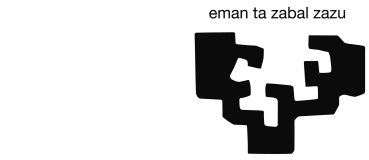

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Bachelor Degree in Computer Engineering

Computation

Thesis

Music composition and interpretation using transformer networks

Author

Ruben Naranjo de las Heras

2020

Abstract

This work presents the development of a deep learning model capable of generating and completing musical compositions automatically through generative algorithms of machine learning from a language modeling approach.

Throughout the document, different neural network structures are studied and compared from vanilla recurrent neural networks to transformers, and the representation of data is discussed, as well as some design aspects for the creation of a model capable of composing and interpreting musical compositions.

The model is trained and tested three times, one for each of the two different datasets and finally one with both together. Then, the three resultant models are discussed and one of them is tested with human subjects to validate the generated musical compositions.

The document also presents the design and implementation of a web-interface aimed at non-technical users, to assist them in the creative process of music composition.

Contents

List of Figures

Table index

CHAPTER 1

Introduction

With the advancements on technology, each day we can delegate more tasks to machines, making our lives easier. Nowadays, computers carry out complex processes and tasks that would be unthinkable not more than a couple decades ago, even imitating human behaviour. Surely, surpassing and overcoming these advances is becoming more complex, but, is there really a limit that bounds what can or cannot a machine achieve?

When presented with a problem whose solution is given by some sort of universal rule, purely logical or mathematical problems for instance, the process of implementation in a machine is completely trivial. The only work is implementing the algorithm via hardware or software.

In contrast, when facing a problem whose answer is not given by a certain rule, the complexity increases. The conversion from text to speech or speech to text, emotion detection in speech, separation of two voices in a conversation, face recognition, etc. are problems that, although we, humans, can solve relatively easy, do not have a simple way of being brought to machine language, since there is no exact rule that always indicates what the correct answer is. In other words, there is no known algorithm that indicates us how to carry out the process.

What is a face? How does a sad or happy voice sound like? And, how do we, humans, learn these concepts? It is about learning things that have no clear definition, about learning from example.

Machine Learning gives us the mechanisms to teach a machine how to learn from example

based induction. Thanks to Machine Learning algorithms, a computer can learn, generalize concepts and carry on tasks that, until relatively recently, where considered unique to living beings.

Thus, we are able to endow machines with mechanisms that mimic brain processes and solve problems of computational perception, as discussed above.

If we apply this to the generation of paintings, sculptures or music, the paradigm changes. We are no longer facing a classification or computational perception problem, there exists a fundamental creative and artistic component, which is associated uniquely with humans.

We have taught machines to identify plants, animals, faces, voices, to diagnose diseases, to translate texts... But, are computers capable of creating something artistic in nature?

Programming an algorithm to generate musical notes is relatively simple. Programming it to generate musical notes that follow a melodic pattern according to the human concept of *musical beauty* is much more complex. The previous examples had a social consensus of what the correct answer was, but in this case, there is not, because beauty is subjective. In other words there is no real correct answer. Although, there is a wrong one.

Every piece of music may not be appealing to everyone, thus the different music genres, but there are certain loose rules that a musical piece needs to follow in order to be *correct*, called *harmony* [1]. Explaining harmony definitely goes out of the scope of this project, but in order to evaluate the results generated, harmonic coherence will be taken into account.

This partial subjectivity, adds a new level of difficulty for the modeling of the data. Thus, generating artistic and creative content, is a challenge for Machine Learning and Artificial Intelligence, which will be discussed throughout this work.

1.1 Objectives of this work

The main goal of this project is to be able to generate compositions and interpretations of original pieces of music in a way that requires no technical knowledge about computer science. The project comprises two parts: The design and validation of a deep neural network model, and the design and implementation of a web-based interface that allows a user to apply the model. Generating music in any style or with as many instruments as wanted goes out of the scope of this project, so the work will be concentrated in generating classical pieces of music for piano. Still, the idea is to build an expandable model, that

could be further developed in the future. To complement this work and provide a user-friendly way of testing the model, a graphical user interface will be developed as a web-app using Flask [2].

CHAPTER 2

Project Management

This section describes in detail the planning used through the development of the project. This planning process is based on an initial planning draft made at the very beginning of the work that has been evolving as the project advanced.

2.1 Planning

2.1.1 Work breakdown structure

Before anything else, the project has been broken down into smaller tasks to organize it into manageable sections. These tasks have been grouped into five sections: Learning process, Research, Development, Documentation and Management.

The identified tasks have been grouped into sections as follows. To better visualize the tasks in its sections an WBS diagram is presented in Figure 2.1.

T1 - Learning process

- T1.1 - Revise Machine Learning Concepts
- T1.2 - Learn about Neural Networks
- T1.3 - Read articles on the topic of music generation

T2 - Research on neural network approximations to music generation

T2.1 - Model research

T2.2 - Dataset research

T2.3 - Representation research

T3 - Development

T3.1 - Implementation

T3.1.1 - Data preprocessing

T3.1.2 - Model implementation

T3.1.3 - Model evaluation

T3.1.4 - Interface Implementation

T3.2 - Testing

T3.2.1 - Model testing

T3.2.2 - Result analysis

T4 - Documentation

T4.1 - Thesis document

T4.2 - Presentation

T5 - Management

T5.1 - Planning

T5.2 - Monitoring

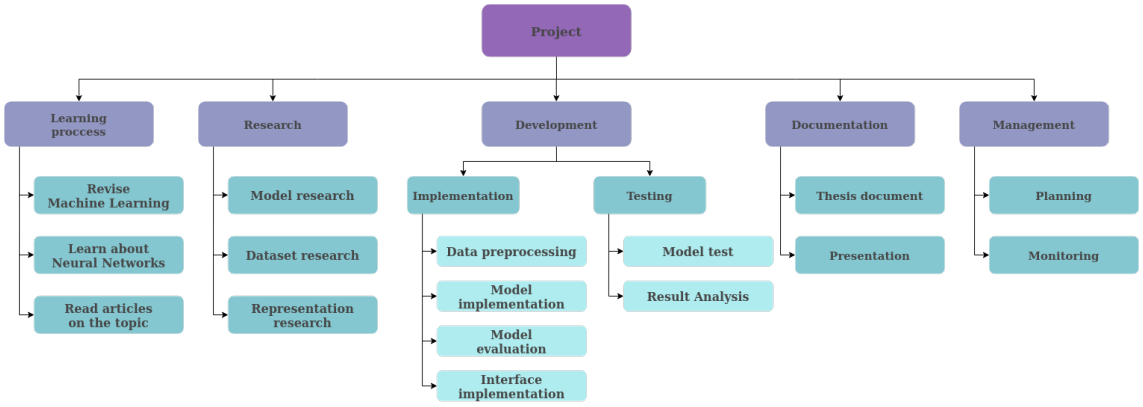

Figure 2.1: WBS diagram of the project

Task T1 comprises a revision on the already known concepts of Machine Learning, a learning process focused on Deep Neural Networks and an extensive research about music generation.

Work on T2 includes the revision of the most significant approaches to music generation that use deep learning. This comprises analyzing the characteristics of the models, research on datasets available and investigation of the most suitable representation for the model.

Task T3 is the development of the project, it includes preparing the dataset and converting it to the representation chosen in T2, the design and validation of the neural network model and the analysis of its results.

Task T4 comprises the writing of this thesis document and preparing the slides for the presentation of the thesis.

Work on T5 includes planning of tasks and time estimations for those tasks, and the later monitoring of that planning.

2.1.2 Deliverables

To fulfill this project some deliverables will be created, the main ones being: an implementation of a machine learning pipeline for training and testing a model, a web-app front-end linked to the model and the thesis document. In the thesis document, the objectives of the project, its management as well as its development process and conclusions will be detailed, written in the *LaTeX* document preparation system. In addition, some slides will be prepared as helpers for the presentation of the thesis.

Apart from the documentation described, the final neural network model for music generation will also be a deliverable. The code used to preprocess the data, the model itself and the various notebooks used to train it and later generate music will be made available on the *gitlab* platform. In addition, the code of the web-app front-end will also be provided.

2.1.3 Deadlines

This project has some administrative deadlines that need to be taken into account:

- 12^{th} of June 2020: Register the Thesis on the GAUR platform.

- 12th of June 2020: Thesis tutor gives permission for presentation.
- 21st of June 2020: Upload the project to ADDI platform.
- 29st of June - 10th of July 2020: Presentation of the Thesis.

In addition to the administrative deadlines, some academic deadlines have been defined:

- 1st of February 2020: implement the data preprocessing.
- 15th of April 2020: finish the neural network model.
- 20th of May 2020: first version of the thesis.

2.1.4 Gantt chart

Figure 2.2 shows all previously detailed tasks spread over the duration of the project.

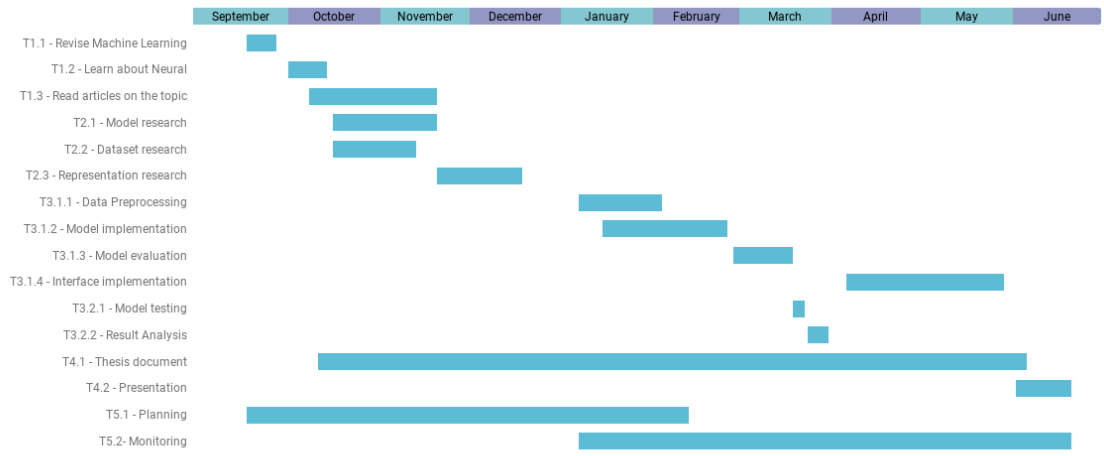

Figure 2.2: Gantt chart of the project

2.1.5 Time estimation

Table 2.1 details the estimated time of each task, grouped into sections.

Task	Estimation
T1 - Learning process	35
T1.1 - Revise Machine Learning	5
T1.2 - Learn about Neural Networks	20
T1.3 - Read articles on the topic	10
T2 - Research	35
T2.1 - Model research	15
T2.2 - Dataset research	10
T2.3 - Representation research	10
T3 - Development	160
T3.1 - Implementation	130
T3.1.1 - Data preprocessing	20
T3.1.2 - Model implementation	70
T3.1.3 - Model evaluation	20
T3.1.4 - Interface implementation	20
T3.2 - Testing	30
T3.2.1 - Model testing	15
T3.2.2 - Result analysis	15
T4 - Documentation	55
T4.1 - Thesis document	50
T4.2 - Presentation	5
T5 - Management	15
T5.1 - Planning	12
T5.2 - Monitoring	3
Total	300

Table 2.1: Time estimation

2.1.6 Risk management

In long-term projects, it is very common to identify risks and prepare a contingency plan to avoid or reduce the impact that complications that may appear throughout development can have.

The main identified hazard that can affect this project gravely is the loss of data, this problem could imply having to restart the project from the very beginning and thus, has been identified as a critical problem. To avoid and prevent it, the dataset and code are stored online, on Google Drive and Gitlab.com; it is also stored in a personal hard drive should those online solutions fail. The thesis of the project is written in the Overleaf.com platform, and also stored in the previously mentioned hard drive; copies of the generated documents are stored in Google Drive and Dropbox.

Other risks include bad results and bad planning; the first issue can be avoided by basing the model on previously tested ones and carefully studying the code to correct potential mistakes. The latter will be prevented by carefully setting deadlines taking into account some extra time just in case.

Another potential risk is the lack or limited availability of hardware resources, the model used in the project is very complex and requires very potent hardware, since it has a very high computational cost. To avoid not satisfying the hardware requirements, the code is executed on the Google Colab online platform which lets the code run in very high-end CPU and GPUs.

Last but not least, personal issues have been identified as risk, and could potentially lead to a delay on the project. This problem is hard to avoid as personal issues are hardly expected and may come in myriad ways. The impact of this risk can be lessened the same way as the previously discussed bad planning risk.

2.2 Monitoring

Section 2.1.5 shows the expected time to spend on each of the tasks. Now, once the project is finished, we are going to see a comparison between that estimate and reality in Table 2.2.

Task	Estimation	Reality
T1 - Learning process	35	31
T1.1 - Revise Machine Learning	5	1
T1.2 - Learn about Neural Networks	20	15
T1.3 - Read articles on the topic	10	15
T2 - Research	35	33
T2.1 - Model research	15	13
T2.2 - Dataset research	10	8
T2.3 - Representation research	10	12
T3 - Development	160	190
T3.1 - Implementation	130	165
T3.1.1 - Data preprocessing	20	25
T3.1.2 - Model implementation	70	80
T3.1.3 - Model evaluation	20	25
T3.1.4 - Interface implementation	20	35
T3.2 - Testing	30	25
T3.2.1 - Model testing	15	20
T3.2.2 - Result analysis	15	5
T4 - Documentation	55	85
T4.1 - Thesis document	50	82
T4.2 - Presentation	5	3
T5 - Management	15	20
T5.1 - Planning	12	15
T5.2 - Monitoring	3	5
Total	300	354

Table 2.2: Time estimation vs Reality

CHAPTER 3

Background

3.1 Music

Music is an art form and cultural activity whose medium is sound. By all accounts there is no single and intercultural universal concept defining what music might be; in fact, the border between noise and music is nebulous and always culturally defined [3]. For the purposes of this work, we will define music as a sequence of sound and silences, expressed through time; or as *organized sound* [4]. What is clear is that music is comprised by some common elements like *pitch*, *rhythm*, *dynamics*, and *timbre* and *texture*.

Pitch lets us differentiate between bass and tremble sounds, and depends on the frequency of the sonic wave. This, determines the tone of a note. In music notation it is represented with a *stave*, *clef*, notes and accidentals of those notes. The clef acts as a reference in the stave, showing which note corresponds on which position.

Such that the *Treble clef*, shown in Figure 3.1, indicates that the note on the second line (counting from below) corresponds to a *G*. Notes in the musical scale (C, D, E, F, G, A, B, [C])[1] are separated by one tone between them, except for the E-F and B-C that are one semitone from each other[2].

[1] There are several musical scales used in music. This one is the C Major scale, one of the few that have no accidentals on the notes.

[2] A semitone or half-tone is the smallest musical interval commonly used in Western music, a tone is twice a semitone.

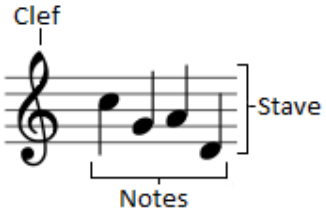

Figure 3.1: Stave, clef and notes in music notation.

Accidentals, *sharp* (♯) and *flat* (♭) allow to indicate variations of a semitone up or down respectively. The accidental *natural*(♮) cancels the previous accidentals on a note.

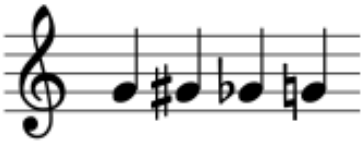

Figure 3.2: Accidentals in music notation.

The second component of music is rhythm, which is associated with several things, such as beat, repetition, and metric structure. For the purposes of understanding this work, we are going to focus only on duration, that is, how long the sounds are. This is represented in music mainly by *tempo*, *note figures* and *time signature*. Tempo is indicated as *beats per minute* (bpm), and establishes the velocity on which notes must be played, such that ♩= 60 indicates that the velocity is 60 quarter notes (or crochets) per minute.

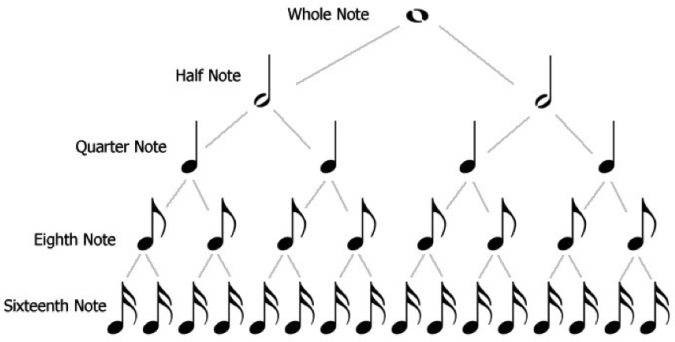

Figure 3.3: Rhythm hierarchy.

Once the tempo is established, the rest of the note figures will have a relative duration as shown in Figure 3.3.

On the other hand, time signature acts as metric, dividing a stave into defined time units, called bars. Time signature is represented at the beginning of the stave, with a fraction, the denominator indicates the unit of measurement and the numerator the number of measures on each bar. For example, time signature $\frac{4}{4}$ means that each bar must have the equivalent of 4 $\frac{1}{4}$ units (quarter notes), such as 1 whole note, 2 half notes, 4 quarter notes, etc.

Dynamics in music refers to the variation in loudness between notes or series of notes. Dynamics are indicated by specific musical notation shown in Table 3.1. However, dynamic markings still require interpretation by the performer depending on the musical context and the period in which the piece was composed.

Abbreviation	Full word	Definition
pp	Pianissimo	Very soft
p	Piano	Soft
mp	Mezzo-piano	Medium soft
mf	Mezzo-forte	Medium loud
f	Forte	Loud
ff	Fortissimo	Very loud
cresc.	Crescendo	Gradually Louder
dim.	Diminuendo	Gradually softer

Table 3.1: Music dynamics

The other component, timbre, is not specially relevant for the case at hand. This is the property by which we differentiate one instrument from another.

3.2 Music representations

Music can be represented in a myriad of ways, from the most common *music score* shown in Figure 3.4 to the less known *pianoroll* which is shown in Figure 3.5. This project follows the advice of Oore et al. [5] and uses the MIDI file standard, whose main advantage apart from the ones described in [5] is the vast variety of datasets that can be found on internet.

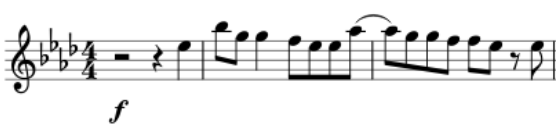

Figure 3.4: Music Score example

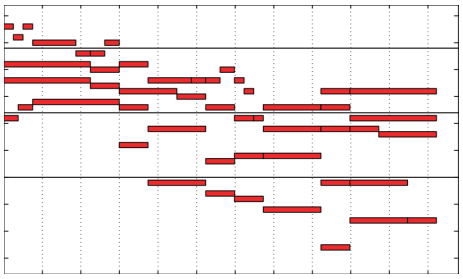

Figure 3.5: Pianoroll example

3.2.1 MIDI file standard

Musical Instrument Digital Interface (MIDI) [6] is a specification designed to exchange information between different electronic musical instruments, computers and various audio recording, editing and playing devices. The information exchanged is not music per se, but rather a series of events that specify the instructions for music, including (but not limited to) note's notation, pitch and velocity. It was originally designed for live performances, but subsequent development has shown that MIDI is a great tool for recording studios, audio and video production and, composition environments; and so, the MIDI File Standard was created [7].

MIDI communication is done through *MIDI messages*, that are intended to be received by only one of perhaps many available devices that can be connected together. The *MIDI channel* provides an easy way to differentiate these devices. A message intended for the device on channel one, for example, will have that MIDI channel number present in its data. Only devices assigned to listen on channel one will respond to any messages with this encoding. The current MIDI specification calls for 16 MIDI channels.

A MIDI message is made up of an eight-bit status byte which is generally followed by one or two data bytes. There are a number of different types of MIDI messages. At the highest level, MIDI messages are classified as being either Channel Messages or System Messages. Channel messages are those which apply to a specific Channel, and the Channel number is included in the status byte for these messages. System messages are not Channel specific, and no Channel number is indicated in their status bytes.

Channel Messages may be further classified as being either Channel Voice Messages, or Mode Messages. Channel Voice Messages carry musical performance data, and these messages comprise most of the traffic in a typical MIDI data stream. Channel Mode messages affect the way a receiving instrument will respond to the Channel Voice messages.

MIDI System Messages are classified as being System Common Messages, System Real Time Messages, or System Exclusive Messages. System Common messages are intended for all receivers. System Real Time messages are used for synchronization between clock-based MIDI components. System Exclusive messages include a Manufacturer's Identification (ID) code, and are used to transfer any number of data bytes in a format specified by the referenced manufacturer.

MIDI files contain one or more MIDI Streams, with added time data for each MIDI event. The most notable advantages of MIDI Files include small file size, ease of modification and manipulation and a wide choice of compatible devices. A MIDI file contains MIDI event messages and other descriptive information.

3.2.2 MIDI file structure

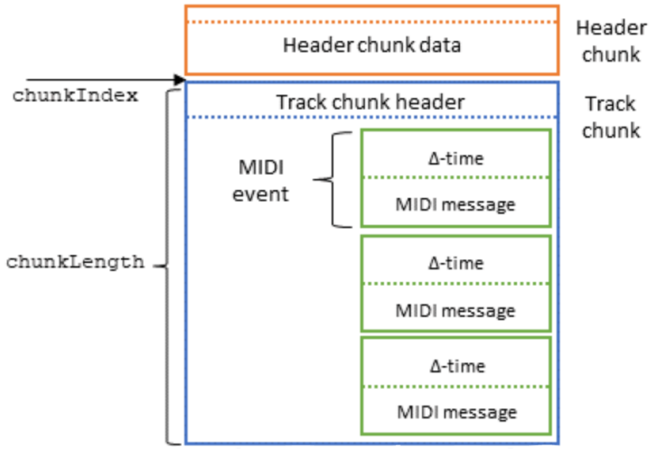

Figure 3.6: Basic structure of a MIDI file

MIDI files are made of two types of chunks, *Header chunks and Track chunks*. A MIDI file always begins with a Header chunk, followed by one or more Track chunks.

3.2.2.1 Header chunk

The Header chunks provides some basic information regarding the entirety of the MIDI file. Here is the syntax of the Header chunk:

chunk_type length format ntrks division

- **chunk_type:** The ASCII word 'MThd'.
- **length:** 32-bit representation of the number 6 (high byte first).
- **format:**

  0 = The file contains a single multi-channel track.

  1 = The file contains one or more simultaneous tracks of a sequence.

  2 = The file contains one or more sequentially independent single-track patterns.

- **ntrks:** Refers to the number of track chunks in the file.
- **division:** specifies the meaning of the time-marks. It has two formats, one for metrical time, and one for time-code-based time:

bit 15	bits 14 through 8	bits 7 through 0
0	ticks per quarter-note	
1	negative frames per second	ticks per frame

Table 3.2: Encoding of the two types of MIDI division

- If bit 15 is zero, the bits 14 through 0 represent the number of delta time "ticks" which make up a quarter-note. For instance, if division is 96, then a time interval of an eighth-note between two events in the file would be 48.
- If bit 15 is one, delta times in a file correspond to subdivisions of a second. Bits 14 through 8 contain one of the four values -24, -25, -29 or -30, representing the number of frames per second in negative. The second byte is the resolution within a frame.

An example header chunk for Sebastian Bach's BMW 847 [8] could be:

MThd 6 1 10 480

or in its hexadecimal form:

4d546864 00000006 0001 000a 01e0

3.2.2.2 Track chunk

The Track chunks are where the song data is actually stored. The Track chunk is simply a stream of MIDI events, preceded by delta-time values. Here is the syntax of a Track chunk:

chunk_type length MTrk_event

- **chunk_type:** The ASCII word 'MTrk'.
- **length:** 32-bit representation of the length in bytes for the Track chunk.
- **MTrk_event:** A MIDI event preceded by a delta-time value.

The syntax of the MTrk_event is very simple:

delta-time event

- **delta-time** represents the amount of time before the following event.
- **event**: the syntax of an event is as follows:

MIDI event | sysex event | meta-event

  - **MIDI event** is any MIDI channel message.
  - **sysex event** is used to specify a MIDI system exclusive message.
  - **meta-event** specifies non-MIDI information useful to software dedicated to creating MIDI files.

3.3 Machine Learning

Machine learning is a field of Computer Science and one of the primary fields of artificial intelligence, whose goal is the development of techniques and mechanisms that provide a system the ability to learn something without being explicitly programmed for it [9] [10].

In this case "learning" refers to identifying complex patterns on large amounts of data. This is, starting from a set of example cases (which we call "training *dataset*"), being capable of recognizing the general patterns (and not the particular ones) that follow said examples. We talk about general patterns, because the goal is to generalize this behaviour, in such a way that the system can reproduce the results for cases included in those examples, as well as for new cases.

"Without being explicitly programmed for it" refers to the possibility of applying said process in a multiple of different cases, without the need to change the algorithm. Instead of programming an algorithm (explicitly) each time to reproduce an specific behaviour or knowledge, the same algorithm can reproduce several behaviours depending on the data on which it has been trained.

Machine learning algorithms can be grouped in two main classes, depending on the objective of the algorithm and the input data it manages [11]:

- Supervised learning: where a set of example data is provided together with a class, category or expected result value, which we call *label*. This kind of algorithms try to identify patters common to each of the classes, with the goal of generalizing this knowledge into a rule or set of rules, that predict the labels of new data.
- Unsupervised learning: where data is not labeled. The algorithm tries to recognize patterns common in all data entries to group them (which is known as *clustering*), with the goal of identifying sets of data with similar characteristics, and then, label the new data with the internal classes created by the algorithm.

There are a lot of different ways to approach these problems, both in supervised learning as well as in unsupervised learning, such as decision trees, association rules, bayesian networks... Within supervised learning we can group learning models into discriminative and generative.

The discriminative models learn the direct conditional probability $p(x|y)$ for data x and label y (on probabilistic models), or a direct mapping of input x to label y (on non-probabilistic models), that is, they learn to categorize, they model the dependency of variable y according to variable x. On the other hand, generative models learn the joint distribution $p(x,y)$, that is, they model the way data is structured and distributed (and therefore, the way they are generated), and can make predictions $p(y|x)$ using Bayes's theorem.

$$P(A,B) = P(A|B) \cdot P(B) = P(B|A) \cdot P(A) \tag{3.1}$$

Generally, the error curve of discriminative models presents a lesser asymptote than that of generative models. However, the latter ones converge at a much faster rate. Therefore, when the quantity of data is lowered, generative models give better results, and vice-versa [12].

On this work, we are going to focus on artificial neural networks that, as we will see, are very potent generative models that, among its multiple uses, are going to let us generate content (music in our case) thanks to their capability of learning data structure and distribution, and to deep learning.

Deep Learning constitutes a specific area within Machine Learning. In the field of neural networks, it is primarily distinguished by the inclusion of multiple layers of neurons that add depth to the neural network. This, as will be seen later, allows the neural network to learn characteristics at much higher levels of abstraction than with traditional neural networks [13].

3.4 Artificial Neural Networks and Activation Functions

Artificial neural networks are network like structures composed by artificial neurons, whose functioning is inspired in the animal brain [14]. Artificial neurons receive a set of numerical inputs as a vector, mimic the behaviour of a neuron using mathematical calculations that we call *transfer function* (usually a weighted sum), and apply an *activation function* to the result to return an output. The most popular activation functions are the *logistic* function or *sigmoid* function (Equation 3.2), the *hyperbolic tangent* function (Equation 3.3) and the *Rectifier Linear Unit (ReLU)* function (Equation 3.4).

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3.2}$$

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3.3}$$

$$f(x) = max(0,x) \tag{3.4}$$

The sigmoid function was the first to gain popularity as an activation function, due to not being linear, having a bounded output in range (0,1) and low computational cost of calculating its derivative (needed for the *backpropagation* algorithm used to train neural networks). The hyperbolic tangent function succeeded the sigmoid, since being symmetric (centered in zero), converges faster.

Both this functions have a very well known downside in common, the *Vanishing gradient problem*. The functions map a very large domain of the function $(-\infty,\infty)$ to an output of range $(0,1)$, where the majority of the inputs is centered at the proximities of the asymptotes of the function. This leads to the fact that a large change in the input value is reflected as a small variation on the output side, which translates to a small gradient.

With multiple linked layers the problem only gets worse, since the mapping of the $(-\infty,\infty)$ domain in the output range $(0,1)$ is produced on the first layer, then mapped again in the next layer and so on. Thus, a large change on the input will result on a tiny variation of the output. Thereby, the gradient slowly disappears during backpropagation and the neurons of the first layers do not learn.

This is why the ReLU function has gained popularity on recent years. It is a simple function to implement and derive [3], it has shown to be more robust to the problem of the vanishing gradient and, in consequence, allows to train networks much faster. On the other hand, neurons that implement a ReLU activation function can reach a point on which the transfer function always returns negative results (usually due to the *bias*), and the output of the activation becomes 0, as well as its gradient, thereby the neuron stops learning and its said to *die*. Thus, recently a new variation has appeared, named *Leaky ReLU* which behaves better in this cases.

3.5 Recurrent Neural Networks

Classic neural networks take in a fixed size vector as input and generate an output. This, limits their usage in situations that involve a series type input with no predetermined fixed

[3] Although not being differentiable at the point $x = 0$, 0 is taken as a derivative at that point.

size, such as in language translation, where translation is done for sentences of any length. In this case, if we want to translate the sentence "*The cat eats the mouse*" into Spanish, we would call the classic neural network five times, one for each word in the sentence; and with this particular sentence it might turn out well, but languages have a lot of ambiguity and to translate the majority of the words on a sentence, our neural network should know which other words came before the current one.

Recurrent neural networks (RNN) introduce a short-time memory mechanism to accomplish this. As we can see in Figure 3.7 , RNNs have loops, allowing information to persist form one iteration to another, this can be though of as calling the same neural network repeatedly as in the example above, but this time, passing some information from call to call. This passed information is called the internal state or hidden state.

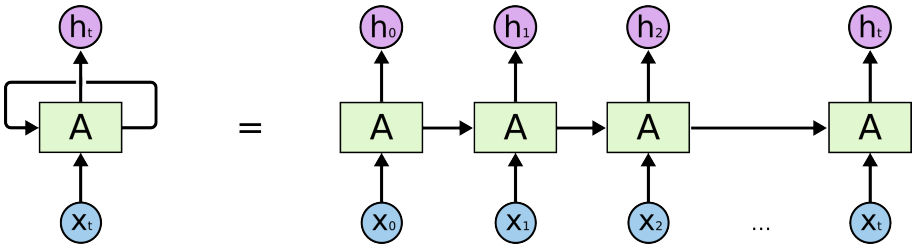

Figure 3.7: Recurrent neural network [4]

Due to its structure, recurrent neural networks are trained by means of *Backpropagation Though Time (BPTT)* [15], which is nothing more than a normal backpropagation adapted to fit recurrent neural networks. The major difference is that complete sequences are taken as training samples; whence, if we take into account that the data generated depends on previous data, to calculate the gradient of the error for a parameter in a certain moment, we will need to compute the partial derivatives of the elements that precede that particular data on the current iteration. Thereby, the number of derivatives increments and, if we consider that the derivative of the sigmoid function return a value between 0 and $\frac{1}{4}$ and hyperbolic tangent returns a value between 0 and 1, the gradient may tend to 0 rapidly.

As a result, recurrent neural networks are specially hard to train, and can also suffer the vanishing gradient problem. Nonetheless, the use of hyperbolic tangent activation functions is very common.

This is due to the fact that, even if ReLU improves the problem of the vanishing gradient, there exists another problem that affects both functions, known as *Exploding gradi-*

[4] Figure taken from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

ent problem, in which the weights of the first layers (or the first temporal elements, in recurrent networks) reach very high values, making neurons unable to learn [16]. This problem, which is specially characteristic of recurrent networks, gets much worse when using ReLU, since its output is not bounded as in the case of the hyperbolic tangent and sigmoid functions.

To solve these problems, Sepp Hochreiter and Jürgen Schmidhuber introduced in 1997 the *Long Short-Term Memory networks (LSTM)* [17]. A type of recurrent network composed from LSTM modules that implement *gating* mechanisms to give the network long-term memory, and solve the gradient problems discussed.

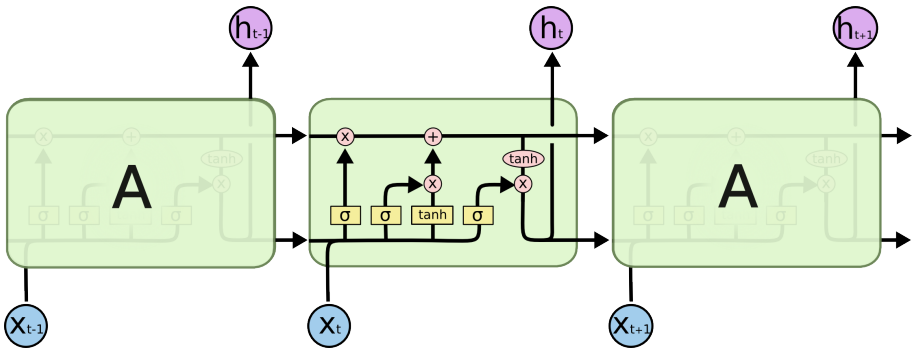

Figure 3.8: Structure of LSTM networks [4]

This long-term memory is implemented through the cell state of the LSTM module that appears represented in Figure 3.9 by the upper horizontal line (C). This cell state is multiplied by the output of the first sigmoid (f_t), which decides what data is erased. Then, it is summed to the result of another two gates (sigmoid i_t and hyperbolic tangent $\tilde{C}_t$) that add new information to the cell state. This new state is passed to the next unit, and is used to filter o_t and produce the output h_t.

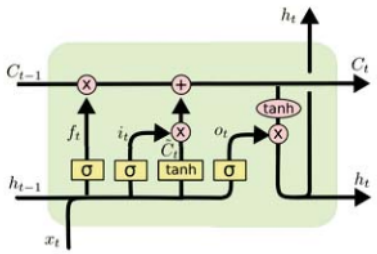

Figure 3.9: LSTM cell [4]

This is mathematically represented in Equation 3.5, f_t being the information to forget (*forget gate*), i_t the information to add *input gate*, $\tilde{C}_t$ the new memory content that goes

though i_t, C_t the memory cell and o_t the *output gate* that controls the quantity of memory that is exposed on output h_t.

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f) \\
i_t &= \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i) \\
\tilde{C}_t &= tanh(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o) \\
h_t &= o_t * tanh(C_t)
\end{aligned}
\tag{3.5}
$$

Later, in 2014, Cho et al. proposed a structure similar to LSTM called *Gated Recurrent Unit (GRU)* [18].

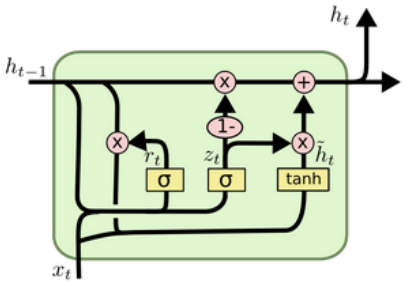

Figure 3.10: GRU cell [4]

GRU combines the input gate and forget gate into a single *update gate*. It also merges the cell state and hidden state, and makes other smaller changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.

$$
\begin{aligned}
z_t &= \sigma(W_z \cdot x_t + U_z \cdot h_{t-1}) \\
r_t &= \sigma(W_r \cdot x_t + U_r \cdot h_{r-1}) \\
\tilde{h}_t &= tanh(W \cdot x_t + U \cdot (r_{t-1} \odot h_{t-1})) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned}
\tag{3.6}
$$

In this case, there is no output gate, so all the content from the inner state is exposed.

3.6 Attention and Transformers

Although LSTM-s solved a large amount of problems for the structure of recurrent neural networks, they still suffer from the vanishing gradient problem; struggle learning long-range dependencies, due to the length of paths that forward and backward signals must traverse in the network; and are not easy to parallelize, due to the sequential nature of their computation.

In order to reduce sequential computation and path lengths, several models where proposed, such as ByteNet [19] and ConvS2S [20], all of which use convolutional neural networks as basic building block, computing hidden representations in parallel for all input and output positions. In these models, the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions, linearly for ConvS2S and logarithmically for ByteNet. This makes it more difficult to learn dependencies between distant positions. In 2017, the *Transformer* model was proposed [21], a neural network structure relying entirely on *attention*. This model reduces the number of operations to a constant, albeit at the cost of reduced effective resolution due to averaging attention-weighted position, this effect is counteracted with *Multi-Head Attention* which will be discussed shortly.

Attention is a mechanism that lets the model learn and later focus on the parts of the input that are important to the current task. An attention function can be described as mapping a *query* and a set of *key-value* pairs to an output, where the *query*, *keys*, *values*, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

The transformer model proposed in [21] has a particular type of attention they call *Scaled Dot-Product Attention*. The input consists of queries and keys of dimension d_k, and values of dimension d_v. A dot product of each query with all keys is made, then each result is divided by $\sqrt{d_k}$ and finally, a softmax function is applied to obtain the weights on the values. In practice, the attention function is computed on a set of queries simultaneously, packed together into a matrix (Q). The keys and values are also packed together into matrices (K and V). The formula for the whole process is shown in Equation 3.7.

[5] Image taken from [21]

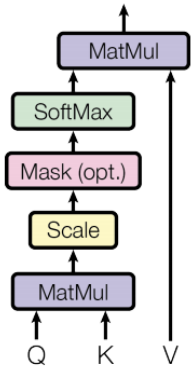

Figure 3.11: Scaled Dot-Product Attention [5]

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.7}$$

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions [21].

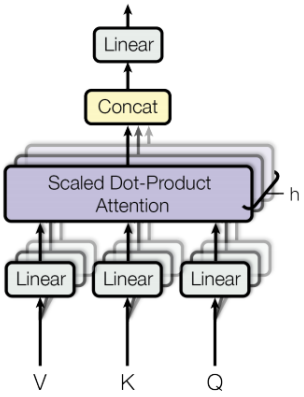

Figure 3.12: Multi-Head Attention [5]

$$\begin{aligned} MultiHead(Q,K,V) &= Concat(head_1,...,head_h)W^O \\ \text{where } head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{3.8}$$

Where $QW_i^Q \in \mathbb{R}^{d_{model} \times d_k}, KW_i^K \in \mathbb{R}^{d_{model} \times d_k}, VW_i^V \in \mathbb{R}^{d_{model} \times d_v}$ are parameter matrices of h times linearly projected queries, keys and values.

The transformer uses multi-head attention in three different ways:

- Encoder-decoder attention: which allows every position in the decoder to attend over all positions in the input sequence.
- Encoder self-attention: which allows every position in the encoder to attend over all position in the previous encoder layer.
- Decoder self-attention: which allows every position in the decoder to attend over all position in the previous decoder layer.

Additionally, the transformer uses common *Position-wise Feed-Forward Networks* on each of the layers of the encoder and decoder, and utilizes learned embeddings to convert input tokens and output tokens into vectors. The complete structure of the Transformer is shown in Figure 3.13.

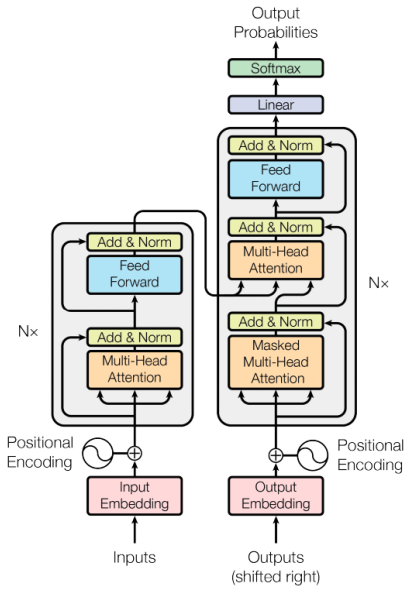

Figure 3.13: Transformer model structure [5]

3.7 One-Hot encoding and softmax activation function

The majority of learning models do not handle categorical variables (cat, dog, bird, mouse,...), thereby, the categorical values are transformed into numerical ones (cat, 1; dog, 2; bird, 3; mouse, 4). This implies a relation in the order of the variables, $cat < dog < bird < mouse$.

As this relation does not initially exist, it is undesirable, so, usually it is transformed into the *One-Hot encoding*.

The One-Hot encoding consists in representing a categorical variable in a binary vector. This vector will have a length equal to the number of values the categorical variable can have, such that each position corresponds to one of the values.

Cat	Dog	Bird	Mouse
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Table 3.3: One-Hot encoding example

This is specially useful for the *softmax* activation function.

$$S(y_i) = \frac{e^{y_i}}{\Sigma_j e^{y_i}} \tag{3.9}$$

The softmax function is a generalization of the logistic function that compresses a vector with real values into a vector of the same dimension but whose values are in range $[0,1]$. Following with the previous example, if we have a neural network with 4 outputs (one for each label), this function will transform the values of the vector into probabilities, so that, the sum of all values of the resultant vector is 1. Therefore, is largely utilized in the last layer of a neural network.

The One-Hot encoding is very adequate when using the softmax function, since it represents data in an equal manner, where the correct variable indicates 1 (100%) and the rest 0 (0%).

3.8 Quality metrics

Quality metrics let us evaluate the performance and results of learning models. While training a neural network is complicated, for the case at hand, evaluation is specially complex, since there is no canonical way of evaluating music quality.

This implies that, with the known metrics, the model whose metrics are better will not necessarily produce better music, but said model will adjusts more to the distribution

of data. Generally speaking, it is expected that the model that adjust better to the data distribution, will produce better music, or at least, will produce music that is more similar to the one used on training. But not necessarily, since music quality apart from being highly subjective, is not directly reflected on metrics.

This also has a special relation with what the metrics represent, which could sometimes be misleading if not correctly interpreted.

The error or *loss* discussed earlier, is commonly used to train models with backpropagation. It is a way of calculating the error made in the predictions, and correct the values. There are many functions that are used as loss functions, such as, *Mean Squared Error (MSE)*, *Mean Absolute Error(MAE)*, *Hinge*, *CrossEntropy*, etc.

It is important to clarify the difference between *accuracy*, *precision* and *recall*.

Considering a problem with two classes as an example, there are 4 parameters that can be computed from the results:

- *True Positive*: Positively classified positive samples.
- *False Positive*: Positively classified negative samples.
- *True Negative*: Negatively classified negative samples.
- *False Negative*: Negatively classified positive samples.

This is expressed generally by means of a *confusion matrix*, where original classes are disposed on an axis and predicted classes on the other axis, distributing the samples by its original and predicted classes.

	Predicted class: Yes	Predicted class: No
Original Class: Yes	True Positive	*False Negative*
Original Class: No	*False Positive*	*True Negative*

Table 3.4: Confusion matrix

With these parameters, the *accuracy* is calculated as the correct responses divided by the total number of responses. *Precision* is computed for each of the classes and corresponds to the number of samples correctly classified in that class divided by the total number of samples classified as that class. *Recall* is also computed for each class, corresponding

to number of samples correctly classified in that class divided by the total samples that originally belong to each class.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{3.10}$$

$$Precision = \frac{TP}{TP+FP} \tag{3.11}$$

$$Recall = \frac{TP}{TP+FN} \tag{3.12}$$

This way, if our model classifies nearly all samples as negative, we will have a high precision (due to the lack of false positives), but that will not mean that our model is working correctly, since recall will be very low due to the large amount of positives classified as negatives (false negatives).

As a combination of precision and recall, we have the *F-measure*, that provides a balanced value between those two:

$$F_1 = 2 \cdot \frac{1}{\frac{1}{recall} + \frac{1}{precision}} = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3.13}$$

On the other hand, it is very common to use the ROC curve, which is a graphic that represents recall (*True Positive Ratio* or TPR) facing specifity (*True Negative Ratio* or TNR), or facing the *False Positive Ratio* (FPR).

$$TNR = \frac{TN}{TN+FP} = 1 - FPR \tag{3.14}$$

$$FPR = \frac{FP}{TN+FP} = 1 - TNR \tag{3.15}$$

One of the most common quality metrics is to calculate the *Area Under Curve (AUC)*, being better the greater the area is.

CHAPTER 4

State of the art

Music generation is a part of a wider research area that applies machine learning to arts. This includes painting classification [22], generation [23] and style transfer [24], music classification [25] and style transfer [26] and many more.

In music generation, different approaches have been investigated such as statistical models [27], genetic algorithms [28], generative adversarial networks [29], markov models [30] and recurrent neural networks, to name a few.

One of the most influential projects on recent years about music generation is the work of Oore et al. [5]. This is a research project by *Google Brain* and *DeepMind* that discusses the differences between the problems of automatically creating music scores and interpreting them, and proposes that, in fact, it is more valuable to work directly in the space of direct *performance* generation, predicting not only notes but also their expressive timing and dynamics. The authors provide results from an LSTM-based network that show great performance compared to previous work. They also evaluate the quality of the model with feedback from professional composers and musicians.

Another important and well known project on music generation is Magenta [31]. A project from Google that offers a set of pre-trained models, and the possibility of training said models with a *dataset*. Among its models we can find both WAV audio synthesis (based on WaveNet, another Google project), and generation based on MIDI files. Several of Magenta's models apply language modeling for generation of music, such as:

- Drums RNN [32]: Generates drum *tracks* with an LSTM. Uses event sequences to represent data.
- Melody RNN [33]: Generates music melody using an LSTM. Uses One-Hot encoding to represent data.
- Polyphony RNN [34]: Generates polyphonic music using an LSTM. Represents data by key words. Inspired by BachBot [35].
- Performance RNN [36]: Generates polyphonic music using an LSTM. Very similar to Polyphony RNN, but represents data as event sequences. Also inspired by BachBot.
- Pianoroll RNN-NADE [37]: Generates polyphonic music using an LSTM in combination with *Neural Autoregressive Distribution Estimator* (NADE) [38]. Represents data with pianoroll binary vectors.
- Music Transformer [39]: Generates polyphonic music using a Transformer network. Represents data with One-Hot encoding.

	Model	Texture	Representation	Reference
Oore et al.	LSTM-based	polyphony	MIDI - one-hot	[5]
Drums RNN	LSTM	persussion only	event sequences	[32]
Melody RNN	LSTM	monophony	one-hot	[33]
Polyphony RNN	LSTM	polyphony	key words	[34]
Performance RNN	LSTM	polyphony	event sequences	[36]
Pianoroll RNN-NADE	LSTM-NADE	polyphony	pianoroll binary vectors	[37]
Music Transformer	Transformer	polyphony	one-hot	[39]

Table 4.1: Details of projects on music generation with language modeling approaches

All of them use an *encoder-decoder* structure called *seq2seq*. All of them acknowledge that evaluating the results of a generative model, specially in an artistic area, is complex. Thus, they provide this tools for artists and musicians.

There are also projects that generate audio manipulating raw audio data (WAV). Examples of such projects are WaveNet [40] by Google, which uses convolutional neural networks, both to generate music as well as to recognize and generate phonemes; or GRUV [41], that uses a combination of LSTM and GRU.

In 2019, *OpenAi* [42] published a project named *MuseNet* [43], a deep neural network that can generate 4-minute musical compositions with 10 different instruments, and can combine styles from country to Mozart to the Beatles. It uses the same general-purpose unsupervised technology as GPT-2 [44], a large-scale transformer model trained to predict the next token in a sequence, whether audio or text.

These publications among others agree on the notable result of recurrent neural networks such as LSTM and GRU, usually on *seq2seq* structures. Although since the publication of '*Attention is all you need*'[21], there have been an increasing amount of projects using the transformer network [39][43] that have shown better results understanding long-term structures than those of previously discussed models.

It is more popular to manipulate data in text or MIDI format than in raw WAV, since the latter requires a more intensive training and a bigger computational cost.

When it comes to results, it is very common for these projects to present samples of the generated audio, but only in a few cases is the system available for public use, and most of the ones that do it, require technical knowledge, that not everybody has, to be able to use them. They also present comparison of *loss* results as a way of justifying the chosen model.

CHAPTER 5

Model Description

Chapter 3 contains a discussion on the most popular models used for music generation. In this work, a modified version of one of the last models presented in said chapter will be used, the Transformer. What drives the use of this model is the incredible results it has achieved in tasks of music generation compared to previous RNN models.

5.1 Model architecture

As explained before, the transformer is a seq2seq model that uses an encoder-decoder architecture implementing attention mechanisms.

The encoder is composed of a stack of 6 identical encoder layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. There is also a residual connection around each of the two sub-layers, followed by layer normalization. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce $d_{model} = 512$ dimensional outputs.

The decoder also includes 6 identical decoder layers, but in addition to the two sub-layers of the encoder layers, the decoder layers insert a third sub-layer, which performs multi-head attention over the output of the encoder stack. There is also a modification in the self-attention sub-layer for it to not attend to subsequent positions, this masking ensures

that the predictions from position i can depend only on the known outputs at positions less that i. This architecture is shown in Figure 3.13.

The Transformer used in this project, is a variation proposed by Huang et al. in December 2018 [39]. The authors proposed a new decoder-wise transformer model that incorporates a relative attention mechanism and immediately achieved state-of-art performance on music generation tasks.

This mechanism involves learning a separate relative position embedding E^r of shape (H, L, D); where H is the number of heads, and L and D the dimension $(L \times D)$ of Q, K and V matrices from vanilla attention (Equation 3.7); this E^r has an embedding for each possible pairwise distance $r = j_k - iq$ between a query and key in position i_q and j_k respectively. The embeddings are ordered from distance $-L+1$ to 0, and are learned separately for each attention head. This relative embeddings interact with queries and give rise to S^{rel}, an $L \times L$ dimensional logits matrix which modulates the attention probabilities for each head as:

$$RelativeAttention = Softmax\left(\frac{QK^T + S^{rel}}{\sqrt{D_k}}\right)V \tag{5.1}$$

This variant of relative attention was originally proposed by Shaw et al. [45] by instantiating an intermediate tensor R of shape (L, L, D) for each head, containing embeddings that corresponded to the relative distances between all keys and queries. Q was then reshaped to an $(L, 1, D)$ tensor, and $S^{rel} = QR^T$, but this incurs an space complexity of $O(L^2D)$, restricting its application to long sequences. However, Huang et al., observed that all of the terms needed for QR^T are already available if Q and E^r are multiplied. After computing QE^{rT}, its (i_q, r) entries contain the dot product of query in position i_q with embedding of relative distance r. Even so, each relative logit (i_q, j_k) in the matrix S^{rel} from Equation 5.1 should be the dot product of the query in position i_q and the embedding of relative distance $j_k - i_q$, to match up with the indexing in QK^T. Therefore, Huang et al., proposed the following *skewing* procedure, to move the relative logits to their correct positions.

5.1.1 Skewing procedure

The goal of this procedure is to transform an absolute-by-relative (i_q, r) indexed matrix into an absolute-by-absolute (i_q, j_k) indexed matrix. The row indices stay the same while

the column indices are shifted according to the following equation: $j_k = r - (l-1) + i_q$. The whole process is illustrated in 5.1. Its outline is the following:

1. Pad a dummy column vector of length L before the leftmost column.
2. Reshape the matrix to have shape $(L+1, L)$.
3. Slice that matrix to retain only the last l rows and al columns, resulting in a (L, L) matrix again, but now absolute-by-absolute indexed, which is the S^{rel} needed.

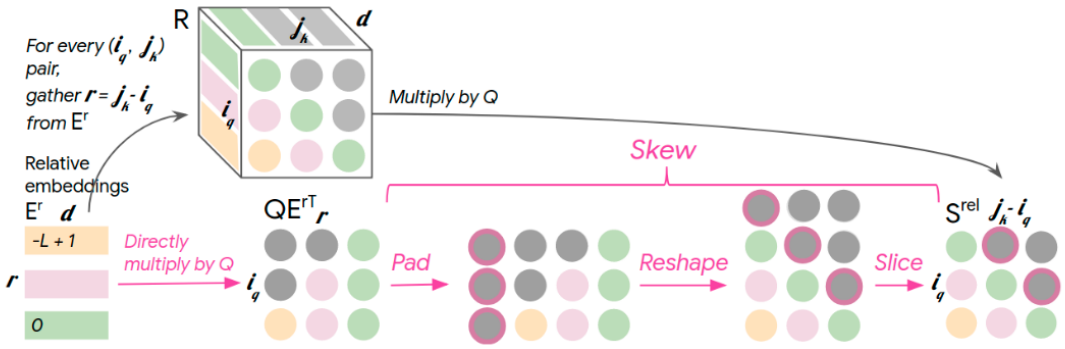

Figure 5.1: Relative global attention: Top row describes original version. Bottom row shows skewing process. Gray indicates masked or padded positions. Each color corresponds to a different relative distance [6]

5.1.2 Complexity of the model

On of the most popular ways to compare and measure the complexity of a model is looking at its trainable parameters, the more parameters a model has to train, the more complex it is.

As an example, the now very famous GPT model [46] from *OpenAI* [42] has around 117 million parameters and its second iteration, GPT-2 [44], around 1,500 million parameters. These, of course, are big-scale models, and such large quantities of parameters are out of the scope of this project.

The model in this work contains nearly 3 million trainable parameters, most of which are from the Relative Global Attention layers of the model. The detailed parameter numbers

[6] Image taken from [39]

are shown in Table 5.1. Parameters for only one Decoder Stack are shown, since all six of the have the same amount of parameters.

Layer (type)	Parameter Number
Embedding	99840
Dynamic Position Embedding	0
Dropout	0
Decoder Stack (x6)	460,672 (x6)
Relative Global Attention	394240
Dense Layer 1	32896
Dense Layer 2	33024
Layer Normalization 1	512
Layer Normalization 2	512
Attention Dropout 1	0
Attention Dropout 2	0
Output Layer	100487
Sparse Categorical Accuracy	2
Total parameters	2,967,433
Trainable parameters	2,967,431
Non-trainable parameters	2

Table 5.1: Model complexity by means of parameters

CHAPTER 6

Experiments

As discussed in Chapter 1, evaluating the performance of a neural network model in the field of music generation is a very complex task. To do so, a variety of experiments have been designed and conducted.

The objective of this work is not so much to create a model capable of generating perfect musical compositions, but rather to create a model that can generate musical compositions capable of making a human believe that they those musical pieces have been done by another human. Human brains have a tendency to be prone to overlooking certain small errors in a piece of art when evaluating its beauty. Thus, the experiments of this work will not only evaluate objective data and rules such as *harmony*, but also, and even more, will take into account the subjectivity of human musical taste and the concept of *musical beauty*.

This Chapter will be divided in three main sections: first the datasets used for the training of the model will be examined and compared and the preprocessing done to them will be described, then, the experimental framework will be discussed and the experiments will be presented, and finally, the results of said experiments will be reviewed and explained.

6.1 Dataset description

For the development of this project two datasets will be used, firstly the very well known *MAESTRO* dataset (V2.0.0), provided by *Magenta* in collaboration with the *International Piano-e-Competition*, contains a 1282 virtuous piano performances, with over a total of over 200 hours, all of it on the MIDI file format. The repertoire is mostly classical music, including composers from the 17th to early 20th century. The second database, *Classic-Piano* is taken from the `www.piano-midi.de` web-page, an archive with 329 classical pieces developed at a digital piano by means of a sequencer on MIDI base.

The MAESTRO dataset contains compositions from over 40 different authors, the majority of them being from the romantic period, with some baroque and classic period pieces as well. Table 6.1 and Figures 6.1 and 6.2 show the data in more detail.

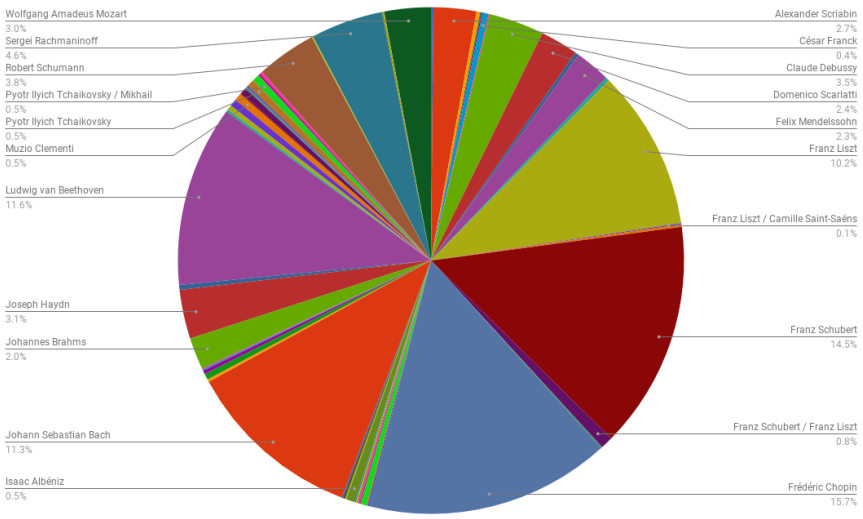

Figure 6.1: Composer chart of the MAESTRO dataset

The Classic-Piano dataset contains compositions from 25 known authors and 5 more well known compositions from unknown authors, this dataset also contains a majority of pieces from the romantic period with some classic ones and very few from the baroque period. More detail can be seen in Table 6.1 and Figures 6.3 and 6.4.

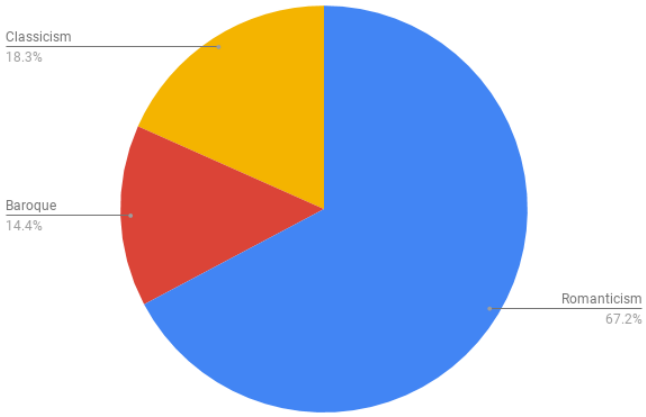

Figure 6.2: Period chart of the MAESTRO dataset

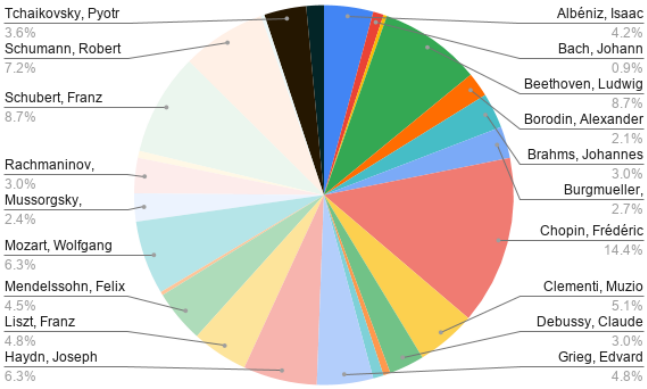

Figure 6.3: Composer chart of the Classic-Piano dataset

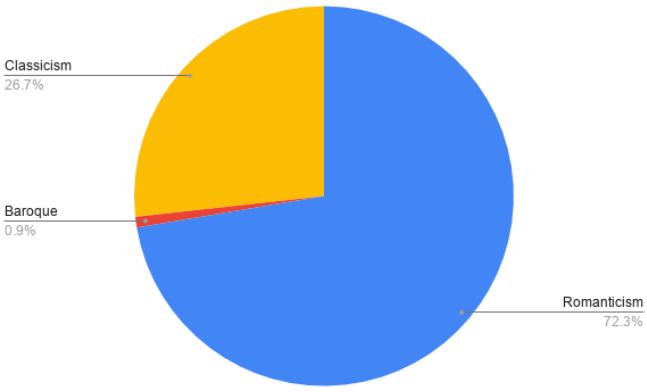

Figure 6.4: Period chart of the Classic-Piano dataset

Dataset	Authors	Baroque	Classicism	Romanticism	Total
MAESTRO	42	185	235	862	1282
Classic-Piano	25+	3	88	238	329

Table 6.1: Dataset detail comparison

One thing to note about the difference between both datasets is that the MAESTRO dataset is comprised of virtuous interpretations of the original pieces, while the classic-piano dataset is just the pieces as they were written by their original composers. This implies that, due to the added layer of complexity from the stylistic choices of interpreters, the model could have a harder time generalizing patterns on that dataset, and could lead to the posterior generated pieces mixing certain incompatible stylistic choices and, therefore, being less appealing for human listeners.

6.1.1 Data augmentation

As shown above, the Classic-Piano dataset only contains 329 compositions as opposed to the 1282 of the MAESTRO dataset, so in order to equate both datasets, a data augmentation process has been conducted on the Classic-Piano dataset. Following the industry-standard for data augmentation on music [5][39], the process was done by transposing the pieces, that is, increasing the pitch of each note on the composition. This was done three times, increasing one semitone each time, and only in the training set, resulting in 789 new pieces for a total of 1118. That way, the two datasets are much closer to one another, as shown in Table 6.2.

Dataset	Authors	Baroque	Classicism	Romanticism	Total
MAESTRO	42	185	235	862	1282
Classic-Piano	25+	9	301	808	1118

Table 6.2: Dataset detail comparison after data augmentation

6.1.2 Preprocessing

This project uses a One-Hot encoding based on the one proposed by Oore et al. [5].That is, a MIDI excerpt is represented as a sequence of events from the following vocabulary

of 388 different events:

- 128 NOTE-ON events: one for each of the 128 MIDI pitches. Each one starts a new note.
- 128 NOTE-OFF events: one for each of the 128 MIDI pitches. Each one releases a note.
- 100 TIME-SHIFT events: each one moves the time step forward by increments of 10 ms, from 10ms up to 1 second.
- 32 VELOCITY events: each one changes the velocity applied to all subsequent notes (until next velocity event).

The preprocessing is done as follows, first, the input MIDI files are preprocessed to extend note durations based on sustain pedal control events. The sustain pedal is a pedal on the piano that when pressed, "sustains" all the damped strings on the piano by moving all the dampers away from the strings and allowing them to vibrate freely; all notes played will continue to sound until the vibration naturally ceases, or until the pedal is released, in other words, it lengthens notes that are already playing. In the MIDI Standard, the sustain pedal is considered to be down whenever a sustain control change is encountered with a value $>= 64$; the sustain pedal is then considered up after a control change with a value < 64. Within a period where the sustain pedal is down, the duration of each note is extended to either the beginning of the next note of the same pitch or the end of the sustain period, whichever happens first. If the original duration extends beyond the time when the sustain pedal is down, that original duration is used instead.

Next, the MIDI note events are converted into a sequence of the previously discussed vocabulary. Finally, these sequences are stored as a binary stream to later be used by the model. This is done by means of the *Pickle* python module.

6.2 Experimental framework

In order to respond to the objectives of this project, the following experiments have been designed:

- Training and validation of the model with the Classic-Piano dataset.
- Training and validation of the model with the MAESTRO dataset.
- Training and validation of the model with both previous datasets.
- Evaluation of the best perceived model with human subjects.

6.2.1 Training experiments

Training with the Classic-Piano dataset is done to validate the capabilities of the model in music generation tasks, while training with the MAESTRO dataset is done to assess the performance of the model with the layer of complexity added by the interpretation of the pieces. Since the performance of the model with the MAESTRO dataset is expected to be lower, due to the stylistic choices of interpreters, training with both datasets is conducted to check if the model is capable of generating musical compositions with minor stylistic choices while retaining the performance obtained with the Classic-Piano dataset.

6.2.2 Experiments with human subjects

The experiment with human subjects will be done based on the model that presents better overall performance, to evaluate the appeal of the compositions generated. It will consist on rating the probability, on a scale of 1 to 5, in which a heard fragment of a piece is from a professional human composition; 1 being composed by a machine or a non professional human and 5 being composed by a professional artist. The fragments will be of about 10 seconds length, and there will be 5 human compositions from the datasets and 5 compositions generated by one of the models. Table 6.3 shows which real human pieces were used on this experiment.

Name of the piece	Compositor
Images, Book II: III. Poissons d'or	Claude Debussy
Suite Bergamasque: II and III	Claude Debussy
Sonata no. 5, Op 53	Alexander Scriabin
Fairy Tales Op. 51 No. 1, 2 and No. 3	Nikolai Medtner
Images Book 1, L 110	Claude Debussy

Table 6.3: Human composed pieces used in the experiment with human subjects

6.2.3 Experiment setup

Implementation and setup of all the experiments, excluding the one with humans, will been done in the *Python* [47] programming language in a *jupyter-notebook* [48] style environment on the *Google Colaboratory* [49] platform. The implementation is done with *TensorFlow 2.0* [50] library as a back-end for the neural network and result analysis, for the managing of data the *pretty_midi* [51] library was used to manipulate MIDI files and the *Pickle* library to store the processed MIDI as binary streams. The hardware provided by Google Colaboratory includes an NVIDIA Tesla K80, a single core hyper threaded Intel Xeon CPU @2.3Ghz (1 core, 2 threads), 13GB of RAM memory and 34GB of storage space. The most useful of this resources has, obviously, been the GPU, since it allowed for a much faster training process.

6.2.4 Training parameters

The three training experiments where done under the same conditions, with a data partition of 80% training 20% evaluation and a micro-batch size of 2, since a larger batch size always resulted in an *Out Of Memory (OOM)* error from Google Colaboratory. This implies that there were 641 iterations per epoch while training with MAESTRO dataset, 658 with the Classic-Piano dataset and 1299 with the joint dataset. Regarding epochs, initially, all models were to be trained up to 50 epochs, but, as we will discuss later, results of the first two training experiments led to the conclusion that around 25 epochs were enough to train the model and more epochs resulted only in the possibility of *overfitting*.

The *Adam optimizer* was used with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\varepsilon = 10^{-9}$. Learning rate was varied over the course of the trainings, according to Equation 6.1.

$$lrate = d_{model}^{-0.5} \cdot \min\left(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}\right) \tag{6.1}$$

This custom learning rate schedule increases linearly for the first $warmup\_steps$ training steps, and decreases thereafter proportionally to the inverse square root of the step number. The used value was $warmup\_steps = 4000$.

Dropout was applied to the output of each sub-layer of the model and to the sums of the embeddings and positional encodings. A rate of $P_{dropout} = 0.1$ was used.

6.3 Results

This section sums up the results obtained in all the experiments discussed above.

6.3.1 Classic-Piano model

During the 7 hour training process, the model achieved a loss value of around 1 and an accuracy of around 66% both in the train and evaluation sets. Figures 6.5 and 6.6 show how these values changed during the process, and how they converge at around 20 epochs into the training.

The results were considered very good since other works on this matter usually achieve a loss of around $1.5-2$ and an accuracy of around $35\%-40\%$. These extraordinary results are probably due to the dataset being an augmented version of a very small one (329 pieces only), as well as not having any stylistic choices from interpreters, which eases the learning of generalized patterns.

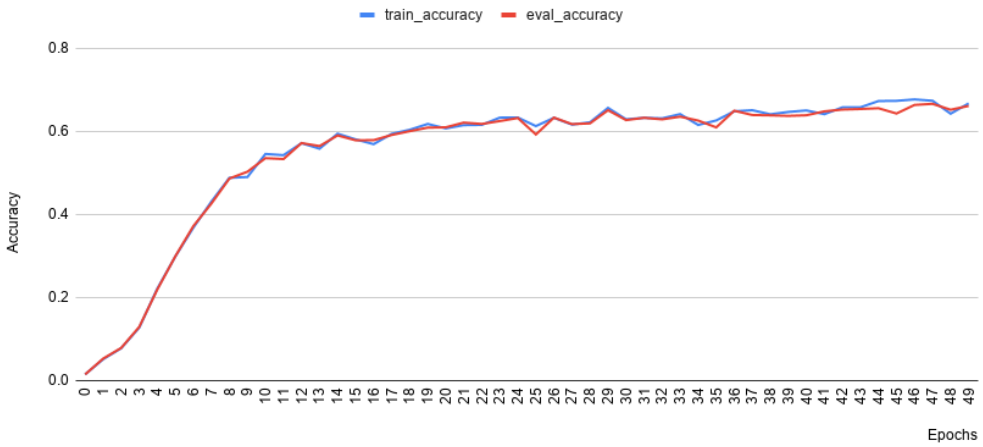

Figure 6.5: Accuracy of the Classic-Piano model

Although objective data results are very good, perceived quality of the music generated is sometimes lackluster; the model has a tendency to repeat the same note or group of notes ad infinitum, making the resultant piece not appealing after a couple of seconds of repetition. Although unpleasant, this effect has a very logical root if thought about it properly; music has a lot of repetition, there are a lot of pieces were the same note or group of notes are repeated, but always in small periods of time, not to be unappealing;

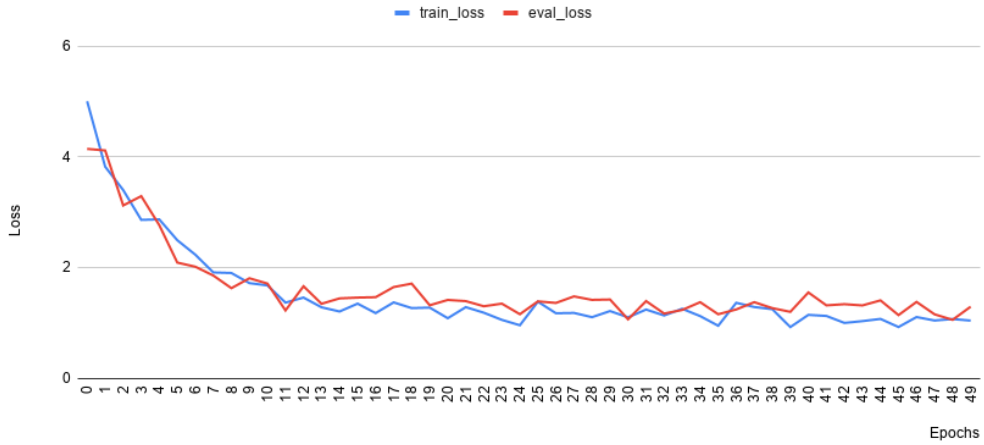

Figure 6.6: Loss of the Classic-Piano model

this could have led to the model giving note in time t more probability to appear again in time $t+1$ than any other note, which, in turn, led to infinite repetition. Apart from this problem, generated pieces that did not suffer from it were beautiful and very pleasant to listen.

6.3.2 MAESTRO model

This experiment grounded the objective data results with a loss value of around 2 and an accuracy of around 38% in both train and evaluation sets. This training process, which also lasted for 7 hours, converged at around 25 epochs, after which the improvement on both loss and accuracy was little, as shown in Figures 6.7 and 6.8.

As for the perceived results, overall this model's generated music suffers less from the repetition problem; this is due to the fact that, even if the music on this dataset also has repetitions, this dataset is virtuously interpreted by musicians, and there is a consensus that all repetitions must be different, that is, with dynamics or *articulation*; since these differences are encoded in MIDI, the model perceives them as different notes, and therefore does not get stuck on repetitions as much as the experiment above. However, this virtuous interpretation creates another issue; as the model is not able to generalize style of interpretation as mush as patterns in music, the generated pieces often show a wild variety of stylistic choices that would not make sense in a human interpretation. This leads to the generated music sometimes sounding odd and unsettling, and therefore, of worse perceived quality.

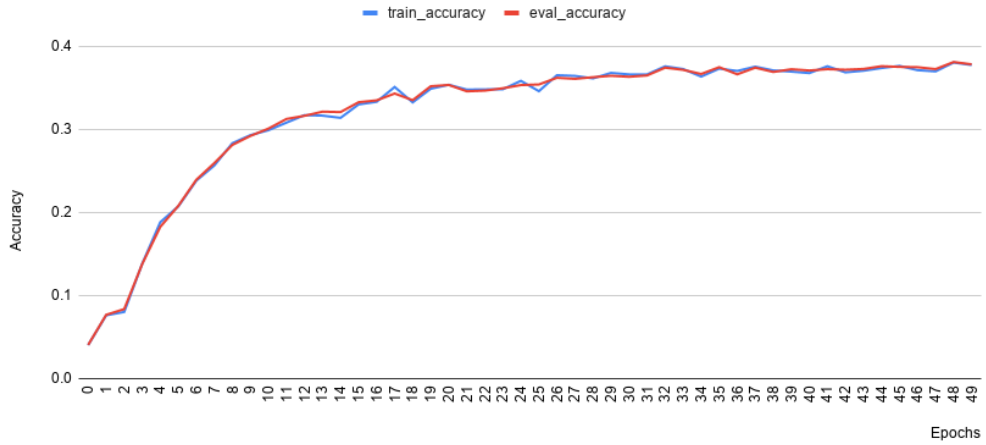

Figure 6.7: Accuracy of the MAESTRO model

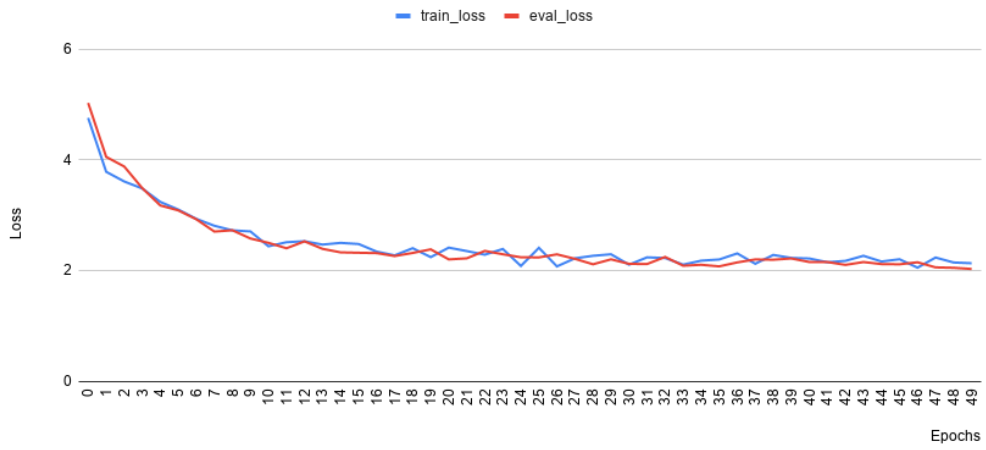

Figure 6.8: Loss of the MAESTRO model

6.3.3 Joint model

This last model was trained with 25 epochs as discussed above, since the first model converged at around 20 and the second at around 25. The results are a bit of a mixture between both previous training processes, which was expected, due to this model training on both datasets.

Accuracy reached a value of 50% in both train and evaluation sets and the loss value stayed at around 1.5 in the case of the train set and around 2 in the case of the evaluation set. Figures 6.9 and 6.10 show more detail, as well as the convergence of both values at around 15-18 epochs.

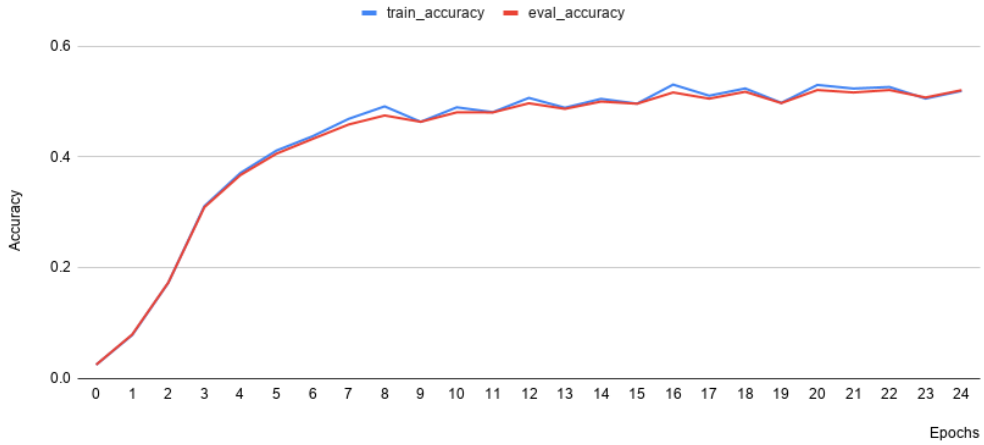

Figure 6.9: Accuracy of the Joint model

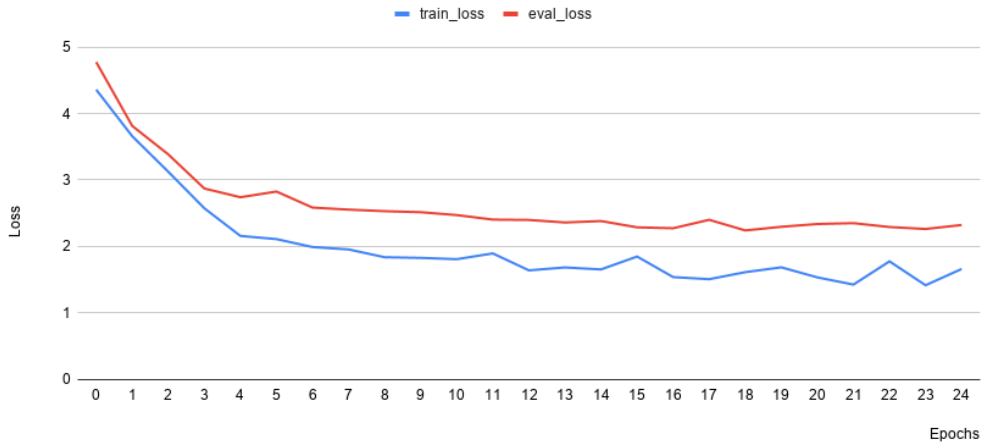

Figure 6.10: Loss of the Joint model

Note that this is the only experiment in which the loss value from the training set and the one from the evaluation set had a notable difference, this could be a side-effect of the training set having more samples of one of the dataset than the other, and the evaluation set having more samples of the other dataset.

The generated music is a meeting point of the two previous experiments. It suffers less than the first experiment from the repetition problem while not mixing too many stylistic choices at once. The overall generated music is beautiful to hear and presents music patterns as expected, which confirms that attention mechanisms are working properly.

6.3.4 Human evaluation

The model used for the human evaluation experiments is the model of the last training experiment, that is, the one trained in both datasets. This model achieved the best results from a human listening point of view, so it was the most fitting for this experiment. The experiment was done with a small set of 20 people, with different age ranges and knowledge about music.

The results were quite uplifting as most subjects found it difficult to differentiate between the fragments generated by the model and the ones from the dataset. The distribution of votes was quite similar for both as shown in Figure 6.11, with an average score of 3.08 for real compositions and 3 for generated ones. This shows how the model achieved its goal of generating realistic music that humans could not differentiate from musical compositions from human artists.

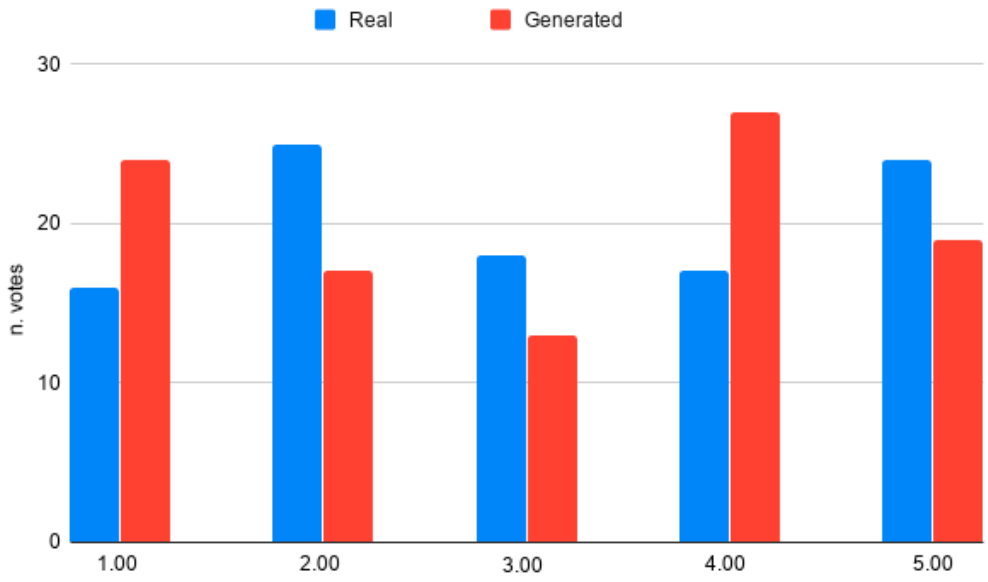

Figure 6.11: Distribution of votes on the human experiment

One thing to note is that while most subjects did not find errors in the fragments presented on the experiment, those with musical training and knowledge found that in some of the generated fragments there were conflicting dynamics like *one-note crescendos* or *one-note decrescendos* as well as sudden unusual changes from *piano* to *forte* and vice versa. These conflicting dynamics are one of the issues discussed above concerning the odd mixture of

stylistic choices that, although less than in the MAESTRO model, are still present in the Joint model.

CHAPTER 7

Web-app development

To complete this work and provide a way of testing the model without the needs of technical knowledge, a graphical user interface was developed as a web-app using the Flask [2] framework.

The interface was developed using HTML5, CSS3, Javascript, the JQuery and Bootstrap libraries and the MIDI.js library [52] to play MIDI files. The Flask framework was used to connect the client-side of the web-app with the server-side in order to utilize the model.

The web-app presents the user with the possibility of introducing a small fragment of music via a score editor and generating music based on the introduced input. When clicking the generate button, a loading page will appear while the model is generating the music, as illustrated in Figure 7.3. Finally, when the model finishes, the user will be presented with the options to play (and pause or stop) and download the MIDI file generated, this can be seen in Figures 7.4, 7.5 and 7.6. The score editor is done completely from scratch while the MIDI player is from the MIDI.js library.

As shown in Figures 7.1 and 7.2, the score editor presents the possibility to introduce several types of notes and to add accidentals on each one.

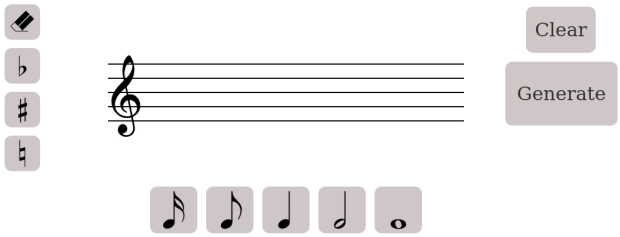

Figure 7.1: Empty main page of the web-app, the score editor

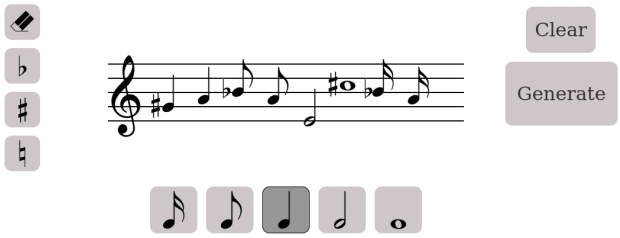

Figure 7.2: Score editor with some notes added to it

Figure 7.3: Loading page of the web-app

Figure 7.4: Web-app options for the generated music

Figure 7.5: Web-app playing the generated piece

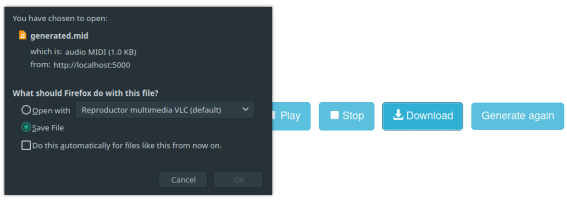

Figure 7.6: Web-app prompt to download the generated MIDI file

CHAPTER 8

Conclusions

As mentioned in the introduction, the main objective of this project was to build a deep neural network model to generate compositions and interpretations of original pieces of music and to provide a way to use it that required no technical knowledge about Computer Science. The first half of the goal was achieved on each of the three training experiments done, to a greater or lesser extent. The second objective was achieved through the development of the web-interface app which uses the result of what is considered the best of the three training experiments.

To improve the obtained results and further develop this project, not having to depend on the hardware limitations of platforms like *Google Colaboratory* would be recommended, since these limitations affected some of the training parameters, most notably the batch size, that could not be increased due to memory limitations.

Additionally, a larger dataset of music interpretations could help the model generalize the stylistic choices and improve the quality of the generated music.

Finally, a larger, more complex model like GPT-2 could outperform the one on this work, as already shown in [43], thereby, it is a direction worth to be explored.

However, all the results obtained were better than expected, since the models are overall capable of deceiving humans to think the generated music was composed by professional composers; thus, it is fair to say that all experiments and both objectives were successful.

Likewise, this work has fostered the development of certain skills necessary to fulfill the

project, such as, a substantial knowledge of machine learning and deep learning, focused on natural language processing models, particularly on the transformer network; a deep understanding on musical representations in digital format and especially on MIDI format and its structure; an introduction to web-app development with python using the Flask micro-framework; and of course, a continued development of project management and monitoring skills.

Bibliography

[1] R. Da Rios, *[Elementa harmonica]; Aristoxeni Elementa harmonica*. Officina Polygraphicae, 1954.

[2] A. Ronacher, "Flask (a python microframework)," *https://flask.palletsprojects.com (Accesed: May 2020)*, vol. 38, 2010.

[3] J.-J. Nattiez, *Music and discourse: Toward a semiology of music*. Princeton University Press, 1990.

[4] C. Wen-Chung, "Varèse: A sketch of the man and his music," *Musical Quarterly*, pp. 151–170, 1966.

[5] S. Oore, I. Simon, S. Dieleman, D. Eck, and K. Simonyan, "This time with feeling: Learning expressive musical performance," *Neural Computing and Applications*, pp. 1–13, 2018.

[6] I. M. Association *et al.*, "Midi musical instrument digital interface specification 1.0," *Los Angeles*, 1983.

[7] I. M. Association *et al.*, "Standard midi-file format spec. 1.1," *Los Angeles: The International MIDI Association*, 1990.

[8] J. S. Bach, "Prelude and fugue in c minor," in *The Well-Tempered Clavier, BMW 847*, 1722.

[9] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[10] C. M. Bishop, *Pattern Recognition and Machine Learning*, vol. 4. Springer New York, 2006.

[11] F. Provost and R. Kohavi, "Glossary of terms," *Journal of Machine Learning*, vol. 30, no. 2-3, pp. 271–274, 1998.

[12] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes," in *Advances in Neural Information Processing Systems*, pp. 841–848, 2002.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[14] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[15] J. Guo, "Backpropagation through time," *Unpubl. ms., Harbin Institute of Technology*, vol. 40, 2013.

[16] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, pp. 1310–1318, 2013.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[19] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, and K. Kavukcuoglu, "Neural machine translation in linear time," *arXiv preprint arXiv:1610.10099*, 2016.

[20] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1243–1252, JMLR. org, 2017.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[22] A. Lecoutre, B. Negrevergne, and F. Yger, "Recognizing art style automatically in painting with deep learning," in *Asian conference on machine learning*, pp. 327–342, 2017.

[23] Y.-w. Guo, J.-h. Yu, X.-d. Xu, J. Wang, and Q.-s. Peng, "Example based painting generation," *Journal of Zhejiang University-Science A*, vol. 7, no. 7, pp. 1152–1159, 2006.

[24] A. Selim, M. Elgharib, and L. Doyle, "Painting style transfer for head portraits using convolutional neural networks," *ACM Transactions on Graphics (ToG)*, vol. 35, no. 4, pp. 1–18, 2016.

[25] T. Li, M. Ogihara, and Q. Li, "A comparative study on content-based music genre classification," in *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 282–289, 2003.

[26] G. Brunner, Y. Wang, R. Wattenhofer, and S. Zhao, "Symbolic music genre transfer with cyclegan," in *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 786–793, IEEE, 2018.

[27] D. Conklin, "Music generation from statistical models," in *Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*, pp. 30–35, 2003.

[28] B. Johanson and R. Poli, *GP-music: An interactive genetic programming system for music generation with automated fitness raters*. University of Birmingham, Cognitive Science Research Centre, 1998.

[29] 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," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[30] A. Van Der Merwe and W. Schulze, "Music generation with Markov models," *IEEE MultiMedia*, vol. 18, no. 3, pp. 78–85, 2010.

[31] Google Brain Team, "Magenta." https://github.com/tensorflow/magenta. Accessed: May 2020.

[32] Google Brain Team, "Drums RNN." https://github.com/tensorflow/magenta/tree/master/magenta/models/drums_rnn. Accessed: May 2020.

[33] Google Brain Team, "Melody RNN." https://github.com/tensorflow/magenta/tree/master/magenta/models/melody_rnn. Accessed: May 2020.

[34] Google Brain Team, "Polyphony RNN." https://github.com/tensorflow/magenta/tree/master/magenta/models/polyphony_rnn. Accessed: May 2020.

[35] F. Liang, "Bachbot: Automatic composition in the style of bach chorales," *University of Cambridge*, vol. 8, pp. 19–48, 2016.

[36] Google Brain Team, "Performance RNN." https://github.com/tensorflow/magenta/tree/master/magenta/models/performance_rnn. Accessed: May 2020.

[37] Google Brain Team, "Pianoroll RNN-NADE." https://github.com/tensorflow/magenta/tree/master/magenta/models/pianorroll_rnn_nade. Accessed: May 2020.

[38] M. Abboud, B. Németh, and J. Guillemin, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," *Chem. Eur. J*, vol. 18, no. 13, pp. 3981–3991, 2012.

[39] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer," *arXiv preprint arXiv:1809.04281*, 2018.

[40] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[41] A. Nayebi and M. Vitelli, "Gruv: Algorithmic music generation using recurrent neural networks," *Course CS224D: Deep Learning for Natural Language Processing (Stanford)*, 2015.

[42] "Openai." https://openai.com. Accessed: May 2020.

[43] C. Payne, "Musenet, 2019," *URL https://openai. com/blog/musenet*, 2019.

[44] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[45] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," *arXiv preprint arXiv:1803.02155*, 2018.

[46] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

[47] G. Van Rossum *et al.*, "Python programming language.," in *USENIX Annual Technical Conference*, vol. 41, p. 36, 2007.

[48] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, *et al.*, "Jupyter notebooks-a publishing format for reproducible computational workflows.," in *ELPUB*, pp. 87–90, 2016.

[49] E. Bisong, "Google colaboratory," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pp. 59–64, Springer, 2019.

[50] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

[51] C. Raffel and D. P. Ellis, "Intuitive analysis, creation and manipulation of midi data with pretty midi," in *15th International Society for Music Information Retrieval Conference Late Breaking and Demo Papers*, pp. 84–93, 2014.

[52] M. Deal, "Midi. js-sequencing in javascript," 2015.