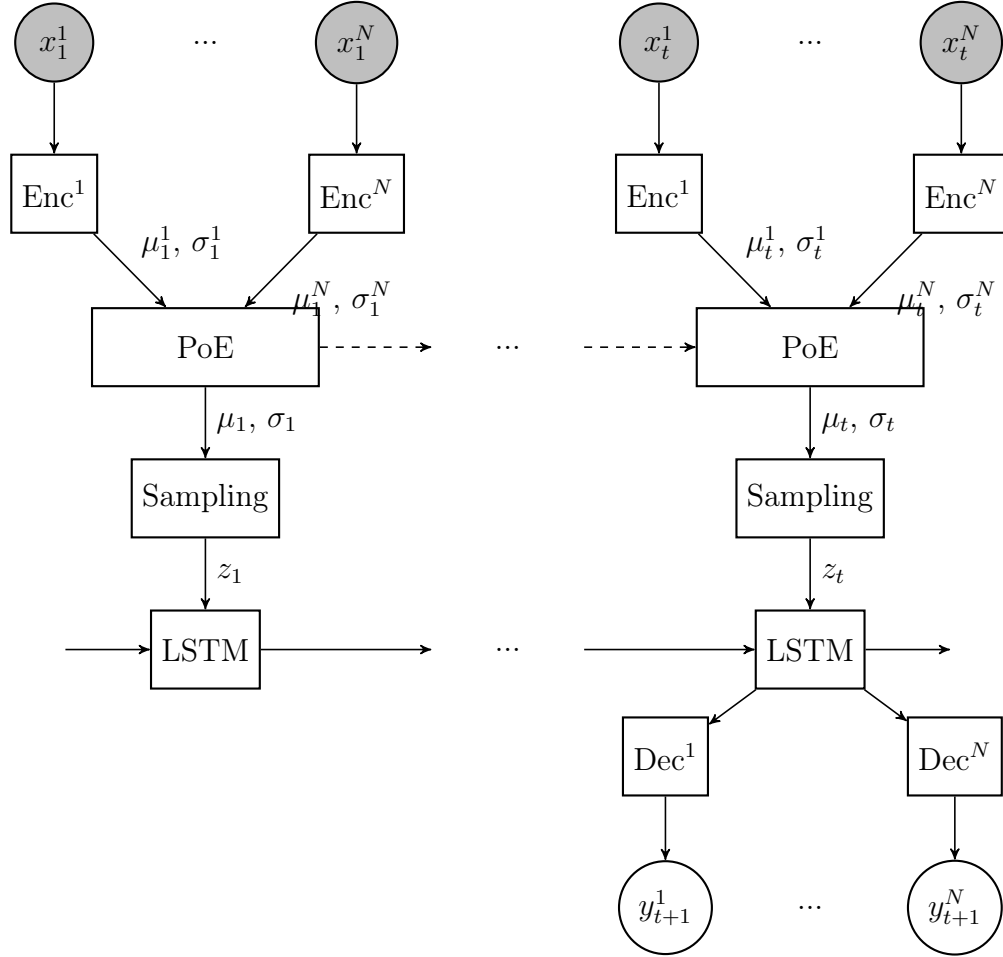


Figure 4.1: Architecture of the RMVAE

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.

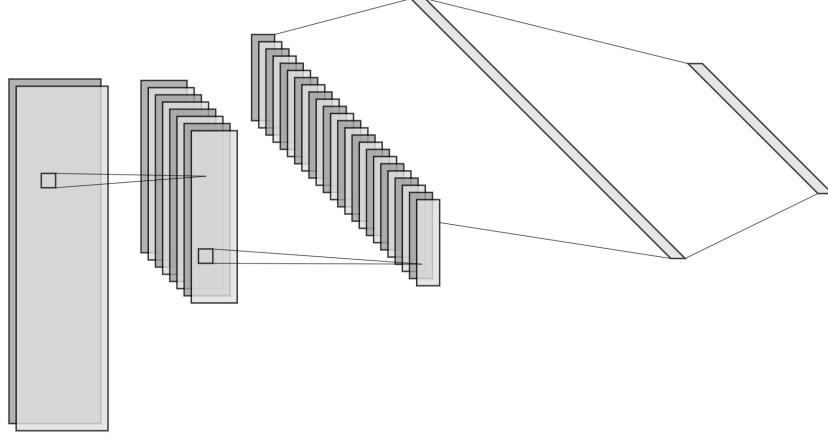


Figure 4.2: RMVAE encoder architecture

The filter sizes of the convolutional layers are $(5, 5)$. I chose this dimension so the NN can process a third major interval and an entier beat with only one layer.

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 distribution (figure 4.3).

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 2.14.

4.3.4 Recurrent layers

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

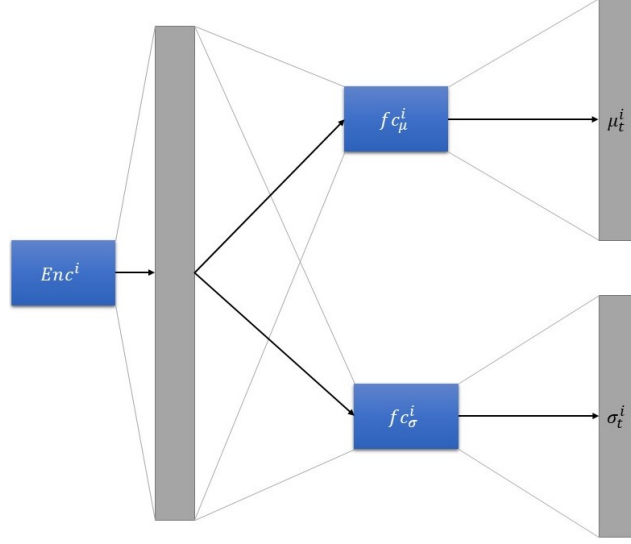


Figure 4.3: RMVAE PoE Fully Connected layers

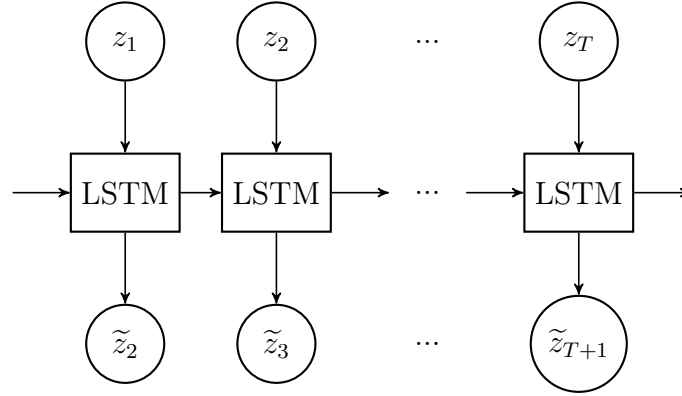


Figure 4.4: RMVAE LSTM layer

LSTM cells

To learn, how this latent space evolves through time, RMVAE uses recurrent layers with LSTM cells (figures 2.10 , 2.11). This is shown in the figure 4.4.

RPoE

To try to catch the time 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 showed in the figure 4.5.

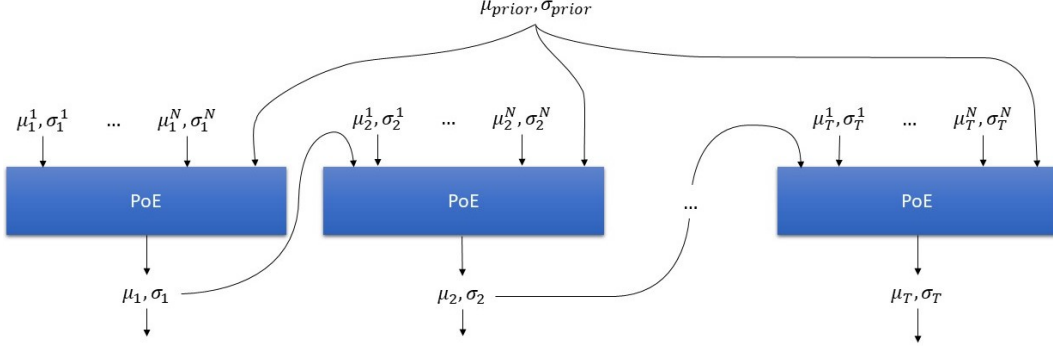


Figure 4.5: RPOE architecture

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 $((16, 128, \text{channels}))$.

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 .

The decoder global architecture is showed in the figure 4.6.

4.3.6 Last layer

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

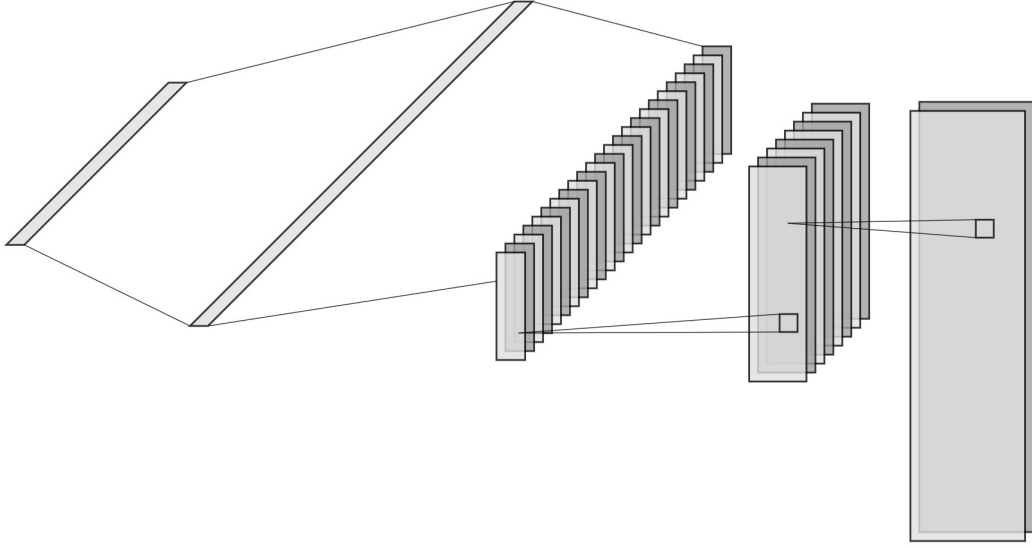


Figure 4.6: RMVAE decoder architecture

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.

Monophonic Music

For monophonic music (section 4.2.2), the activation function is a *sigmoid* function for the *additional note* (index 129: *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 apply a softmax across all the notes. But the result was disappointing: the highest frequency of *note_{continue}* in the dataset was too high. The network wasn't generating anything because of the *note_{continue}* had the highest probability and the network didn't want to generate any new note anymore.

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 will 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 *unacceptable mistakes*. In other words, during the training part, if the model doesn't hit the note it should have hit, I don't want to *punish* him 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 a 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.

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.

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
```

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:

$$Scale(truth, predict)$$

where the shape of the input tensors is `(nb_instruments, 16, 128)` for both which the represent the activated notes (where there is a 1) of every instruments for the next measure. The figure 4.7 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.

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

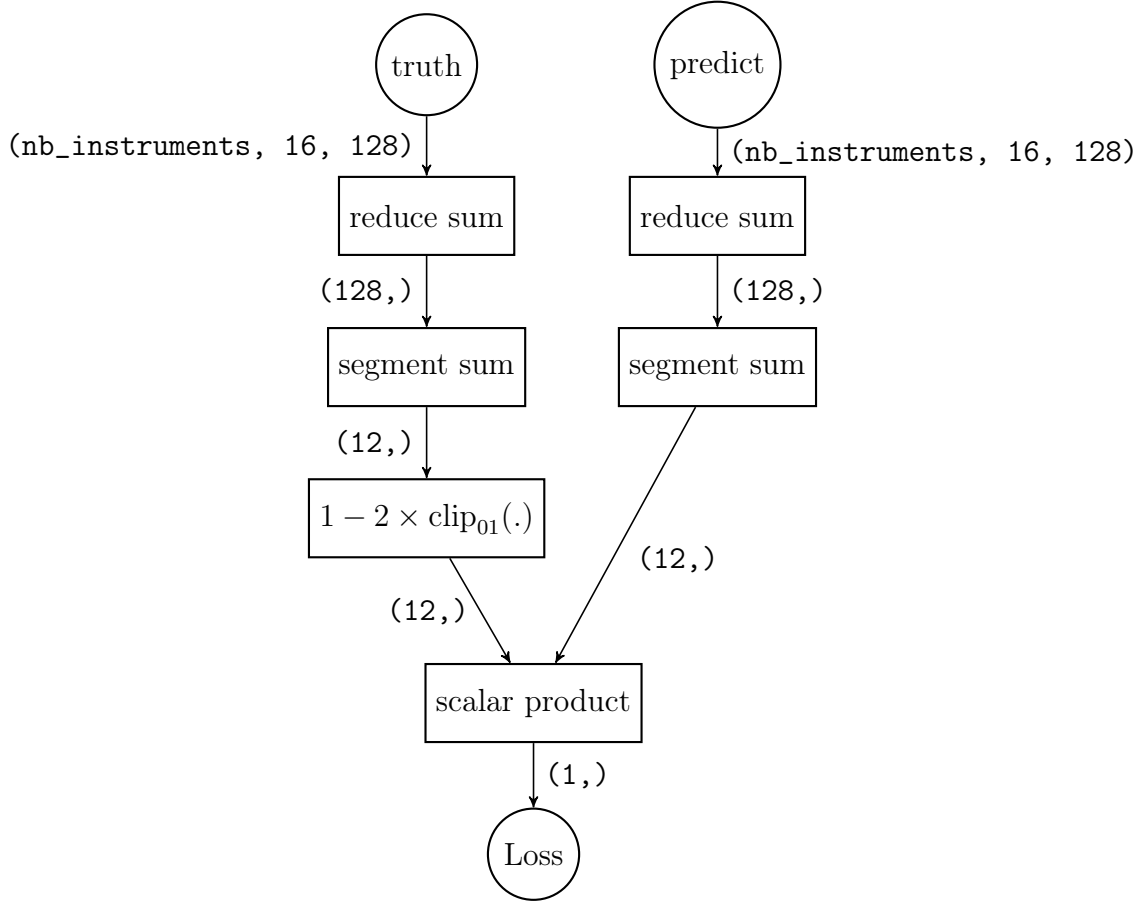


Figure 4.7: Scale Loss

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 4.8 is very similar as the Scale implementation.

4.4.3 Harmony

I explained in the section 2.2.3 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

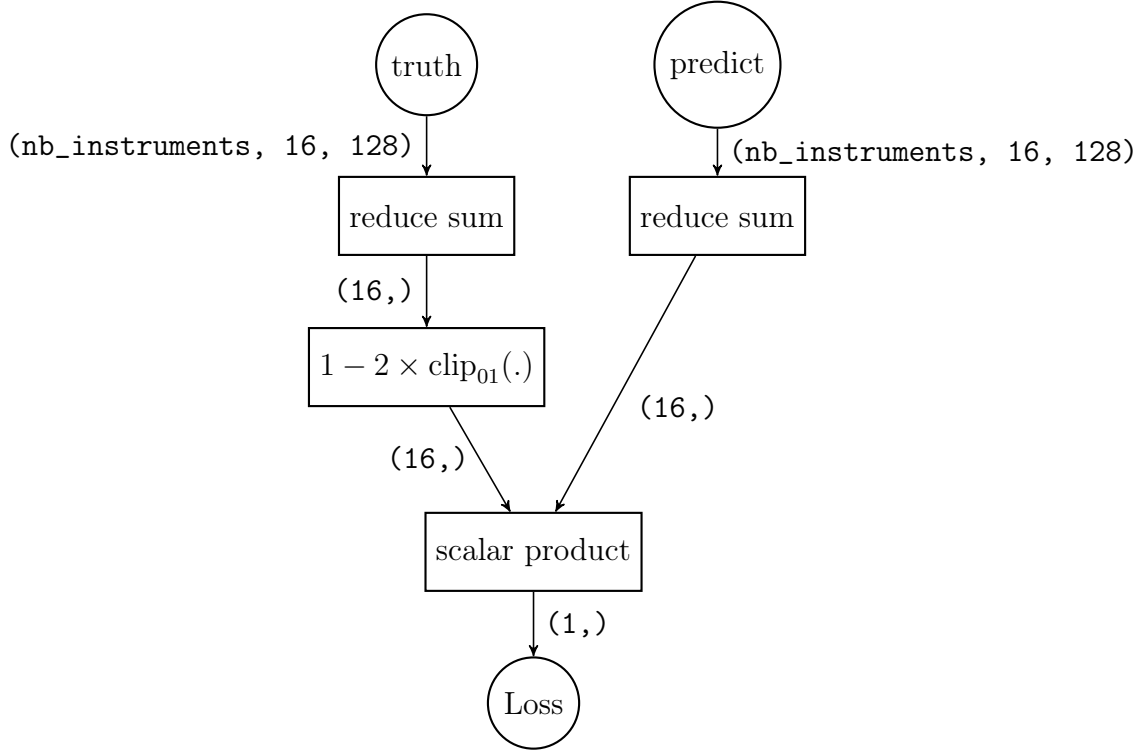


Figure 4.8: Rhythm Loss

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 4.9 shows the acceptable and non-acceptable musical intervals for the note *A*.

From this figure, I realized there are actually *3 forbidden musical intervals*:

- 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.

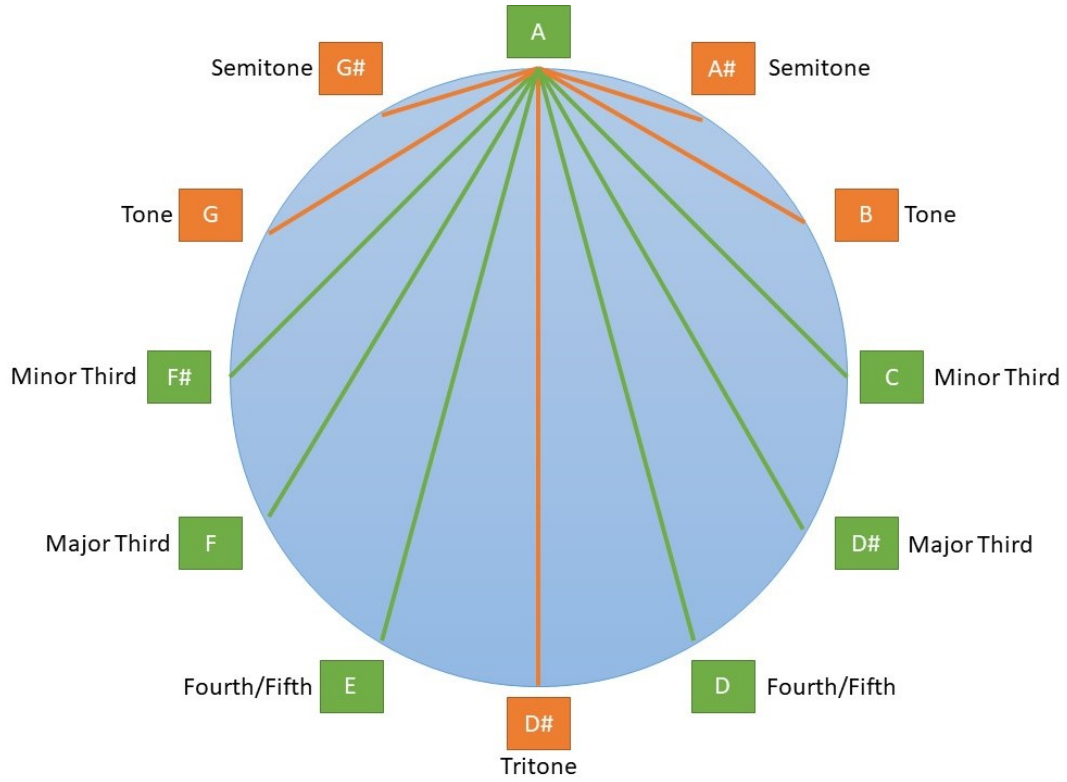


Figure 4.9: Harmony Circle for A

To prevent the model to generate such intervals, I created a loss function which gives a penalties every time there is one of this intervals. To do so, I created a subfunction harmony_n which penalize 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 4.10 and 4.11. The roll_n operation is explained in the appendix .3.

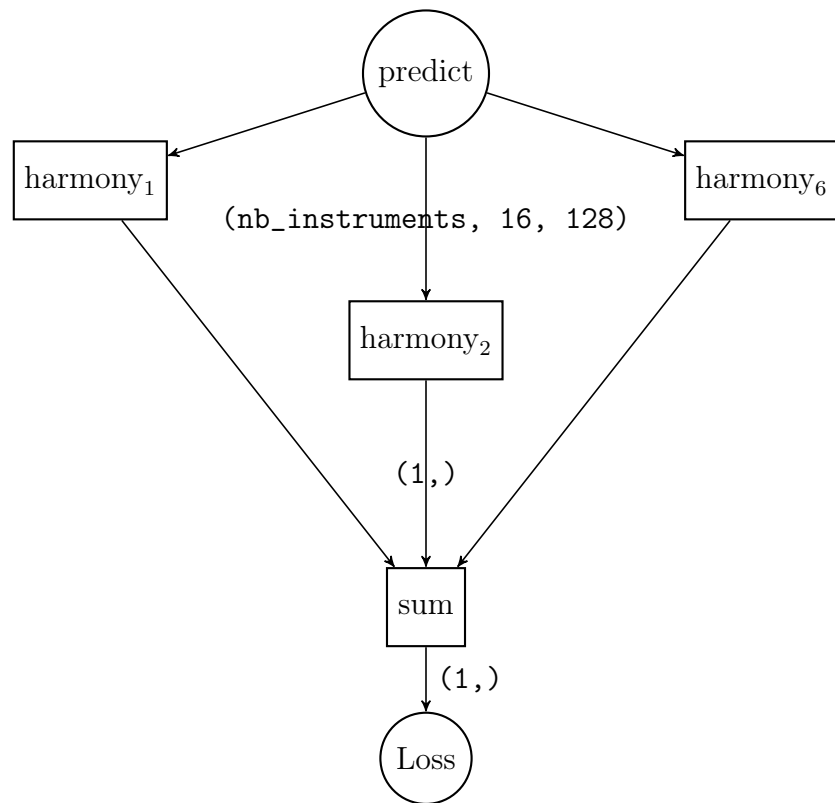


Figure 4.10: Harmony Loss

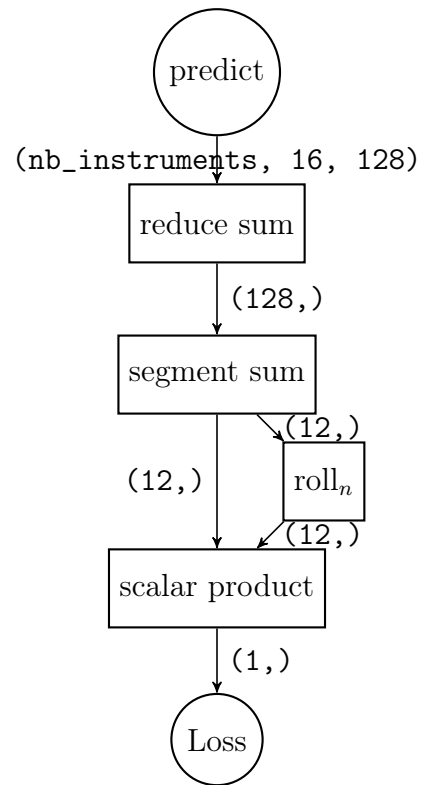


Figure 4.11: Harmony_n Loss

CHAPTER 5

Experiments

CHAPTER 6

Conclusion

Conclusion

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

Bibliography

- [1] A beginner’s guide to word2vec and neural word embeddings. Library Catalog: pathmind.com.
- [2] *GAN Deep Learning Architectures - review*.
- [3] Restricted boltzmann machine tutorial | deep learning concepts. Library Catalog: www.edureka.co Section: Artificial Intelligence.
- [4] Tutorial - what is a variational autoencoder? Library Catalog: jaan.io.
- [5] The variational auto-encoder.
- [6] Kamil Adiloglu and Ferda Alpaslan. A machine learning approach to two-voice counterpoint composition. 20:300–309.
- [7] Haleh Akrami, Anand A. Joshi, Jian Li, Sergul Aydore, and Richard M. Leahy. Robust variational autoencoder.
- [8] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription.
- [9] Ching-Hua Chuan and Dorien Herremans. Modeling temporal tonal relations in polyphonic music through deep networks with a novel image-based representation. page 8.
- [10] Sander Dieleman, Aaron van den Oord, and Karen Simonyan. The challenge of realistic music generation: modelling raw audio at scale. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and

- R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7989–7999. Curran Associates, Inc.
- [11] Carl Doersch. Tutorial on variational autoencoders.
 - [12] Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis.
 - [13] Asja Fischer and Christian Igel. An introduction to restricted boltzmann machines. pages 14–36.
 - [14] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.’s negative-sampling word-embedding method.
 - [15] Ian Goodfellow. Generative adversarial networks (GANs). page 86.
 - [16] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks.
 - [17] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. DeepBach: a steerable model for bach chorales generation.
 - [18] D Herremans. Modeling musical context using word2vec. page 8.
 - [19] D. Herremans and K. Sörensen. Composing fifth species counterpoint music with a variable neighborhood search algorithm. 40(16):6427–6437.
 - [20] Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron Courville, and Douglas Eck. COUNTERPOINT BY CONVOLUTION. page 8.
 - [21] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron van den Oord,

- Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis.
- [22] Dhruvil Karani. Introduction to word embedding and word2vec. Library Catalog: towardsdatascience.com.
- [23] Stefan Lattner, Maarten Grachten, and Gerhard Widmer. Imposing higher-level structure in polyphonic music generation using convolutional restricted boltzmann machines and constraints. 2(2).
- [24] Feynman T. Liang, Mark Gotham, Matthew Johnson, and Jamie Shotton. Automatic stylistic composition of bach chorales with deep LSTM. In *ISMIR*.
- [25] Wenqian Liu, Runze Li, Meng Zheng, Srikrishna Karanam, Ziyang Wu, Bir Bhanu, Richard J. Radke, and Octavia Camps. Towards visually explaining variational autoencoders.
- [26] Laura Felicity Mason. Essential neo-riemannian theory for today’s musician. page 98.
- [27] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. SampleRNN: An unconditional end-to-end neural audio generation model.
- [28] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.
- [29] Roni Mittelman, Benjamin Kuipers, Silvio Savarese, and Honglak Lee. Structured recurrent temporal restricted boltzmann machines. page 9.

- [30] Guido Montufar. Restricted boltzmann machines: Introduction and review.
- [31] Michael Moor, Max Horn, Bastian Rieck, and Karsten Borgwardt. Topological autoencoders.
- [32] Mohammad Norouzi. CONVOLUTIONAL RESTRICTED BOLTZMANN MACHINES FOR FEATURE LEARNING. page 61.
- [33] Mohammad Norouzi, Mani Ranjbar, and Greg Mori. Stacks of convolutional restricted boltzmann machines for shift-invariant feature learning. page 8.
- [34] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio.
- [35] Xin Rong. word2vec parameter learning explained.
- [36] Marco Rudolph, Bastian Wandt, and Bodo Rosenhahn. Structuring autoencoders.
- [37] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning - ICML '07*, pages 791–798. ACM Press.
- [38] Ilya Sutskever, Geoffrey Hinton, and Graham Taylor. The recurrent temporal restricted boltzmann machine. page 8.
- [39] Zhi-Xuan Tan, Harold Soh, and Desmond C. Ong. Factorized inference in deep markov models for incomplete multimodal time series.

- [40] Michael Tschannen, Olivier Bachem, and Mario Lucic. Recent advances in autoencoder-based representation learning.
- [41] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation.
- [42] Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator.
- [43] Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu. Neural discrete representation learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6306–6315. Curran Associates, Inc.
- [44] Mike Wu and Noah Goodman. Multimodal generative models for scalable weakly-supervised learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 5575–5585. Curran Associates, Inc.

CHAPTER 7

Appendices

.1 Interpolation project

.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.

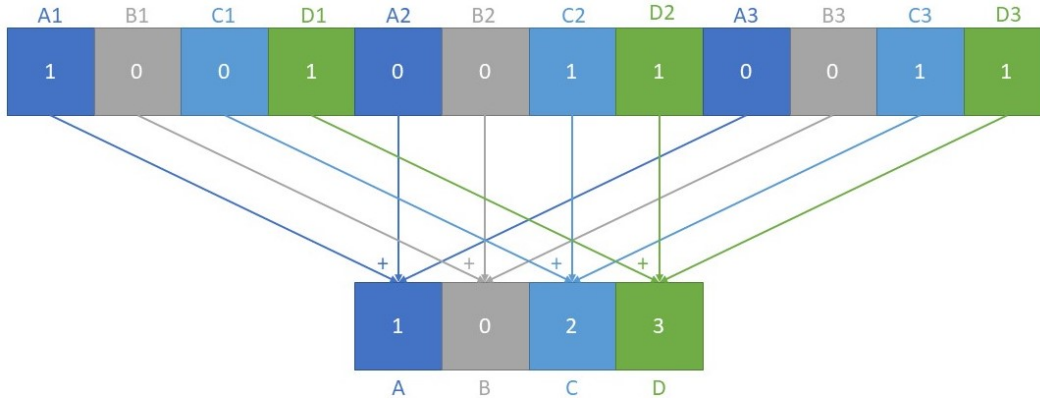


Figure 1: Segment sum operation

.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 2 explains how it works and the figure 2 shows an example with $n = 2$.

Algorithm 2 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
```

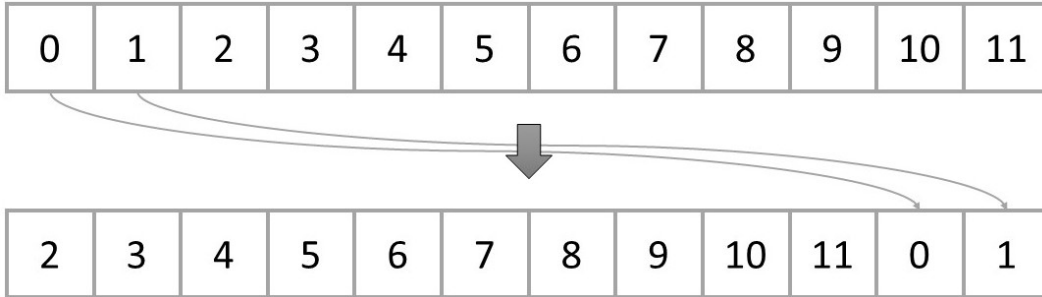


Figure 2: Roll₂ example

.4 BandPlayer

.5 Coconet Process

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

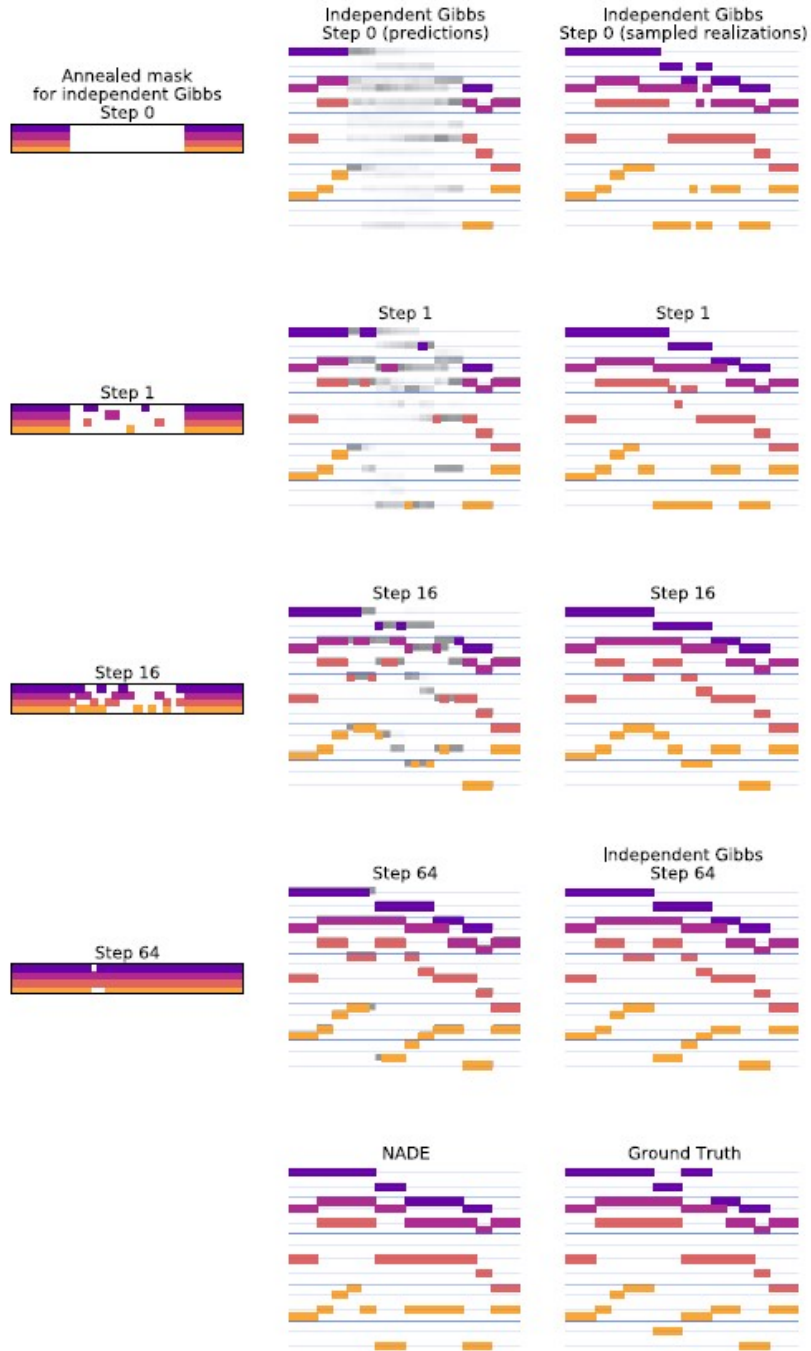


Figure 3: COCONET process
Source: Cheng-Zhi Anna Huang et al.'s paper [20]