Advanced Computer Science (Computational Intelligence) - Master of Science

Dissertation
Using deep neural networks to produce novel music

Student :
David Goerig
djg53@kent.ac.uk

Supervisor :
Dominique Chu
D.F.Chu@kent.ac.uk

Colleague :
Bastien Lecussan
bavl2@kent.ac.uk

University of Kent - School of Computing
- 2020 -

Word count : approximately 8000 words

Abstract

Music generation has been a neural network topic since 1988, Lewis and Todd created the first wave of work. They aimed to use neural networks for automatic music composition. The technique deployed by Lewis was a multi-layer perceptron (MLP) [1], and Todd used an RNN (recurrent neural network) [2].

Today the answers to this problem have grown increasingly in number and complexity thanks to technological advancement in Artificial Intelligence (AI) systems and deep neural networks. The new machine learning frameworks permit to implement different generation methods. We chose to focus on the generative adversarial network after researching other modern alternatives such as VAE (Variational Autoencoder) [3].

Our methodology comprises in improving a generative adversarial network (GAN) by using a Bidirectional LSTM in the generator [4] and a Convolutional Neural Network (CNN) in the discriminator. From our research, this model has never been tested or used in this area, although his conception fits perfectly with the music generation problem. Subsequently, we aim to assess the capacity of this architecture to generate novel music efficiently, by comparing obtained results with other models, but also with known solutions for this problem as the MuseGan [5].

Acknowledgements

I would like to thank my supervisor Dominique Chu for providing us with important support throughout the development of this project and the dissertation.

I would also like to thank Bastien for working with me during this project and my family for their support and for providing me a healthy working environment.

Finally, I want to thank the School of Computing who helped us to achieve our goal during this pandemic, and all of the tester of our research survey, which took time for helping us in the evaluation of our algorithm.

Contents

1 Introduction

1.1 Problem description

Generating novel music with artificial intelligence is a task that interests many researchers. Indeed, it is a viable solution that can and is being used by producers to help in the creative process. Moreover, it is a real challenge in term of conception because of the complex and various music architectures.

Nowadays, deep learning methods permit researchers to have novel solutions to this problem and to have better results. Famous musician already uses these solutions, and an entire industry built around AI services for creating music emerged (IBM Watson Beat, Google Magenta's NSynth Super, Spotify's Creator Technology Research Lab and many more). The aim of this generated musics are mostly to find novel inspiration, but they are also sometimes used as generated [1].

However, every song is going to vary widely, what tend to complicate learning. And there are also a lot of intricacies in music to be considered, from melody and harmony to tempo and timing. There are still a lot of differents way to do this generation, and even if researches made great progress, we are still far from replace human.

1.2 Analyses of music structure

The vast and complex music architecture is the reason of this problem complexity. Indeed, it is hard to decompose the music in different parameters that can be given as input to the neural network and keeping the links between these parameters. This part will be our main focus in this research.

As Jean-Pierre Briot clearly described it [6], the most prominent part of work is linking each element of a song : chords, harmonies, notes and rest. Almost all song are polyphonic, which consist of having two or more

1. Taryn Southern - Break free,

simultaneous lines of independent melody. This polyphony can be translated either by several instruments or by different melodies of the same instrument at different pitches to give depth to the music.
Indeed, the polyphonic and multi-track part is a vast subject and was treated independently by many researchers. However, we followed the indication given by Lee [7] and in the MuseGan [5] article for our implementation.

Our objective was to find a way to extract this information from the music to allow our model to perform the learning. For this, we chose the MIDI format and particularly the MIDI Polyphonic Expression.

1.3 Related work

Different conception and application approaches of music generation have been explored and investigated since the end of the 20th century. However, the most recent developments have more elaborate and permanent architectures and thus require more attention. The improvement the music generation quality has increased proportionally with these advancements.

The use of deep learning to generate musical content is exposed in recent surveys, which permit to assimilate different opinions and way of implementation. Briot et al. [6] expose various systems through a multicriteria analysis focusing on :

— objective (defining the musical content : melody, accompaniment, etc.) ;

— representation (the musical parameters taken as model input and how to encode them : note, chord, duration, offset, piano roll, etc.) ;

— architecture (the deep neural network models to use, which layers, etc.) ;

— challenge (the desired properties and issues : adaptability, variability, incrementality) ;

— strategy (how to handle the final music generation)

Papadopoulos and Wiggins [8] survey expose different AI-based methods for music generation and gave a try to music sequence generation with genetic algorithms [9]. Graves [10] reviewed the application of recurrent neural networks architectures and gave higher importance to the sequence generation. Fiebrink and Caramiaux [11] accorded priority to the learning part and information extraction from data (in other words to the pre-processing).

More recently MuseGan [5] became a great reference of a copyrightfree generative adversarial network used for music generation, and improved researches in polyphonic extraction by creating consistent tracks which contain bass, drums, guitar, piano and strings. A generative adversarial network (GAN) is a machine learning frameworks where two neural networks contest with each other to allow each other to learn and improve.

In parallel to this, improvements were made on generative adversarial networks by finding applications in a wider range of fields, such as image generation or unpixellization. These new applications can bring new architectures, such as the cross-domain relations used for detecting face identity patterns on images [12], or deep residual network [13].

1.4 Technology review

Nowadays, there are more modern generative models to produce novel music. As presented before different types of technology are used to produce novel music with deep learning methods. The GAN (Generative adversarial network) [14], created by Yang et al. in 2017, and in 2018 and the VAEs (variational autoencoders) [15] looked very promising for the music generation problem, this is why we had to review these technologies in order to chose the one to use for our project :

— **Auto encoder** :
  An auto-encoder is an artificial neural network used for unsupervised learning of discriminant characteristics. The objective of an auto-encoder is to learn a representation of a set of data, in order to reduce the dimension of this set. This model does not offer a lot of possibility since it reduce our objective on parameter optimization and and has a fixed output size. It is mostly used to classify, detect anomalies or remove noise from an input. [16]

— **CNN** :

The Convolutional Neural Networks (CNNs) are used to recognized and define patterns. They are mostly used in image recognition or any type of recognition. They work sequentially by applying filters to different part of the input in order to detect patterns and predict what the input is. They learn fast and provide different filters / pooling methods that make them very versatile.

— **RBM** :
This model is composed of two parts : visible units and hidden units. By design it can learn stochastic pattern. However, its simple mechanism and its small size have been surpassed by new models that also fulfill the same purpose with better performance and parameters.

— **RNN** :
The RNN are used mostly for predictions, which is good in our case. As their name says, they work recurrently by passing an input to a neural networks and a state. This state is provided by the RNN itself after each iteration in order to give the next prediction. However the RNNs present a problem due to the passing of the previous state which is the vanishing gradient problem. It is caused by the fact that the previous state is taken into account even if it is not interesting for the current prediction.

— **LSTM** :
The LSTMs are based on the same architecture as RNNs but provide a new feature called the forget gate. This gate is used to avoid the vanishing gradient problem by passing the previous state only when it is relevant.

— **Bidirectional LSTMs** :
Bidirectional LSTMs are an extension of basic LSTMs that improve model performance on sequence classification issues

by using two LSTMs instead of one. While one LSTM compute on the input sequence as-is, the second one computes on a reversed copy of the input sequence, what permits to link the data before and after by saving it.

1.5 Possible approaches and objectives

The research shows that GAN models are very promising since it is used in addition to the generator to add a second learning method. We started by having a look at : how is working an auxiliary classifier GAN [17]. This GAN aims to generate digits with label conditioning, which is a good start for music generation because it also use a sequence system.
The similar problem (digit generation) has been solved thanks to conditional generative adversarial networks. We found the model fairly easy to understand [18], but we want to avoid classification. Although the project was useful to understand the potential of generative adversarial networks and interesting in its application range.
Next to the reading of this article, we looked at different model applications to the same problem. Then, we found more convenient techniques for music generation and looked at the DCGAN [19]. The DCGAN allows us to understand the arithmetic properties of a GAN and how to manipulate the generated samples. Cycle-Consistent generative Adversarial Networks shown us another aspect of the generation capability : by applying the pattern obtained by learning a domain X to a domain Y (resulting in a new domain G) G : X -¿ Y [20]. It is applied here to image translation, but it can also be applied to music generation.

In addition to this, we decided to create a novel architecture. As far from our researches, combining a bidirectional-LSTM for the generator and a CNN for the discriminator was never tested. But the combination of these models seems very promising. The way of learning of the bidirectional-LSTM, in term of sequence management, taking into account the element before and after each element, is close to the human way of music analyses. We can easily draw the parallels to the human way of approaching music. Not analyzing music sequences note by note or chord by chord, but focusing on the links of a note or a chord with the one of before and after. The CNN is known for his performance and is often used in image analysis. However, we want to prove that it can lend itself to the task of the musical generation. Indeed, the CNN analyzes note or chord sequence chunks and tries to extract patterns from them, which is also a good answer expected for the task of a discriminator, who must understand whether it is real music or not. By using this new architecture, we hope to hear a difference in the generated song architecture, by comparison with the multilayer perceptron (MLP). We are using the comparison with the MLP model because it is a classical (direct propagation model, feedforward) and powerful model, using backpropagation, a powerful method for the calculation of the weight of neuronal links. This quantization method makes it possible to solve a problem, in this case of musical generation, therefore involves determining the best weights applied to each of the inter-neuronal connections.

Moreover, the complexity of the music lies in the fact that the generated music is often of lower quality because there is a loss of quality compared to the learning music. Indeed, many parameters are omitted/left to count or just not given as input. We are thus focused on not losing data or any music parameter. We did not focus on the instruments but the relationship of the parameters between them.

We decided to use Keras (Tensorflow) because of its good result in performance and its ease of use. Moreover, Tensorflow and Keras are the most updated at the moment, and they have a huge community behind it [21]. Additionally, it has some good additional tools such as

Tensorboard which allow us to see the model on a web interface.

2 Methods and design

2.1 Scrum methodology and project management

We wanted to follow the SCRUM methodology for this project. Indeed, we already both worked with this framework, and it permits to products faster, since each set of goals must be completed within each sprint time frame. Defining sprint and tasks permit to work efficiently, but also to plan each task and to have a broader vision on them. Nevertheless, the SCRUM methodology is not only sprint and task planning, but it also encompass group communication.

Each sprint started with sprint planning, where we setted our objectives, and distributed the tasks. The goal is to estimate the effort to allocate to each task so that it can be carried out. Then, we started each day by a daily stand-up where we tried to answer these questions : What did I finish yesterday ? What am I going to work on today ? Is there something blocking my work ? Answering these basic questions permit the coworker to have a look at the other work and thus to have an overview of the project throughout it. However, also to clarify our ideas by forcing to enunciating them.

In addition to these meeting, we planned weekly meetings that we called "Follow up" and which were used for retrospection. The aim of this follow-up is to review every task achieved since the last session in order to understand what worked or not. Thereby these sessions were used to having a review together and aimed to fulfil different methodology aspects : code review, pair-programming on important parts, objectives until the next weekly meeting, task review, checking / accepting merge requests.

We used on top of that different managing tools. To have a better view of the tasks, we used a kanban. Our gitflow helped us to work easily with branches for every project part. And we created a dynamic environment strategy, by scripting all our processes, so we could both work on the project locally on our machine and run the programs easily on a remote server. During the testing phase, we used a cloud shared folder to share schemes and plots of our work/results.

2.2 Design principles

In order to maintain a stable project, we followed all the convention and principle of a well structured and documented architecture. All scripts and appendices are named and organized according to their functionality. Each of the python scripts follows the PEP 8 convention, including code layout, naming convention, comments and programming convention. Every method and function in every script are commented following the PEP 8 style guide, but also include sphinx documentation, which permits us to export all of these code documentation lines in a document, or an HTML file. The comments tend to give additional information for a better understanding but also to explain uncertainties and rationalise decisions.

In addition to these style and programming conventions, we automatised each necessary tasks we encountered in our project. So, we scripted each step and functionality of the project :

— installation of modules (Windows / Linux) ;
— launch of the model learning phase ;
— launch of the music generation ;
— push/pull project file and resources on/from the remote server ;
— launch of the web test and survey platform.

Moreover, in order to facilitate the testing phases, and avoid time-consuming tasks, we created a configuration file, including all the constants of our project. Whether they are coefficients, file paths for music generation or reports, the name of the pickles to save the training data, these parameters are accessible in a file but can also be configured in the launch scripts.

2.3 Continuous testing, logbook and issue tracking

Through all the project, each iteration of the project was tracked in a physical notebook (to have easier note-taking). Each working day entry in the logbook contains details on the reflections of the day, the progress in our objectives but also writings and diagrams to have a better definition of our code and thoughts. To facilitate the crossing of all thoughts and changes of the project, I added the results and brainstorming of our group meetings. A start and an end header surround each daily entry. The start header allows for planning daily objectives. The end header allows talk about the result of the day, but also the tasks to be accomplished following the added project chunk. Therefore, the logbook was an ally for crossing each element and idea of the project but was also used as an issue tracker. Each difficulty was documented, highlighted, and linked to the founded solution. It permits us to draw conclusions on each issue and never be faced with it again.

The learning phase of a model and the pre-processing are time and resources consuming tasks. To adapt to this, and to have feedback on the daily changes made to the code and to the model, we had to adapt our way of testing and relying on the results of learning because a test takes hours. We thus launched tests, in a planned and recurring manner on remote servers, and scripted the launch but also the postponement of this learning. In addition to having these results generated, integrating them into the logbook was useful and saved time. It allowed to have a global vision of the tests and to analyse the evolution of the results.

2.4 Comparison between Bidirectional-LSTM/CNN and MLP/LSTM

In addition to our objectives, we have adapted our implementation in order to be able to quickly change the discriminator and generator model. Indeed, it was build in a modular way to change at any time our Bidirectional LSTM and CNN models by an MLP / LSTM, which is one of the most common models for the musical generation of generative adversarial networks. Being able to have this comparison will allow us to adapt our results better, and to be able to integrate these models on our musical generation directly, and on our sequencing will allow us to test them independently of our models. Besides, the comparison between the Bi-LSTM / CNN and MLP / LSTM models may bring more consistency to the results of our GAN. For the comparison of our final results, we also used MuseGan results in order to have a comparison.

3 Implementation

This section discusses each part of our solution and justifies our decisions made during their implementation. Our solution allows to generate music and to be able to compare our models and results to different solutions and models. Thus in this section will be described :

— Deep Learning tools and libraries which allowed us to implement all the desired functionalities ;

— the sequencing ;

— the normalization ;

— the model and layers of the generator ;

— the model and layers of the discriminator ;

— the generation of the music by feeding the resulting model.

This section also includes additional features that allowed us to perform our human tests. These features are surveys allowing the quantification of subjective human feedback, but also a web platform to allow our testers to easily request the test batteries. In addition to in-depth interviews about the opinion of testers, the process includes a set of analyses to understand and compare this data.

3.1 Libraries

The model implementation, as well as the pre-processing and the subsequent phases, are constructed based on using Python programming language version 3.7. Python has a large number of scientific libraries for data processing and machine learning approaches. The following section aims to present these libraries and to describe their relevance in this project.

3.1.1 Scientific python

This section presents the libraries that have been developed specifically for scientific work and which are used in our implementation.

Python plays a vital role in deep neural network coding languages by offering many advantages to the users. Python crosses the simplicity of the language with the speed of compiled programming languages, but also with maintained and performant scientific libraries. Therefore, highly complex algorithms can be created and test in a short time. For this project, we chose to use the combination of Scipy, Numpy and Matplotlib. The Scipy module concerns scientific computing in general (interpolation, fft, optimization, linear algebra). Some functions not present in Numpy are present in Scipy. While Matplotlib intended to plot and visualize data in the form of graphs, which is used for testing and analyzing returns

3.1.1.1 Numpy

NumPy (Numerical Python) is an extension of the Python programming language, intended to handle multidimensional matrices or arrays as well as mathematical functions operating on these arrays. More precisely, this free and open-source software library provides multiple functions making it possible to create an array directly from a file or, on the contrary, to save an array in a file, and to manipulate vectors, matrices and polynomials. NumPy is the base of SciPy, a grouping of Python libraries around scientific computing3.

3.1.1.2 SciPy and Matplotlib

SciPy is a project aimed at unifying and federating a set of Python libraries for scientific use. Scipy uses the arrays and matrices of the NumPy module. This distribution of modules is intended to be used with the interpreted language Python in order to create a scientific working environment very similar to that offered by Scilab, GNU Octave, Matlab or even R. It contains, for example, modules for optimization, linear algebra, statistics, signal processing or even image processing. It also offers advanced visualization possibilities thanks to the matplotlib module. In order to obtain excellent execution performance (weak point of the interpreted languages), most of the algorithms of SciPy and NumPy are coded in C. The NumPy module makes it possible to apply operations simultaneously on an entire table allowing to write more readable code, easier to maintain and therefore more efficient. The project follows the same philosophy as the Python language, namely code clarity, ease of learning and extensibility.

3.1.2 Tensorflow and Keras

Keras is the 2nd most widely used Python tool in the world for deep learning. This open-source library allows you to easily and quickly create neural networks, based on and allowing access to the main frameworks and in our case to Tensorflow. These libraries thus work jointely.

Tensorflow is an open-source library developed by the Google Brain team who used it initially internally. It implements machine learning methods based on the principle of deep neural networks (deep learning). A Python API is available. We can use it directly in a program written in Python.

Keras is a Python library that encapsulates access to functions offered by several machine learning libraries, in particular Tensorflow. In fact, Keras does not implement natively the methods. It simply interfaces with Tensorflow. The goal of using this additional layer is to be able to save time during the creation of the architecture by using functions and procedures that are relatively simple to implement. work to quickly achieve a result which can then be optimized and modified in detail using Tenserflow's low-level parameters and methods.

3.2 Generative adversarial network architecture

In a generative adversarial network (GAN), two models, a discriminator model and a generator model contest each other in a game to improving during the learning phase. The two models are trained and tested simultaneously. The discriminator helps to improve the generator and vice versa. The generator creates fake music samples by learning from a MIDI file database. The discriminator is feed by real samples from the database, but also by the generator fake samples and aims to recognize if it's a generated sample or not. Being fed by more and more generated music help him to improve. Parallelly, the generator learns from the output of the discriminator, and try to generate music samples that are not recognized by the discriminator, and also improving his music generation ability. The generator and discriminator help each other in learning in an adversarial way. The final result sought in our case is to have a generator which manages to fool the discriminator by giving false samples which are not detected as fake, and thus may be able to trick the human. This section aims to describe our model's implementation, but also the preprocessing and the subsequent phases to generate music. A general idea of the architecture of the complete solution is schematized in a diagram in Figure 1.

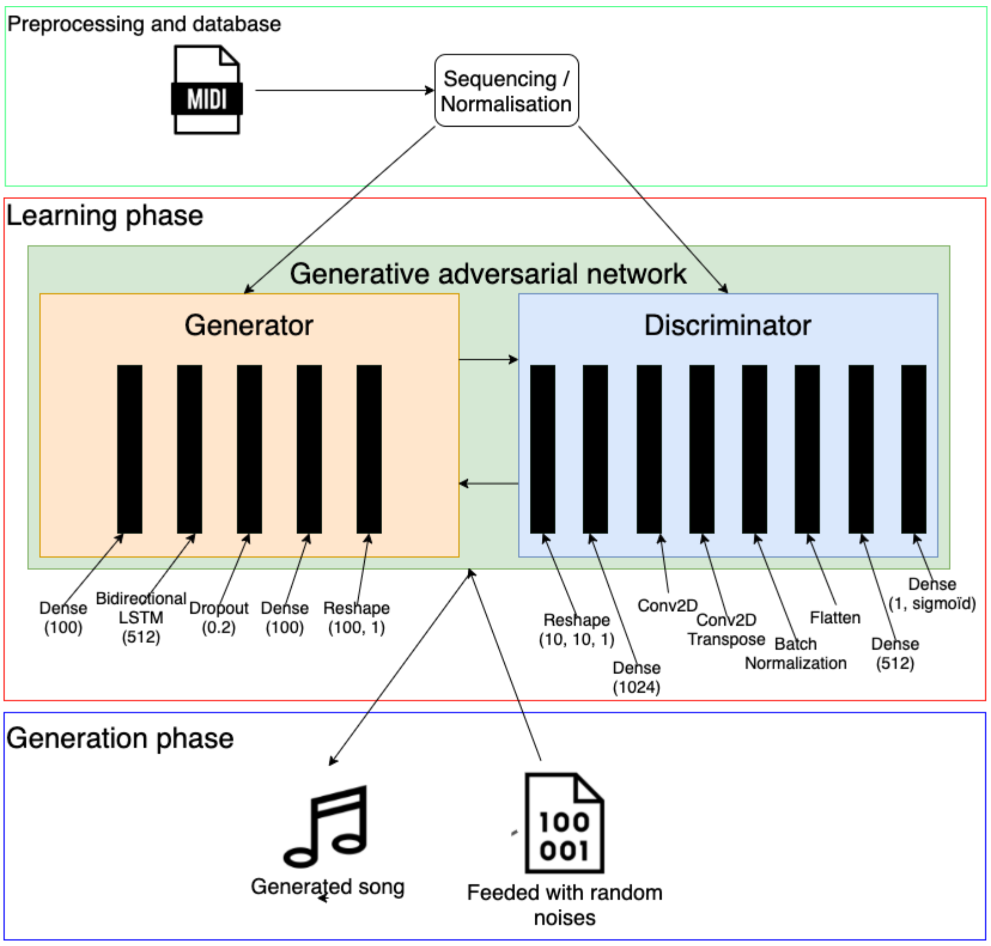


FIGURE 1 – Schematic diagram of our implementation of a generative adversarial network

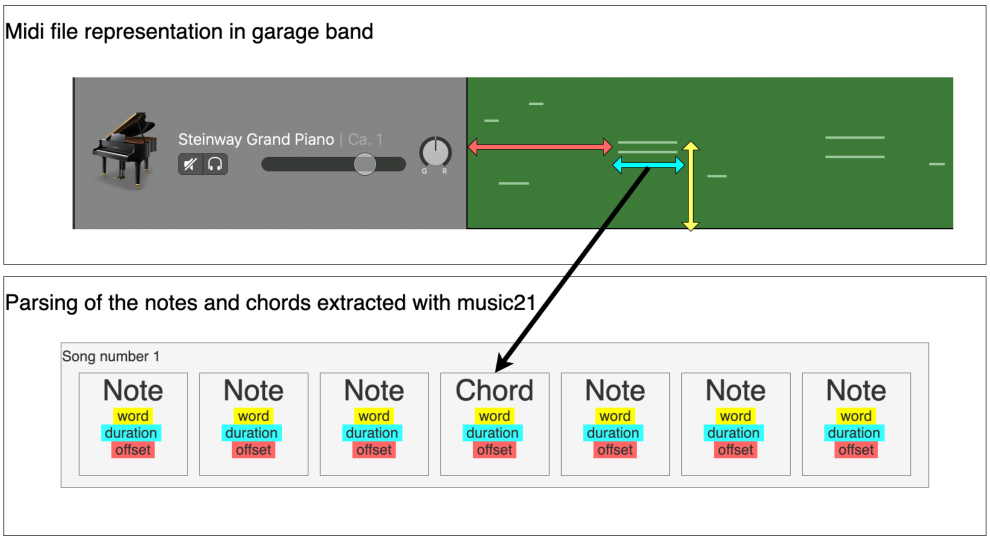
3.2.1 Data pre-processing

Pre-processing is required in order to allow the music information contained in MIDI to be usable. This step is preponderant in order to obtain a suitable result and permit our solution to gain control, expressiveness but also the randomness of music structures.
The quality of the final output is proportional to the quality of the learning of the model, to the generation, but also to the quality of the integration and the quantity of the musical parameters. In many instances, generated music tend to lose information. The data pre-processing for music generation follow three steps described in this section : data extraction from MIDI file, sequencing and normalisation.

3.2.1.1 Music parameter extraction using Music21

To extract musing information of the MIDI files, and to have a discrete sequential data that can be used as input of the models, we used Music21. Music21 is a python musicology toolkit, which allows us to acquire the musical notation of MIDI files, but also to create novel Note and Chord object and convert them to MIDI files. What fits exactly for the data extraction but also the music generation.

Music21 permits to extract in an easy manner notes and chords (groups of notes) from each instrument in the MIDI file into Python object while maintaining their sequence. Moreover, we analyzed in the generative adversarial networks range of software, treating the data as word result in promising algorithm behaviour and performant results [22]. The Music21 package permits us to extract the note and chords as text data and interprets them as "words". In addition to this, we can link the other notes and chords parameters (as offset and duration) to this word (see Figure 2).



Midi file representation in garage band
Steinway Grand Piano | Ca. 1
Parsing of the notes and chords extracted with music21
Song number 1
Note
word
duration
offset
Chord

FIGURE 2 – Schematic diagram of the main parameter extraction with music21

The choice to transcribe each music into a sequence of notes and chords that we have our hands on was made with a view to scalability. Indeed, the extracted parameters are easily configurable, and all the subsequent sequencing, normalization and generalization phases are adapted to the number of extracted parameters. We limited, in our tests, to the extraction of the duration, the offset and the octave (word note) to align ourselves with the solutions chosen to compare.

3.2.1.2 Music sequencing

To be able to feed the model with convenient input, we have created sequences of sizes n. This size n is used to define the size of the sequences, and to split each MIDI input into sequences of this size. In order to only have sequences of the same size, sequences of size less than n have a padding of size : n - missing size, which will not be taken into account by the models during the learning phase.

We chose to cut the music in sequence because what interests us is the link between the notes and chord forming a sequence and not their individuality, and having sequences of equal sizes facilitates learning. This size was set to 100 following test batteries to see which sequence size is optimal to have enough relationship between each note and chords without having to use too large sequences which could overload and overfit the models.

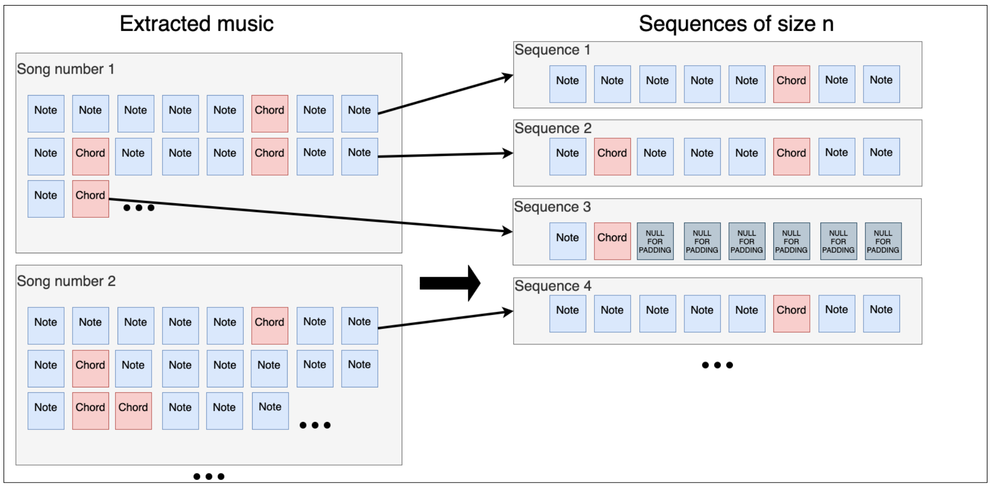


FIGURE 3 – Schematic diagram of the sequencing

3.2.1.3 Normalisation

After creating these sequences, the last step of pre-processing is data normalization. Normalization is a method of pre-processing data that helps reduce the complexity of models. It is also a prerequisite for the application of certain algorithms. Normalization standardizes the mean and standard deviation of any type of data distribution, which simplifies the learning problem by overcoming these two parameters.

To perform this transformation, we subtract from the data their empirical mean and divide them by their standard deviation (see Figure 4)). We create at the same time as the normalization of the dictionaries allowing to cross the notes and chord with their normalized value to be able to do a de-normalization work during the generation.

It is opposed to principal component analysis (PCA), which is precisely based on the relative importance of each dimension. Finally, normalisation finds the utility for us not to modify the result of the data but to convert it to a [-1 ; 1] range, which can be given as input to the generator.

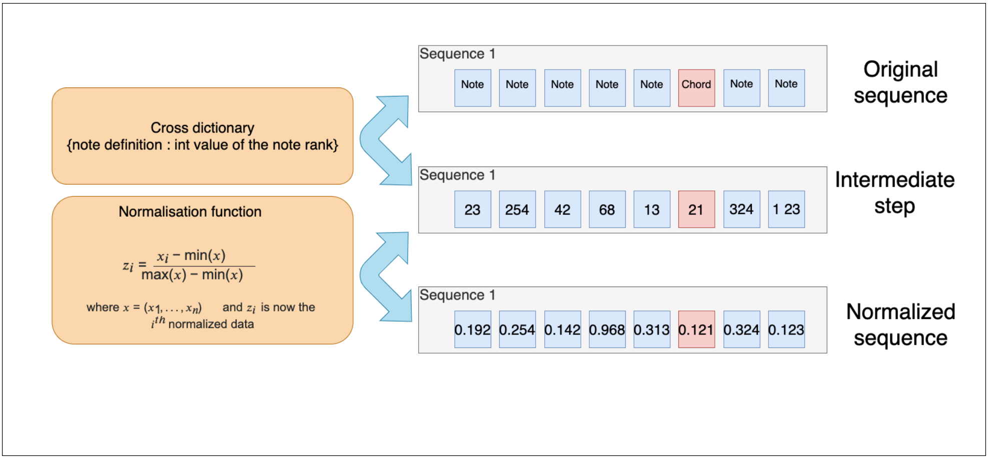

Cross dictionary
{note definition : int value of the note rank}
Normalisation function
$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$
where $x = (x_1, \ldots, x_n)$ and z_i is now the i^{th} normalized data
Sequence 1
Note Note Note Note Note Chord Note Note
Original sequence
Sequence 1
23 254 42 68 13 21 324 1 23
Intermediate step
Sequence 1
0.192 0.254 0.142 0.968 0.313 0.121 0.324 0.123
Normalized sequence

FIGURE 4 – Schematic diagram of the sequence normalisation

3.2.2 Models - Discriminator and Generator

To choose the models, we wanted to follow our personal experience to create music, but also to try to understand the human approach. In a music group, musicians can play different instruments. Still, they all aim to improve music without a predefined arrangement but mostly by improvising. This improvisation is not natural but translates with knowledge of harmonic structure and instrumentation. We wanted to follow this approach of ”improvisation with knowledge” by choosing the Bidirectional LSTM and the CNN. The reason and implementation are described in this section.

3.2.2.1 Generator : Bidirectional LSTM

The choice of the bidirectional LSTM model was made because we can make the parallel to the human way of approaching music, by not analysing a sequence note by note but by focusing on the links of a note or of a chord with that of before and after. Indeed the bidirectional LSTM evaluate the before and potential after notes. To model long-term dependencies, it is necessary to give recurrent neural networks the ability to maintain a state over a long period. This is the purpose of LSTM (Long Short Term Memory) cells, which have an internal memory called a cell (or cell). The cell helps maintain a state as long as necessary. This cell consists of a numerical value that the network can control depending on the situation. In most sequence analysis problems, it is useful to have a memory of the past to make good decisions at time t. But in certain recognition problems, like in ours with music analysis, it is sometimes interesting to look at future observations, when they are available, hence the use of bidirectional LSTM. Two layers are combined to make the best local decisions, having analysed the entire signal.

But a model can only contain one layer and must know what form of input it should expect, prepare the output and also contain other processing layers. This model is composed of (see Figure 1) :

— A first **Dense** layer which makes it possible to carry out first processing of the raw data and to evaluate the different relation. It also helps stabilise the loss of data generation. The first layer of a sequential model must be given the characteristics of the input shape (other layers can determine the shape of their inputs by inference).

— A **Bidirectional LSTM** layer, which is the central and main layer of the model. This layer is used to generate sequences by evaluating the before notes and the potential after notes of each note. It contextualise each note by using the data before and after.

— A regularisation technique layers, **Dropout** type, occurs to avoid overfitting. This step allows differentiation of the generated sequence from the original inputs, and thus to create music different from that given in input.

— A **Dense** layer, also named fully connected layer, is a classic neural layer, completely connected with the previous layer and the next layer. Each neuron can be activated or not depending on the activation function. We have chosen for this to use a hyperbolic tangent activation function. After these treatments, the generator can produce a probability distribution which make it possible to make predictions (see Section 3.2.3).

— A **Reshape** layer to set the output to a sequence of the chosen size.

The sizes of the layers are evaluated related to the output of the previous layers, the model input or the desired output. While the loss and the accuracy following adaptability tests.

3.2.2.2 Discriminator : Convolutional Neural Network (CNN)

The purpose of the discriminator is, therefore, to extract the structure of melodies to be able to deduce whether it is real or fake music. But the goal is also to train the model to a stage of convergence where the generator can fake the discriminator. So we wanted to use a convolutional neural network (CNN). It is a type of acyclic (feed-forward) artificial neural network, in which the visual cortex of animals inspires the connection pattern between neurons. Neurons in this region of the brain are arranged regarding overlapping regions when tiling the visual field, which corresponds to a piece-of-sequence filter system. A significant advantage of convolutional neural networks is the use of a single weight associated with the signals entering all the neurons of the same convolutional nucleus. This method reduces the memory footprint, improves performance and allows translation processing invariance. Our research has shown that the use of this model corresponds perfectly to the problem of pattern detection thanks to the filter, which applies to musical analysis. This model was, for instance, used in the MuseGAN [5]. Biological processes inspire their operation ; they consist of a multi-layer stack of perceptrons, the purpose of which is to preprocess small amounts of information without loss. The preservation of model information and inspiration of biological process corresponds to the subjective human approach of his relationship to music, what made us choose this model. The formation of the discriminator layers, used in the principle of extracting the structure of the music, is composed of :

— a **Dense** input layer used like that of the generator, which allows the first layer of the model to gain stability and to take care of the relations of the input data. Indeed, it allows a better evaluation by the kernels because there is no pre-training).

— A first layer of convolution in 2 dimensions (**Conv2D**) to apply a fairly large kernel (5 * 5) to search for global patterns

— A second, smaller two dimension convolutional layer (**Conv2DTranspose**) which does not reshape the output (because this layer transposes the data), which makes it possible to methodically compare the patterns between neighbouring notes with a stride of (2,1). Using a stride of this size allows you to perform a step of two and thus refine the results because the correlation is less punitive.

— A normalization layer (**BatchNormalization**) to improve the performance of the learning of our final activation function, which is of sigmoid type.

— A **Flatten** layer which allows to completely reshape the data on a single level and thus adapt the output because the CNN requires an internal vector in 4 Dimensions, which is not suitable for the activation function.

— To finish, a fully connected layer (**Dense** layer), using the sigmoid activation function, allow the model to produce the final results. The filters of fully connected layers are determined by tests and optimizations.

We have chosen this model because we believe it may be a better alternative than MLP. Although sufficient for image processing, multilayer perceptrons (MLP) have difficulty handling large sequences, due to the exponential growth in the number of connections with image size, since each neuron is "fully connected" to each of the neurons of the preceding and the following layer. And this problem is all the more present in music due to the number of parameters it contains.

Convolutional neural networks aim to limit the number of entries while maintaining the strong "spatially local" correlation determining the sequence, and in our case music parameters. As opposed to MLPs, CNNs have the following distinguishing features :

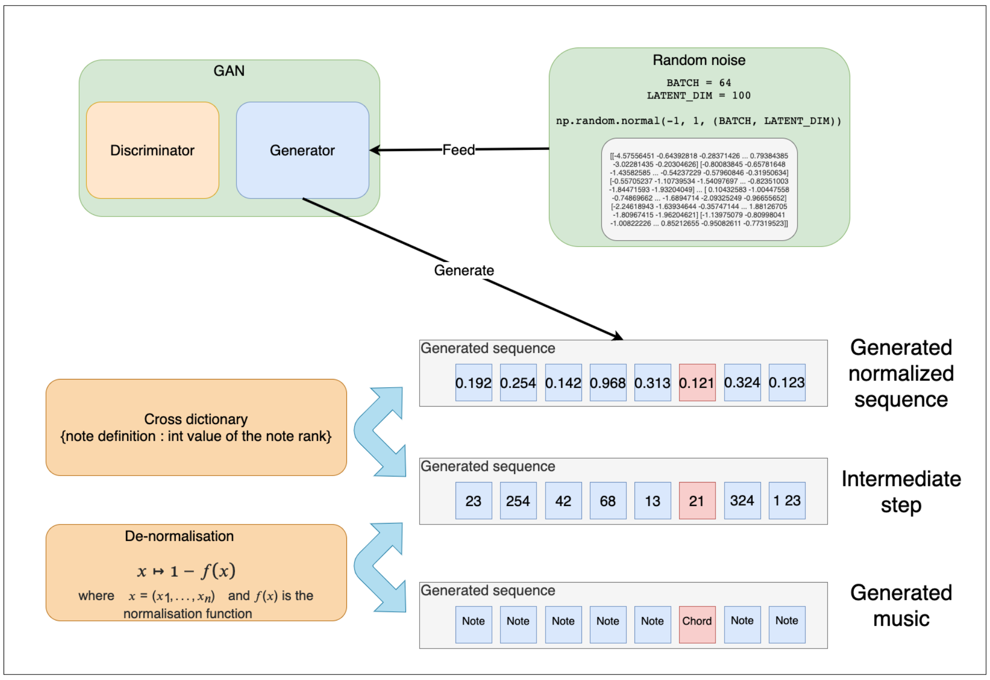
— **Local connectivity** : thanks to the receptor field which limits the number of inputs to the neuron, while retaining the MLP architecture, convolutional neural networks thus ensure that the ”filters” produce the strongest response to an input pattern spatially localized, which leads to a parsimonious representation of the entrance. Such a representation occupies less space in memory. Besides, the number of parameters to be estimated is reduced, their estimation (statistical) is more robust for a fixed volume of data (compared to an MLP).

- **Shared weights** : in convolutional neural networks, the filtering parameters of a neuron (for a given receptor field) are identical for all the other neurons of the same group. This setting (weight vector and bias) is defined in a ”function card”.
- **Translation invariance** : as all the neurons of the same nucleus (filter) are identical, the pattern detected by this nucleus is independent of spatial location in the sequence.

The results of the training tests showed us that these properties allow convolutional neural networks to obtain better robustness in the estimation of parameters on training problems since, for a fixed training corpus size, the quantity of data per parameter is greater. Weight sharing also makes it possible to considerably reduce the number of free parameters to be learned, and thus the memory requirements for the operation of the network. The reduction of the memory footprint allows the learning of larger networks and therefore, more powerful.

3.2.3 Generation

After training both of the generator and discriminator, we aim to generate music sequences. But for creating these sequences and convert them to MIDI files with music21, we need to generate a sequence of note and chords predicted by the models. In the case of our GAN network, we feed the generator with a random sequence of wanted sequence size (see Figure 5) to make the prediction. This random sequence is composed of numbers sampled from a standard normal distribution. After feeding the models, we obtain a normalized sequence. The last task is to reverse the normalization of the notes and to link the result with the assigned notes and parameters stored in the cross dictionaries.



```
BATCH = 64
LATENT_DIM = 100

np.random.normal(-1, 1, (BATCH, LATENT_DIM))
```

```
[[-4.57556451 -0.64392818 -0.28371426 ... 0.79384385
-3.02281435 -0.20304626] [-0.80083845 -0.65781648
-1.43582585 ... -0.54237229 -0.57960846 -0.31950634]
[-0.55705237 -1.10739534 -1.54097697 ... -0.82351003
-1.84471593 -1.93204049] ... [ 0.10432583 -1.00447558
-0.74869662 ... -1.6894714 -2.09325249 -0.96655652]
[-2.24618943 -1.63934644 -0.35747144 ... 1.88126705
-1.80967415 -1.96204621] [-1.13975079 -0.80998041
-1.00822226 ... 0.85212655 -0.95082611 -0.77319523]]
```

FIGURE 5 – Schematic diagram of the music generation by feeding the generator with random noise

3.3 Features

To have human results efficiently (see Section 4.1) by giving an attractive appearance to testers to help us in our research, we have developed and implemented tools which are described in this section.

3.3.1 Web application

We created a web platform [2] to allow testers to have easy access to forms, but also to be able to access the music to be evaluated. Indeed, our web server hosts a pre-trained version of our GAN which will generate a MIDI file with prediction and generation on demand. The saved model is updated thanks to the automatic deployment after each training. The web platform is hosted on Heroku, and we used a python backend with the Flask framework to easily adapt to Keras models.

3.3.2 Forms for subjective analyses

To be able to analyze our results in a subjective way, we created forms as well as analysis tables aimed at questioning users on the musical aspect. To have relevant results, we target users with a musical background. We have therefore focused our questionnaires on simple questions but allowing us to have answers to specific different issues :

— **How real ?** To have subjective results of our generation of music, focusing on the result of the combination of parameters which is one of the main characteristics of music that appears real, and our main challenge.

— **How pleasing ?** To have indications on how sounds the link between the notes and chords of our result, and analyze whether the use of a new model (Bidirectional LSTM), emphasizing the treatment of parameters by linking them with those of before and after, had the desired effect.

2. https ://backend-dissertation.herokuapp.com/

— **How inspiring ?** To analyze if the results have the expected interest for musical composition, and that they differ from the already existing music. It is needed that the generated melodies are not copies of the music given as input.

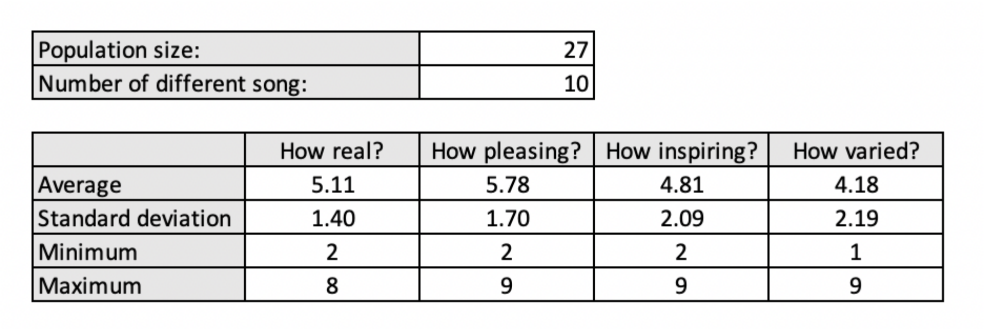
— **How varied ?** To evaluate the capacity of our model to create different results by feeding the model with random sequences, but also to test the convergence of the model.

User research was done in our private social networks as well as in forums music-oriented. To be able to process the results obtained, we created Excels scripts using the results of the forms to calculate data (average, variance, maximum, minimum) and give us graphical representations.

4 Results and analysis

4.1 Human evaluation/survey

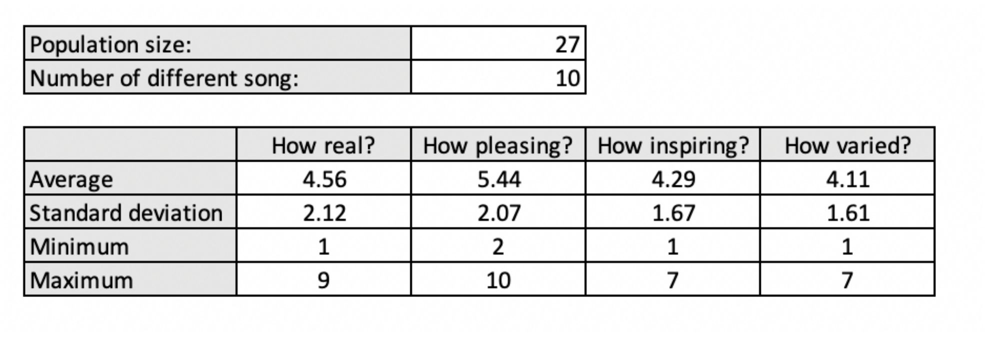
Finally, we conducted a listening test of 27 subjects recruited from the Internet via forums and our social circles. All of them were categorised as "user with music background" according to a simple questionnaire probing their musical background. Each subject has to listen to 30 MIDI files of the same sequence size, which is not equal to the music duration depending on the tempo. These MIDI files contains ten results from our tested GAN Bi-directional LSTM / CNN (see Figure 6), ten results for our implementation of MLP/LSTM with the same pre-processing, normalisation and generation (see Figure 7) and ten results of the MuseGAN [5] (see Figure 8). The subject rates the MIDI file requested on the web platform in terms of whether they 1) sounds real ; 2) have unified rhythm ; 3) have definite and coherent musical structure ; 4) are varied from each other ; on a 10-point scale with one being "not real/unified/coherent/varied at all" and ten being "absolutely real/unified/coherent/varied".



Population size:	27
Number of different song:	10

	How real?	How pleasing?	How inspiring?	How varied?
Average	5.11	5.78	4.81	4.18
Standard deviation	1.40	1.70	2.09	2.19
Minimum	2	2	2	1
Maximum	8	9	9	9

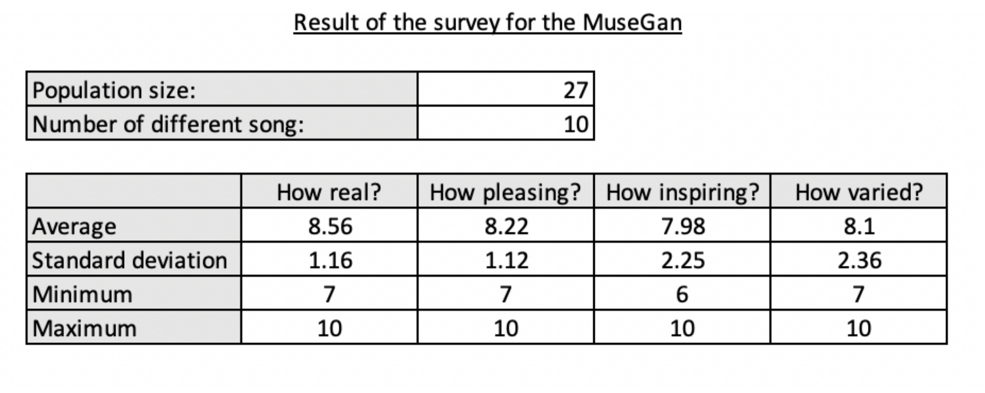
FIGURE 6 – Result of the survey for the Bidirectional LSTM / CNN model



Population size:	27
Number of different song:	10

	How real?	How pleasing?	How inspiring?	How varied?
Average	4.56	5.44	4.29	4.11
Standard deviation	2.12	2.07	1.67	1.61
Minimum	1	2	1	1
Maximum	9	10	7	7

FIGURE 7 – Result of the survey for the MLP / LSTM model



Result of the survey for the MuseGan

Population size:	27
Number of different song:	10

	How real?	How pleasing?	How inspiring?	How varied?
Average	8.56	8.22	7.98	8.1
Standard deviation	1.16	1.12	2.25	2.36
Minimum	7	7	6	7
Maximum	10	10	10	10

FIGURE 8 – Result of the survey for the MuseGan

Thanks to the result of these tests we can see that our model gives convincing results and has a contribution on the quality of the melody by taking into account the relationship between notes and chord, and by not being overloaded. Besides, we have proof that our efforts on adding numerous musical parameters while avoiding losses allow melodies to appear relatively real. However, having taken the MuseGAN as a comparison distorted the subjective opinion of our results. Indeed, the presence of several instruments in a melody (which is a particularity of MuseGAN and which knows only a few implementations) adds a real impression of reality because it is in this form that the music is mostly consumed. We think our results are conclusive because we produced a better result with the new architecture (Bi-directional LSTM/ CNN) than our implementation of the control model (MLP / LSTM). However, we saw the limits of subjective tests. The evaluator must identify their definition of creativity, identify the standards, and then test the system using those standards. Our goal was the diversity of the parameters and the coherence of the sound but not the multiple instrument management. The allotted time of the project does not allow us to concentrate on this part of the music generation.

Consequently, we noted that from a musical point of view, models produced with only one instrument created much better results than when we used all instruments. The fact that we have deliberately omitted the generation of melody by instrument and their linking, to concentrate on the presence of the multiple parameters and the performance of our architecture, gave results that are less pleasant to the ear because they are monophonic. However, when we compared these monophonic results, we could notice that the results are convincing and that the scores in the surveys increased considerably compared to music with multiple instruments.

4.2 Evaluation of the loss of the training

For a model, learning means determining the correct values for all weights and bias from labeled examples. GANs attempt to reproduce a probability distribution. They thus use loss functions that reflect the distance between the distribution of data generated by the GAN and the distribution of actual data.

So the loss is a number that indicates how poor the model prediction is for a given example. If the model prediction is perfect, the loss is zero. Otherwise, the loss is greater than zero. The aim of training a model is to find a set of weights and biases for which the loss, on average over all examples, is small.

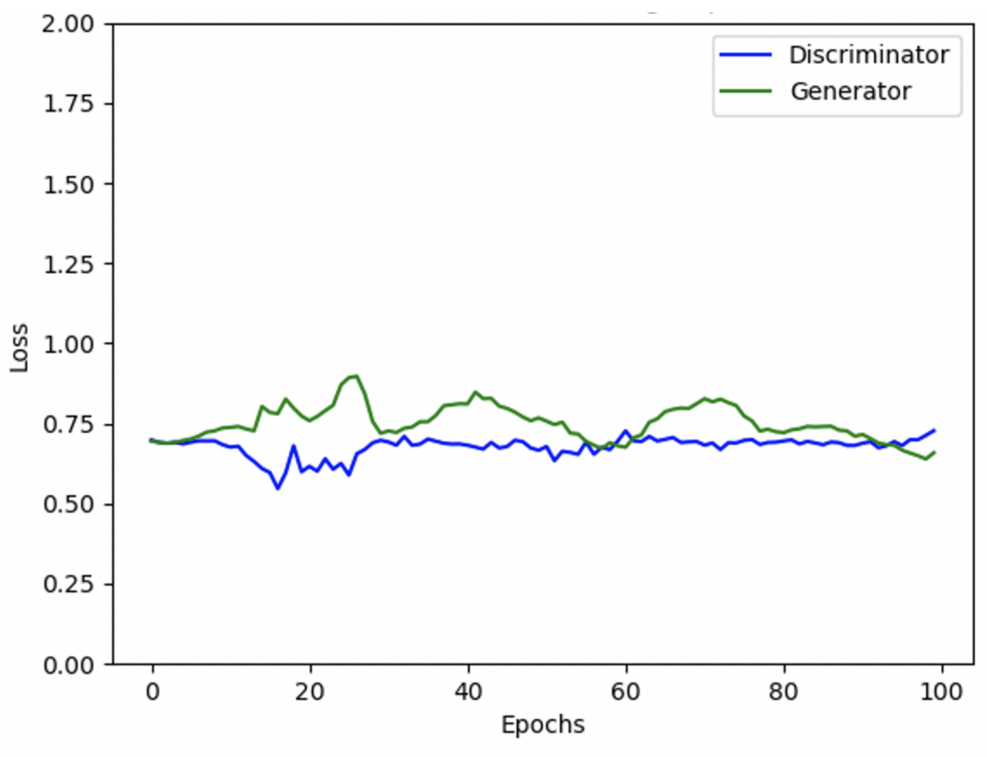


Figure 9 – MLP / LSTM model : GAN loss evolution through epochs

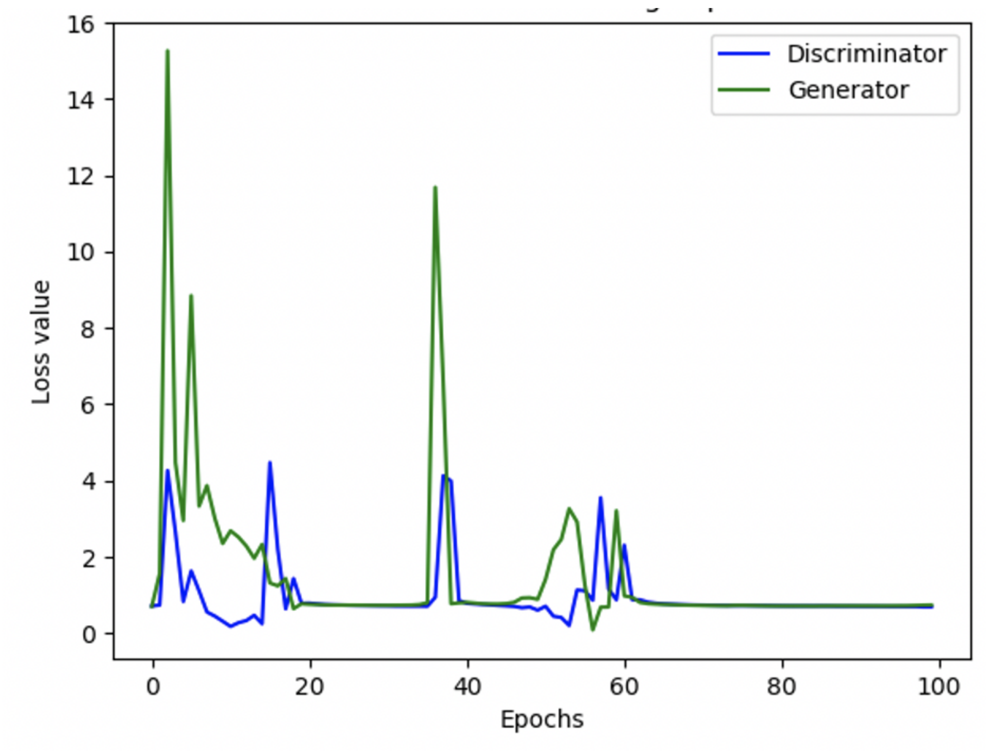


FIGURE 10 – Bidirectional LSTM / CNN model : GAN loss evolution through epochs

These results show us that the Bidirectional LSTM is more victim of outliers and architecture changes during learning, we can see it in figure 10 by the presence of spikes at the start of the learning. However, we can notice that the two GANS converge correctly and quite quickly around 0.75, which proves their functionality. The fact that the presence of pikes disappears after a few epochs shows that we use the collapse mode (on both models) to converge to a model with the expected results quickly. We can see better stability over time and epochs for the combination Bi-directional LSTM / CNN, which is undoubtedly due to the greater power of CNN as it does not have to multiply the layers depending on the number of parameters and avoids being overloaded.

Moreover, our GAN presents a design problem resulting from the loss ; it converges to a set of solutions due to the "collapse mode". "Collapse mode" describes the blocking of the discriminator in a local minimum. Then with each new generation, the generator generates a better result which blocks the discriminator from unblocking. This concern does not prevent us from evaluating the architecture of the generated song and the quality of the output. Still, it limits the capacity of the generator to produce different results. One possible solution is the loss of Wasserstein [23], which will focus on discriminating harder against repeats to avoid blocking.

4.3 Evaluation of the architecture of the generated songs

In order to prove the relevance of Bi-directional LSTM and to be able to quantitatively compare the structure of the generated music sequences, we made an analysis of the notes, chords and their connection (see Figure 11 and 12). The generated MIDI files also make it possible to have an opinion on the structure of the data in the sequences thanks to an overview given by the piano roll (see Figure 13 and 14).

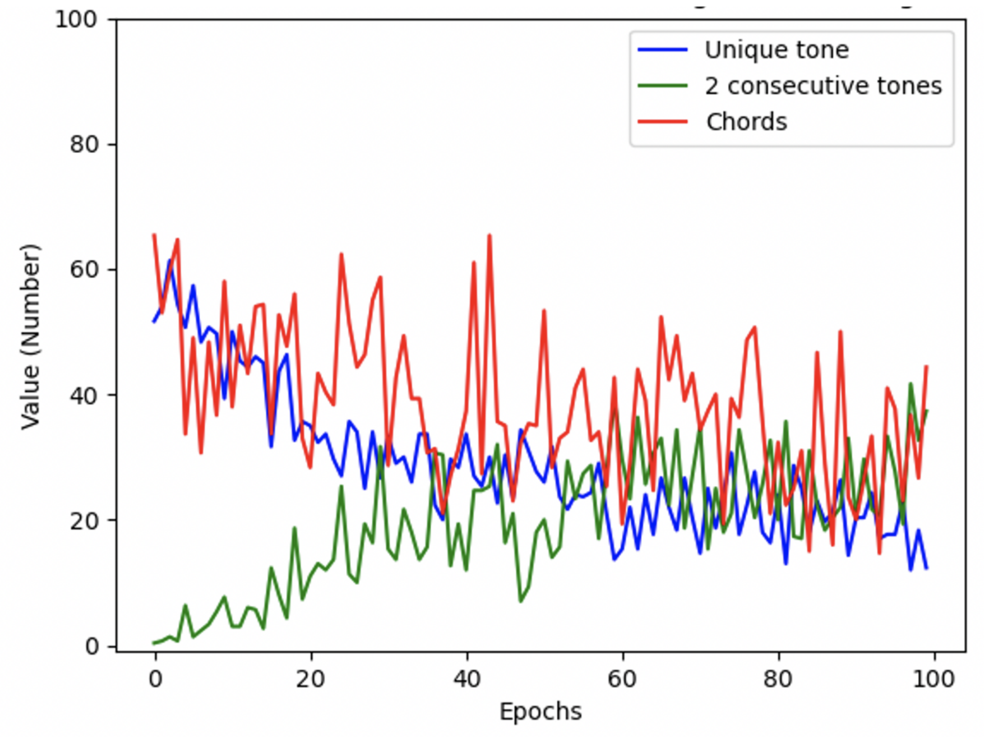


FIGURE 11 – MLP / LSTM model : Evaluation of the generated song architecture

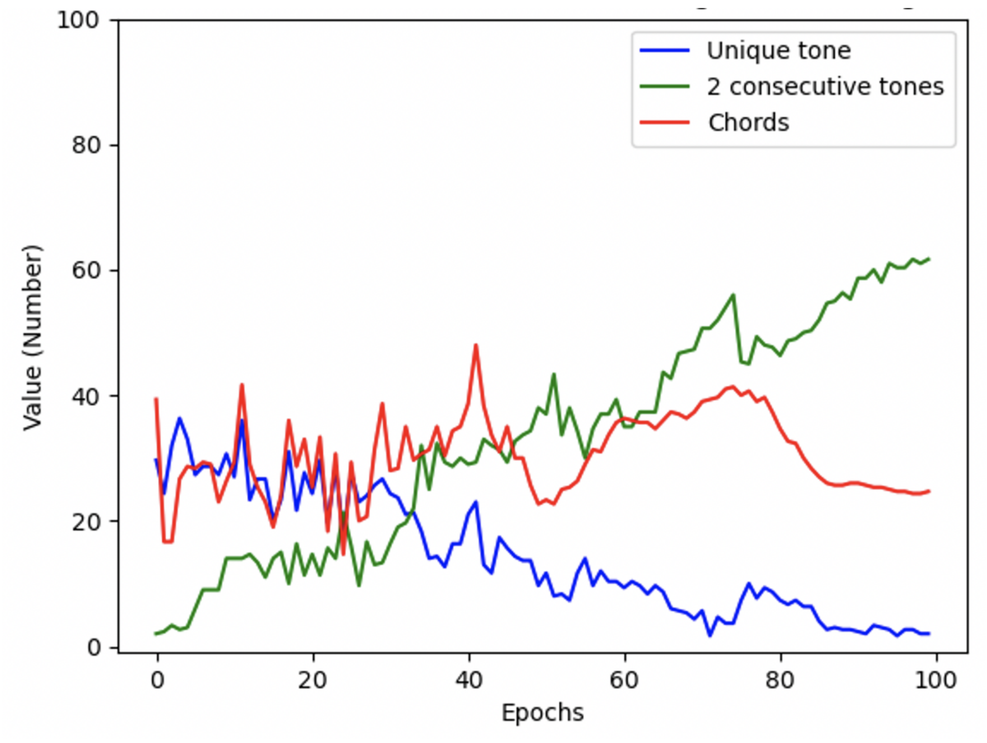


FIGURE 12 – Bidirectional LSTM / CNN model : Evaluation of the generated song architecture

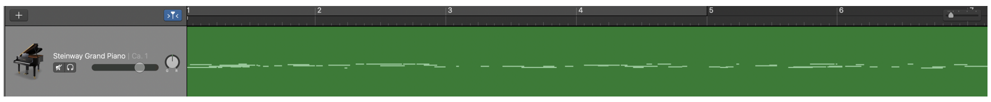


FIGURE 13 – MLP / LSTM model : Piano roll of a generated sequence

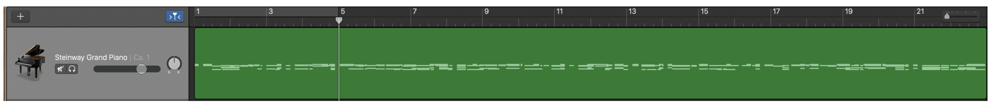


FIGURE 14 – Bidirectional LSTM / CNN model : Piano roll of a generated sequence

Thanks to this data we can affirm that the two models nevertheless differ in the ability to learn music ; while the MLP will try to converge towards an average for all the parameters returned by the discriminator, the Bi LSTM will use the past and future correlations to arrive at extracting a more complex architecture from the sound. Thus our model better manages polyphony, which consists of playing several notes at the same time. Polyphony is different in our learning than chords, which are treated like notes. Thus this polyphonic character clearly shows an advance in musical architecture.

5 Further researches conclusion

5.1 Further researches

We aimed to take into account as many parameters as possible and to prove that the combination of a Bidirectional LSTM and a CNN is a viable solution. Our next step in further researches is to add a multiple instrument management system. It was not our focus, but human survey results are altered when mixing each instrument of a song and result where much better when we used instruments separately. We wanted first to combine sequences of the three mainly used instrument (guitar, bass and battery) in a single track for having a multi-instrument result, but we concluded that it is not the right way to doing it. In further researches, we want to do analyses on models for understanding the link between instrument in a piece of music.

5.2 Conclusion

This project allowed us to experiment with generative algorithms, and to deepen our knowledge on the vast topic of music generation. In this work we presented a new generative architecture, and a new approach to limit loss and optimize the learning of many musical parameters. Bidirectional LSTM produces better results than LSTM in terms of prediction accuracy. But above all, it improves the consistency between the different elements of the melody by taking into account past and future information. CNN solves the overloading problem encountered with too many musical parameters by the multi-layer perceptron (MLP).

6 Figures

Figures

Références

[1] Lewis, *Creation by Refinement : A creativity paradigm for gradient descent learning networks.* International Conference on Neural Networks., 1988.

[2] Todd, *A sequential network design for musical applications" in Proceedings of the Connectionist Models Summer School.*, 1988.

[3] M. W. Diederik P Kingma, "Auto-encoding variational bayes," *arXiv :1312.6114v10*, 2014.

[4] D. Eck and J. Schmidhuber, "A first look at music composition using lstm recurrent neural networks," *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, vol. 103, p. 48, 2002.

[5] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "Musegan : Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," *arXiv preprint arXiv :1709.06298*, 2017.

[6] J.-P. Briot, G. Hadjeres, and F.-D. Pachet, "Deep learning techniques for music generation–a survey," *arXiv preprint arXiv :1709.01620*, 2017.

[7] S.-g. Lee, U. Hwang, S. Min, and S. Yoon, "Polyphonic music generation with sequence generative adversarial networks," *arXiv preprint arXiv :1710.11418*, 2017.

[8] G. W. George Papadopoulos, "Ai methods for algorithmic composition : A survey, a critical view and future prospects," *AISB symposium on musical creativity*, 1999.

[9] ——, "A genetic algorithm for the generation of jazz melodies," *Proceedings of STEP*, 1999.

[10] A. Graves, "Generating sequences with recurrent neural networks," *arXiv :1308.0850*, 2013.

[11] B. C. Rebecca Fiebrink, *CHAPTER 12 : The machine learning algorithm as creative musical tool.* The Oxford Handbook of Algorithmic Music, 1988.

[12] H. K. J. K. L. J. K. Taeksoo Kim, Moonsu Cha, "Learning to discover cross-domain relations with generative adversarial networks," *arXiv :1703.05192*, 2017.

[13] F. H. J. C. A. C. A. A. A. A. A. T. J. T. Z. W. W. S. Christian Ledig, Lucas Theis, "Photo-realistic single image super-resolution using a generative adversarial network," *arXiv :1609.04802*, 2017.

[14] Y. et al., *MidiNet : A convolutional generative adversarial network for symbolic-domain music generation.* International Society for Music Information Retrieval Conference (ISMIR), 2017.

[15] D. et al., *Synthesizing audio with Generative Adversarial Networks.* ICLR Workshops, 2018.

[16] J.-P. Briot, G. Hadjeres, and F.-D. Pachet, *Deep learning techniques for music generation–a survey*, 2017.

[17] J. S. Augustus Odena, Christopher Olah, *Conditional Image Synthesis With Auxiliary Classifier GANs*, 2016.

[18] S. O. Mehdi Mirza, *Conditional Generative Adversarial Nets.* Departement d'informatique et de recherche operationnelle - Universite de Montreal, 2014.

[19] S. C. Alec Radford, Luke Metz, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.* Under review as a conference paper at ICLR 2016, 2016.

[20] P. I. A. A. E. Jun-Yan Zhu, Taesung Park, *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, 2017.

[21] I. Vasilev, D. Slater, G. Spacagna, P. Roelants, and V. Zocca, *Python Deep Learning : Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow.* Packt Publishing Ltd, 2019.

[22] T. C. A. T. K. C. Y. B. R Devon Hjelm, Athul Paul Jacob, “Boundary-seeking generative adversarial networks,” *arXiv :1702.08431*, 2017.

[23] L. B. Martin Arjovsky, Soumith Chintala, “Wasserstein gan,” *arXiv :1701.07875*, 2017.