Music completion 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

Supervisor:

Assistant Professor Harold Soh

Examiners:

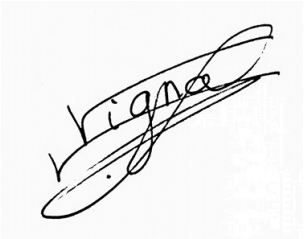
Assistant Professor Gim Hee Lee

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

22 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

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.

However, most of the related works about music generation with a deep neural network aim to perform one and only one task. The main goal of this Dissertation is to create a neural network architecture able to handle several tasks a musician or a composer could perform with only one trained model. These tasks are, generating one or several musical parts, creating an accompaniment from a melody, creating a melody from an accompaniment, creating one or several musical parts from one or several other musical parts.

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

As a Musician, I have some knowledge about musical rules and tendencies. Therefore, I tried to incorporate some prior knowledge to the neural network with some additional custom loss functions. I came up with 3 different functions that I named *Scale*, *Rhythm* and *Harmony*.

For this project, I used a 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`.

I constructed the **RMVAE** architecture in a way that, it is easy to im-

plement a script to handle different tasks. The tasks I have implemented are, generate several musical parts, complete several musical parts, create a musical part from other musical parts, and re-write a song by replacing one by one all the musical parts. Other tasks can easily be derived from the one I have already implemented.

Unfortunately, the created samples from the **RMVAE** don't match my expectations. The model is consistent through its generation, but the quality of the created samples is poor. The created notes are mostly Quarter notes (♩). The model has the tendency to generated only around 3 different notes per voice and repeat a simple sequence.

I conducted some experiments and created some google forms to evaluate whether the additional loss functions actually help the model or not. Much to my disappointment, the results of these experiments and polls weren't in favour of my custom losses. Indeed, the results with a custom loss function are either equivalent or worse than without.

LIST OF TABLES

LIST OF FIGURES

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 Bolzmann 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
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, harmonized 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], and 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, which means create a song, harmonize a melody, arrange a music or jam with someone...

While I was spending my time learning about music, training my musical skills or playing with other people, I have learnt, understood and integrated some musical concepts. Instead of letting the model learn everything from scratch from the data, I also want to give it some prior knowledge about music.

Thus, the first objective of my Dissertation is to create a unique trained model able to generate or complete a melody, generate or complete 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 introduce to the reader some background knowledge about music and deep learning. The chapter 3 will summarize what has already been done about music generation in deep learning, including how the data are represented, what are the objectives of these different works, what architecture they chose, and what is the generation process used. The chapter 4 will present what are my contributions and what novel ideas I had for my Dissertation. This will include the architecture I created (RMVAE) and how I created 3 additional loss functions to give some prior knowledge to the model. 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 musicians 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 my Dissertation.

2.1 Music representation

In this section I will explain how musicians represent music on paper, and from it, how it is possible to represent 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).

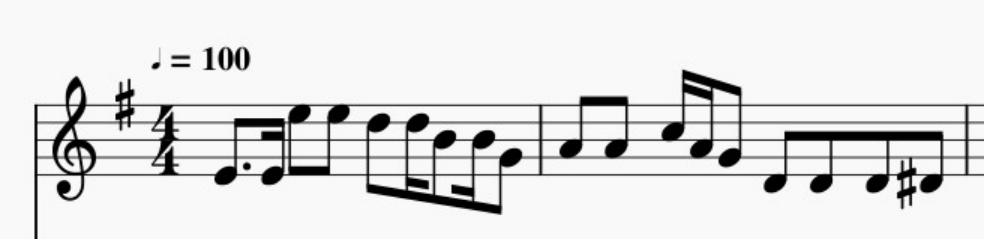


Figure 2.1: Musical Stave example

The vertical axis is the frequency axis an 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 ♯) and the measure are divided with 4 beats (4/4 inscription). As said previously, the vertical position of a note indicates its frequency.

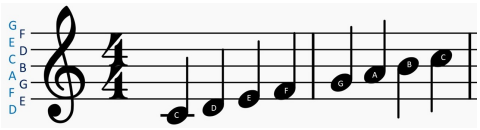


Figure 2.2: Notes on a musical stave

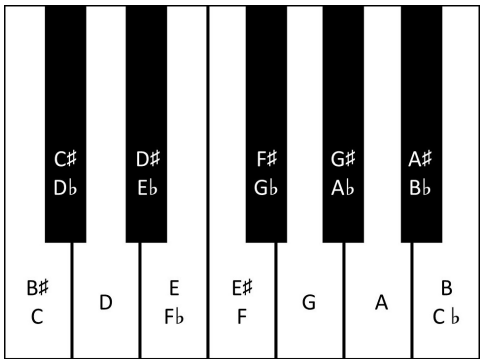


Figure 2.3: Notes on Piano

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 describes the relation between the length of a note, and its shape on the musical stave.

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 durations

In this work, I 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 works.

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 any musical devices. These messages are musical events. It can contains note information as the pitch, the velocity, or the panning, and some other parameters (for example, vibrato, volume...). The most important messages I will consider are:

- *Note on* which indicates 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 :

$$< NoteOn, 0, 50, 127 > \tag{2.1}$$

to describe the event *"Start to play the note D*3 *with the maximum velocity (127) for the channel (instrument) 0"*.

- *Note off* which indicates 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.

$$< NoteOff, 0, 50, 127 > \tag{2.2}$$

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

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

2.1.3 Pianoroll

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

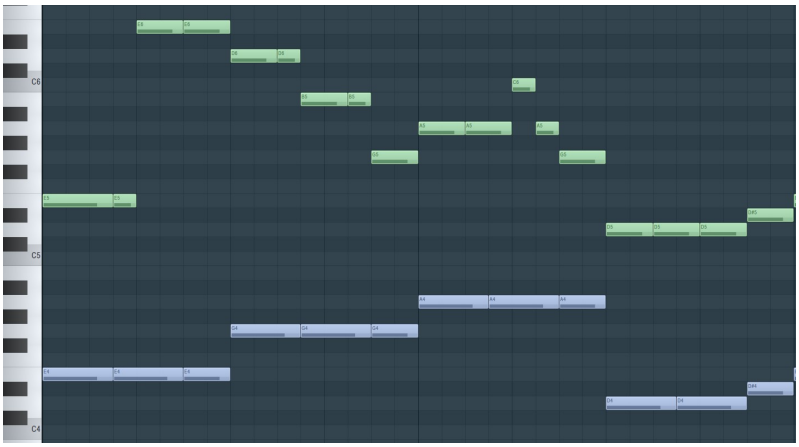


Figure 2.4: Pianoroll example from the software FL Studio [6]

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 and so on 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 one 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 scale, often used by the musicians to creates their *solos*, is the *Pentatonic* scale.

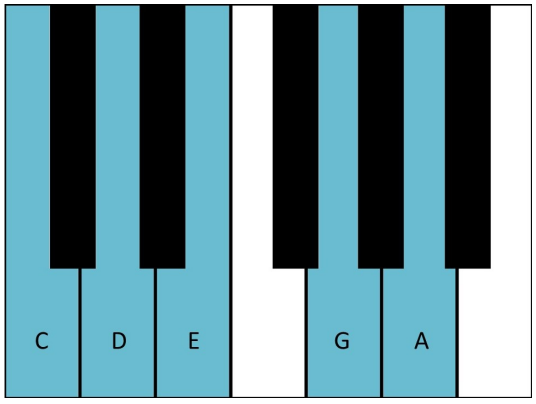


Figure 2.5: C Major Pentatonic scale (or A Minor Pentatonic scale)

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 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 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.

I 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 timbres and then illustrate how it can explain some musical rules.

2.2.2.1 Harmonics

A sound is a sum of harmonics (equation 2.3).

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

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 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 references the first harmonics of the $A4$ note.

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	$C\sharp 6$	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 $C\sharp$ notes are linked together (*Major Third* interval).

2.2.2.2 Chords

The links between notes illustrated in the table 2.2 explain why a Major chord sounds *nice* or *smooth*. A $A\ Major$ chord is composed with 3 notes : A, $C\sharp$ 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, a resonance phenomena is observed.

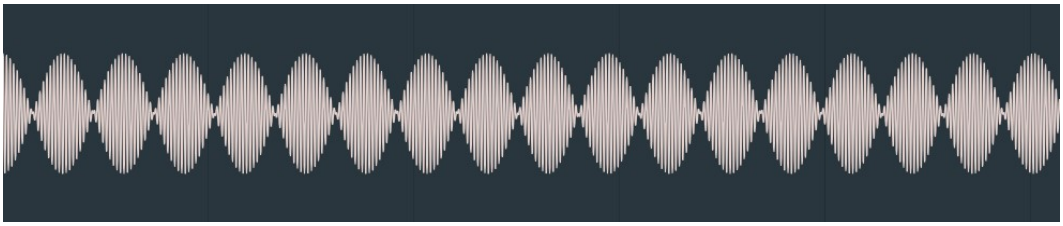


Figure 2.6: Resonance waveform

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 written in the equation 2.4.

$$\cos(f) + \cos(f + \delta f) = 2\cos(\frac{2f + \delta f}{2})\cos(\frac{\delta f}{2}) \tag{2.4}$$

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

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

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.

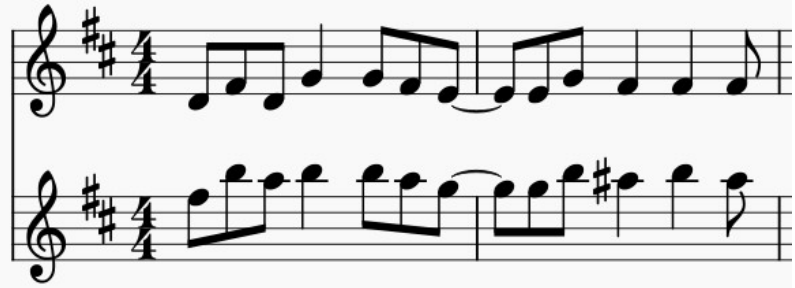


Figure 2.7: Lead voice and its harmony part

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 to an existing melody musical variety. To put it more simply, it means creating a accompaniment from a melody. To do so, the musician can include chords, change the rhythm, add other musical parts...

To give an example, arranging a song could be creating piano, guitar, drums and bass parts from a 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 neural networks are often used on images because of their ability to 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, a 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`.

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

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 pooling operations are:

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

The figure 2.8 illustrates how the max pooling operation works.

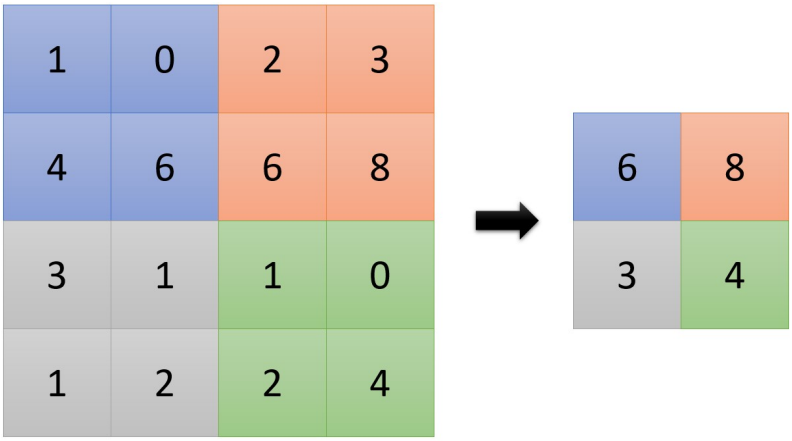


Figure 2.8: Max pooling operation

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 layer), 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. It forces the network to compress the input and reduce its dimensions. Therefore, the AE has to learn "high level features" about the inputs. The figure 2.9 summarizes this architecture.

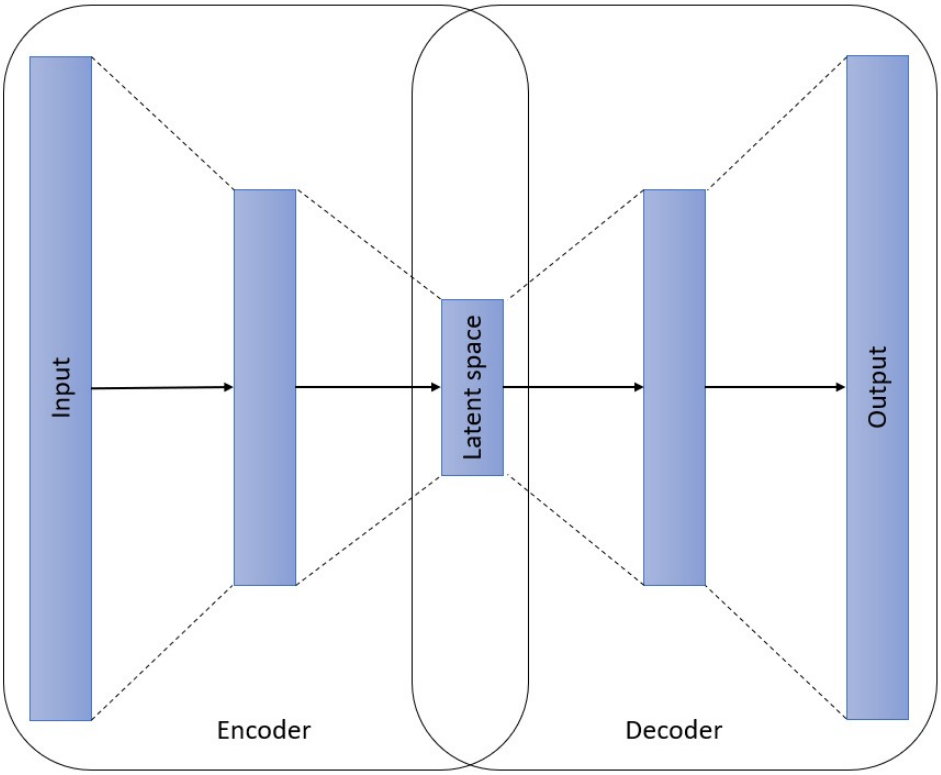


Figure 2.9: AutoEncoder
Source: Wikipedia

2.4.3 Variational AutoEncoder

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

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

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 $p_\theta(x|z)$ is a neural network, with the parameters θ composed with a simple likelihood (for example, a Bernoulli or Gaussian distribution). The training's objective is to maximize the marginal likelihood of the data (called the *"evidence"*) (equation 2.7).

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

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

$$\log(p_\theta(x)) = \log\left(\int_z p_\theta(x|z)p(z)dz\right) \tag{2.8}$$

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

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

The figure 2.10 shows the reparameterization trick. At training-time, VAE is implemented as a feed forward 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 feed forward behavior of these networks is identical, but the backpropagation can be applied only to the right network.

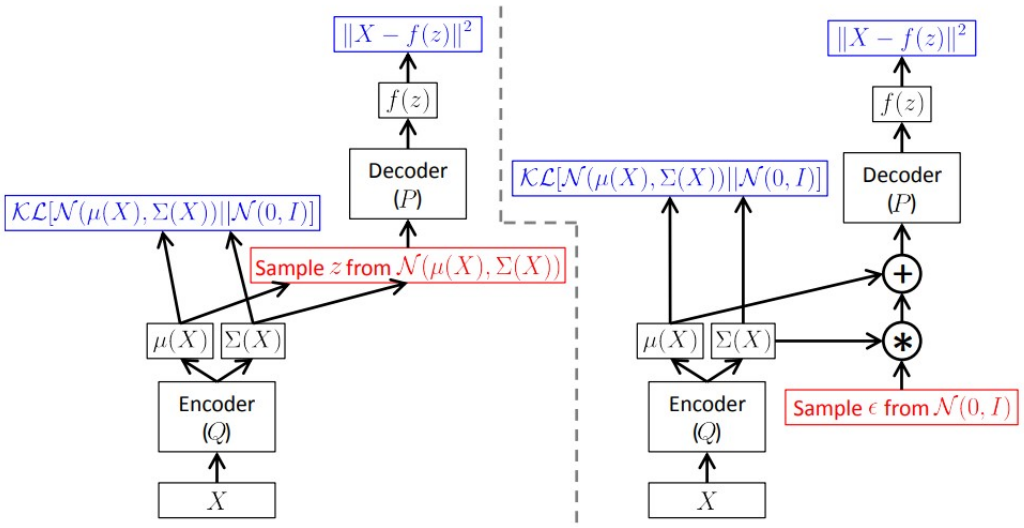


Figure 2.10: Reparameterization trick for VAE
Source: Tutorial on Variational Autoencoders [16]

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 to 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 :

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

2.4.4 Generative Adversarial Network

The idea of GANs [22, 23, 24, 25] is to set up a game between two players. The first player is called the *generator* and creates samples. These samples are intended to come from the same distribution as the training data. The second player is called the *discriminator*. He 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 differentiable functions. 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 a cost function which depends on the parameters of both players. The discriminator tries to minimize $J^{(D)}(\theta^{(D)}, \theta^{(G)})$ by controlling $\theta^{(D)}$. The generator tries 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 can only modify its own parameters, this scenario can be described as a game. The solution to a game is a called a Nash equilibrium. For us, 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.

2.4.4.1 Discriminator cost function

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

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

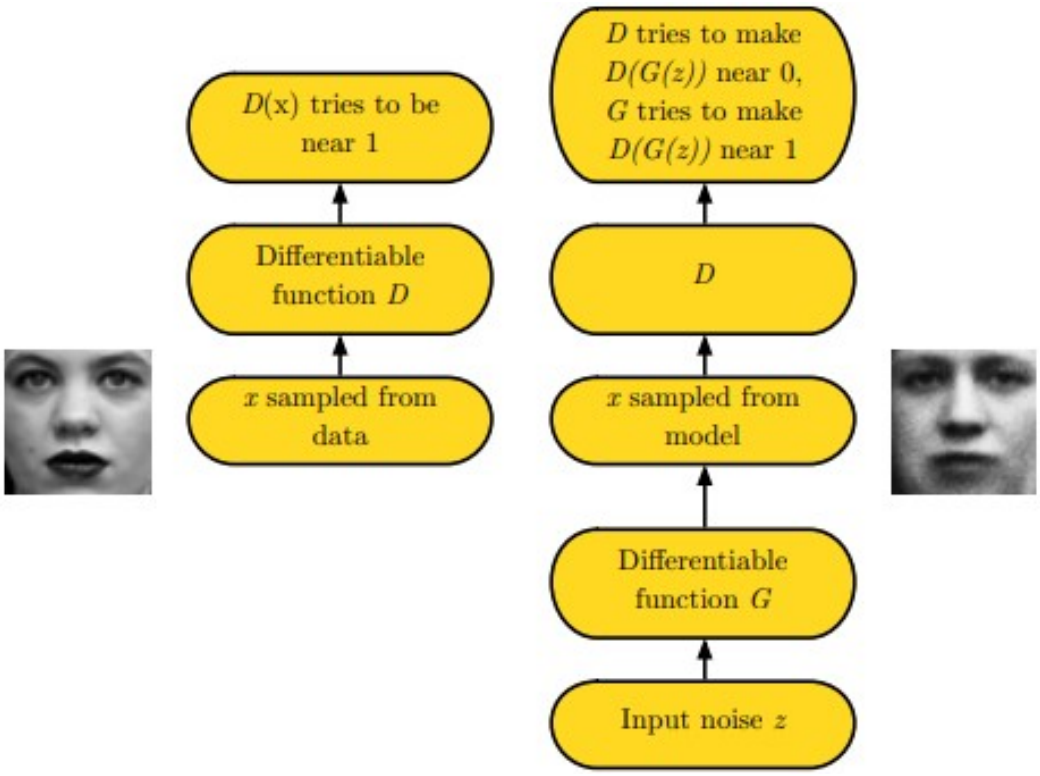


Figure 2.12: GAN framework
Source: Ian Goodfellow's paper [25]

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: $J^{(G)} = -J^{(D)}$. This a *zero-sum game*. Zero-sum games are also called **minimax** games because of the form of the solution (equation 2.12).

$$\theta^{(G)*} = \text{argmin}_{\theta^{(G)}} \left[\text{argmax}_{\theta^{(D)}} \left[J^{(G)} \left(\theta^{(D)}, \theta^{(G)} \right) \right] \right] \tag{2.12}$$

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. If the discriminator performs too well compared to the generator, it will successfully reject the samples from the generator with high confidence. The gradient of the generator will then start to vanish. In the Heuristic, non-saturating game version, the generator cost is the equation 2.13

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

The motivation of this cost function is to ensure that each player has a strong gradient when 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. It means it minimizes the KLD between the data and the model. In its paper [24], Goodfellow showed that using the equation 2.14 is equivalent to minimize the KLD between the data and the

model. In the equation 2.14, σ is the logistic function.

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

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 to

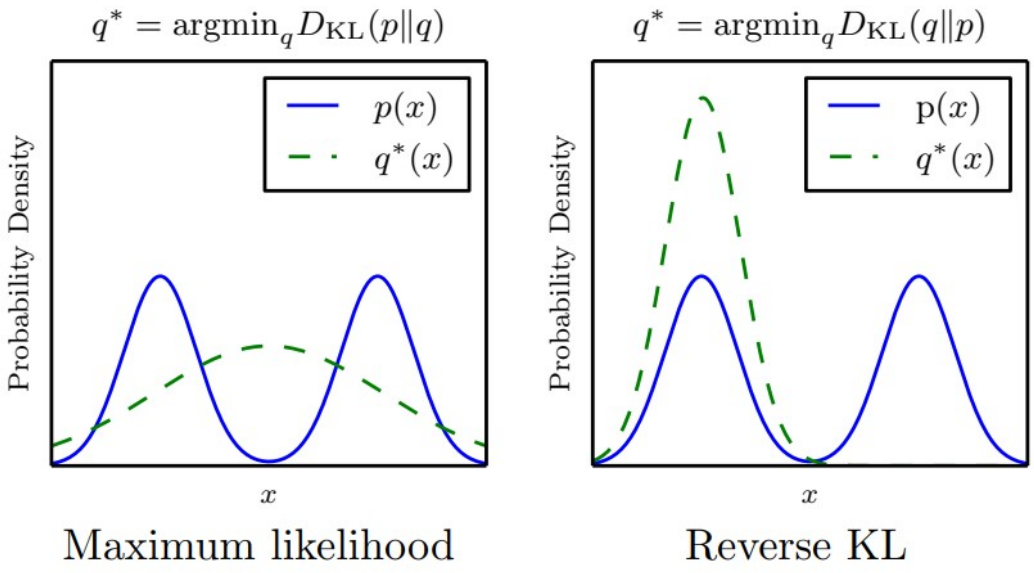


Figure 2.13: Differences between the two direction of the KLD
Source: Ian Goodfellow's paper [25]

yield better samples because the model will generate samples that come only from modes in the training distribution. It will ignore some modes and won't try to include all of them with the risk of generating some samples that don't belong to 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.

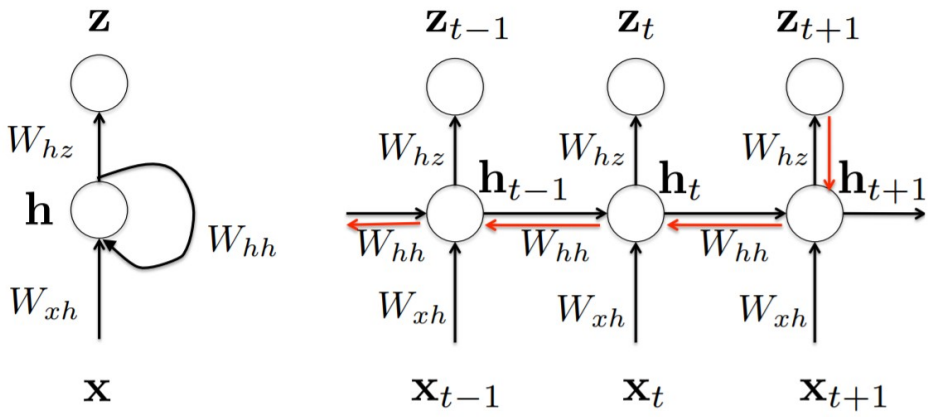


Figure 2.14: Recurrent Neural Network
Source: Gang Chen's paper [26]

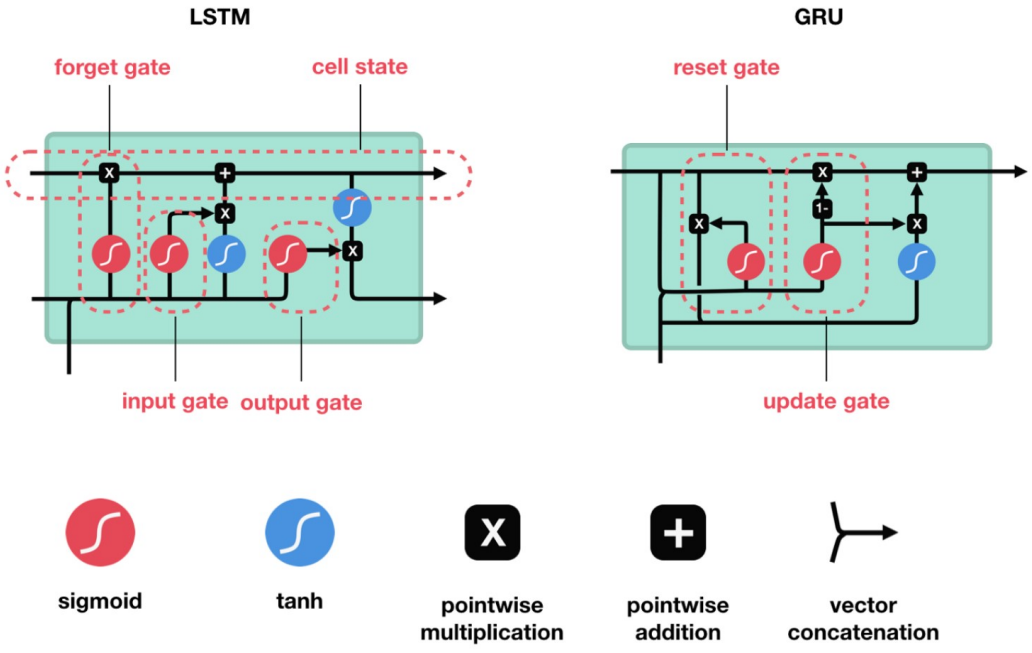
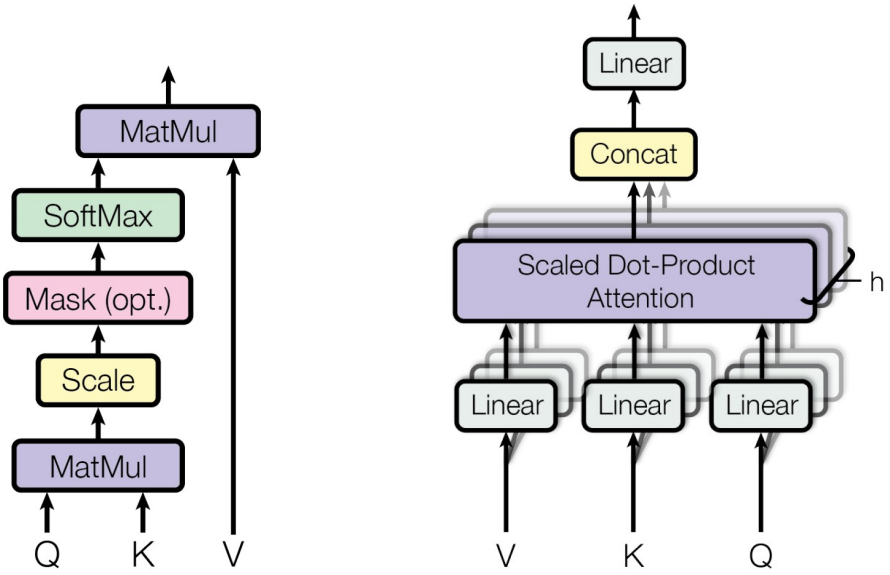


Figure 2.15: LSTM and GRU cell
Source: Michael Nguyen's tutorial [27]



(a) Scaled Dot-Product Attention
Source: Ashish Vaswani et al.'s paper [32]

(b) Multi-Head Attention
Source: Ashish Vaswani et al.'s paper [32]

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 is the follows the equation 2.15.

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

where d_k is the number of dimensions.

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

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

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 queries, keys, values, and outputs are all vectors. The output is a weighted sum of the values. 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 which 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 in the input.

2.4.7 Multimodal 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. To solve the multi-model inference problem, he MVAE uses a *Product of Experts* (PoE) inference network and a sub-sampled training paradigm.

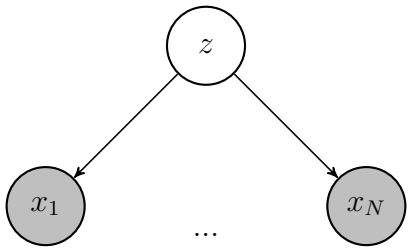
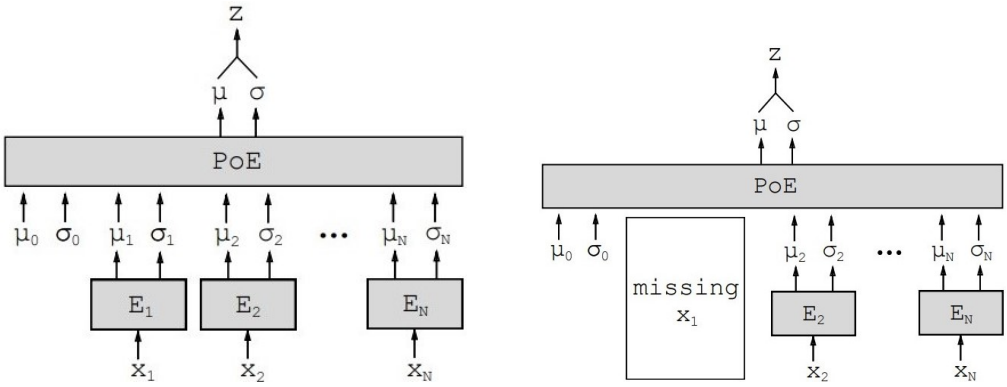


Figure 2.17: Graphical model of the MVAE

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

$$
\begin{aligned}
p(z|x_1,...,x_N) &= \frac{p(x_1,...,x_N|z)p(z)}{p(x_1,...,x_N)} \\
&= \frac{p(z)}{p(x_1,...,x_N)} \prod_{i=1}^{N} p(x_i|z) \\
&= \frac{p(z)}{p(x_1,...x_N)} \prod_{i=1}^{N} \frac{p(z|x_i)p(x_i)}{p(z)} \\
&= \frac{\prod_{i=1}^{N} p(z|x_i)}{\prod_{i=1}^{N-1} p(z)} \frac{\prod_{i=1}^{N} p(x_i)}{p(x_1,...,x_N)} \\
&\propto \frac{\prod_{i=1}^{N} p(z|x_i)}{\prod_{i=1}^{N-1} p(z)} \approx \frac{\prod_{i=1}^{N} (\widetilde{q}(z|x_i)p(z))}{\prod_{i=1}^{N-1} p(z)} = p(z) \prod_{i=1}^{N} \widetilde{q}(z|x_i) \\
&\text{With } \widetilde{q}(z|x_i) \text{ the model approximation of } \frac{p(z|x_i)}{p(z)}
\end{aligned}
\tag{2.17}
$$

Thus, the joint posterior distribution can be approximated by a product of experts (PoE) which includes a *"prior expert"*.



(a) With all the modalities (b) With missing modalities

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 the 2.18.

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

It is a feed forward network parameterized as in the equations 2.19 and 2.20.

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

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

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

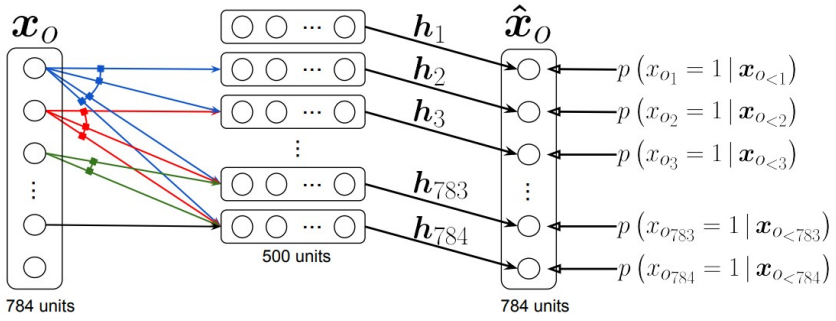


Figure 2.19: NADE architectue
Source: Benigno Uria et al.'s paper [33]

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 Bolzmann 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:

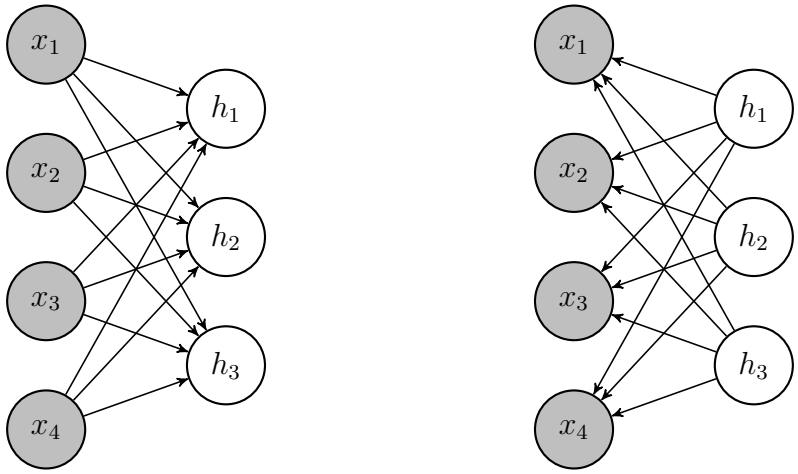
$$h_i = \text{sigmoid}(x^T W_i + a) \tag{2.21}$$

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 endoding phase.

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

$$x_i^* = \text{sigmoid}(hW^T + b) \tag{2.22}$$

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



(a) RBM forward pass (b) RBM backward pass

Figure 2.20: RBM architecture

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

energy given by:

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

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

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

where Z is the partition function:

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

Training a RBM consists 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 present 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 or complete music with several parts (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 [39].

3.1.2 Completion

The completion challenge is the same as the generation challenge. The goal is to generate one or several musical parts (they can be either monophonic or polyphonic). The difference with the generation challenge is that the model takes as an input a *seed*. The model will continue the musical parts present in the seed.

Completion is the most common objective choice among the existing works [8, 40, 41, 42, 43]

3.1.3 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).

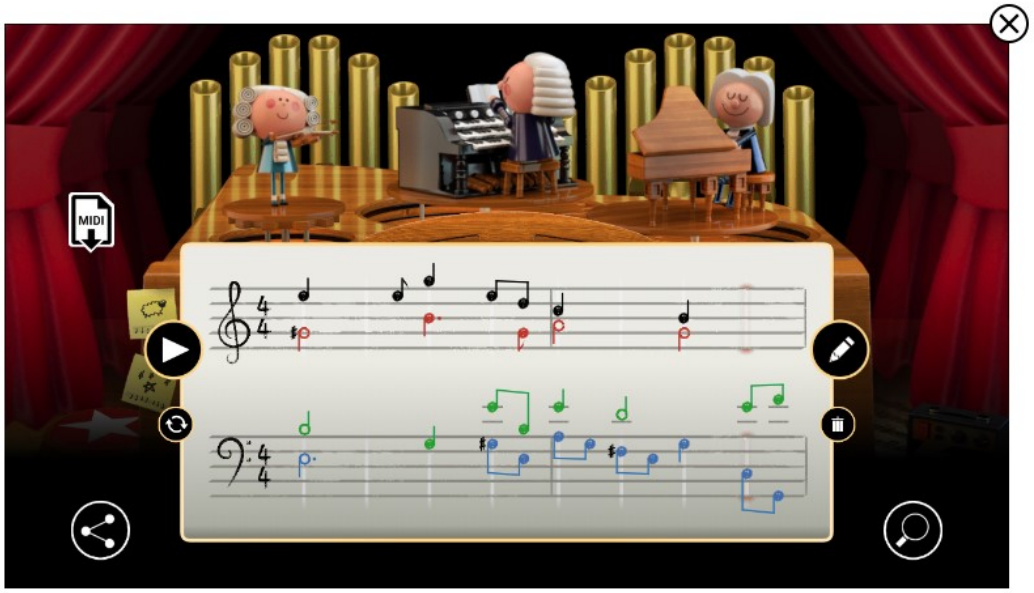


Figure 3.1: Bach Doodle application
Source: BachDoodle

3.2 Music Representation

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

3.2.1 Audio Signal

The first data used to create music is the audio signal. Some works [44, 45, 39, 46, 47, 48] have been done to generate music audio signal. This signal can be represented either by its waveform [44], its Fourier Transform, or its spectrogram (figures 3.2, 3.3).

Figure 3.2: Example of an audio waveform
Source: Needpix.com

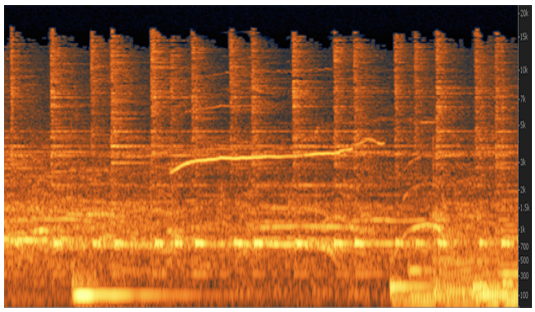


Figure 3.3: Example of audio spectrogram
Source: iZotope's tutorial [49]

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). [40, 7, 41, 8, 50, 51, 52, 42, 43, 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 [41, 40, 42, 43, 39]. The figure 3.4 shows a very simple 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. [40] 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"*.

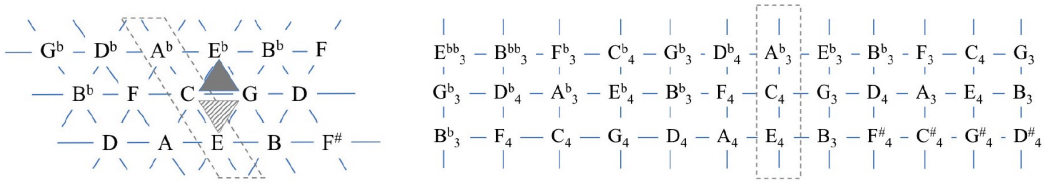
Instead of encoding notes in a one-dimensional tensor (figure 3.5a), they encode them in a 2-dimensional tensor where the relative positions between two notes is meaningful.

The figure 3.5a illustrates a common form of tonnetz. In the tonnetz network, each node represents one of the 12 pitch classes. The nodes on the same horizontal line follow the Fifth circle ordering. Two adjacent nodes on the horizontal line is a Fourth or Fifth interval An upside-down triangle filled with diagonal lines in the figure 3.5a is C major chord, and the solid triangle on the top is C minor chord.

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

3.2.2.2 Text

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



(a) (b)

Figure 3.5: (a) tonnetz and (b) the extended tonnetz matrix with pitch register Source: Ching-Hua Chuan's paper [40].

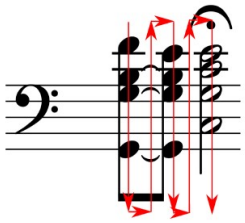
B5, _, B5, _, E6, _, E6, _, D6, _,
D6, B5, _, B5, D6, _

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 work 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 constraints, 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) `<Pitch; Tie>` tuples where `Pitch` $\in \{0; 1; \ldots; 127\}$ represents the MIDI pitch of a note and `Tie` $\in \{$`True`;`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.



(a) Three musical chords in traditional music notation. The red arrows indicate the order in which notes are sequentially encoded.

```
START           (59, True)      (55, False)
(65, False)     (55, True)      (48, False)
(59, False)     (43, True)      |||
(55, False)     |||             END
(43, False)     (.)
|||             (64, False)
(64, False)     (60, False)
```

(b) A corresponding sequential encoding of the three chords in an eighth-note time quantization (for illustration, broken over three columns). Each line within a column corresponds to an individual token in the encoded sequence. ||| delimit frames and (.) indicate a fermata is present within the corresponding frame.

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.

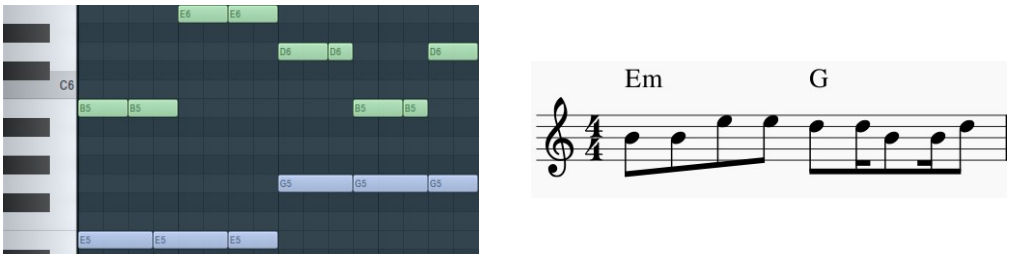


Figure 3.8: Example of correspondence between a pianoroll and its chords representation

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] considers 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 give to the model the information about the beats number and subdivisions by considering the entire measure as one entire object or step.

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 [40, 42] 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 sin 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. [42] 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. [43] 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. [41] 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 COCONET 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, 40, 7, 46, 47] as Feynman Liang et al.'s BachBot [8] or not [45].

For instance, BachBot [8] and DeepBach [7] use an encoder/decoder architecture with LSMT layers on the latent space. [40]

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. [45] use a VAE (or more precisly a VQ-VAE [67] or a AMAE [45]) to encode audio waveform and reconstruct it with a stylistic consistency accross tens of seconds.

3.4.5 GAN

The Generative Adversarial Network architecture is explained in the section 2.4.4

For instance, the model WaveGan from Chris Donahue et al. [39] 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).

3.4.6 Transformers

The Transformer's architecture is explained in the section 2.4.6. Some works have already been done on music generation 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 perfomance by using a Transformer with relative attention (equation 3.1).

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

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 [39].

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 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]. However, as Shuqi et al. [76] say, music is more complex and has several level of style (timber, performance and composition). Therefore, 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, 40, 41, 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 of the music.

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

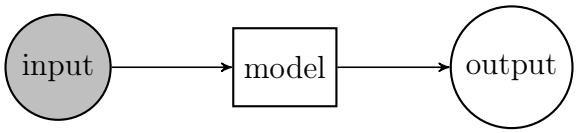


Figure 3.10: Feed forward generation process

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.

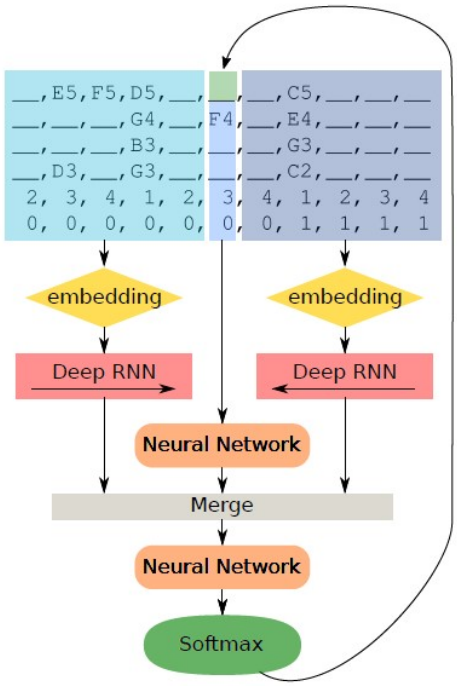


Figure 3.11: Graphical representation of DeepBach's neural network architecture for the soprano prediction
Source: DeepBach paper [7]

3.6 Thesis contributions and relation to prior work

As explained in the introduction, I aim to create only one model which can handle several tasks as:

- Generate or complete 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

Unfortunately, the related works don't have a such objective and the used neural architectures couldn't fit my needs. That is why I created a new archi-

tecture called Recurrent Multimodal Variational AutoEncoder. This architecture is explained in the section 4.3.

Because of the difficulty of obtaining clean data and of extracting meta data from a musical score, I decided to not work on that part and mostly focus my efforts on the deep learning part. However, the results I obtained didn't satisfied me. That is why I decided to transpose all the songs from my datasets in the C major key or the A minor key.

I decided to use the pianoroll representation (section 3.2.2.1) because this is how I (as a pianist) personally visualize the music). And I chose to consider the measure as an entire object (and not a frame) because this is how I conceptualize music.

Finally, all along my project, I had the hope to be able to play and jam with a trained model, that is why I chose my generation process to be feed-forward (section 3.5.3). This is the simplest way to consider the notes I play and build a real time application.

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 data 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. In the section 4.4, I'll illustrate how I gave some prior knowledge to the network with custom loss functions. And finally, in the section 4.5, I will explain how the RMVAE is trained and how it can be used to create a new song.

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 considered 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 to predict the next one.

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	𝅘𝅥‿𝅘𝅥𝅯
6	𝅘𝅥.
7	𝅘𝅥‿𝅘𝅥𝅮.
8	𝅗𝅥
9	𝅗𝅥‿𝅘𝅥𝅯
10	𝅗𝅥‿𝅘𝅥𝅮
11	𝅗𝅥‿𝅘𝅥𝅮.
12	𝅗𝅥.
13	𝅗𝅥.‿𝅘𝅥𝅯
14	𝅗𝅥.‿𝅘𝅥𝅮
15	𝅗𝅥.‿𝅘𝅥𝅮.
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.

$$\texttt{channel}_{\texttt{activation}} = 0 \iff \texttt{channel}_{\texttt{duration}} = 0 \tag{4.1}$$

4.2.2 Monophonic Music

The goal was too 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 musical 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 $note_{continue}$. 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.

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

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

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

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

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 Multimodal Varational 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 Va-

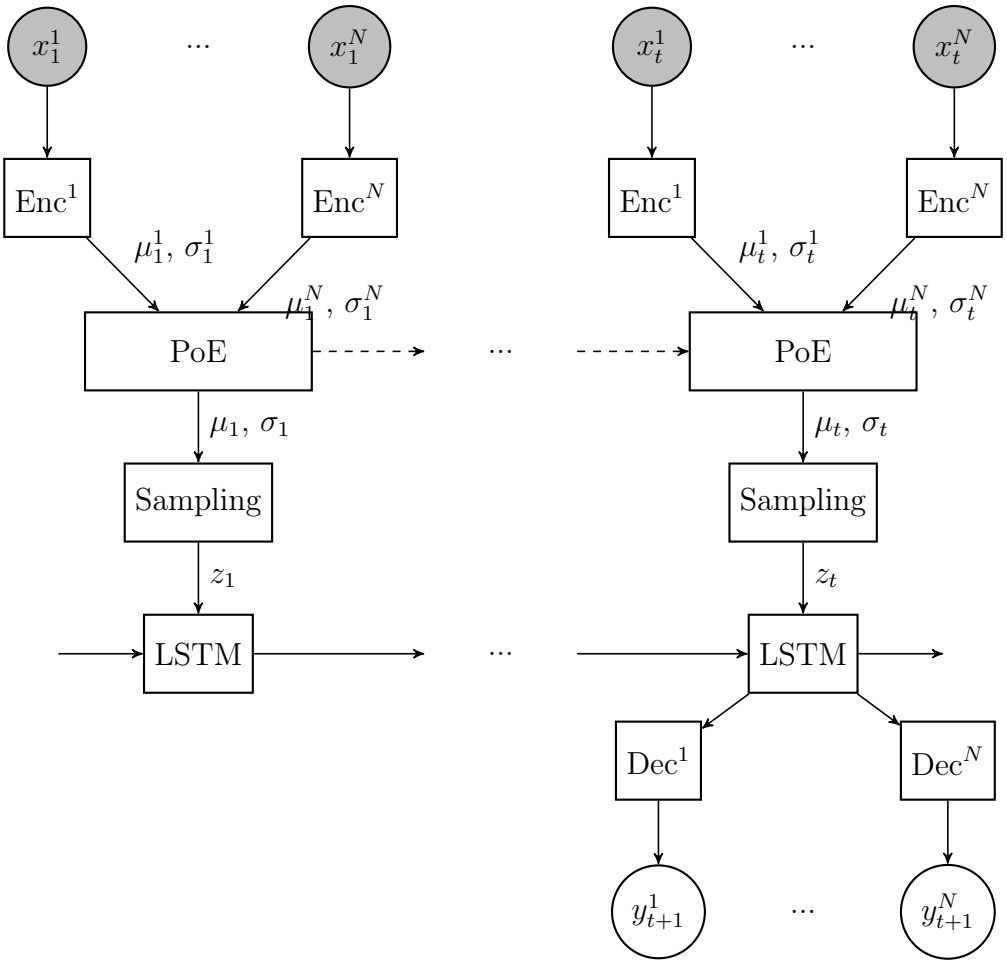


Figure 4.1: Architecture of the RMVAE

rational AutoEncoder). This architecture is described in the figure 4.1.

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 uses an RNN architecture to

encode it.

Each instrument or musical voice has its own couple of encoder and decoder. During the training, each couple of encoder and decoder will specialize themselves to represent the specificity of each voices in the best possible way.

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 processes 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.

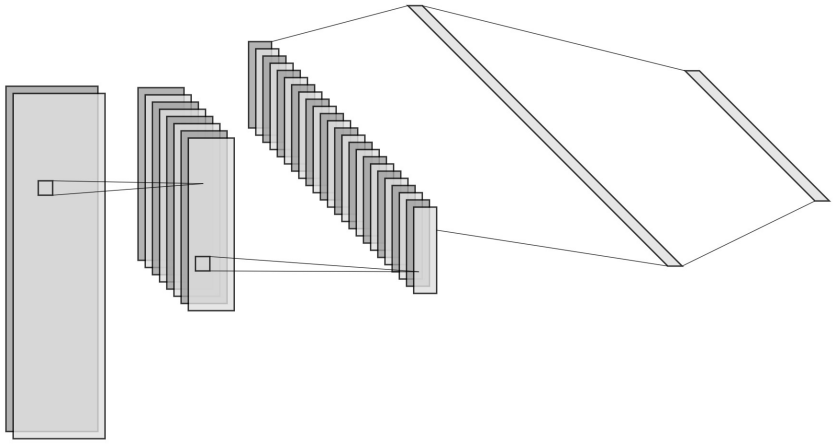


Figure 4.2: RMVAE encoder architecture

The filter sizes of the convolutional layers are `(5, 5)`. I chose this dimension so the 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 dimension (`(16,)`) corresponds of a sixteenth note (𝅘𝅥𝅯) in time. Therefore, I created an encoder using bidirectional LSTM cells. This architecture is shown in the figure 4.3.

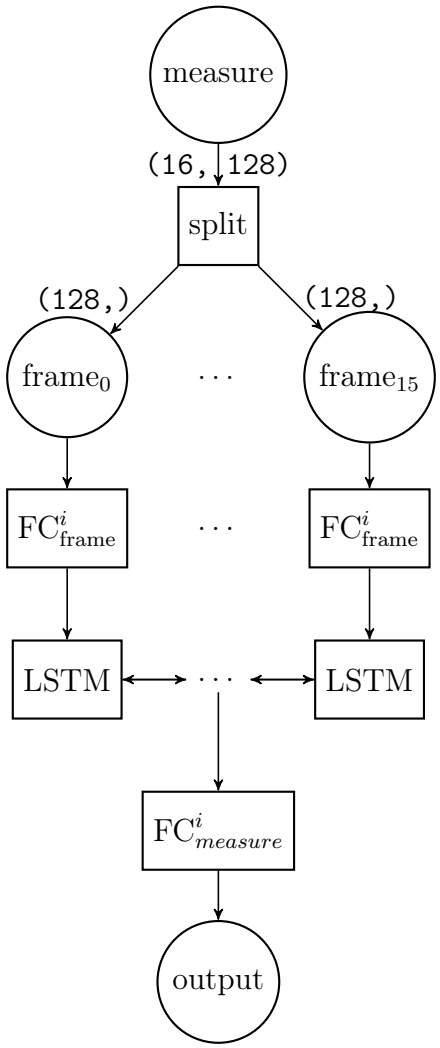


Figure 4.3: Measure Encoder using bidirectional LSTM
Encoder for a measure of an instrument i. A measure has a shape of `(16, 128)` and a frame as a shape of `(128,)`.

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^i_μ and fc^i_σ which will return the mean and variance of the encoded normal distribution (figure 4.4).

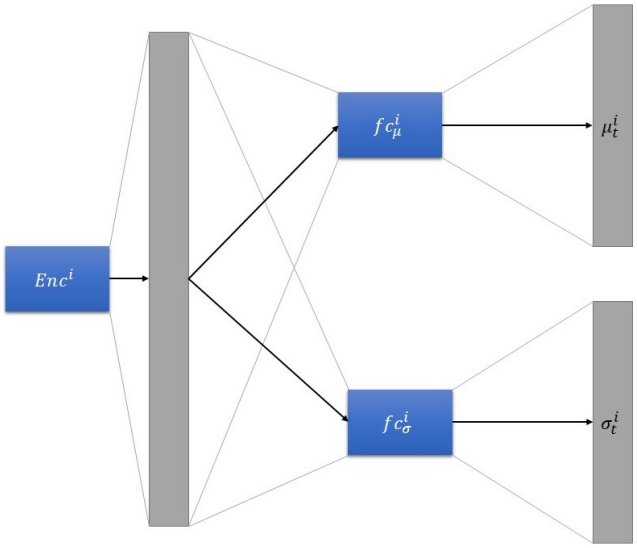


Figure 4.4: RMVAE PoE Fully Connected layers

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.

4.3.4 Recurrent layers

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

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 , 2.15). This is shown in the figure 4.5.

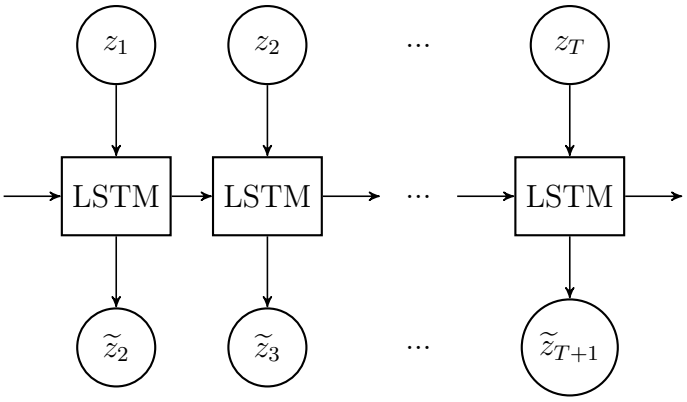


Figure 4.5: RMVAE LSTM layer

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.

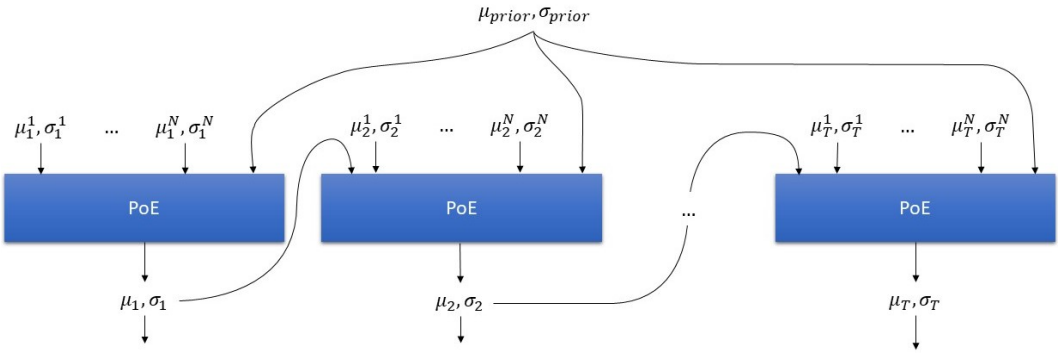


Figure 4.6: RPoE architecture

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.

The decoder global architecture is showed in the figure 4.7.

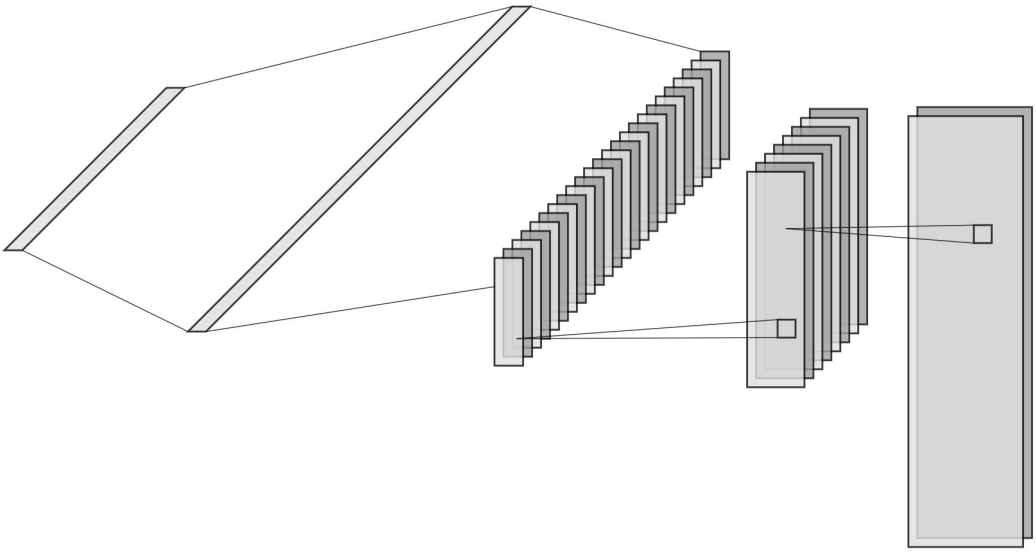


Figure 4.7: RMVAE decoder architecture

This decoder was associated with the convolutional encoder from 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

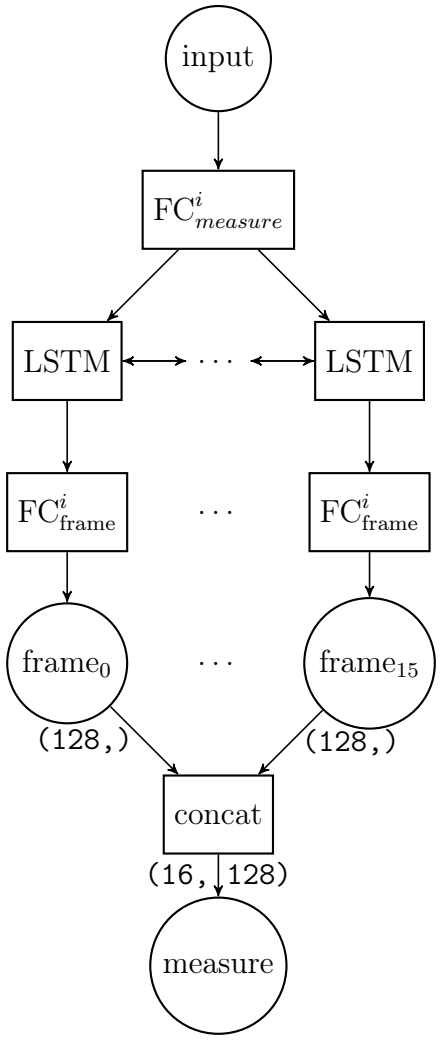


Figure 4.8: Measure Decoder using bidirectional LSTM
Decoder for a measure of an instrument i. A measure has a shape of `(16, 128)` and a frame as a shape of `(128,)`.

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 s *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 I was simply applying a softmax activation across all the notes. But the result was disappointing: the frequency of $note_{continue}$ in the dataset was to 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 the musician has to follows it. From the chords played, the scale, some notes sounds 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. 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 words, during the training part, if the model doesn't hit the note it should have hit, I don't want to *punish* it to 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 ← commonLoss(noteTruth, notePredict)
4:     if soundsGood(notePredict) then
5:         loss ← loss − reward
6:     else
7:         loss ← 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:

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

where the shape of both input tensors is `(nb_instruments, 16, 128)`. It represents 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. The result of this function will be positive for a note in the predict tensor which is not in the truth tensor. Conversely, if the note in the predicted tensor is present in the truth tensor, the result will be negative.

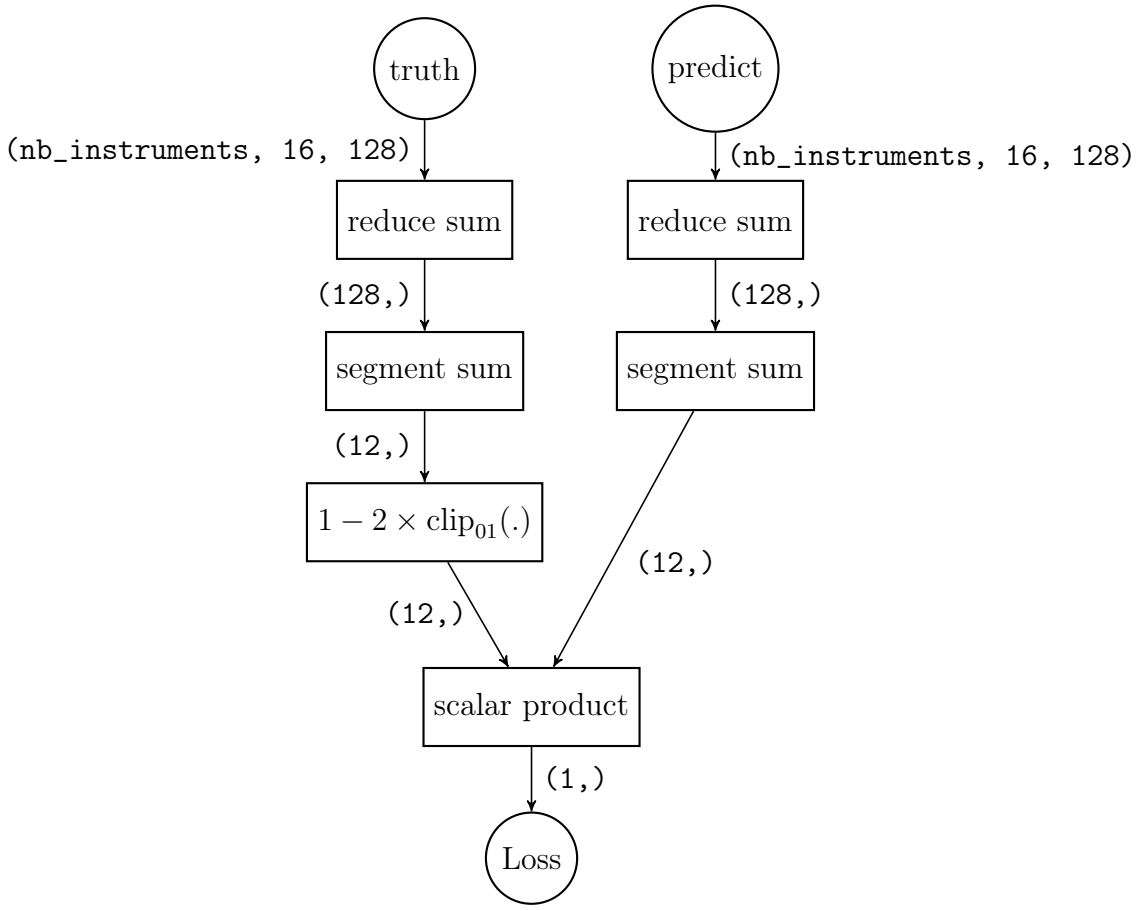


Figure 4.9: Scale Loss

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 needed 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.

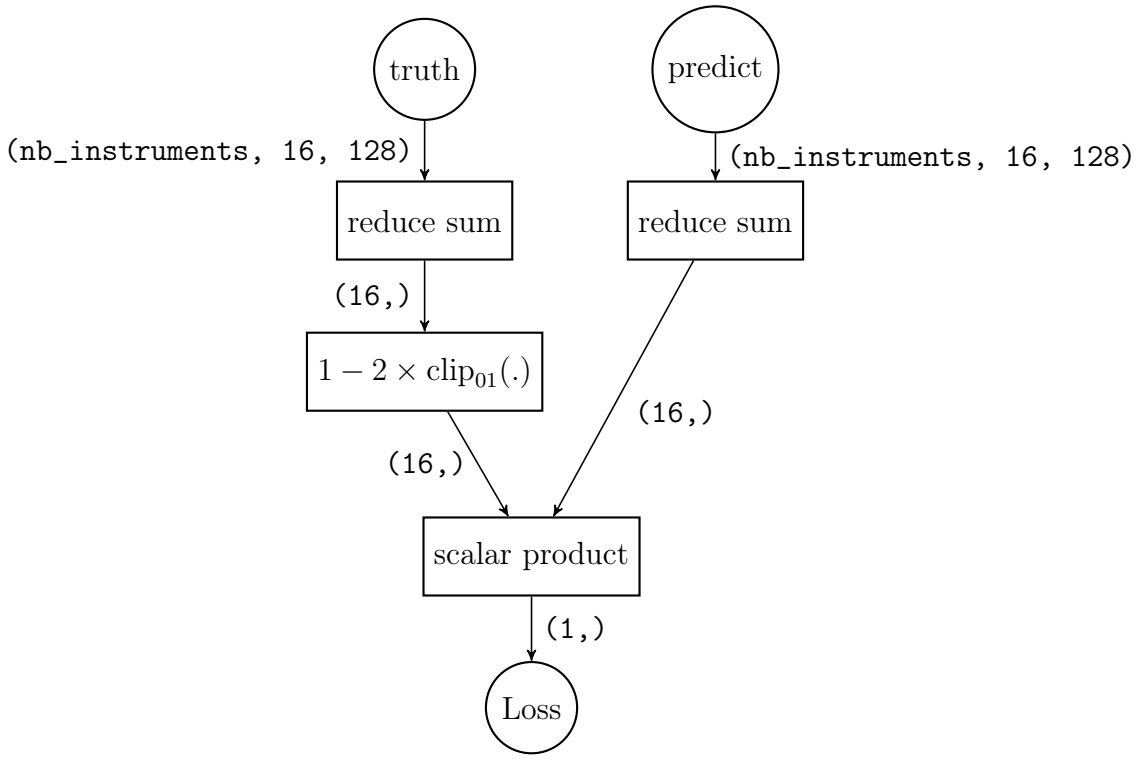


Figure 4.10: Rhythm Loss

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 a A note.

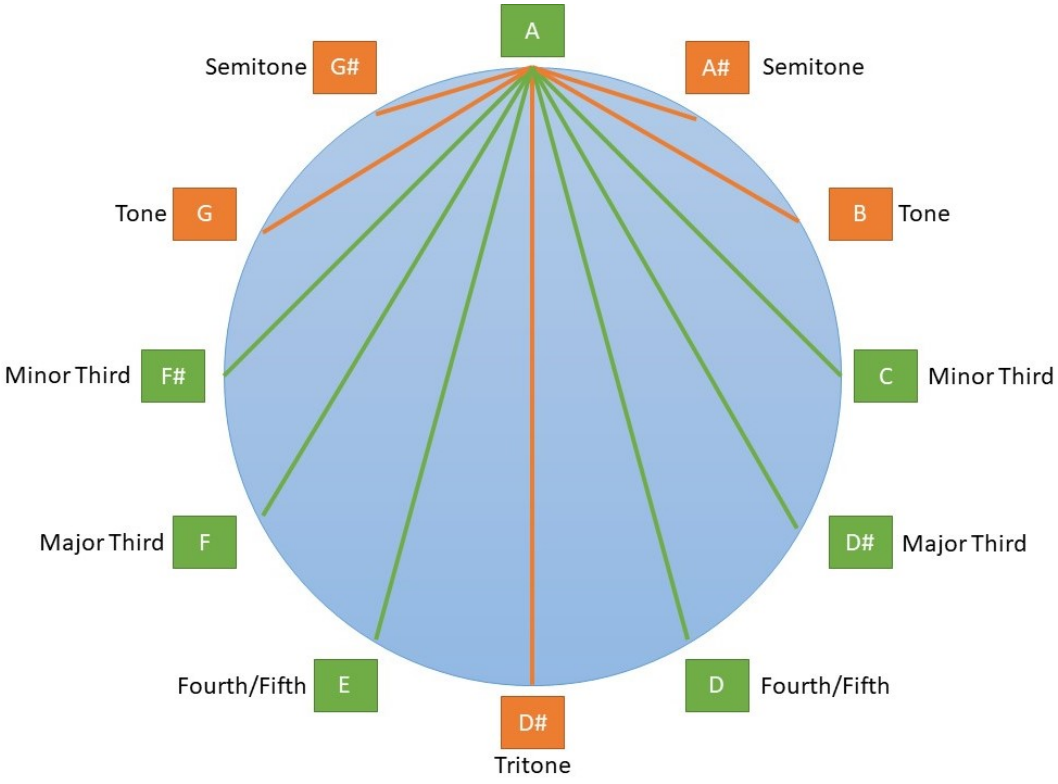


Figure 4.11: Harmony Circle for A

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

- The *semitone* interval

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

To prevent the model to produce 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. As for the Scale loss, the harmony_n use a segment sum operation to get rid of the octaves. The trick is to apply a roll_n operation on the result and compute a scalar product with the original. The roll_n operation is explained in the appendix .3. It will return a positive and high value if an interval of n semitones is present in the tensor.

The function Harmony do the sum of the functions harmony_1, harmony_2 and harmony_6 which respectively handle the semitone, tone, and tritone intervals.

4.5 Learning process

In this section I will describe how I train the Recurrent Multimodal Variational AutoEncoder. I use the model differently between the training time and the creation time.

During the training, the model is trained to predict the next measure of a song from a fixed number of measures. In practice, I give 8 measures of a song from the Bach Chorale corpus and ask the model to predict the next one. The goal is not to *over fit* the training dataset. During the training time, the model will try to *guess* and learn what is the next measure from the previous ones for the songs in the dataset. This idea is that the model will not learn all

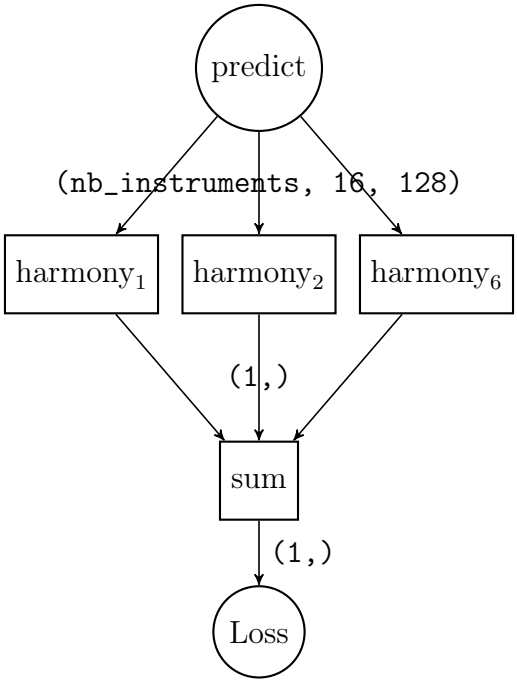


Figure 4.12: Harmony Loss

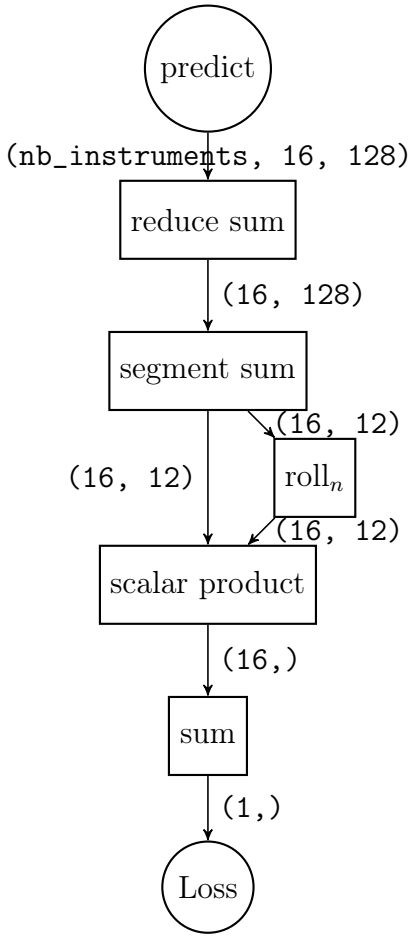


Figure 4.13: Harmony$_n$ Loss

the songs *by heart*, but it will learn music in a more general and meaningful way. It will learn musical rules tendencies, and styles from the dataset.

Then, when the model is correctly trained, it will be able to complete a song in a consistent way from the given seed and the train dataset. When 8 measures from the test dataset are given to the model, it will create the next measure which can be a possible continuation of the given seed. Obviously the model won't be able to guess and recreate the correct measure from the song because it is impossible to *guess* a song without hearing it before. So the model will create an alternative continuation of a song and this is how it can be used to create a new song.

Alternatively, the seed given to the model can be random so the RMVAE will generate new measures on its own.

CHAPTER 5

Results

In this chapter I will share the results I got. In the section 5.1, I will explained how I decided to evaluate the music samples created by my model. In the section 5.2, I will describe what sample the model usually produces. In the section 5.3, I will show the results obtained when the model is using or not the $\text{note}_{\text{continue}}$ note. Finally, in the section 5.4, I will enumerate what tasks I have actually implemented. To be fair, the results are quite poor and disappointing.

5.1 Music evaluation

Music is by definition an art, it is therefore subjective.

5.1.1 Prediction evaluation

5.1.2 Creation evaluation

5.2 Created Samples

5.2.1 CNN Encoder Decoder

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 completed sample by a RMVAE.

As we can see on the figure 5.1, the global shape of the the created part

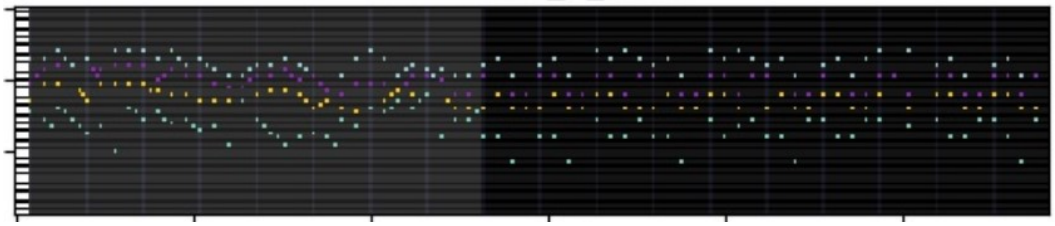


Figure 5.1: Created pianoroll by the RMVAE with CNN encoder decoder The grey part is the seed given to the model and the black part is the created part (Link)

is very different from the seed part. The main reason is a lack of rhythm and variation. Most of the created notes are quarter notes (♩), and the model produces always the same notes.

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

5.2.2 Recurrent Encoder Decoder

For the recurrent encoder and decoder, the results have been slightly better but not much. The created samples are mostly composed of quarter notes (♩) and the variations are too poor. One sample is showed in the figure 5.2. The generated sample seems more complex with more variations. And this is verified when you listen to it. There are more small variations in the rhythm and the played note than the sample from the cnn encoder decoder.

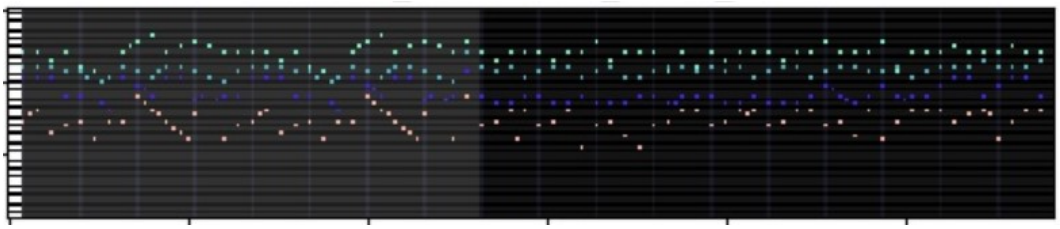


Figure 5.2: Created pianoroll by the RMVAE with recurrent encoder decoder The grey part is the seed given to the model and the black part is the created part (Link)

5.3 Activation function of the $\text{note}_{\text{continue}}$ for monophonic music

5.3.1 CNN Encoder Decoder

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 5.3 shows a created sample. As explained, the produced part (black part) contains only few notes.

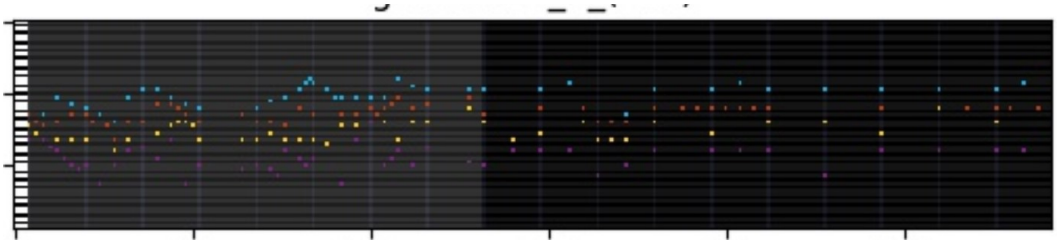


Figure 5.3: Created pianoroll by the RMVAE with a softmax activation The grey part is the seed given to the model and the black part is the created part (Link)

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 chose what is the note to be played (if $note_{continue} < 0.5$).

By doing so, I helped the model to produce new notes and get results like the figure 5.1. However, the lack of rhythm is still present and most of the notes have a length of one beat (♩).

5.3.2 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 5.4 and 5.5 illustrate it. We can see on the pianoroll views that the created music is more consistent when the activation function on the note $note_{continue}$ is the sigmoid function.

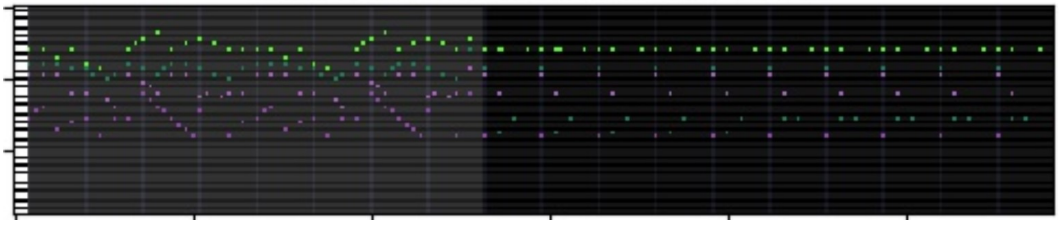


Figure 5.4: Created pianoroll by the RMVAE with recurrent architecture with softmax activation
The grey part is the seed given to the model and the black part is the created part (Link)

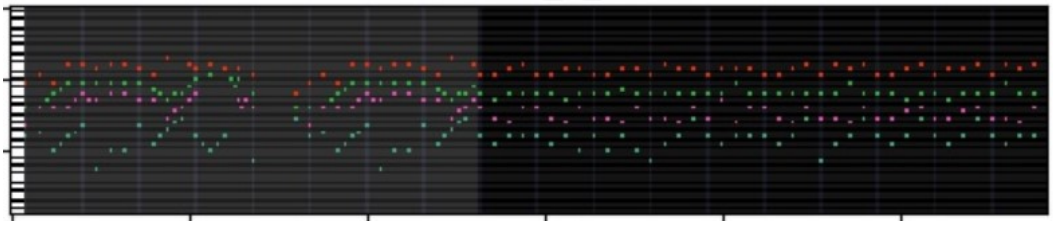


Figure 5.5: Created pianoroll by the RMVAE with recurrent architecture with softmax and sigmoid activation
The grey part is the seed given to the model and the black part is the created part (Link)

However, the issues stay the same. There is absolutely no rhythm, all the notes are quarter notes (♩) and there is no variation. For instance, in the figure

5.5, the first voice (the pink one at the top) is only playing 3 different notes.

5.4 Tasks

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

- Generate (section 5.4.1)
- Fill (section 5.4.2)
- Redo (section 5.4.3)

5.4.1 Complete

This is the simplest task I have created for the model. The figure 5.6 is the result of it.

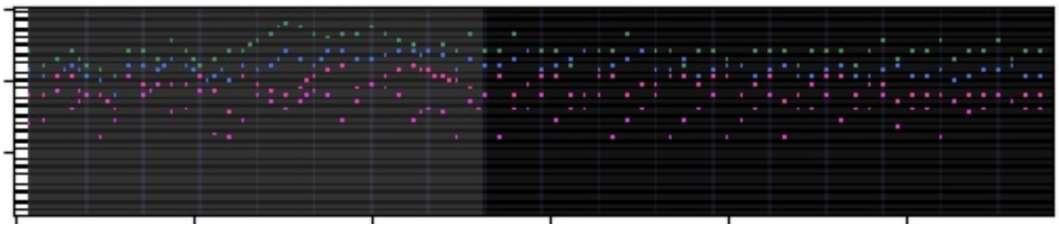


Figure 5.6: Task *Complete*
(Link)

The model is given a seed (grey part of the figure 5.6). From this seed it creates the next measure. It will then consider the seed and the previously created measure to create 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 to do is to put 0 at the positions $mask[i, :]$ for every instrument i we want to ignore.

Algorithm 2 Complete function

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

```
1: function COMPLETE(seed, n)
2:     seed_length ← len(seed)
3:     generation ← seed
4:     for k ← 1 to n do
5:         input ← generation[−seed_length :]
6:         output ← model.predict(input)
7:         generation.append(output)
8:     end for
9:     return generation
10: end function
```

However, as you can see in the figures 5.6, the created sample is poor in variation and complexity. The accuracy of the model to create the next measure is around 0.8.

5.4.2 Fill

The *Fill* task creates a musical part (an instrument voice) from the others. As an example, the figures 5.7 and 5.8 illustrate this task.

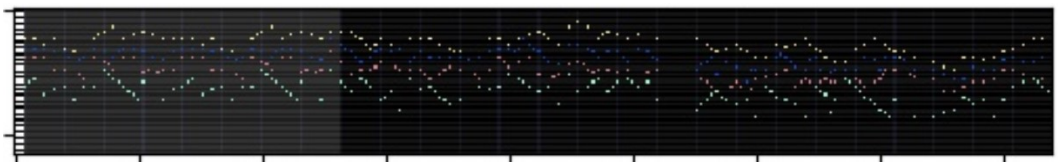


Figure 5.7: Song given to the model for the *Fill* task
(Link)

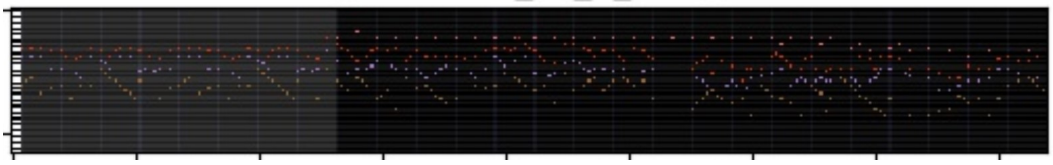


Figure 5.8: Output of the model for the *Fill* task of the first voice
(Link)

I gave to the model the figure 5.7 and apply the fill algorithm on the voice

0 (the one at the top, yellow in the figure 5.7 and pink in the figure 5.8). To do so, the model deletes the voice to fill and uses the remaining ones as a seed to create 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 ← song
4:     for k ← 0 to len(song) − seedLength − 1 do
5:         input ← song[k : k + seedLength]
6:         output ← model.predict(input)
7:         filled[k + seedLength, instrument] ← output[instrument]
8:     end for
9:     return filled
10: end function
```

As you can see in the figure 5.8, the filled voice has a big lack of complexity. Most of the notes are quarter notes (♩) 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.

The accuracy of the model on the prediction on the next measure for one voice is usually around 0.7.

5.4.3 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 5.9, 5.10, 5.11, 5.12, 5.13 illustrate this task.

The song from the figure 5.9 as been given to the model. It has 4 voices (4 different instruments). Here are the for steps:

Algorithm 4 Redo function

Input: *song* (list of measures)
Output: *redone* (list of measures)

```
1: function REDO(song)
2:     redone ← song
3:     for k ← 0 to nb_instruments − 1 do
4:         redone ← Fill(redone, k)
5:     end for
6:     return redone
7: end function
```

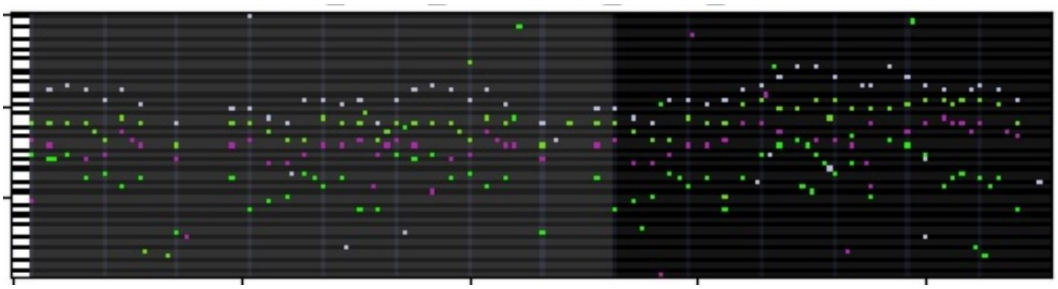


Figure 5.9: Song given to the model for the *Redo* task
(Link)

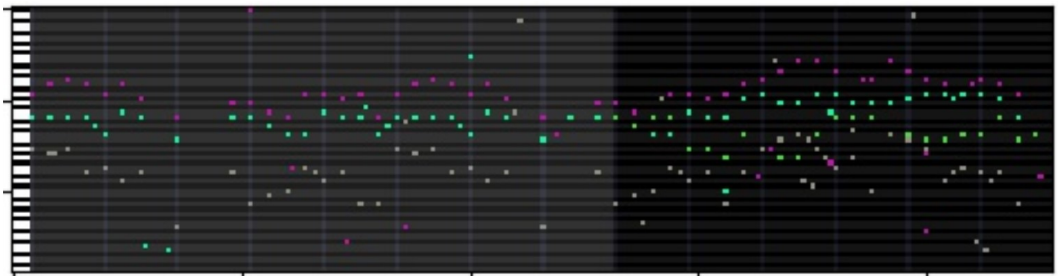


Figure 5.10: Iteration 1 of the *Redo* task
(Link)

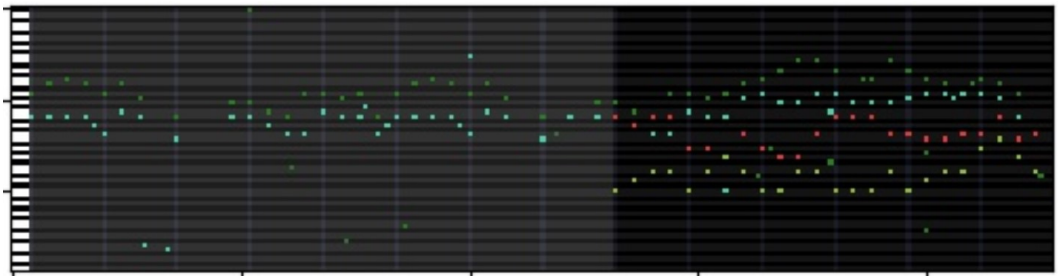


Figure 5.11: Iteration 2 of the *Redo* task
(Link)

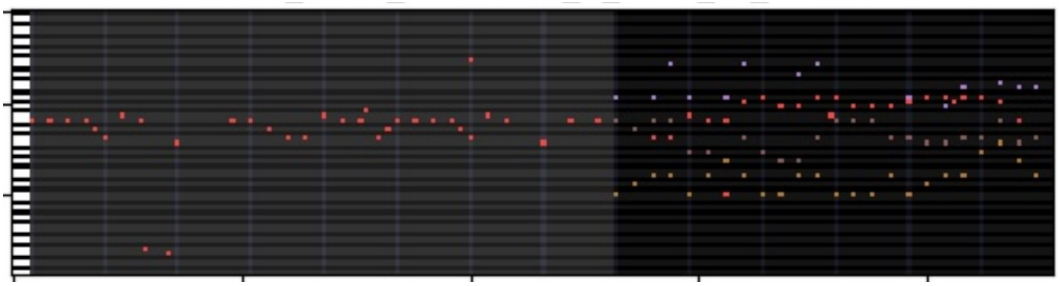


Figure 5.12: Iteration 3 of the *Redo* task
(Link)

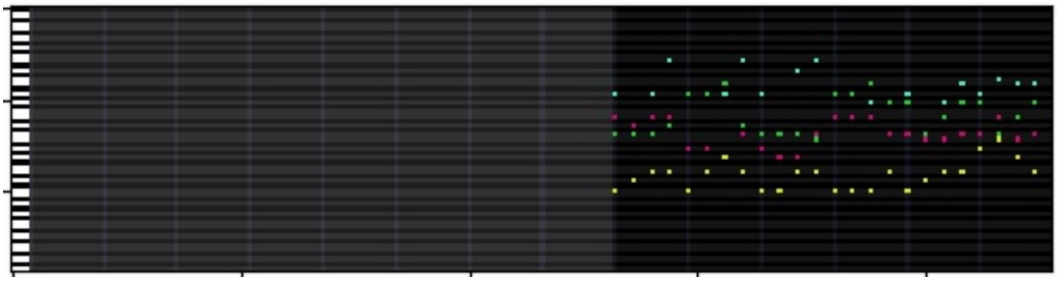


Figure 5.13: Iteration 4 of the *Redo* task
(Link)

1. (figure 5.10) The model apply the *Fill* task on the voice 3 (from the top)
2. (figure 5.11) The model apply the *Fill* task on the voice 4 (from the top)
3. (figure 5.12) The model apply the *Fill* task on the voice 1 (from the top)
4. (figure 5.13) 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 voice mostly stays on few notes. This is very repetitive.

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

5.4.4 Generate

The *generate* task is used to generate a song from a noise. The model task a random seed from a noise, and apply the generate task on this seed. In that way, the model creates its own music and doesn't continue an existing song.

The results from generate task are quite surprising because they are consistent with the previous tasks.

I tried 3 different kind of noise:

- Gaussian
- Uniform
- Binomial

As a reminder, for a monophonic music, there is only one `channel`: the channel *activation*. The dimension of a tensor for an instrument is `(16, 128 + 1, channel=1)`. `16` is the number of Sixteenth notes (𝅘𝅥𝅯) in a measure, `128` is the number of possible pitches provided by the MIDI format. An extra note ($\text{note}_{\text{continue}}$) is added to symbolize the fact that the previous note is continued and no new note is played. for the pitch dimension (`128 + 1`), all the values are 0, and there is only one 1 at the adequate position.

The results from the different noises are similar. Except for the Gaussian loss which produces samples less complex than the others.

5.4.4.1 Gaussian

I give to the model a Gaussian noise with a mean of 0.5: $\mathcal{N}(0.5, 1)$.

The figure 5.14 shows a sample generated by the model from this noise. The grey part at the beginning is the noise given as a seed.

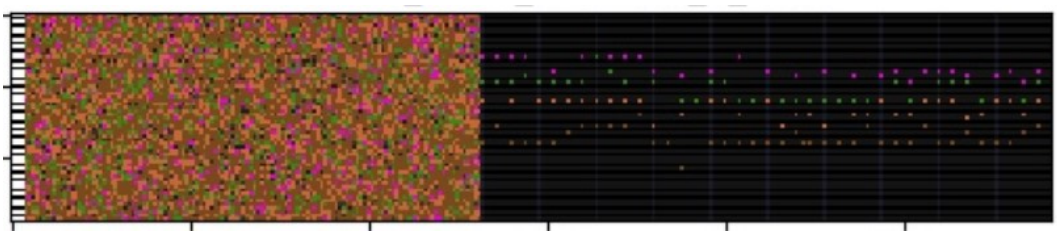


Figure 5.14: Generation of music by the model from a *Gaussian* noise
Link

5.4.4.2 Uniform

The second noise is a Uniform noise. At the beginning, all the values of the seed are 0. For each Sixteenth notes (𝅘𝅥𝅯) for each measures of each instruments, a value is chosen between $\{0, 1, \ldots, 128\}$ with a uniform probability ($\frac{1}{129}$). A 1 is put at the index of the chosen value.

The figure 5.15 shows as sample generated by the model with the *Uniform* noise. The grey part at the beginning is the noise given as a seed.

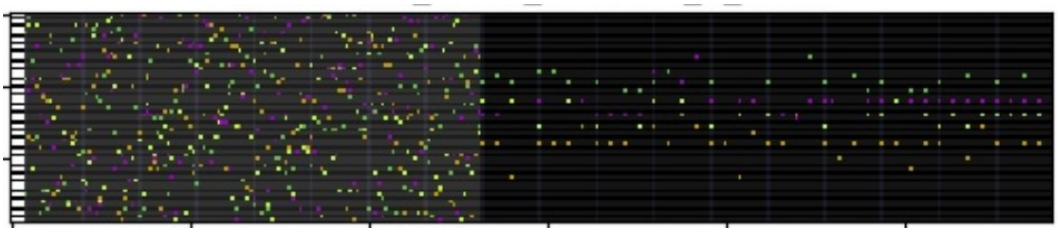


Figure 5.15: Generation of music by the model from a *Uniform* noise
Link

5.4.4.3 Binomial

The last noisy seed I created is called *Binomial*. First, for each sixteenth notes of each measure of each instruments, the value of the $\text{note}_{\text{continue}}$ as a probability $p_0 = 0.5$ to be set to 0 and a probability $p_1 = 0.5$ to be set to 1. If $\text{note}_{\text{continue}} = 1$, then all the others notes are set to 0. And if $\text{note}_{\text{continue}} = 0$ then, one of the other notes is set to 1 and the others to 0. The activated note is chosen uniformly among the `128` remaining ones ($p = \frac{1}{128}$).

The figure 5.16 shows as sample generated by the model with the *Binomial* noise. The grey part at the beginning is the noise given as a seed.

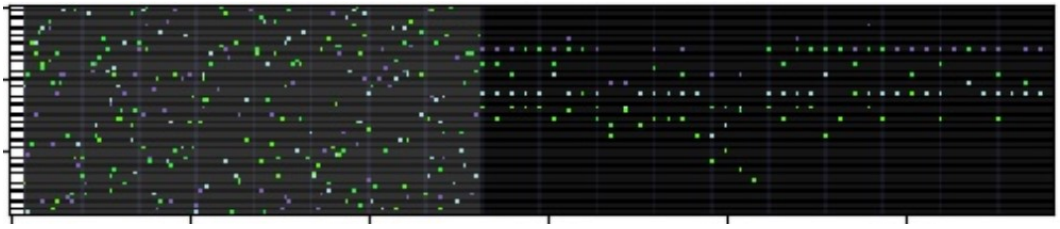


Figure 5.16: Generation of music by the model from a *Binomial* noise
Link

CHAPTER 6

Experiments

In this chapter I expose the experiments I have conducted for this project. Evaluating a produced 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 papers working on music use the population to evaluate whether their model is qualitative enough. [41, 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 many polls to evaluate the musical samples my model creates. The appendix .9 contains the list of all the different polls I have made for this Dissertation.

Conducting several of experiments was very laborious and slow. I use the python framework Tensorflow [4] and Keras [5] and faced a lot of memory leak issues [77, 78]. Because of that I had to launch every training manually. Moreover, training a model is very long, for example, the average time to train a model for 50 epochs was about 1 day and a half. Also, I was sharing the GPUs with limited memory provided by NUS with other students, so I couldn't use all the memory space to train all my different models in parallel.

The appendix .10 contains some figures useful for this chapter.

6.1 Transposed data

The appendix .10.1 contains the useful plots for this section. The first model is trained on a transposed dataset while the second one is trained on a non

transposed dataset.

- The figure 6 shows the value of the loss of an output.
- The figure 7 shows the accuracy value of an output.
- The figure 8 shows the global loss value.
- The figure 9 shows the KLD value.
- The figure 10 shows the value of the harmony loss.
- The figure 11 shows the scale and rhythm loss value.

To help the model and get sharper distributions, I created a script to transpose all the musics in C major or A minor scale.

Unfortunately, it didn't help much the model. The output and accuracy of an output stays the same for the train dataset, and becomes worse for the validation dataset (figures 6, 7, 8). The value of KLD and the harmony loss is unchanged for the training and validation dataset (figures 9 and 10). However, it looks like that for the scale and rhythm loss, the model performs better on the transposed validation set.

From the metric figures, it seems that transposing the data doesn't help the model to train. However there is no noticeable difference between the samples created of the two models. The figures 6.1 and 6.2 respectively show a sample created by the model trained and transposed data and a sample generated by the model train on the non transposed data.

I created a Google Form so people can evaluate what is the best model. The figure 6.3 displays the result of the preference of the persons who answered the survey.

14 people answered the it. And the results match the expectations from the plots of the metrics. People didn't prefer a model to another.

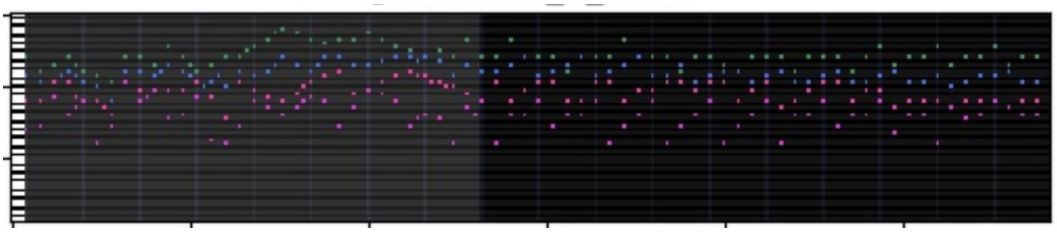


Figure 6.1: Pianoroll created by the model which was trained on transposed data

Link

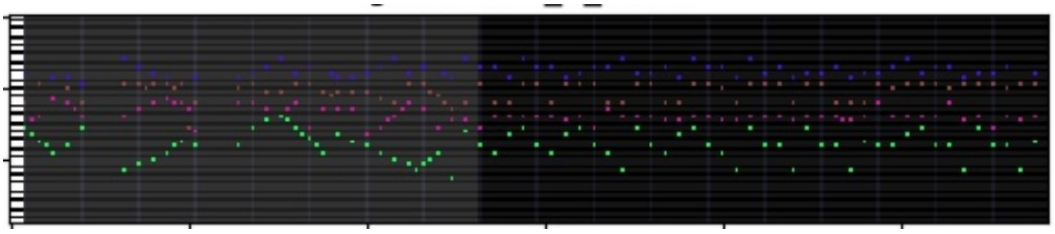


Figure 6.2: Pianoroll created by the model which was trained on non transposed data

Link

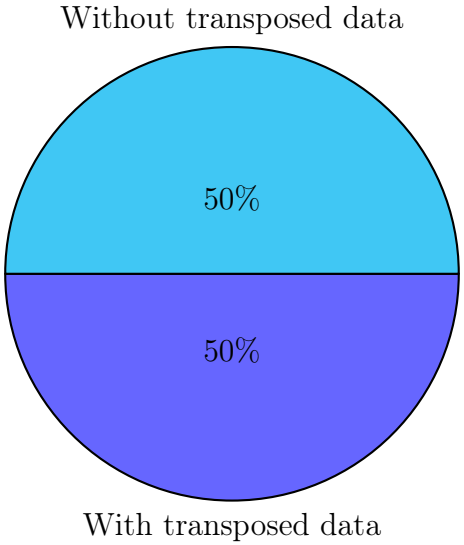


Figure 6.3: Google Form result on transposed data
The preferred model by the people

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.

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

Those memory sizes correspond to a batch size of 64.

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

The graphs for this section are in the appendix .10.2.

- The figure 12 shows the value of the loss of an output.
- The figure 13 shows the value of the accuracy of the loss.
- The figure 14 shows the global loss value.
- The figure 15 shows the KLD value.

From these different figures, we can see there is no concrete differences 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 despite the difference of value between the different metrics, the created samples are not very different, the notes are mostly quarter notes (♩) and there is no variation is the generated music.

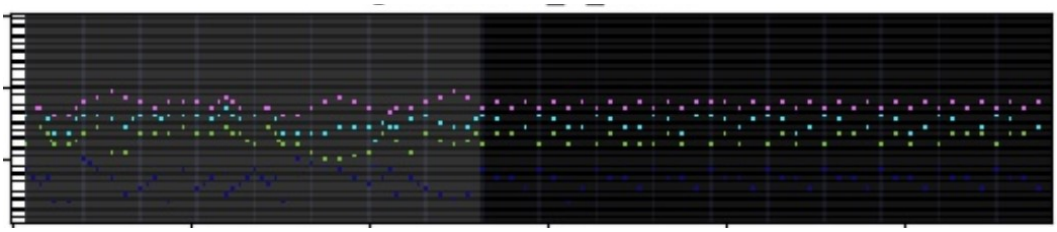


Figure 6.4: Pianoroll created by the small model
Link

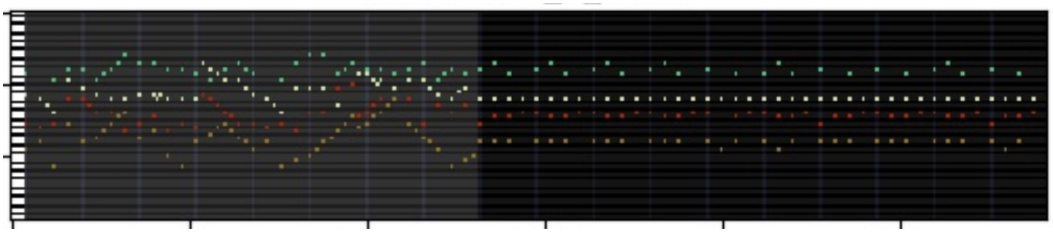


Figure 6.5: Pianoroll created by the medium model
Link

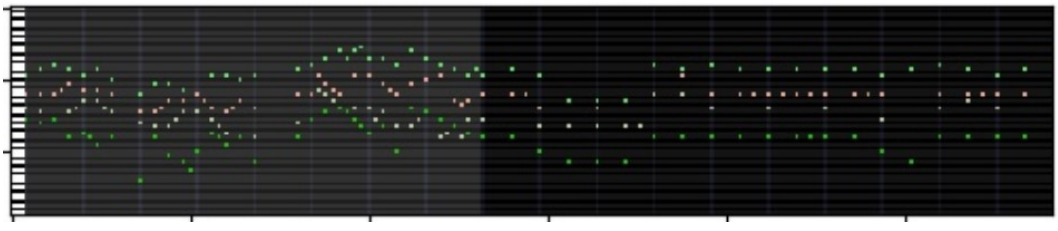


Figure 6.6: Pianoroll created by the large model
Link

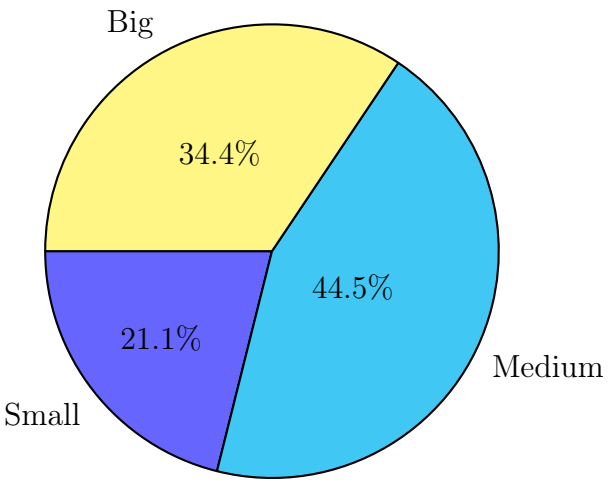


Figure 6.7: Google Form result on model size
The preferred model by the people

As shown in the figures 6.4, 6.5, 6.6, the created samples are quite similar through all the models. The generated part is looping over the same notes.

I created a Google Form so people can evaluate what is the best model. The figure 6.7 displays the result of the preference of the persons who answered the survey.

32 persons answered the google form, and surprisingly, the results follow the metrics even though the generated samples are pretty similar. This confirms that the metrics could actually be meaningful.

6.3 Custom losses

6.3.1 Scale and Rhythm

The appendix .10.3 contains useful graphs for this section. The following figures display the difference of some metrics through the training of two models. The first one doesn't use the Scale and Rhythm losses while the second one uses them.

- The figure 16 shows the loss of an output
- The figure 17 shows the accuracy of an output
- The figure 18 shows the global loss
- The figure 19 shows the value of the KLD
- The figure 20 shows the Harmony value. Both of the models are using the harmony loss
- The figure 21 shows the value of the Scale and Rhythm losses.

As we can see on the figures, there is no much difference between the two models. However, for the validation set, the model which doesn't use the scale and rhythm losses performs better than the other one for the accuracy and the loss of an output (figures 16 and 17) and the global loss and the KLD value (figures 18 and 19).

Both of the trained models here are using the harmony loss. From the figure 20, we can see that both of the models have the same harmony value on the train and validation dataset. It would mean the scale-and-rhythm losses and the harmony loss don't affect each other.

The scale and rhythm loss value in the figure 21 are obviously different between the two models since the first one doesn't use those loss functions.

The figures 6.8 and 6.9 show respectively 2 generated samples from the model using the Scale and Rhythm losses and the model which doesn't use it.

From the samples showed in the figures 6.8 and 6.9, it looks like the sample from the model using the scale and rhythm losses is more complex and has more variations than the one generated by the other model.

I created a Google Form so people can evaluate what is the best model. The figure 6.10 display the result of the preference of the persons who answered to the poll.

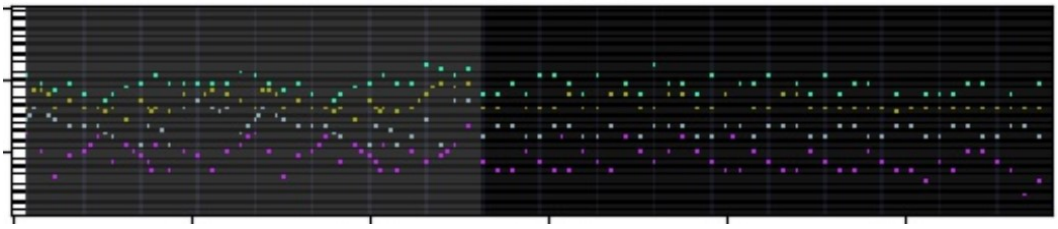


Figure 6.8: Created sample with a model trained using the Scale and Rhythm losses

(Link)

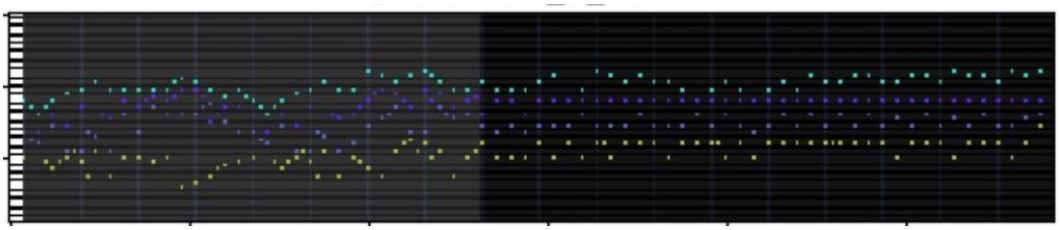


Figure 6.9: Created sample with a model trained which doesn't use the Scale and Rhythm losses

(Link)

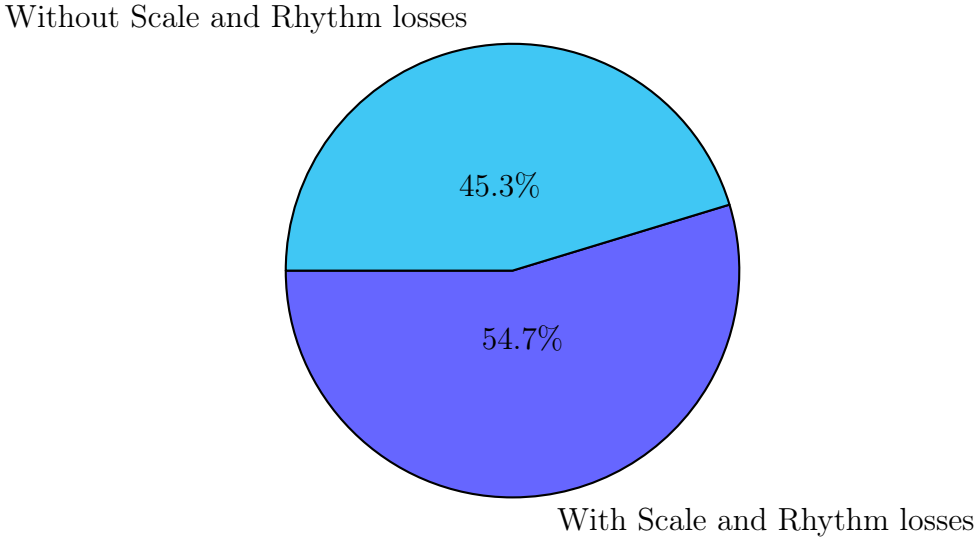


Figure 6.10: Google Form result on Scale and Rhythm losses
The preferred model by the people

17 persons answered the google form. And in that case, the result of the survey doesn't match the metrics but match my feeling about the generated samples. People slightly preferred the samples created by the model with the scale and rhythm losses.

6.3.2 Harmony

The appendix .10.4 contains useful graphs for this section. The following figures display the differences for some metrics through the training of two models. The first one uses the custom loss *Harmony*, and the second one doesn't use it.

- The figure 22 shows the accuracy of an output
- The figure 23 shows the loss of an output
- The figure 24 shows the global loss
- The figure 25 shows the harmony loss value
- The figure 26 shows the KLD value

As we can see in the figure 22, there is no difference between the two models, the accuracy of an output is basically the same. The same ascertainment can be done with the loss of an output (figure 23), the values are pretty similar.

However, the value of the loss function (figure 24) of the model which uses the Harmony loss is higher. It can been explain because the second model doesn't use the harmony loss (see figure 25), therefore the harmony loss value is not added to the global loss.

Because of time issues, I couldn't train the model using the Harmony loss too long. So I trained the first model for 20 epochs and the second one for 50 epochs. This is why the KLD value 26 between the models is different.

Therefore the β annealing (appendix .8.2.3) follows a different evolution. What we can notice, it that the final value of the KLD is basically the same for the validation set and the training set.

The figures 6.11 and 6.12 show respectively 2 generated samples from the model using the Harmony loss and the model which doesn't use it.

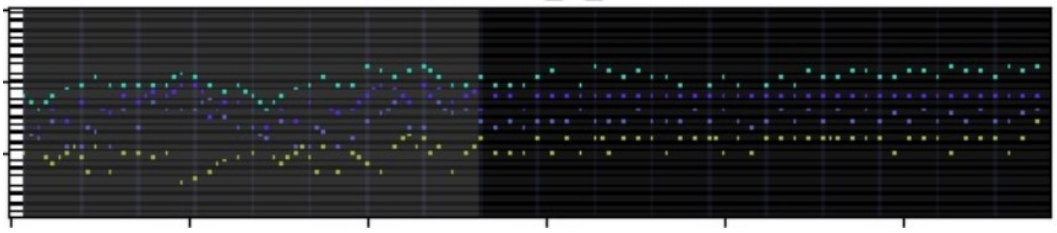


Figure 6.11: Created sample with a model trained using the Harmony Loss
(Link)

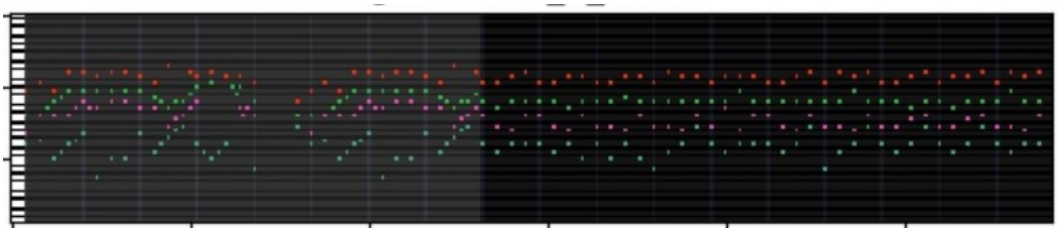


Figure 6.12: Created sample with a model trained which doesn't use the Harmony loss
(Link)

From these samples, it is not obvious to determine whether the loss is helping the model or not. Maybe the produced samples from the model using the Harmony loss are *"smoother"* than the other one, but this is very subtle.

I created a Google Form so people can evaluate what is the best model. The figure 6.13 displays the result of the preference of the persons who completed the survey.

22 persons answered the google form. Unfortunately, the people have preferred the model which doesn't use the harmony loss. It is quite disappointing and I was hoping the harmony loss would have helped the model.

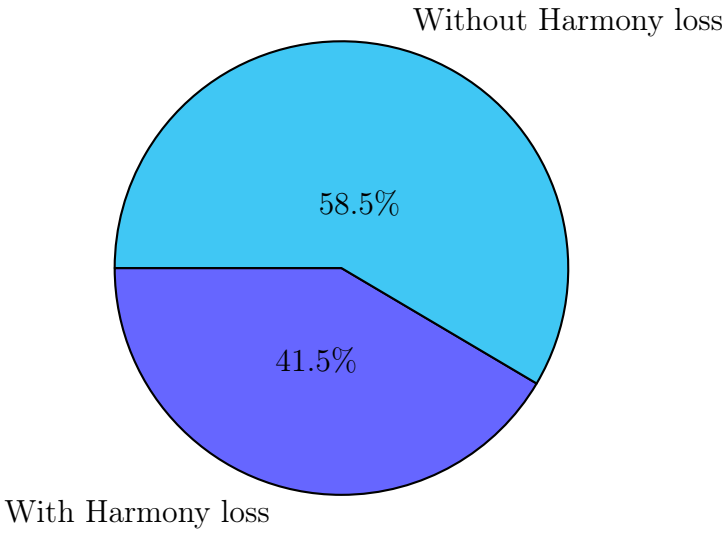


Figure 6.13: Google Form result on Harmony loss
The preferred model by the people

6.4 RPoE

The appendix .10.5 contains useful figures for this section. The following figures display the differences of the value of the several metrics during the training of two models. The first model doesn't use a RPoE layer, and the second model is using it.

- The figure 28 shows the accuracy of one output
- The figure 27 shows the loss of on output
- The figure 29 shows the global loss value
- The figure 30 shows the KLD value
- The figure 31 shows the harmony loss value
- The figure 32 shows the scale and rhythm losses value

As we can see on the figures 28, 27, 31, the accuracy and loss values of an output are not different between the two models for the training and validation

set.

On the figure 32, it appears that on the validation set, the value of the scale and rhythm losses and seems to be better for the model which doesn't use the RPoE layer.

Finally, on the figures 29 and 30, it appears the value of the KLD of the model using the RPoE is 2 times bigger than the one which doesn't use the RPoE.

From these results, it seems that the model using a RPoE layer doesn't perform better than the model that doesn't use the RPoE layer.

However, the samples sound a bit better to me when they are produced by the model containing a RPoE layer. In my opinion, there are more variations and the repeated sequence is more complex. The figures 6.14 and 6.15 show 2 generated samples. The first one comes from the model with the RPoE layer and the second one comes from the model without the RPoE layer.

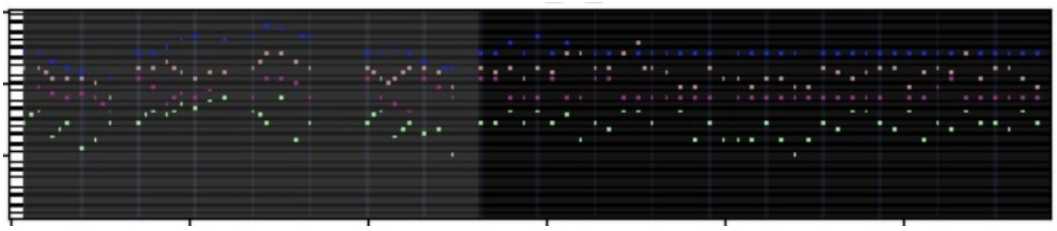


Figure 6.14: Created sample with a model trained using the RPoE layer
(Link)

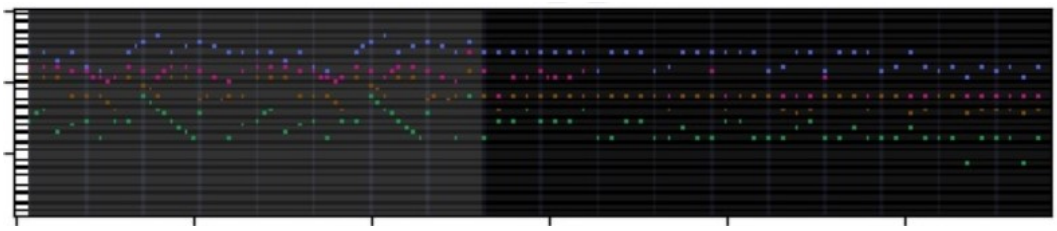


Figure 6.15: Created sample with a model trained which doesn't use the RPoE layer
(Link)

I created a Google Form so people can evaluate what is the best model. The

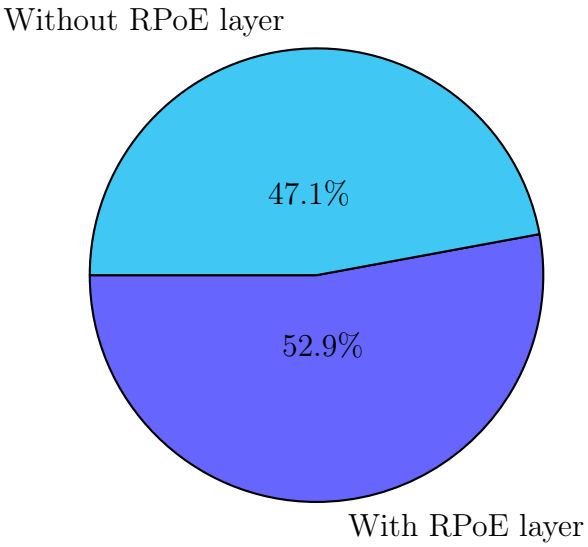


Figure 6.16: Google Form result on PRoE layer
The preferred model by the people

figure 6.16 displays the result of the preference of the persons who completed the survey.

14 persons answered the google form. In that case the results of the survey match the suppositions from the metrics. Using a RPoE layer doesn't add anything to the generation.

CHAPTER 7

Conclusion

The objective of my dissertation was to create a unique model to perform several tasks a musician could do as create a song, arrange one, create an accompaniment from a melody or create a melody from an accompaniment... However, the current works done about music generation using deep learning solution mostly focus on one objective.

In this work, I have created a new architecture called **RMVAE** (Recurrent Multimodal Variational AutoEncoder). This architecture allows me to train only one model to perform several music generation tasks as:

- Generating or completing one or several musical parts
- Creating one or several musical part(s) from one or several musical part(s). Each of them could be an accompaniment or a melody part
- Redoing a song by iteratively change a musical part after another

Moreover, the agile architecture and implementation of the **RMVAE** easily allows us to implement other tasks like reconstructing a missing measure, only consider some specific parts to generate others... I created a new neural network layer called RPoE (Recurrent Product of Expert). And finally, I came up with 3 additional loss functions to give prior knowledge to the model. These loss functions are based on my understanding of music and on some physical phenomena as the musical harmonics and the resonance phenomena.

However, even though the **RMVAE** is general and can perform many tasks, the results are far from my expectations. While DeepBach [7] or BachBot [8]

create realistic Bach's style music, my model gets stuck and usually produces the same sequence over and over. The looping sequence is usually composed of Quarter notes only (♩) and each musical parts use around 3 or 4 different notes.

I have conducted some experiments and created several surveys to allow the population to evaluate the contribution from the custom losses. Unfortunately, the results showed that a model using a custom loss were equivalent or worse than a model that doesn't use it.

Future work

I believe that this approach can lead to a well trained model able to generalize music and handle most of the tasks someone could ask to a neural network.

It looks like the text encoding (section 3.2.2.2) is the one which performs the best for Bach's style music generation [7, 8]. This is something I can't explain because letters and words structure is, in my opinion, not linked in any way to musical notes structure. Even though I don't understand it, this would be a possible solution to improve the model's performance.

The *Scale* and *Rhythm* loss functions could be ameliorated. A scale template could be created to fit the local scale. This template could contain some weights related the importance of the notes. A *strong* weight would give a *strong* reward to the produced notes. Those weights would be decreasing following the harmonics order (Tonic, Fifth, Major Third...) or the Third cycle (Tonic, Major Third, Fifth...) to fit music and chords theory. This would encourage the model to produce notes that are not already in the target tensor but will (hopefully) still be musically correct.

If a dataset contains some additional information about music (like chords, local scale...), it would allow us integrate a deeper prior knowledge to the

neural network.

The slowness of deep learning training, the leak memory issues, and the limited resources have been the biggest constraints for my Dissertation. A deep and large investigation about all the different hyper-parameters used in my project, using for instance a Grid-Search or a Bayesian-Search algorithm, would dramatically help to trained a powerful **RMVAE**.

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: towards-datascience.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: towards-datascience.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. [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 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. 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

[40] C.-H. Chuan and D. Herremans, "Modeling Temporal Tonal Relations in Polyphonic Music through Deep Networks with a Novel Image-Based Representation," p. 8.

[41] C.-Z. A. Huang, T. Cooijmans, A. Roberts, A. Courville, and D. Eck, "COUNTERPOINT BY CONVOLUTION," p. 8, 2017.

[42] 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

[43] 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

[44] 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

[45] 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

[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] "Memory leak on TF 2.0 with model.predict or/and model.fit with keras · Issue #33030 · tensorflow/tensorflow," library Catalog: github.com. [Online]. Available: https://github.com/tensorflow/tensorflow/issues/33030

[78] "Memory leak · Issue #33009 · tensorflow/tensorflow," library Catalog: github.com. [Online]. Available: https://github.com/tensorflow/tensorflow/issues/33009

[79] 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

[80] 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

[81] R. P. Adams, "A Tutorial on! Bayesian Optimization! for Machine Learning," p. 45.

[82] "scikit-optimize: sequential model-based optimization in Python — scikit-optimize 0.7.4 documentation." [Online]. Available: https://scikit-optimize.github.io/stable/

[83] 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/

[84] 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

[85] 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

[86] 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 first tried another project which was to create music with a waveform representation. Subham S. Sahoo et al. [79] present their improved network for equation learning $\text{EQL}^{\div}$, 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.

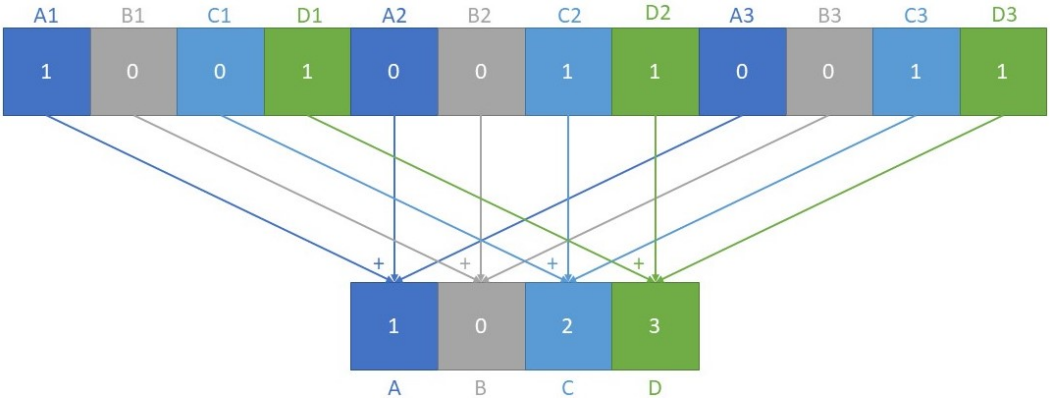


Figure 1: Segment sum operation

.3 Roll$_n$

The Roll$_n$ operation takes a one dimensional tensor, takes the first n values and put 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 ← 1 to n do
3:         firstElement ← tensor.pop(0)
4:         tensor.append(firstElement)
5:     end for
6:     return tensor
7: end function
```

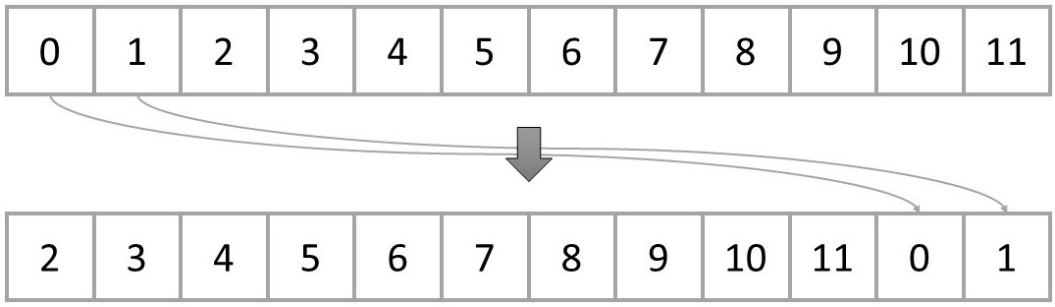


Figure 2: Roll_2 example

.4 BandPlayer

Training a model and generating music is interesting. But as musician, I wasn't fully satisfied with this application. To fulfil my needs, 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 produced 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. Therefore, 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 a researcher can have is his intuition and his understanding on how a specific parameter will influence a specific process.

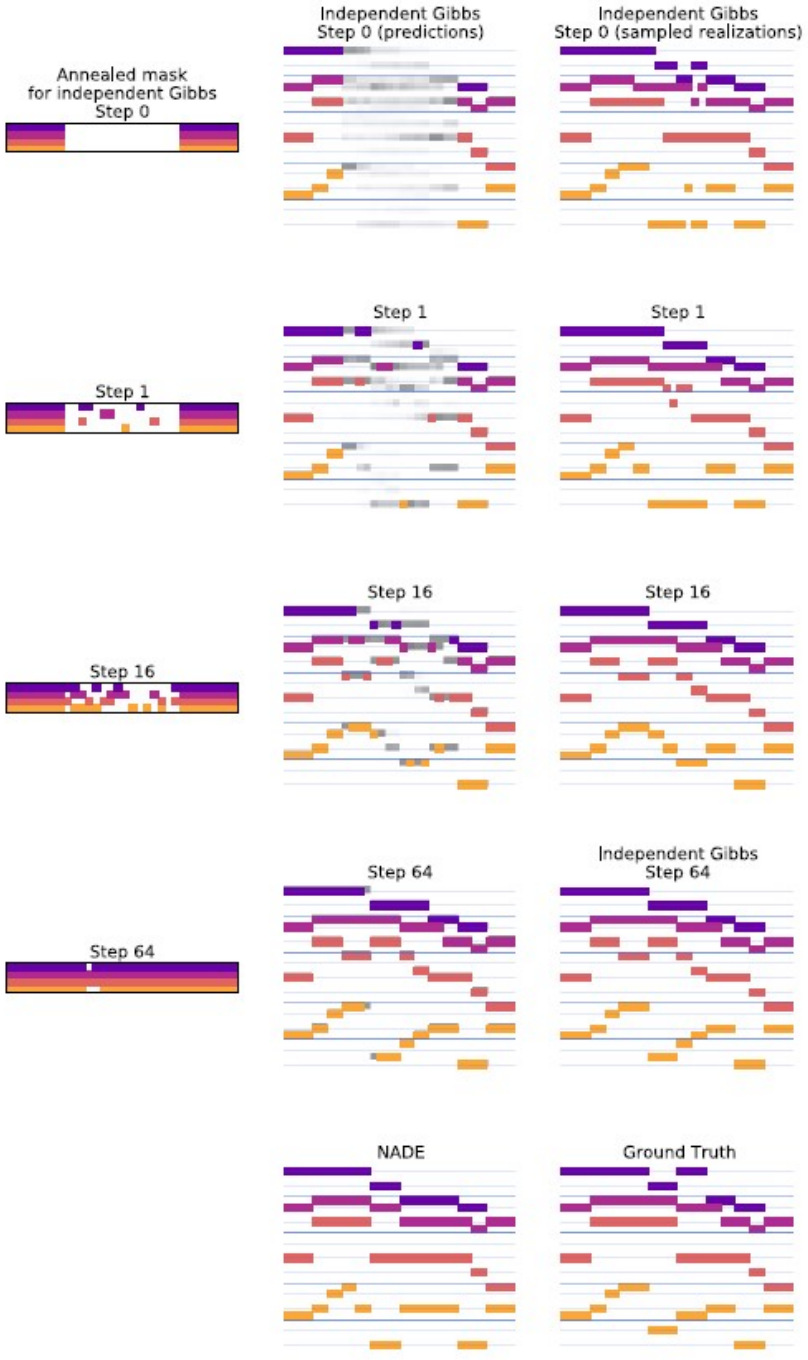


Figure 3: COCONET process
Source: Cheng-Zhi Anna Huang et al.'s paper [41]

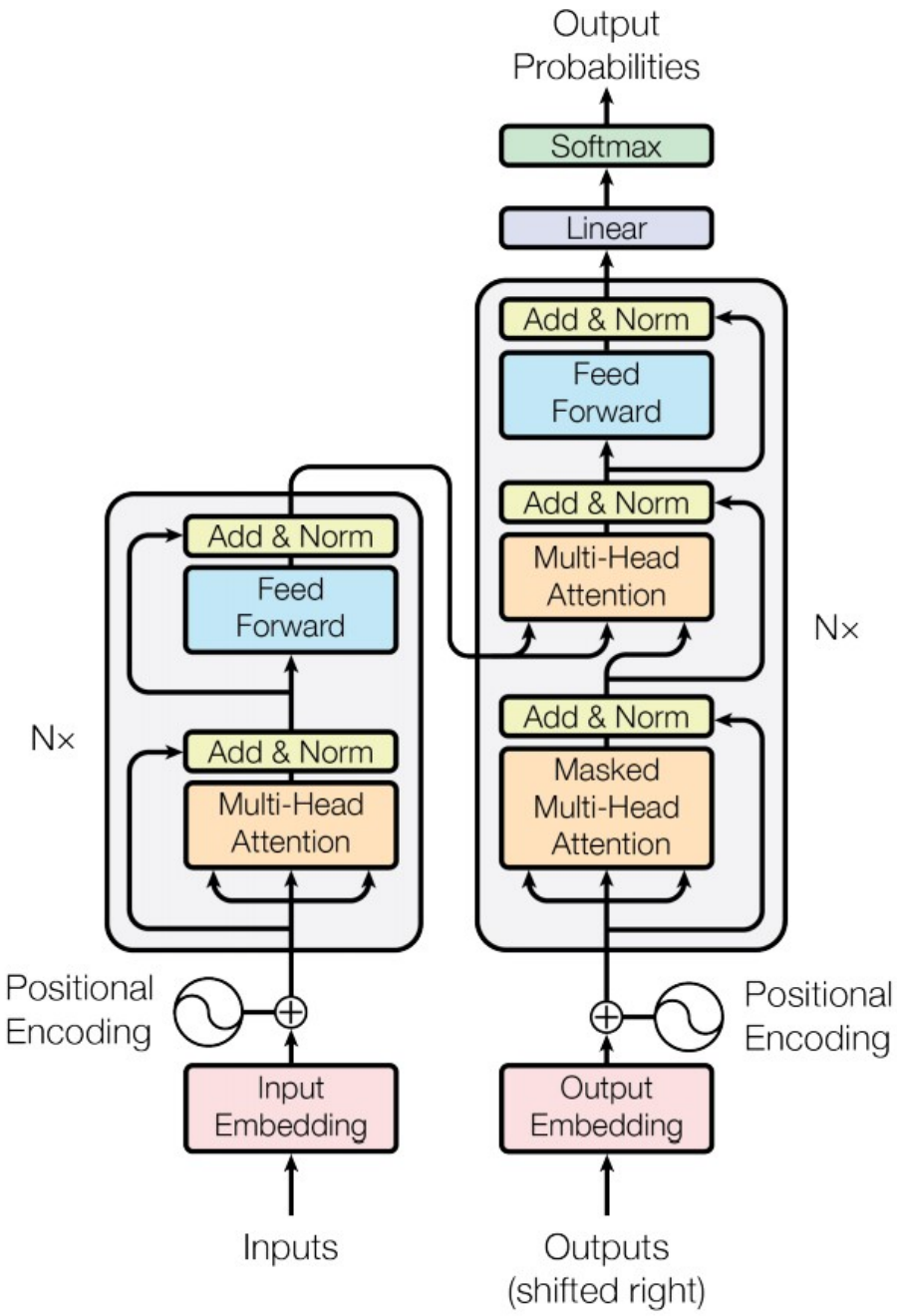


Figure 4: Transformer architecture
Source: Ashish Vaswani et al.'s paper [32]

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 train the model with all the possible combinations of parameters. However, the number of iterations grows 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 [80, 81]. 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 [82]. 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 Implementation details

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

.8.1 Network details

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

.8.1.1 Hyper Parameters

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

Hyper Parameter	Value
optimizer	Adam optimizer [83, 84, 85]
learning rate	10^{-3}
fully connected layer dropout	10^{-2}
convolutional layer dropout	10^{-3}
lstm dropout	10^{-3}
decay	10^{-1}
λ_{scale}	10^{-1}
λ_{rhythm}	10^{-2}
$\lambda_{\text{semitone}}$	10^{-2}
λ_{tone}	10^{-2}
λ_{tritone}	10^{-2}
$\text{epoch}_{\text{kld start}}$	10^{-2}
$\text{epoch}_{\text{kld stop}}$	0.4

Table 1: Neural Network Hyper-Parameters

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.

.8.1.2 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 and the model will ignore it.

.8.2 Training details

.8.2.1 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.

.8.2.2 Tensor size

For simplicity, I have written in all my dissertation that the shape of a tensor representing a measure was `(16, 128, channels)`. However, because I have been sharing a GPU from NUS with other students, I tried to reduce the size of my models. To do so, decided to reduce the pitch dimension (`(128,)`). Instead of considering all the possible MIDI notes, I consider the minimum range of notes which contains all the notes actually present in the dataset. This allows me to reduced the size from `128` to `58` (the range of the MIDI notes is $[\texttt{10}, \texttt{68})$).

.8.2.3 KLD Annealing

It is usually difficult to train a VAE because of the KLD which adds a substantial constraint on the latent space. To help the model to converge, I implemented a *KLD annealing* [86].

$$loss = loss_{output} + \beta \mathbb{D}_{KL} \tag{1}$$

I linearly increases β from 0 to 1 through the epochs from $\text{epoch}_{\text{kld start}}$ to $\text{epoch}_{\text{kld stop}}$. Before $\text{epoch}_{\text{kld start}}$ $\beta = 0$, and after $\text{epoch}_{\text{kld stop}}$, $\beta = 1$. For my project, I have set $\text{epoch}_{\text{kld start}} = 10^{-2}$ and $\text{epoch}_{\text{kld stop}} = 0.4$.

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

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

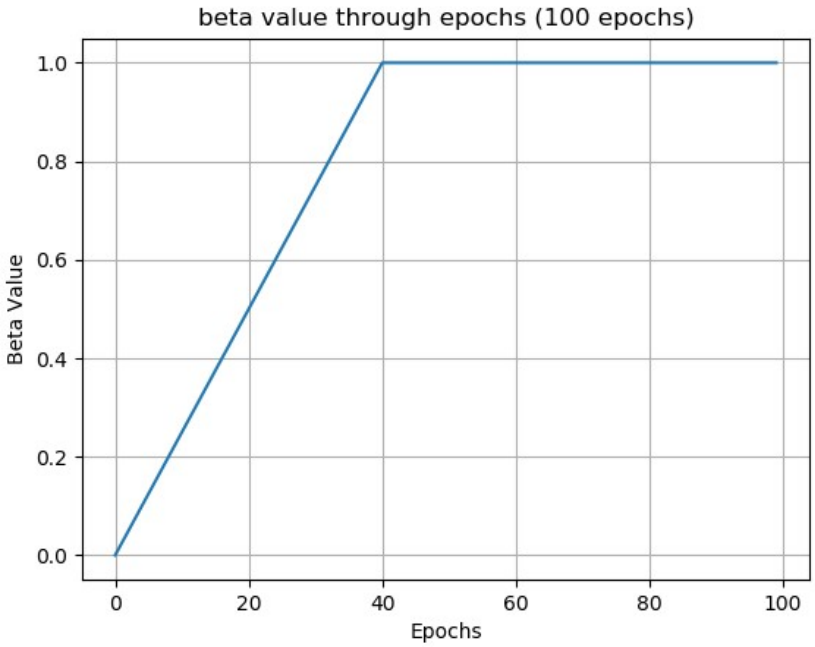


Figure 5: β value through training for KLD annealing

.8.2.4 Sub-sample Training Paradigm

As explained in Mike Wu's paper for the MVAE [1], a specific training paradigm is used to fully train the different experts.

Given a complete dataset with no missing modalities (i.e. missing instruments in my work), optimizing the equation 2.9 is not optimal: the individual inference networks are never trained and thus, the model does not know how to use them when it encounters any missing data at test time. Conversely, if the model is trained only on independent modalities, it will fail to capture the relationship between modalities in the generative model. The idea is to combine the two extremes, including ELBO terms for whole and partial observations. However, if there are N modalities, training on every possible subsets would be computationally intractable (there are 2^N subsets). To reduce the cost, the ELBO terms to optimize are sub-sampled. Specifically, they chosen ones are (1) the ELBO using the product of N Gaussians, (2) all ELBO terms using a single modality, and (3) k ELBO terms using k randomly chosen subsets X_k. For each minibatch, I evaluate a random subset of the 2^N ELBO terms. In expectation, it approximates the full objective.

The sub-sampled objective can be written as in the equation 3.

$$ELBO(x_1, \ldots, x_M) + \sum_{i=1}^{N} ELBO(x_i) + \sum_{j=1}^{k} ELBO(X_j) \tag{3}$$

In practice, $k = 2$

.9 Google Forms

For this Dissertation I created several google forms to determine what model was better than the other. Here are the links of all the google forms I have created:

- For the tranposed data (section 6.1): Google Form Link
- For the size of the model (section 6.2): Google Form link
- For scale and rhythm loss (section 6.3.1): Google Form link
- For the harmony loss (section 6.3.2): Google From link
- For the RPoE layer (section 6.4): Google Form Link

.10 Experiment plots

This section contains the plots of the different experiments from the chapter 6.

.10.1 Transposed data

This section contains the graphs from the section 6.1 (figures 6, 7, 8, 9, 10, 11).

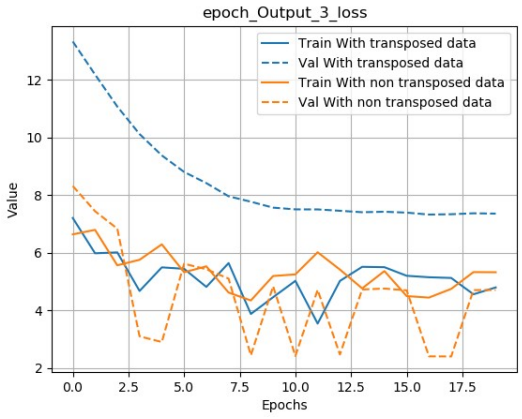


Figure 6: Loss value of an output for models training on transposed and non transposed data

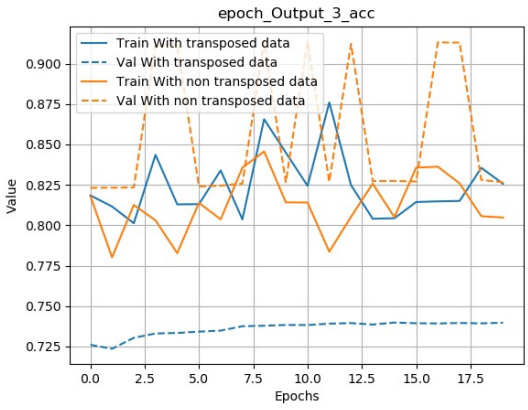


Figure 7: Accuracy value for an output for models training on transposed data and non transposed data

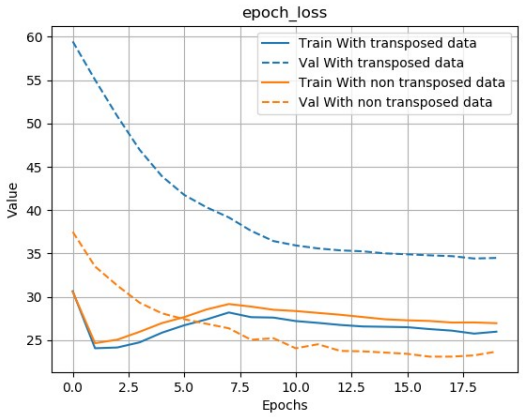


Figure 8: Global loss for models training on transposed and non transposed data

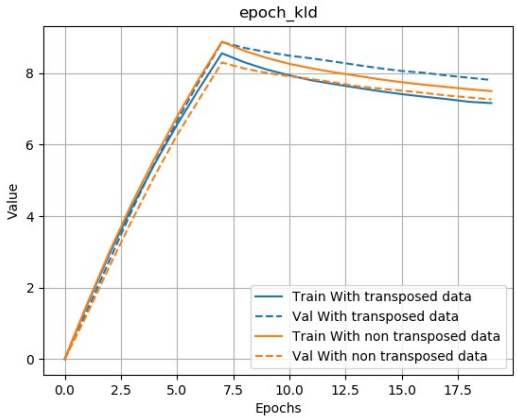


Figure 9: KLD value for models training on transposed data and non transposed data

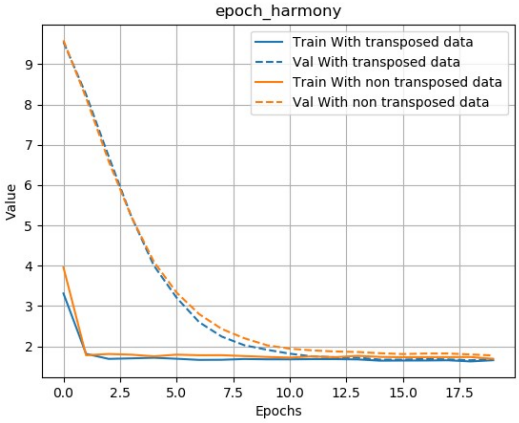


Figure 10: Harmony loss for models training on transposed and non transposed data

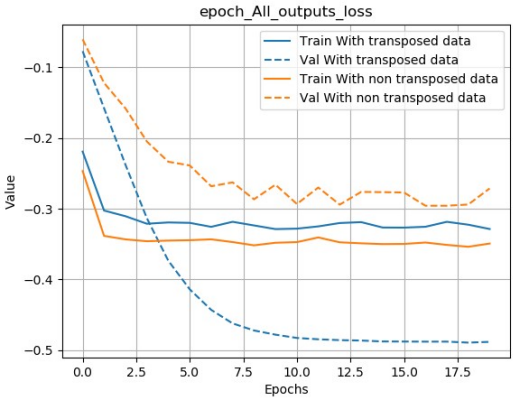


Figure 11: Scale and Rhythm loss value for models training on transposed data and non transposed data

.10.2 Model size

This section contains the graphs from the section 6.2 (figures 12, 13, 14, 15).

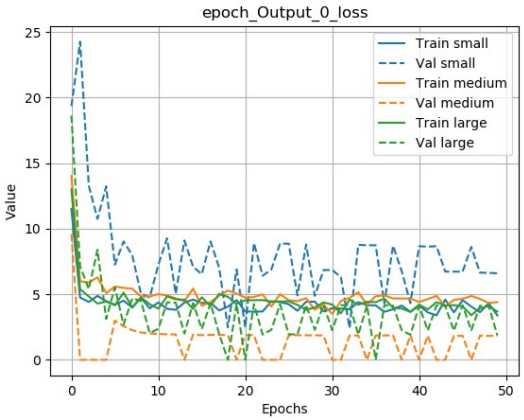


Figure 12: Loss of one voice for different size of models

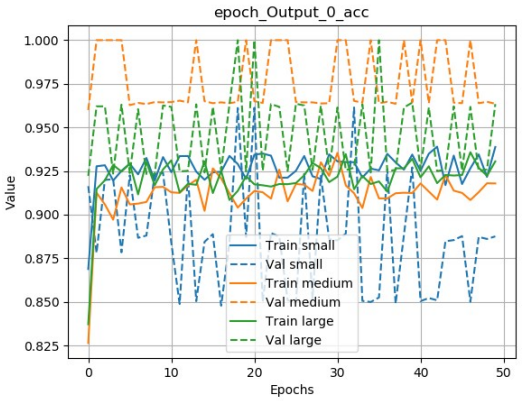


Figure 13: Loss of one voice for different size of models

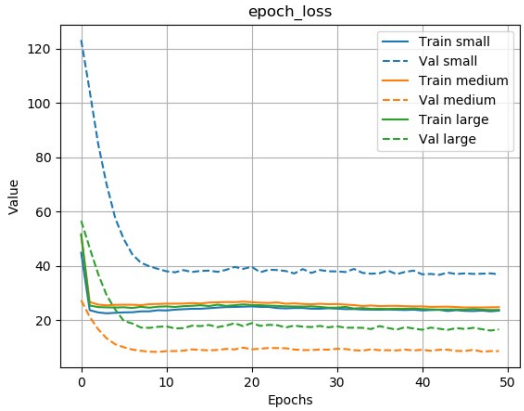


Figure 14: Global loss for different size of models

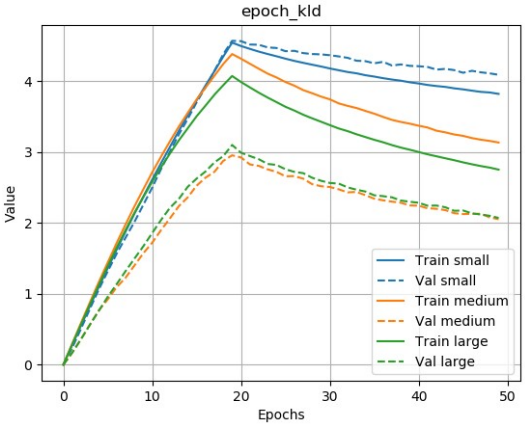


Figure 15: KLD value for different size of models

.10.3 Scale and Rhythm

This section contains the graphs from the section 6.3.1 (figures 16, 17, 18, 19, 20, 21).

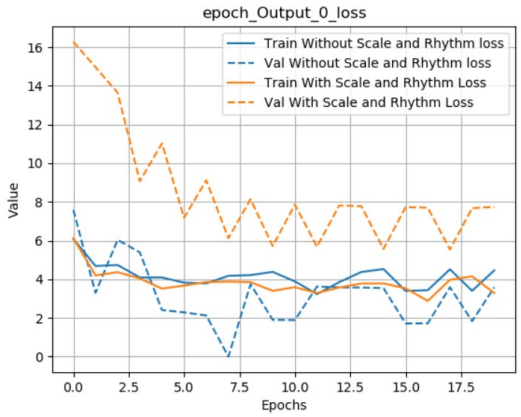


Figure 16: Loss value for one output comparison with and without scale and rhythm loss

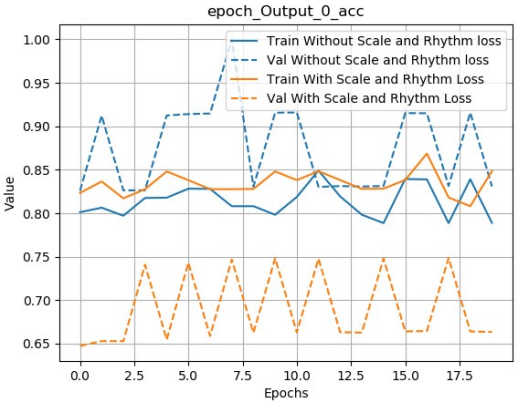


Figure 17: Accuracy value for one output comparison with and without scale and rhythm loss

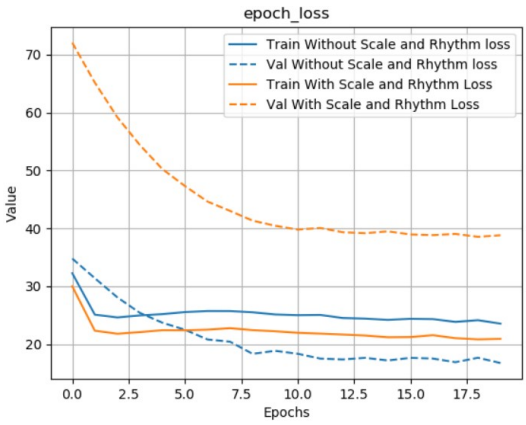


Figure 18: Loss value for one output comparison with and without scale and rhythm loss

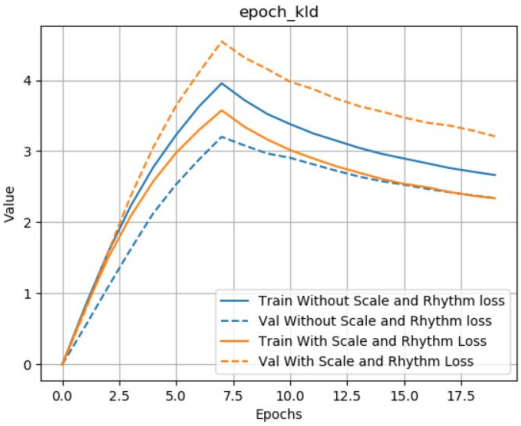


Figure 19: Loss value for one output comparison with and without scale and rhythm loss

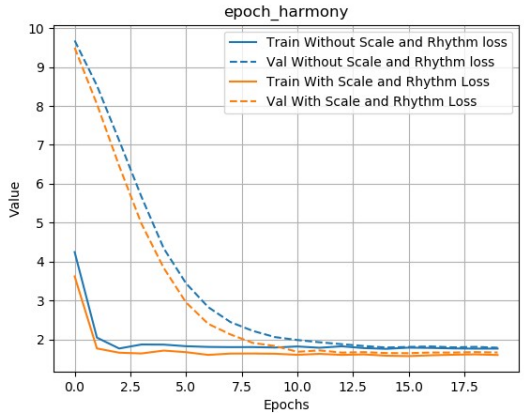


Figure 20: Harmony value for one output comparison with and without scale and rhythm loss

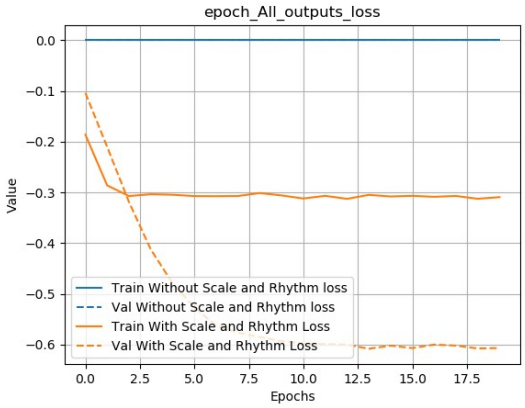


Figure 21: Scale and Rhythm value for one output comparison with and without scale and rhythm loss

.10.4 Harmony

This section contains the graphs from the section 6.3.2 (figures 22, 23, 24, 26, 25).

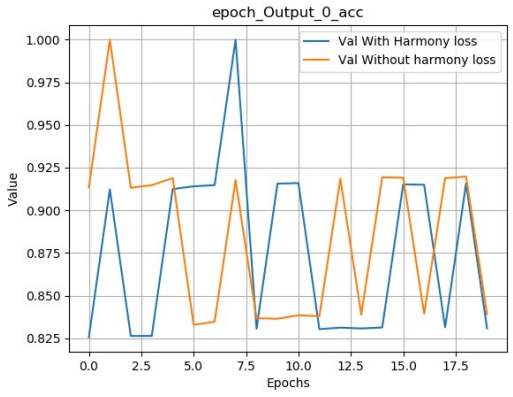


Figure 22: Accuracy value for one output comparison with and without harmony loss

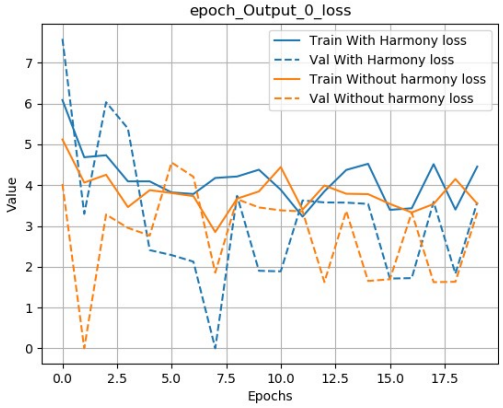


Figure 23: Loss value for one output comparison with and without harmony loss

.10.5 RPoE

This section contains the graphs from the section 6.4(figures 27, 28, 29, 30, 31, 32).

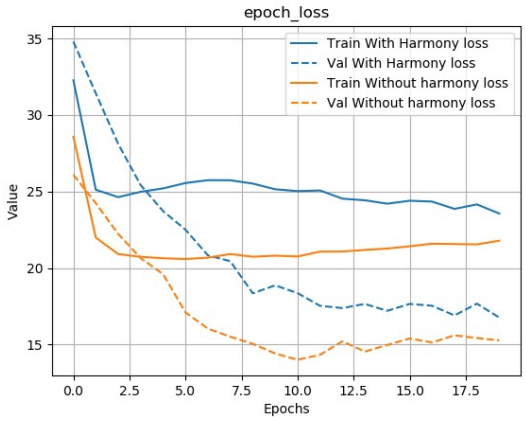


Figure 24: Global loss value comparison with and without harmony loss

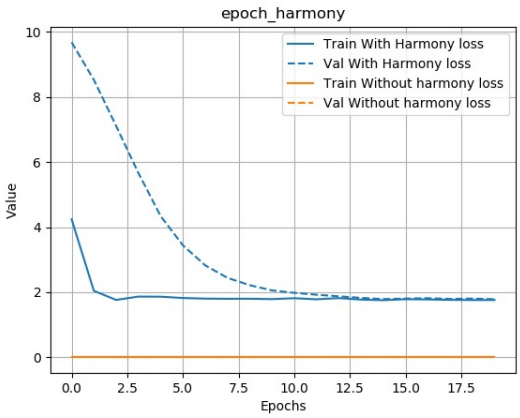


Figure 25: Harmony loss value comparison with and without harmony loss

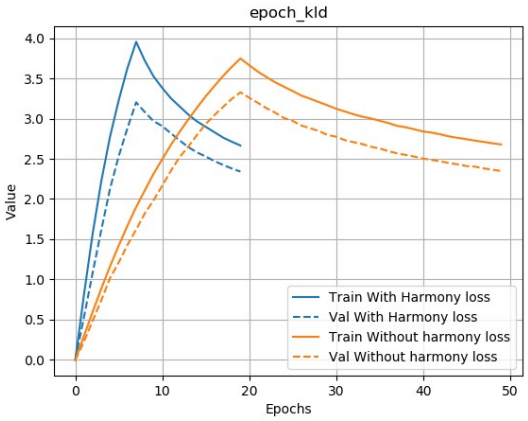


Figure 26: KLD value comparison with and without harmony loss

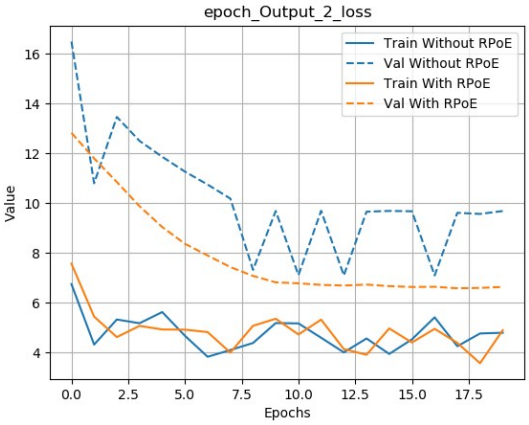


Figure 27: Loss value of an output comparison with and without RPoE layer

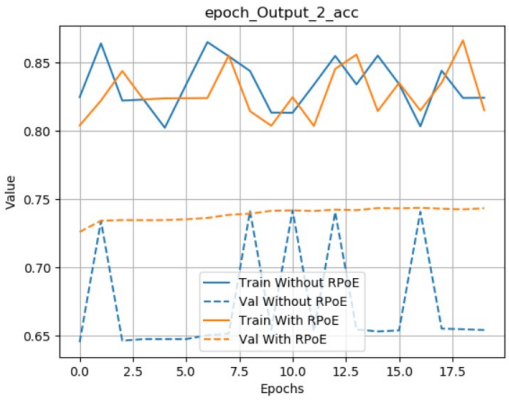


Figure 28: Accuracy value of an output comparison with and without RPoE layer

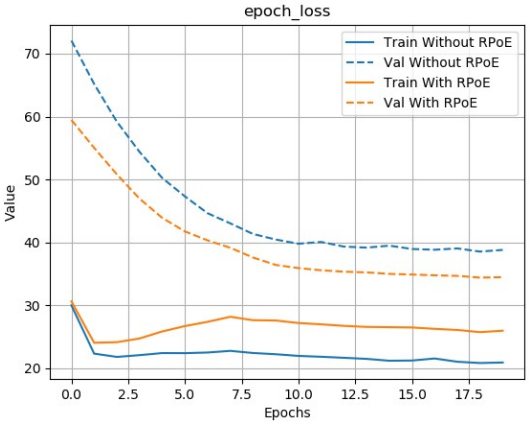


Figure 29: Global loss value comparison with and without RPoE layer

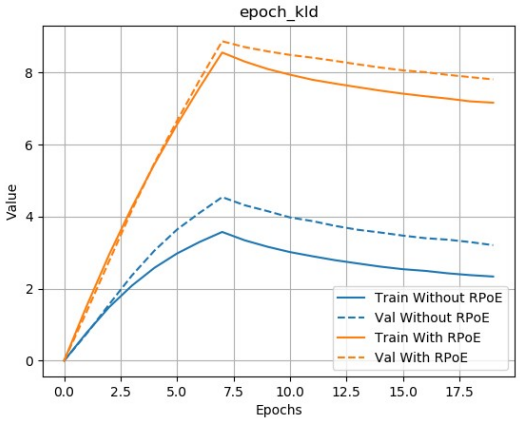


Figure 30: KLD value comparison with and without RPoE layer

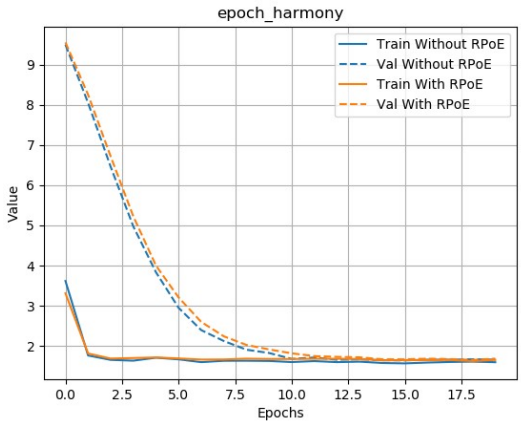


Figure 31: Harmony value comparison with and without RPoE layer

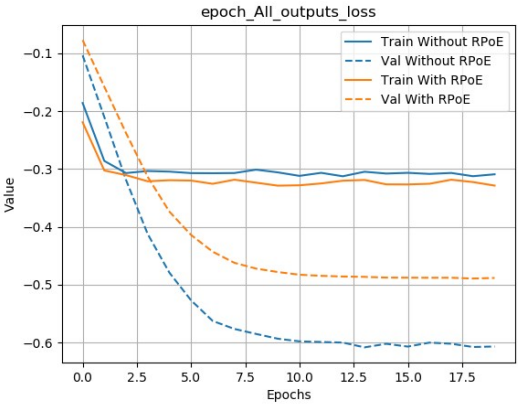


Figure 32: Scale and Rhyhtm losses value comparison with and without RPoE layer