Universidade da Beira Interior
Departamento de Informática

Nº317 - 2022: Music Generated based on Emotions

Developed by:

Gabriel Alexandre Araújo Ribeiro

Supervisor:

Professor Doutor João Carlos Raposo Neves

July 11, 2022

Acknowledgements

This project and its development would not have been possible without the help and contribution of many people. People which I will express my deepest regards.

To my Project supervisor Professor Doutor João Carlos Raposo Neves. His help, guidance, support and mentoring has served me well and its lessons will remain throughout my academic life and career. My sincerest gratitude, Professor.

To all the professors that shaped me throughout this past years that led to this final stage.

To my friends, and particularly my roommate. They were there in the good times and during the tough times.

And at last, to my parents that I love deeply and more than anything in the world. Every sacrifice they made, every lesson, every expression of love and support. None were taken for granted. Thanks to them I am in the wake of an endless dream that they fought their whole life for me to achieve, and these words don't even begin to describe how much I admire them and cherish them. To my parents, I love you.

Contents

List of Figures

List of Tables

Acrónimos

AI	*Artificial Intelligence*
CS	*Cell state*
CNN	*Convolutional Neural Network*
DS	*Data set*
DL	*Deep Learning*
HS	*Hidden state*
KS	*Key Signature*
LSTM	*Long short-term memory*
ML	*Machine Learning*
MG	*Music Generation*
RNN	*Recurrent Neural Networks*
SPN	*Scientific Pitch Notation*
UBI	*Universidade da Beira Interior*

Chapter

1

Introduction

1.1 Framing

It's not a novel concept to create music using an algorithm. It was in 1959 that the first algorithmic composition model was computed. Neural networks were also utilized to create music later on, starting in 1989. But it was only in recent years that the use of neural networks to create music has become increasingly relevant, mostly because deep neural networks were demonstrated to be capable of learning from large-scale datasets . Since then, a lot of *Deep Learning* (DL) network models have been proposed for *Music Generation* (MG). Therefore, this Project anticipates using implementing one of these MG methods as well as adapting it in order to generate music that can elicit an emotional response, according to the user's input.

This Project was developed within the scope of my Bachelor's Degree in Computer Science and Engineering in *Universidade da Beira Interior* (UBI).

1.2 Motivation

Music is a way for human beings to express themselves. It is a part of what we are, and how we think. Understanding how the mind of a great composer works when composing a beautiful piece, is understanding how part of the human mind works. So being able to make *Artificial Intelligence* (AI) generate music that never existed before, is one of the many steps towards a world where AI can think like a human being.

This Project combines my interest in the world of AI, DL and *Machine Learning* (ML), and my love for music and everything that encompasses it.

1.3 Objectives

These are the objectives that were outlined in order to develop this MG based on a given emotion Project:

- Study of state-of-the-art methods on MG
- Implementation of one (or more) MG methods
- Adaptation of the implemented method(s) to allow the inclusion of a parameter that conditions the emotional response that the generated music provokes
- Tests and Debugging

1.4 Document Organization

In order to reflect on the work done this document is structured in the following manner:

1. First Chapter – **Introduction** – Presents the Project, motivation for its choice, framework, objectives and organization.
2. Second Chapter – **State-of-the-Art** – Presents the main methods for MG developed in the past years, and describes important concepts of Music Theory and Deep Neural Networks, in the context of this Project.
3. Third Chapter – **Tools and Technologies used** – Describes in great detail all the tools and technologies used in the context of this Project.
4. Fourth Chapter – **Developed Work** – Explains the implementation of the MG method, shows and explains results.
5. Fifth Chapter – **Conclusions and Further Work** – Final thoughts regarding the Project as a whole, as well as discussion of possible future work.

Chapter

2

State of the Art

2.1 Introduction

A big part of this Project is researching current methods of MG and concepts necessary to apply them. In this chapter, these concepts and methods are layed out.

2.2 Music Theory

2.2.1 Overview

Music Theory is the study of the practices, possibilities, and elements of music. This section contains a brief introduction to some important concepts of Music Theory for this Project.

2.2.2 Emotion in Music

Emotion in music is a very complex topic on its own. Although it is common knowledge that *Key Signature* (KS) convey, in general, different emotions to the listener [2], the result can, and possibly will vary depending on the chords, the notes disposition, length, etc. Other factors completely unrelated to Music Theory can also affect the way a given song or musical content is perceived, such as the mood of the listener.

Determining a specific emotion is in of itself a task that seems straightforward, but falls short of being easy when it is consider the ramifications of a given emotion.

Because of limited time and resources, this Project's approach will be solely based on the Music Theory aspect regarding emotion and feelings in Music.

2.2.3 Melody

A melody is essentially a combination of notes and rests (example of a melody in Figure 2.1). The melody is the part of the song, piece or musical content that is best identifiable as well as easily recognized by an untrained ear. Some might say it is the very soul of music. This Project will focus heavily on handling music as just a melody, as a personal choice.

Although it is possible to generate polyphonic music using DL networks[3] (and there already exists developed methods to do so [4]) , it would demand a much better understanding of music theory as a whole and years of experience in this field of study in order to execute such an ambitious project.

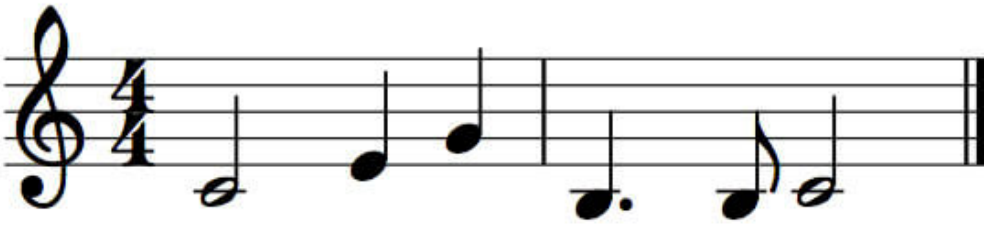

Figure 2.1: A simple melody with notes,KS and Time Signature

2.2.4 Pitch / Scientific Pitch Notation

A note's pitch indicates how high/low a note is. In standard notation, the notes are named sequentially from A to G, unlike the *solfège* scale which is taught in Musical Education to 5th graders in Portugal (shown below in the figure 2.2. After the note **G**, it resets to an **A**. An **A** that is an octave above the previous **A** has two times the frequency.

To distinguish between all the existing notes and octaves, we use *Scientific Pitch Notation* (SPN), which combines the note's name(or letter), with the respective octave. E.g. C1,A3,F2

The number of the octave is determined by the standard piano keys. The note farthest to the left of the Piano is an **A**, in the octave numbered as 0.

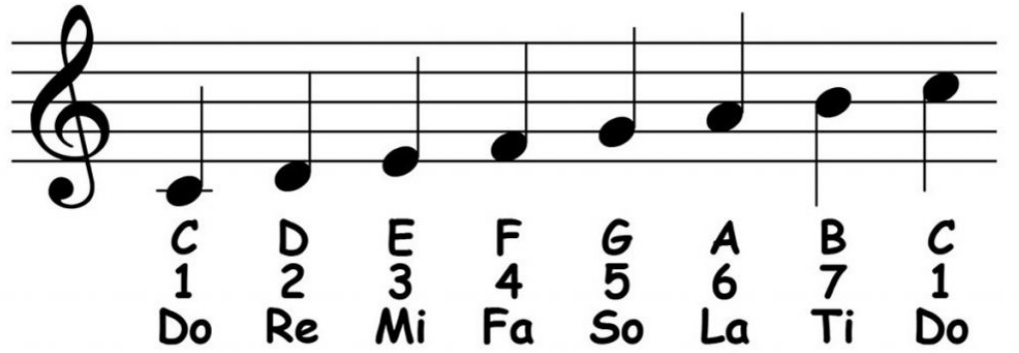

Figure 2.2: A C Major Scale with standard notation, regular numbering, and *solfège* notation, respectively

2.2.5 MIDI and Kern

MIDI (Musical Instrument Digital Interface) is a protocol developed in the 1980's which allows electronic instruments and other digital musical tools(such as computers, samplers, and synthesizers) to communicate with each other and sound-controlling equipment„ so that they may communicate with one another using MIDI messages. This enables easy note editing, flexible orchestration, and song composition. It makes it possible to record music in this manner. MIDI messages are also used by virtual instruments, (e.g. computer programs) that simulate hardware synthesizers and samplers, to connect with computer sequencing software operating on the same machine.

Midi itself does not make a sound. The messages are interpreted by a MIDI instrument to produce sound. A MIDI instrument can be a piece of hardware (electronic keyboard, synthesizer) or part of a software environment.

Kern is a data format for music representation, which will be the data format of some of the *Data set* (DS) used in this Project.

2.2.6 Key Signature

The Key is a group of pitches that form the centre of the piece and it is formed by two elements, the tonic and the mode. E.g. C Minor

The tonic note is often found at the beginning and end of a piece/phrase, and it is the "centre" of the entire piece, or in other words, it is the centre of the harmonical part of the piece.

Depending on the mode(E.g Major/Minor), the intervals of the notes in a scale vary. For example, in the C Minor Scale we find 3 flat notes, whereas in

the C Major Scale we find none. Both start and finish with a C note, and have similiar tones.

The concept of Key is extremely important to this Project, not only because of its relevance in the emotional aspect of the music generated, but also because we want to make sure the Neural Network has been able to learn the concept of keys, in order to use it to form melodies/pieces that make "sense", so it doesn't just generate a lot of notes with no structure whatsoever. This gives stability in the music, and as mentioned before it can convey a particular emotion to the listener.

2.2.7 Transposition

If a collection of notes is moved a tone up or down, the key changes, but the musical content remains the same. However, there are many variables at play here. If a given listener listens to both pieces(the original and the transposed), its opinion might be biased towards feeling the same emotion despite its key. If the same listener only listens to one version of the piece, the emotion that the piece conveys to that particular listener has a much higher chance of being the emotion that Music Theory says it will convey.

Overall, this process is very ambiguous and there is a lot of Psychology at play, so there will not be an iron-clad law that we ought to follow when it comes to generation music that conveys a given emotion.

2.3 RNN's and LSTM's

2.3.1 Overview

If an individual is listening to a particular music and starts singing it, it is because the individual remembers the music that is currently being played. This task would be impossible if you could only hear one note of the song. However, if you take the time to listen, you will be able to pick up and sing along much more easily.

Similarly, a traditional neural network is incapable of remembering information that is presented over time. This is true not only for music, but for any sequential information in which the next piece of information is dependent on what came before.

Recurrent Neural Networks (RNN)'s can be applied to information that comes in a sequence to overcome this limitation of traditional neural networks.

Music has long-term structural patterns, which means there are several patterns that repeat iteself in a musical phrase or in a melody overall. The idea

of a *Long short-term memory* (LSTM) is to have a memory cell that enables learning longer term patterns.

2.3.2 Time Series

There are some type of data that can change over time(time-dependent data). Since we're dealing with notes/melodies, it's useful to find patterns that repeat over time. A Time Series is a set of observations that are collected based on regular intervals of time.

2.3.3 RNN's architecture

As shown in Figure 2.3(where **t** is time, **Xt** is the input, **ht** is the output and **A** is the hidden layer), an RNN is a Neural Network with loops in it[5], which will then allow information to persist. A loop enables data to be transferred from one step of the network to the next, which means that the information that "leaves" the hidden layer at a given step can be reused at a later point(the next step). The information is given step-by-step(sequentially) to the hidden layer, and the output will be both the **output** and an **hidden state**, which is the part of the output that will be used at a later point. The hidden layer in an RNN is the same layer but it is used in a recursive manner. In Figure 2.4 we see the same RNN but in an unrolled state, so it is easier to understand the process previously explained.

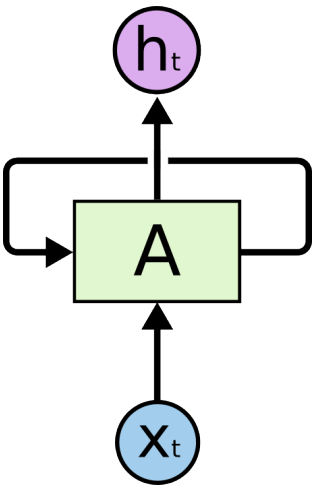

Figure 2.3: Diagram of a simple rolled RNN with a single layer[1]

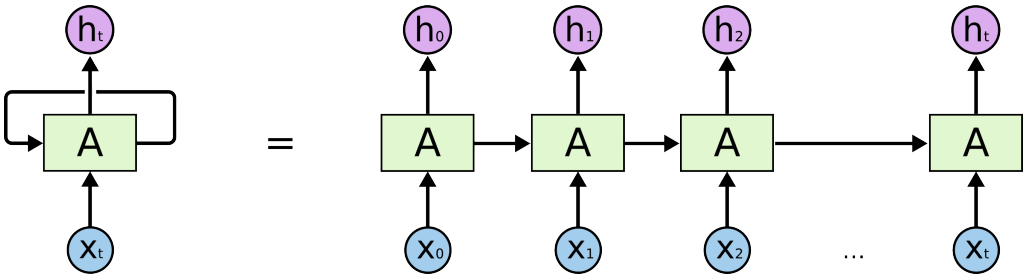

Figure 2.4: Diagram of a simple rolled RNN with a single layer [1]

2.3.4 Activation Function

An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network [6]. An activation function is usually nonlinear but it can vary depending on the purpose of it. An activation function can be used for the hidden layer, input layer or output layer of the neural network.

In RNN's, the activation function for hidden layers is usually the **Tanh Activation**.

2.3.5 Problem with a simple RNN

A simple RNN has its advantages and in theory, should be capable of handling "long-term dependencies". However, in practice, it has been shown the opposite for a number of reasons [7]. A simple RNN can't bridge more than 5-10 time steps (Vanishing Gradient Problem)[8]. In sum, it struggles with long-term memory. The network can't use information from a "distant past". This leads to the major issue in using a simple RNN for this Project. A simple RNN is unable to learn patterns in long dependencies, and since a time-series based problem(such as MG) involves long-term dependencies, a different type of RNN has to be used.

2.3.6 Simple RNN vs LSTM

The LSTM is a special type of RNN first introduced in 1997 [9]. They are specifically designed to avoid the problem mentioned in sub section 2.3.5. The basic architecture of an LSTM is similar to the simple RNN, as show in Figure 2.5 and Figure 2.6. Figure 2.7 is the notation displayed in the diagrams.

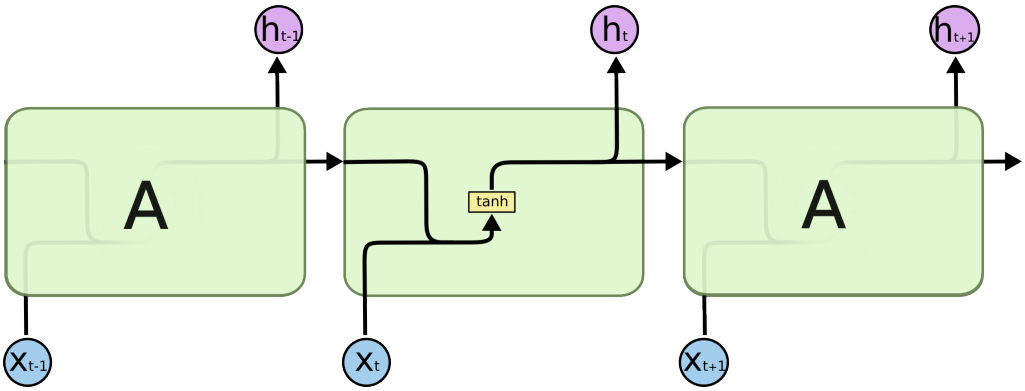

Figure 2.5: Diagram of a simple unrolled RNN with a single layer[1]

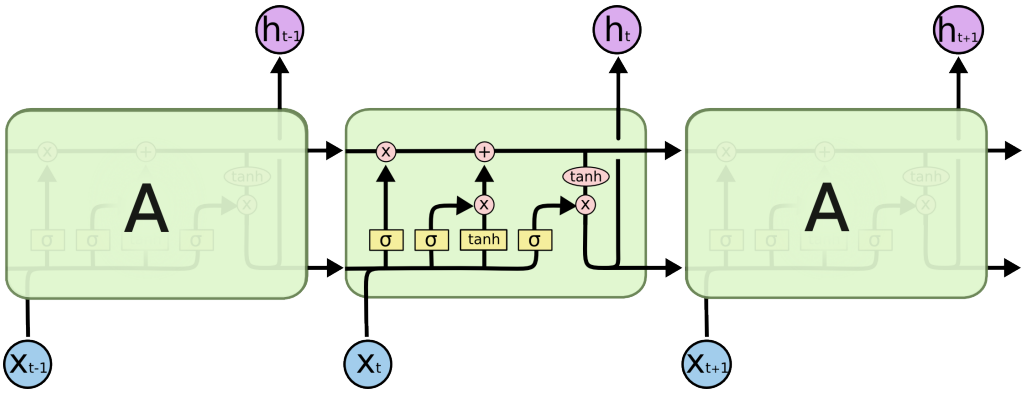

Figure 2.6: Diagram of an unrolled LSTM[1]

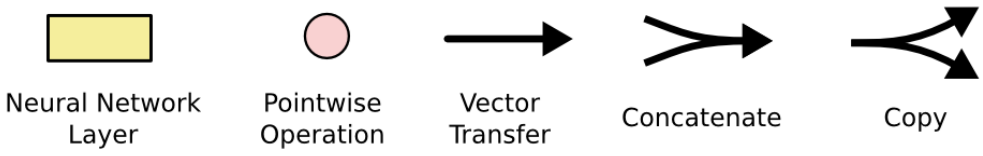

Figure 2.7: Notation used in diagrams of Figure 2.5 and Figure 2.6[1]

Each line in the aforementioned diagrams carries an entire vector from one node's output to another node's input. The yellow boxes are learned neural network layers, whereas the pink circles are pointwise operations like for example, vector addition. Concatenation is indicated by lines merging, whereas lines forking indicate that their content has been copied and is being sent to other locations.

2.3.7 Cell State

While the *Hidden state* (HS) (represented in Figure 2.8 as H_t) is in a way, the short-term memory of the LSTM, the key to LSTM's(and the reason it is capable of long-term memory) is the ***Cell state*** **(CS)**, represented as C_t and also referred to as second-state vector. Since the CS runs through the entire chain of the LSTM, with only some minor linear interactions, it is very easy for information to proceed unchanged throughout the whole network. The LSTM can add or remove information from the CS with the help of the structures called **gates**, which are explained below in subsection 2.3.8.

Since the CS is only changed in two instances of the LSTM cell, it has very few computations behind it, so it is able to stabilize the gradients(solving the problem mentioned in sub section 2.3.5). In the pointwise multiplication (2.7) occurring in the CS, it basically "decides" what to forget from this long-term memory. In the pointwise addition, it "decides" what new information to remember.

When training an LSTM, the network becomes very proficient in learning what patterns to learn and what patterns to forget, and these operations occur in the CS

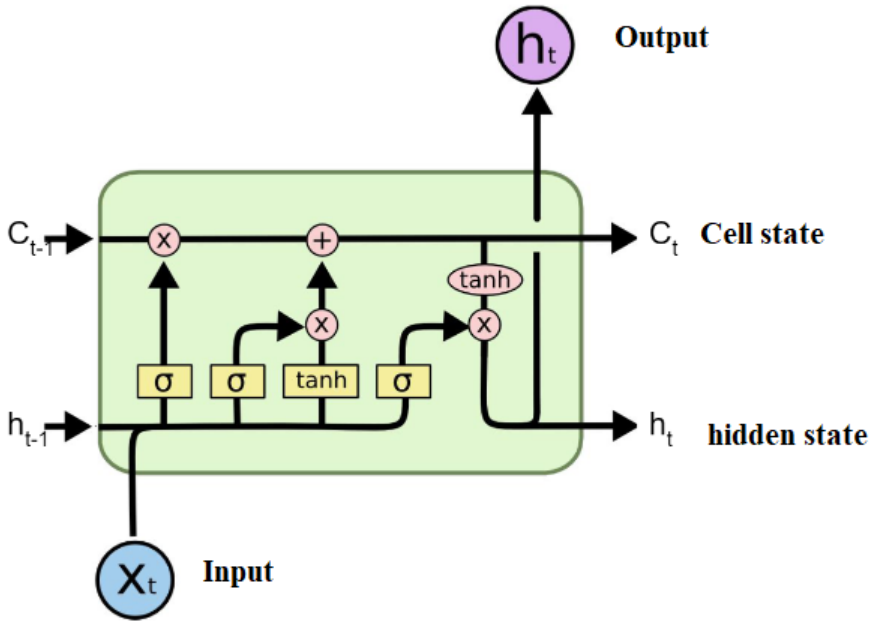

Figure 2.8: LSTM cell[1]

2.3.8 Gates and information processing

Gates(2.9 are a way to let information through in a more controlled manner. They consist of a pointwise multiplication operation and a sigmoid neural network layer. Indicating how much information should be allowed through, the sigmoid layer outputs numbers between zero and one. When that value is zero, it means to let no information through, and when that value is one , it means to let everything through [8]. These three gates serve to control the cell state in an LSTM. In short, these gates act as "filters" for the information that flows in the LSTM. They decide what information to input, output and forget.

The decision of what information gets forgotten from the CS is made by a sigmoid layer called **forget gate layer**. The information that reaches the pointwise multiplication operation in CS has the shape of a matrix (f_t) and it is calculated by the following equation:

$$f_t = \sigma(W_t \cdot [h_{t-1}, x_t] + b_f) \tag{2.1}$$

Equation 2.1 is for f_t where σ is a sigmoid function, W_t is a weight matrix, and b_f is a bias matrix. The hidden state from the previous time step h_{t-1} is concatenated with the input x_t, and then the sigmoid function σ is applied to everything. After the equation is applied, we get a matrix which is the "filter" for what the LSTM is supposed to forget.

All the values in f_t will be between 0 and 1 (because of the sigmoid function), so the indexes that are closer to 0 are the indexes that will be forgotten. This "forgetting" process is done through the Hadamard product(elementwise multiplication) after the sigmoid layer, as shown on the following equation:

$$C_t^f = C_{t-1} \circ f_t \tag{2.2}$$

The next step is to choose the new information that will be stored in the CS. Two sections make up this(Figure 2.8). The **input gate layer** which is a sigmoid layer, first determines which values will be updated. Afterwards, a tanh layer creates a vector of potential new values, $\tilde{C}_t$, that could be added to the state. The subsequente phase combines these two to create an update to the state. The information that "leaves" the input gate layer has the shape of a matrix i_t and it is calculated by the following equation:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.3}$$

Equation 2.3 will act as a "filter" for the simple RNN component present in the LSTM (Figure 2.5)

The information that "leaves" the layer with the tanh function is calculated with the following equation:

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.4}$$

After i_t and $\tilde{C}_t$ are calculated, the Hadamard product is calculated, and the information that is relevant will be "filtered":

$$C_t^i = \tilde{C}_t \circ i_t \tag{2.5}$$

The final thing regarding this stage of the LSTM is to calculate C_t (the CS at the current time step):

$$C_t = C_t^f + C_t^i \tag{2.6}$$

The last part to cover regarding the LSTM is the output. The **output gate layer** is also a sigmoid layer. First, the sigmoid layer decides what parts of the cell state it is going to output. Then, the cell state is put through Tanh (to push the values to be between 1 and 1) and is multiplied by the output of the sigmoid gate, so that we only output the parts we decided to. The information that "leaves" the output gate has the shape of a matrix o_t and it is calculated by the following equation:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.7}$$

Finally, we get h_t(Figure 2.8) by calculating the Hadamard product of the information that left the output gate layer, and the cell state at the current time step after being passed through a Tanh function(different than the Tanh layer) to it, as shown in the following equation:

$$h_t = o_t \circ tanh(C_t) \tag{2.8}$$

The result will then be copied(Figure 2.7) and it will be both the **output** and the **hidden state** that will be used at a later point(next time step), as it happens in a simple RNN aswell (2.3.3)

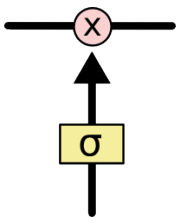

Figure 2.9: Gate in an LSTM[1]

2.4 Existing methods of Music Generation based on Emotion

2.4.1 Overview

While there exists a substancial amount of MG papers and approaches, some even using different type of neural networks [10], there are only two major methods proposed for MG based on emotion (to the best of my knowledge), and both will be summarized below.

2.4.2 SentiMozart: Music Generation based on Emotions

The two models that make up the suggested framework in this paper [11] are the Image Classification model and the MG model. The first model classifies the person's image into one of the seven sentiment classes: angry, disgust, fear, happy, sad, surprise, and neutral—and the second model creates music that fits the sentiment class that was determined.

A *Convolutional Neural Network* (CNN) is implemented for the classification of images into its respective class. For the Music Generation model, a doubly-stacked LSTM is used. The input given to the LSTM is a One-hot enconded representation(cross reference implementation here) of the MIDI (2.2.5) files in a particular dataset belonging to the specific sentiment.

In sum, in the proposed approach, an image would be taken as an input to the Image Classification Model, which would in turn output a certain emotion. Afterwards, the MG model would generate a song of that sentiment, but not before training the LSTM with the specified dataset.

In the context of this Project(putting aside the use of an Image Classification Model) this approach would mean that the LSTM would have to be trained "on the spot" after the user has chosen the designated emotion, and time-wise it is not viable. However, the concept of having differently trained LSTM models for different emotion will be taken in to account.

2.4.3 Learning to generate music with sentiment

In this paper [12], the authors propose to train the LSTM with a DS comprised of about 800 pieces, in which 95 are annotated according to a 2-dimensional model that represents emotion using a valence-arousal pair. Valence indicates positive versus negative emotion, and arousal indicates emotional intensity and the valence-arousal model is one of the most common dimensional models used to label emotion in music.

This approach used a combination of mLSTM and logistic regression proposed by Radford et al. [13] to compose music with sentiment. To do this, they treated the music composition problem as a language modeling problem.

This showed to be the most accurate in terms of generating music based on emotion, since it uses the valence-arousal model to determine emotion. However, this approach was dismissed in the context of this Project due to the proficiency in DL techniques and Music Theory required to elaborate it.

2.5 Conclusions

In this chapter, it was given a breakdown of the most important concepts of Music Theory, RNN's and LSTM's in regards to this Project. The logic and mathematics behind an LSTM (which will be put to use in the practical part of this Project) were succinctly explained and demonstrated. Furthermore, there was given a brief overview on the most relevant and current methods towards MG based on emotion, and a brief explanation on why certain aspects of the aforementioned approaches would not be replicated in this Project.

Chapter

3

Used Technologies and Tools

3.1 Introduction

This chapter describes the various technologies and tools used in the development of this project, presenting a brief summary of its characteristics.

3.2 Python

Python [14] is a powerful, interactive, object-oriented, and interpreted scripting language. Python has been designed to be very readable. It was developed by Guido van Rossum at the Netherlands' National Research Institute for Mathematics and Computer Science in the late 1980s and early 1990s. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

3.3 Music21

Music21 [15] is a Python-based toolkit for computer-aided musicology. It is used to examine and study vast databases of music, generate musical examples, teach the foundations of music theory, alter musical notation, research and compose music(both algorithmically and directly) and among other things.

3.4 Tensorflow (GPU)

TensorFlow [16] is an open-source library developed by Google primarily for deep learning applications. Traditional machine learning is also supported. TensorFlow was first created without having deep learning in mind in order to handle huge numerical computations. However, it turned out to be quite helpful for the development of deep learning as well, so Google made it open-source.

Tensorflow takes data in the form of Tensors, which are multi-dimensional arrays of higher dimensions. When handling a lot of data, multi-dimensional arrays come in quite helpful.TensorFlow works on the basis of data flow graphs that have nodes and edges.

Since the execution mechanism is in the form of graphs, it is considerably easier to execute TensorFlow code in a distributed manner across a cluster of computers while using GPUs.

Deep learning applications are quite complex, and the training process requires a lot of computation. Due to the amount of the data, it requires a lot of time and involves numerous iterative processes, mathematical calculations, matrix multiplications, and other operations.

It would normally take substantially longer to do these tasks on a conventional Central Processing Unit (CPU). In the context of gaming, where you need a high-resolution screen and image, graphic processing units (GPUs) are common. However, they are also employed in the creation of deep learning applications. Support for GPUs as well as CPUs is one of TensorFlow's major advantages.

3.5 Keras

Keras [17] is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation, and it is used to make the implementation of neural networks easy. Because it offers a python frontend with a high level of abstraction while having the choice of multiple back-ends for computation purposes, Keras is relatively easy to learn and use. As a result, Keras is considerably more beginner-friendly but slower than other deep learning frameworks.

Keras has been adopted by TensorFlow as its official high-level API. Because Keras has built-in modules for all neural network computations, it can be used to perform deep learning quickly.

3.6 Other relevant tools

- **MuseScore**: Software that generates sheet music from a Midi file.
- **Pickle**: Pickle is a Python module that implements binary protocols for serializing and de-serializing a Python object structure.
- **Numpy**: NumPy is an open source project aiming to enable numerical computing with Python. [18]
- **Github**: GitHub is a code hosting platform for collaboration and version control. GitHub lets you (and others) work together on projects. Some code implemented in this Project derived from a few repositories available [19] [20] [21]

3.7 Conclusions

In this chapter all the relevant tools and technologies that allowed the development of this Project were succinctly described.

Chapter

4

Developed Work

4.1 Introduction

In this chapter, the implementation of the techniques chosen and utilized for Music Generation [6], are succinctly presented and explained. The implementation involves preparing the dataset, Pre-processing the dataset, generating the training sequences from said pre-processed dataset, training two LSTM models 2.4.3 and last but not least, generating samples.

4.2 Dataset

There were two different dataset used in the implementation of this project. The first one was comprised of folksongs from Germany, England, France, Italy, Sweden and The Netherlands [22] and it was used to train the model that would generate "happy music". The second one, was comprised of blues melodies [23], and videogame music 3.6 and it was used to train the model that would generate "sad music". The total statistics of the dataset are shown in Table 4.1

Dataset	**Major songs**	**Minor songs**	**symbols**
Folksongs	749	87	43
Blues/Videogame music	31	91	99

Table 4.1: Dataset statistics

There is a significant difference in major and minor songs, but since most of the data given to the first dataset is major songs, and minor songs to the second dataset, the model learns as intended. Also, there are more elements to

Music Theory involved in, for example, the blues music that inherently makes the song seem "sadder" and the model captures those patterns and parameters.

4.3 Pre-Processing

Data preprocessing is a crucial stage in Machine Learning because the quality of data and the relevant information that can be obtained from it has a direct impact on our model's capacity to learn. Consequently, preprocessing our data before feeding it into our model is critical. In the following sub sections the sequential steps in Pre-processing the musical data to a state that the computer can interpret it is explained.

4.3.1 Converting and Parsing files

Firstly, we go through all the files in the MIDIPATH, which will contain our Dataset. Since our Dataset consists of Midi/Kern files, we need to convert them to a format that can be used as information. Parsing consists of using Music21's library 3.3in order to convert the midi/kern file in to a stream, which then can be handled properly with this library.

4.3.2 Filtering files with unacceptable durations

In this section, the entire parsed file is checked to see if there are any notes that don't have a duration said to be among the most common note durations(quarter-note, half-note, etc). When encoding the symbols in later steps of the implementation, a note will be stored as a 16th note, and if it is held longer than a 16th note, it is stored the symbol "_" for as many 1/4 beats(16th notes) as the note is sustained, because this means that the note is carried on the next time-step(at least), since each time-step in the time series is set to 0.25. This is how the time series is standardized.

If there are irregular durations that fall outside the common note durations, it could jeopardize the training of the model and its ability to learn specific patterns.

4.3.3 Transposing parsed file

In the context of this Project, there are two main reasons as to why we need to transpose (2.2.7) each piece of music in the Dataset. In order to have some sort of musical consistency in the generated music, and since the approach

on emotion is based on Key Signature (2.2.6), it is important that the LSTM learns Key Signatures. We can analyze the Key Signature with Music21 3.3

All the music pieces are transposed to C Major (2.2.4) if they have a Major mode , and to C Minor if they have a Minor Mode. This is because that C Major is correlated with "happy emotions" and C Minor with "sad emotions" [2].

4.3.4 Encoding Symbols

After the parsed file is transposed,it is time to encode all the relevant information. The parsed file is composed of tempo, Key Signature, Notes, Rests, etc. The notes and rests are separated from everything else and afterwards we go through the entire file that contains just the notes and rests, evaluate each symbol, and save them in to a list.

If the symbol is a Note, we save the pitch value of the given note. If the symbol is a rest, we save the string "r". If the note is a Chord, we save the integers values of the normal order of the Chord which are the pitch class numbers of each note in the chord, separated with a +. As mentioned in 4.3.2, a note held for more than a 1/4 beat is encoded, as well as the symbol "_" for as many 1/4 beats as it is held. If the file has reached the end, we save the string "/" 64 times (Each of our sequences have a length of 64) which is our delimiter for each musical piece.

All the symbols are appended in to a list, which will then be "pickled" 3.6 to later use.

4.3.5 Mapping

An LSTM does not understand these type of symbols, so they will have to be converted to integers eventually. This process is made easier by mapping each different symbol to an incremental integer. In this process, a dictionary which contains all the different mappings is created and saved into a .json file so that it can be easily accessed later on.

4.4 Generating Sequences

The notes are then "unpickled" 3.6 in order to start generating training sequences out of the already pre-processed dataset. The sequence length is set to 64, since it is usually the length of an entire musical phrase. In Table 4.2 we see the number of sequences that will be generated in order to train both models.

In order to start generating sequences first the notes are converted to their integer counterpart by using the mapping .json file that it was previously created 4.3.5. Afterwards we start generating the inputs and the targets.

It is necessary to pass some "historical information" about the musical events happening in the dataset so the model can learn the existing patterns, so each input will be a list containing a slice of the pre-processed dataset(already in integer form), and the target will be the following note of that sequence. This process will continue incrementally until it reaches the number of sequences that can be created from the dataset.

All that is left to do is one-hot encode the input array. One hot encoding is one method of converting data to prepare it for an algorithm and get a better prediction. In Music Generation, it is usually best to one hot encode the input array in the training sequence [6]. The target array remains a integer array.

4.4.1 Python Generator

One-hot encoding the input array 4.4 raises a problem. The size of the data increases immensely. So, to work around this issue of hardware capacity, it was implemented a python generator which will later be a parameter in the model, instead of the inputs and targets. Essentially, what the generator does is the process of generating the training sequences mentioned above 4.4, but doing so while the model is getting trained. Instead of having the all the sequences ready by the time the model starts training, it generates "on the spot".

Dataset	Number of symbols	Traning Sequences
"Happy Model"	133965	133901
"Sad Model"	17966	17902

Table 4.2: Sequences statistics

4.5 LSTM model

As mentioned before, two LSTM models are trained(one with the Folksong dataset, and one with the Blues/videogame music dataset). The layers of the model are the same, but some parameters change. The model is built using Keras (3.5) Functional API.

4.5.1 Layers

The model has an Input Layer where the output shape will be an array **[None, None, output unit]** (output units is the number of symbols in the vocabulary

4.3.5, and setting the input layer like this enables the model to have as many time steps/cells 2.8 as we want), an LSTM layer, a Dropout layer(prevents overfitting [11]) and a dense output layer with a **softmax** activation function 2.3.4 (Equation 4.1).

4.5.2 Epochs and steps-per-epoch

Since we generate the training sequences whilst training the model 4.4.1, there is no need to specify a batch size. We calculate the steps-per-epoch(as a rule of thumb, it should be the number of samples divided by the batch size that it was intended to be used).

The training time for both models are in Table 4.3 and Table 4.4. The epochs on both models had to be larger because the model was failing to generate any music.

Training	Training time
Epochs = 50	19 minutes
Epochs = 70	33 minutes

Table 4.3: Training time for "Happy Model"

Training	Training time
Epochs = 30	55 minutes
Epochs = 50	82 minutes

Table 4.4: Training time for "Sad Model"

4.5.3 Other parameters

Since the target array4.4 is comprised of integers, the Loss functions in the LSTM model is sparse categorical crossentropy(as documented in the Tensorflow source code) [16]. The optimizer used is Adaptive Moment Estimation or **Adam**(a variant of Gradient Descent) and the Learning Rate is set to **0.001**.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{4.1}$$

4.5.4 Training statistics

By analysing the performance graphs of both final models during training (figure 4.1 and figure 4.2), is evident that the accuracy and loss follow, for the

most part, a steady elevation and decline respectively. When there's a sudden decrease in accuracy, there exists a sudden increase in loss, but overall the training runs smoothly with not many deviations from its regular behaviour.

The use of a test set was excluded from this project and by association, from this performance graphs. The reason behind this choice are succinctly explained in section 4.7

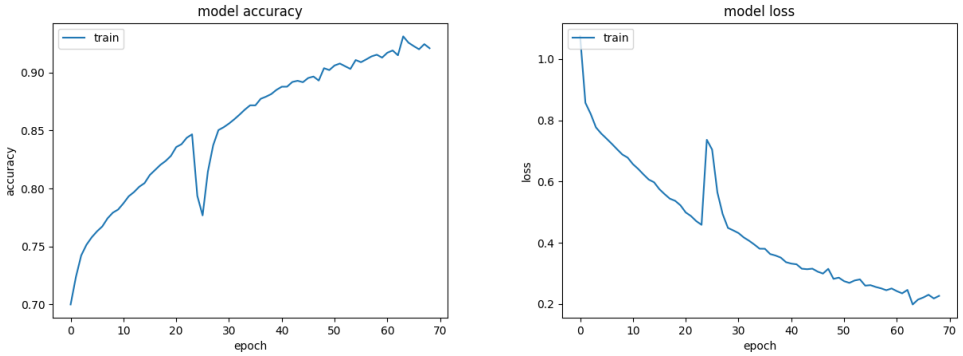

Figure 4.1: Performance graphs during training for the "happy model"

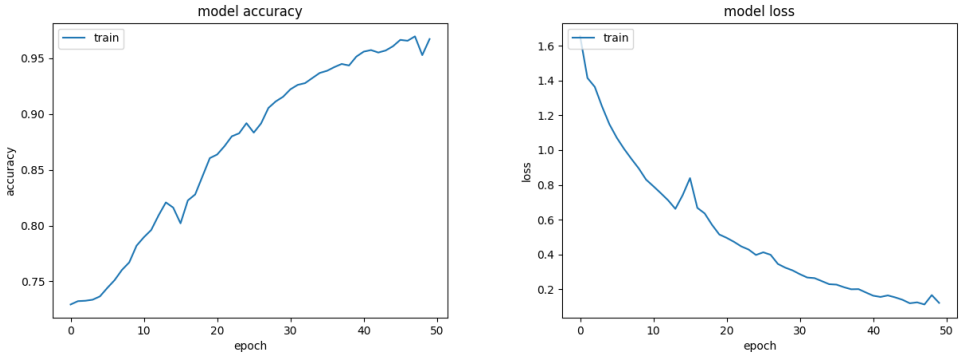

Figure 4.2: Performance graphs during training for the "sad model"

4.6 Generating a music sample

Lastly, it is time to generate a music sample. In order to generate a sample of music we need to specify the following parameters: a random **seed**(a small sample of the pre-processed DS), the number of **steps** in the time series music representation that we want the network to output, the max sequence length(to limit the seed to a given length) and the temperature.

The temperature gives a certain control to the user as to randomness of the events generated by the model [24] and makes the generation more flexible. As such:

- a temperature of 1.0 uses the exact distribution predicted;
- a value smaller than 1.0 reduces the randomness and thus increases the repetition of patterns;
- a larger value increases the randomness and decreases the repetition of patterns.

The seed is converted to the integer array counterpart using the mappings .json file 4.3.5, and then one hot encoded, so it can be fed to the model and get a prediction.This prediction is a probability distribution (an array). The probability distribution is then sampled with temperature. This process is done by getting the **logarithm** of the probability distribution and divide it by the temperature value assigned (we will call this value x. Afterwards, we arrive at the probability distribution sampled with temperature, by applying a **Softmax** function (Equation 4.1 with x as the previous calculated value. Then, we sample one of the values of the distribution, and that is the value of the output.

The seed is then updated by appending the output to it, the output is mapped 4.3.5 to the original encoding, appended to an array which will contain the entire generated music and this process is repeated for the number of steps specified.

The last step is to save the music in to a Music21(3.3) stream and convert it in to a Midi (2.2.5) file.

The user chooses wether they want a happy sample or a sad sample and the program generates accordingly, choosing the specific model.

4.7 Tests and results

In the State-of-the-art methods for music generation 2.4, the authors disregard the use of a test set for the evaluation of the LSTM model and its generative capabilities. Due to its core nature, the results of this implementation are bound to be ambiguous so its evaluation is best suited for an hearing task.

Yang et al. [25] states that the difficulty in designing evaluation methodologies has hampered research on automatic music generation systems, and there exists a lot of valid subjective approaches evaluations. As mentioned by Madhok et al. [11], exploring better evaluation metrics for judging emotion of generated music remains a challenge.

As such, in the figures 4.3 and 4.4 are provided QR Codes that will redirect to a playing of .mp3 files containing two samples of "happy music" (Figure 4.3a and Figure 4.3b) and two samples of "sad music" (Figure 4.4a and Figure 4.4b)generated by both models, using two random seeds that were picked out of the pre-processed DS, and fed to the two models.

Since a melody (2.2.3) can repeat its patterns and intricacies several times through out itself, the temperature that should be assigned when generating a music sample should be a number below 1. It was assigned a value of **0.7** to all the samples below. In order to generate the "happy samples", it was assigned 2000 steps to the model, since the "happy dataset" has more sequences. 500 steps were assigned to the "sad model" in order to generate the "sad samples".

As we can hear the music "resolves" or, in other words, it finishes in a pleasant way, which means the model was capable of understanding the patterns that comprises a melody and generates them accordingly. The "sad music" emotes a dark tone, and the "happy music" emotes a cheerful, innocent tone. This is a good demonstration of different emotions displayed in the music generated.

A good subjective approach to evaluate the model is creating a seed that does not exist in the DS but it is comprised of symbols of the vocabulary, and check if the model is able to generate music samples with the same characteristics as it normally would have.

In figure 4.3c and figure 4.4c is shown music samples generated with two seeds of the aforementioned type. It is shown that the models still generates music where the melody "resolves", contains the characteristics mentioned above(depending on what emotion it is) and is fairly pleasant to hear, so we can deduce that the model is not underfitting or overfitting.

(a) QR Code of Happy song #1

(b) QR Code of Happy song #2

(c) QR Code of Happy song #3

Figure 4.3: QR Codes for happy songs

(a) QR Code of Sad song #1

(b) QR Code of Sad song #2

(c) QR Code of Sad song #3

Figure 4.4: QR Codes for sad songs

4.8 Conclusions

In this chapter, it was described step-by-step the implementation of this Project, from the DS, to the music generated by the models trained with said DS. The final results were presented via QR Code for anyone interested in listening to them.

Chapter

5

Conclusions and Further Work

5.1 Conclusions

Generating music with a learning algorithm is a challenging task. It is a inherently subjective area, and there is a lot of work to be done in this particular field of research. Generating music with a specific parameter is not excluded from the challenges that need to be faced.

This project does what is intended to do. MG based on a given emotion is a very complex problem/topic and there were a lot of limiters in the development of this project. There were a lack of consistent DS available for the very specific task at hand, having to resort to different "genres" of music for the different emotions. Evaluating a MG model is challenging on its own, and the lack of concrete objective methodologies was a problem in the final stages.

The subjective methodology elaborated was carefully pondered and executed in order to analyse and show the results that were considered satisfactory.

5.2 Further Work

This work was heavily focused on monophonic music, but monophonic music is just the tip of the iceberg when it comes to music complexity and feeling. Research and development in the area of polyphonic MG is a valid next step and good idea for further work regarding this project.

In order to improve this project facing the ambiguity of "emotion", diving deeper in to both of the approaches mentioned in 2.4 and possibly combining them both would take this Project to greater lengths. With enough time and resources, it is definitely possible and something worth to research about.

Bibliography

[1] Christopher Olah. Understanding LSTM Networks, 2015. [Online] `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Last access in June 30th,2022.

[2] Rita Katherine Steblin. *Key Characteristics in the 18th and Early 19th Centuries: A Historical Approach*. PhD thesis, University of Illinois at Urbana-Champaign, 1981.

[3] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

[4] Hao-Wen Dong and Yi-Hsuan Yang. Convolutional generative adversarial networks with binary neurons for polyphonic music generation. *arXiv preprint arXiv:1804.09399*, 2018.

[5] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5:64–67, 2001.

[6] Jean-Pierre Briot. From artificial neural networks to deep learning for music generation: history, concepts and trends. *Neural Computing and Applications*, 33(1):39–65, 2021.

[7] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[8] Ralf C Staudemeyer and Eric Rothstein Morris. Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*, 2019.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for

symbolic music generation and accompaniment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[11] Rishi Madhok, Shivali Goel, and Shweta Garg. Sentimozart: Music generation based on emotions. In *ICAART*, pages 501–506, 2018.

[12] Lucas N Ferreira and Jim Whitehead. Learning to generate music with sentiment. *arXiv preprint arXiv:2103.06125*, 2021.

[13] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017.

[14] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[15] Michael Scott Cuthbert and Christopher Ariza. Music21: A toolkit for computer-aided musicology and symbolic music data. In J. Stephen Downie and Remco C. Veltkamp, editors, *ISMIR*, pages 637–642. International Society for Music Information Retrieval, 2010.

[16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[17] François Chollet et al. Keras. `https://keras.io`, 2015.

[18] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[19] Akshay Mattoo. Ai music generator, 2020. [Online] https://github.com/matakshay/AI_Music_Generator, Last accessed on July 11th 2022.

[20] Valerio Velardo. Generating melodies with rnn lstm, 2020. [Online] https://github.com/musikalkemist/generating-melodies-with-rnn-lstm, Last accessed on July 11th 2022.

[21] Sigurður Skuli. Classical piano composer, 2017. [Online]https://github.com/Skuldur/Classical-Piano-Composer, Last accessed on July 11th 2022.

[22] Folksongs from the continent of europe. [Online]https://kern.humdrum.org/cgi-bin/browse?l=essen/europa, Last accessed on July 11th 2022.

[23] Blues genre midi melodies. [Online]https://www.kaggle.com/datasets/function9/blues-genre-midi-melodies, Last accessed on July 11th 2022.

[24] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. Deep learning techniques for music generation–a survey. *arXiv preprint arXiv:1709.01620*, 2017.

[25] Li-Chia Yang and Alexander Lerch. On the evaluation of generative models in music. *Neural Computing and Applications*, 32(9):4773–4784, 2020.