M.Sc. Project Investigation Report

Generating Piano Melodies In MIDI Format With Deep Learning

Ludovico Lanni

A report submitted as part of the requirements for the degree
of MSc in Data Science: Artificial Intelligence
at the School of Computing
Robert Gordon University
Aberdeen, Scotland

July 2020

Supervisor Dr. Carlos Moreno-Garcìa

Abstract

This body of work consists of the investigation report for the MSc Final Project in Data Science, titled "Generating Piano Melodies In MIDI Format With Deep Learning". Music generation using neural networks has been a significant research topic for the past 10 years, when the rapid progresses in deep learning techniques allowed for more comprehensive and accurate models. The focus of this report is on researching, summarising and analysing the theoretical and the experimental background in the context of neural music generation, with the aim of defining the content and the directions to follow in the development of this project. First of all, the basis of music theory are presented and MIDI is identified as the more structured and scalable format for music files given the limited resources available for this project. After that, the long-term capability of gated recurrent neural networks is discussed in comparison to feed-forward networks and traditional recurrent networks. As a result, LSTMs and GRUs are chosen as protagonists of the music language model that will work with MIDI-derived text sequences in experimental phase of this project. These considerations are supported by literature, as existing solutions are examined and compared in terms of efficiency and effectiveness. In conclusion, the aim of the project is to build a light end-to-end music language model based on gated recurrent cells that is able to process MIDI files in order to generate plausible music.

Acknowledgements

First of all, I would like to show my gratitude:

To my family, for the constant support and for all the possibilities they give me.

To my girlfriend, for accepting and sharing my life choices and for coloring my days even if miles away from me.

To my supervisor, for embracing my project idea and providing useful and insightful advice.

Then, I would love to dedicate this work to my grandparents. Their love and their struggle shine in my eyes and remind me how many obstacles can increase the difficulty of our life journey but, at the same time, how they can be overcome with patience, suffering and resilience.
A special mention goes to my grandmother "Mamie" Clarisse, who recently left us despite being the strongest person on Earth. Love you.

The first step is growing up and realising that life is hard. The next step is learning how to make it simpler, together.

Again, thank you.

Ludovico

Declaration

I confirm that the work contained in this MSc project report has been composed solely by myself and has not been accepted in any previous application for a degree. All sources of information have been specifically acknowledged and all verbatim extracts are distinguished by quotation marks.

Signed .....*Ludovico Lanni*..... Date ... 11-July-2020 ...

Ludovico Lanni

Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

This chapter provides an introduction to the topics covered in this report. First, a short section will state the author, the context and the purpose of this body of work. After that, the second section will expand on the motivation behind the choice of the topic. Then, a brief problem overview will be provided. Lastly, the list of all the following chapters and their content will be illustrated.

1.1 About this Thesis

This is the thesis of *Ludovico Lanni*, submitted as part of the requirements for the degree of M.Sc. Data Science at the School of Computing, Robert Gordon University, Scotland. The project focuses on the sub-field of data science which is commonly referred to as artificial intelligence, or AI.

It consists of the review, the formulation and the experimentation of mathematical and computational techniques for teaching the machine how to generate simple music melodies in such a way that human listeners would find it hard to tell whether it was composed by a human or not.

1.2 Motivation

Music is art, mathematics is science. As I grew up, I often stumbled across this widely spread paradigm describing the comparison between creativity and rationality. According to this traditional perspective, the two sides of the human nature coin are clearly separable: on the one hand the ability to create, on the other hand the ability to reason. For the past 40 years, some prestigious researchers, including the Nobel Prize owner Roger Sperry, formulated theories [1] regarding the dichotomy of the brain: while the left-side should be associated with analytical thinking and rational sciences, the right-side should be associated with qualitative thinking and arts. This idea has been on the edge between myth and fact for many years. However, recent studies [2] demonstrate the similarity in the activity of the two hemispheres in the human brain, regardless of subjects personalities - whether analytical or creative.

From a data scientist's - with an engineering background - point of view, it seems difficult to imagine such separation in our way of reasoning and the potential prevalence of one side on top of the other. An excellent problem solver is an out-of-the-box thinker who can combine quantitative and qualitative thinking in order to create, innovate and inspire. Sciences and arts are not two opposite worlds, two entities far away from each other. There can be science in art and art in science, and this is what usually happens. One could easily say that music is science, mathematics is art.

Therefore, the extremely fascinating opportunity to give a mathematical meaning to music creativity is at the core of my choice to jump on this project.

It is important to note that I am not a musician and I have never studied music theory in depth. However, I have recently developed an interest in beat-making using digital audio workstations and I have managed to produce music pieces using multiple instruments, plug-ins and controls just by playing a few hours with the software. This

demonstrates how technology and science can determine and lead a significant transformation even in those activities that are commonly associated with arts. My lack of expert knowledge in music theory should not affect the development of the project: in fact, the main goal is to let the machine learn and decide the structure of the composition it generates based on minimal information about the songs it has previously listened to.

Before going to the next section that will give a very general overview of the problem at the center of this project, I would like to give a final formulation for the comparison between music and mathematics. We started with "music is art, mathematics is science"; we then said that one could state that "music is science, mathematics is art"; now we summarise by claiming: music and mathematics are two examples of the bridge between what is commonly called art and what is commonly called science.

1.3 Problem Overview

This section gives a brief and general description of the problem that this project is meant to address. The technical concepts that are mentioned here will be explained in detail in chapter 2 and the peculiar methodology will be discussed in chapter 4.

In few words, this project aims at leveraging state-of-the-art deep learning techniques in order to teach the machine how to compose brand new piano melodies so that the result could not be easily distinguished - in terms of credibility - from human-made compositions by a sample of human listeners.

The development of such project implies the ability to solve preliminary questions and issues such as:

- What is the most appropriate format for music files?
- What dataset should be considered?
- What is the best way to mathematically represent such music files so that they can be ingested by a deep learning architecture?
- What deep learning architecture should be used for training and testing?
- What deep learning architecture should be used for generation?
- Is it possible to successfully build the model with limited computing resources, limited size of the dataset and limited time?
- What kind of credibility test should be used for generated music?

Following the investigation scheme illustrated above, the project consists of using a light deep learning model that ingests musical files and is able to compose new pieces of music based on the knowledge it has gained during training. Lastly, a sample of human listeners evaluates their perception about the origin of the compositions.

Chapters 2 and 3 provide a full study and description of the theoretical and experimental background for this problem.

1.4 Chapter List

To conclude the introductory part of this report, a very brief description of the following chapters is provided.

Chapter 2 Domain Analysis. This chapter consists of the explanation of the key-principles and techniques involved in the development of the project.

Chapter 3 Literature Review. This chapter examines the existing systems for solving problems that are similar to the one addressed in this project by highlighting their strengths, weaknesses and feasibility given my set of resources.

Chapter 4 Project Specification. This chapter draws accurate specifications for the problem, including objectives, methodology and requirements.

Chapter 5 Summary. The conclusions of the report are presented.

Chapter 2

Domain Analysis

This chapter provides the explanation and the analysis of the theoretical concepts that are involved in the development of this project. First of all, the necessary foundations of music theory, the MIDI standard and the dataset are presented in section one. After that, the second section introduces neural networks and deep learning, with a special focus on recurrent neural networks and their gated variants. Then, section three gives a contextual overview of generative deep learning systems and tools, expanding on neural language models for text data. Lastly, the ending section offers a summary of the whole chapter.

2.1 Music and the MIDI standard

The aims for this section are: providing the explanation of few music theory concepts that may help in better understanding this machine learning project; introducing the MIDI standard and presenting the MAESTRO dataset.

2.1.1 Musical Notes

Musical notes are with no doubt the most important atomic element to understand when trying to learn music theory. Basically, a note is an object associated with a specific sound, which is in turn determined by the pitch, the velocity, the duration and the timbre.

Let us start with the pitch. At the core, it is a measure for discriminating between low and high notes [3]. Although the pitch is related to the frequency of the sound wave, which is a continuous variable in physics, it is commonly described as a discrete and ordered set of steps. In order to better understand what the pitch is, we first need to understand the intervals between these steps. A semitone is the measure that indicates the distance between two adjacent discrete values, while a tone is twice this interval. Pitches are commonly organized into groups, called octaves, which are made of 12 semitones. This is graphically illustrated in Figure 2.1, that shows a sample piano keyboard with 12 semitones, i.e. 1 octave.

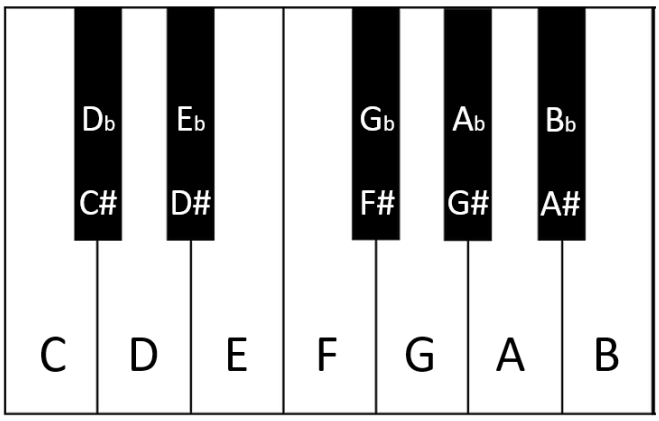

Figure 2.1: The 12 pitches of an octave displayed on a keyboard

Author: Ludovico Lanni. Original Vector by Clker-Free-Vector-Images from *Pixabay*. Available at pixabay.com. Download date: 25 June 2020

As figure 2.1 shows, the pitches associated with a standard octave are commonly named with letters going from A to G plus five special keys which can be referred to using the words *flat* or *sharp*.

In digital music standards like MIDI, a piano keyboard is virtually made of 128 different keys, with 9 sequential octaves, i.e. 128 semitones, repeating similar patterns. This can be illustrated with figure 2.2.

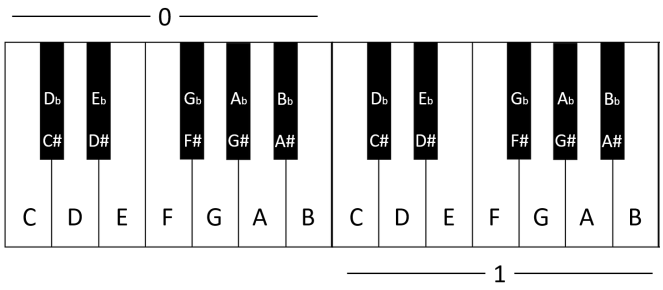

Figure 2.2: Two sequential octaves side by side on a keyboard

Author: Ludovico Lanni. Original Vector by Clker-Free-Vector-Images from *Pixabay*. Available at pixabay.com. Download date: 25 June 2020

After understanding the pitch, we now focus on velocity. This idea should be even easier to grasp. In fact, velocity is a synonym for volume, that is the intensity a certain note with a certain pitch is played with. When playing the piano, for instance, this can be associated with how hard the pianist hits a key. Theoretically speaking, while the pitch is related to the frequency of the sound wave, velocity is related to the its amplitude.

The duration of a note basically indicates how long it lasts [3]. The length of a note is usually expressed in relation to the duration of what is called the whole note. As a consequence, there can be half notes (which last half the time of whole notes), quarter notes (which last a quarter the time of whole notes), eight notes, sixteenth notes and so on. With that being said, the only duration aspect that we are interested in for this project is time measured in seconds. This is going to be explained in the next subsection when we discuss about rhythm.

Lastly, the timbre. Two notes with the same pitch, the same velocity and the same length can still be easily perceived as different. How is that possible? For example, when the two notes are played with two different instruments. This is what the timbre, or color, is about. The explanation is that each note from a musical instrument is a complex wave containing more than one frequency [3], and different combinations of these frequencies, i.e. same notes played with different instruments, are perceived as two different colors of the sound. With respect to this project, only mono-instrument compositions are considered, thus the timbre parameter is out of our scope.

2.1.2 Melody, Harmony and Rhythm

After we discussed the key-concepts regarding musical notes, we now move to consider the three building blocks of music composition: melody, harmony and rhythm.

The term melody technically indicates a set of notes played one after the other. However, people often use the same term with a more general meaning, referring to the entire song.

On the other hand, the term harmony is used when a set of notes is played simultaneously. In this case, the set of notes is called chord.

The concepts of melody and harmony fade one into the other when a chord progression is played: here the harmonic element represented by the chord is given the time component typical of the melodic element, i.e. the sequence.

The building block that takes care of time patterns in music is rhythm. To be specific, the rhythmic elements we are interested in with respect to this project are time signature and tempo. From a general point of view, musical time is organized in beats, that are in turn grouped into measures. Time signature is expressed as the ratio between two numbers, a numerator and a denominator: the first indicates how many beats are in one measure, while the second indicates what type of note equals one beat [3]. Therefore, a time signature of 4/4 indicates that a measure is made of 4 beats, that equal four quarter notes or any combination of notes that sum up to four quarter notes. It is important to note that only knowing the time signature is not enough to represent time in seconds. Tempo, in fact, defines how many beats are in one minute and, for this reason, it is usually measured in bpm (beats-per-minute). For example, the duration in minutes of a song played at 120 bpm is obtained dividing the total number of beats by the tempo, i.e. 120.

2.1.3 The MIDI standard

MIDI literally means Musical Instrument Digital Interface. It is a standard protocol that allows for communication between electronic devices capable of composing music, thus including not only musical instruments like keyboards or guitars but also computers [4].

To a certain extent, MIDI can be seen as a proper language that has its own syntax and rules. For simplicity, we are not going to discuss this language in its low-level and original representation. Instead, in order to give an overview of MIDI, we consider the representation of its messages obtained by parsing them as a *.csv* file [5]. This way, the meaning of MIDI messages is easy understandable because the low-level and bit-level

language is translated into high-level text sequences containing natural language words and numbers.

Before expanding on the midi-csv messages, it may be useful to briefly introduce the natural hierarchy of elements in a MIDI object, i.e. a music composition in MIDI format.
Intuitively, the song itself sits at the top of the hierarchy. Now, a song is made of one or more tracks, with each track being a set of notes played over time and being regulated by a set of parameters. These parameters, sitting on lower levels of the hierarchy, can refer to the specific instrument, or timbre, a chord progression should sound like when played, for instance. Musical notes are at the lowest level and are described by a set of attributes, such as pitch, velocity and duration. Note that we cited all the four elements we described as key specifications for musical notes in subsection 2.1.1.

Having that in mind, midi-csv messages can be classified into three main groups [5]:

- File structure records: they give information about the start and the end of the entire song and the single tracks.
- File meta-events: they provide meta-information about tracks - such as title, instrument name, time signature, tempo - that allows to contextualize channel events.
- Channel events: they give information about what note is played at a certain time, encoding also all the necessary attributes such as velocity and duration.

All midi-csv messages consists of at least three objects separated by a comma: *track-number*, *time*, *message-type*. When the track number is not meaningful or not definable for a certain message, it is assigned the value of zero. Same happens for time object. Then, depending on the message type, other objects are added to the line, always separated by commas. Listing 2.1 shows an example of midi-csv file containing all three kinds of events: file structure related, meta-information and channel data.

```
0, 0, Header, 1, 2, 480
1, 0, Start_track
1, 0, Title_t, "Close Encounters"
1, 0, Text_t, "Sample for MIDIcsv Distribution"
1, 0, Copyright_t, "This file is in the public domain"
1, 0, Time_signature, 4, 2, 24, 8
1, 0, Tempo, 500000
1, 0, End_track
2, 0, Start_track
2, 0, Instrument_name_t, "Church Organ"
```

```
2, 0, Program_c, 1, 19
2, 0, Note_on_c, 1, 79, 81
2, 960, Note_off_c, 1, 79, 0
2, 960, Note_on_c, 1, 81, 81
2, 1920, Note_off_c, 1, 81, 0
2, 1920, Note_on_c, 1, 77, 81
2, 2880, Note_off_c, 1, 77, 0
2, 2880, Note_on_c, 1, 65, 81
2, 3840, Note_off_c, 1, 65, 0
2, 3840, Note_on_c, 1, 72, 81
2, 4800, Note_off_c, 1, 72, 0
2, 4800, End_track
0, 0, End_of_file
```

Listing 2.1: Example of a midi-csv file.

Author: John Walker. Original example available at fourmilab.ch.
Extraction date: 25 June 2020. See also [5].

The last concept that it is worth mentioning in this subsection is piano-roll. This is a mathematical representation of MIDI notes-on and notes-off messages that basically tell what notes are played at a certain time-step and at what velocity, or volume, they are played. Therefore, a piano-roll representation consists of a 2D tensor with dimensions (`pitches`, `timesteps`) where each cell value is an integer corresponding to the velocity a note of a certain pitch is played at a certain time-step. A velocity of zero indicates that a note is not being played.
Piano-roll graphical representation is always used in digital audio workstation software tools in order to compose music clicking and dragging with the mouse (see Figure 2.3).

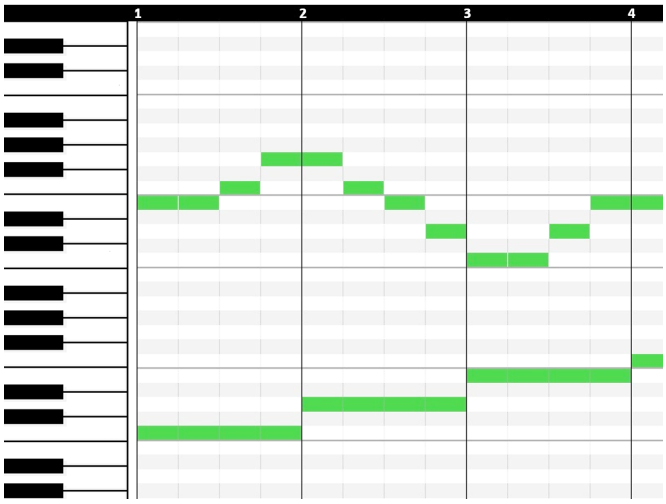

Figure 2.3: A piano-roll visual example as it appears in digital audio workstations

Author: Ludovico Lanni. Original Image by Briot et al. from *Deep Learning Techniques for Music Generation: A Survey* [6].
Available at researchgate.net. Download date: 25 June 2020.

2.1.4 The MAESTRO Dataset

Now that we have a general understanding about music key-principles and MIDI language, it is time to conceptually introduce the dataset used in this project. The MAESTRO dataset, namely MIDI and Audio Edited for Synchronous TRacks and Organization, is a large collection of classical music piano performances captured with fine alignment between note labels and audio wave forms [7].

The dataset has been built by Google AI team, working on the project called *Magenta*[1], in collaboration with the International e-Piano Competition[2]. Although the potentiality of the MAESTRO dataset is enormous due to the fine alignment between MIDI messages and audio waves, this project only focuses on downloading the MIDI compositions from `version 2.0.0` in order to use them in a machine learning framework. The entire dataset consists of more than 200 hours of mono-instrument and polyphonic compositions in MIDI format, for a total of more than 7 billion notes. Given the massive quantity of data and the limited resources for this project, only a subset of MAESTRO will be considered.

[1] For more information visit the following link: magenta.tensorflow.org
[2] For more information visit the following link: piano-e-competition.com

2.2 Deep Learning

Chollet (2017) [8] identifies two main orientations in the field of artificial intelligence: *symbolic AI* and *Machine Learning*. In order to explain the difference, let us consider three key-elements: questions, answers and rules (link between questions and answers). On the one hand there is *symbolic AI*, which focuses on providing questions and rules to the machine so that it can elaborate them and come up with the answers. On the other hand, the much more interesting approach of *Machine Learning* consists of providing questions and answers - when applicable - to the machine so that it can process them and discover the hidden rules. In other words, the focus of machine learning is not on hard-coding a finite set of rules in order to compute the output given a specific input, but on letting the intelligent system learn these hidden rules from examples.

This project, however, focuses only on a peculiar set of machine learning tools belonging to the scientific scope of deep learning. Thus, how does deep learning differentiate from the rest of machine learning techniques? This section will provide the answer for this question, expanding on neural networks and the relevant neural architectures with respect to the scope of this project.

2.2.1 Neural Networks

All machine learning systems share the same goal: learning a mathematical representation for input data that can explain in the most efficient and effective way the relation with output data. Once this representation has been learnt, it can be applied to new input data, of which both answers and rules are not known (see Section 2.2). Given what has been said, different machine learning systems use different representations and different ways to learn them.

The key concept that differentiate deep learning from other machine learning fields is what is called layered representation [8]. In other words, a number of representative layers - the larger this number, the "deeper" the learning mechanism - is organized into a mathematical architecture that defines the flow of data from input to output: each layer is responsible for a functional transformation, and the whole hierarchy of layers progressively allows for a more meaningful and accurate representation. The mathematical layered architecture we mentioned is called neural network. The term "network" is easily understandable: a layer can be connected to one or more layers, drawing a network-shaped structure. The term "neural" most likely refers to the analogy with how the human brain works. However, while the way our brain works is still unclear, what we know is that every neural network is simply a set of mathematical operations.

With that in mind, let us briefly summarise the structure and the key concepts of a neural architecture.

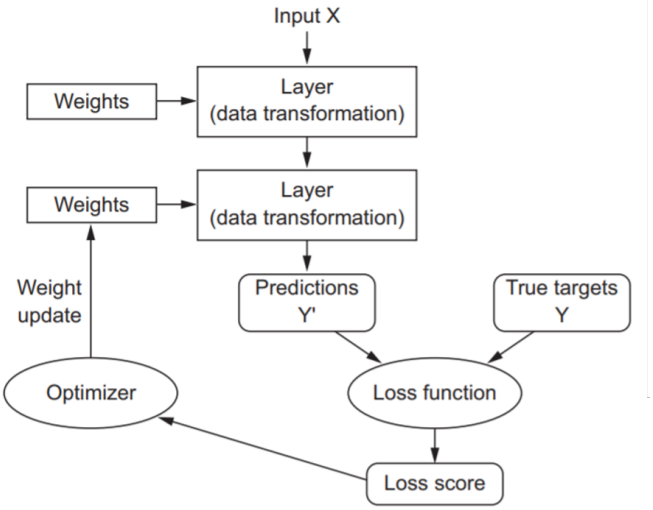

Figure 2.4: The main elements of a simple neural network architecture.

Author: Francois Chollet. Original image by Chollet from *Deep Learning With Python* [8]. Figure 1.9. Available at faculty.neu.edu.cn. Download date: 26 June 2020.

Figure 2.4 illustrates the most important elements in a simple neural network, made of two layers and suitable for a supervised learning task. How does it work?
First of all, input data in form of a tensor - i.e. an n-dimensional vector - are sent to the first layer, where a mathematical operation computes a defined transformation. This operation usually consists in linear combinations of the input features using a *weight matrix* and a *bias vector*, followed by an activation function that maps the results of the previous operation to the desired range of values. Note that, once the type of tensor operation and the type of activation have been fixed, the weights and biases can change their value according to an optimization criterion, which we will discuss in a moment. After that, the output of the first layer becomes the input of the second layer, which in this case works in the exact same way as discussed above. Of course, a new weight matrix and new biases will be used. In this very basic, or "shallow", example the second layer is the last layer, therefore it is in charge of shaping its output tensor, i.e. the predictions of the network, in such a way that can be compared with the true targets. At this point, a feed-forward iteration is completed: data crossed the entire network and predictions are computed. However, these have been calculated using totally random weights and biases and, for this reason, they are far from being an expression of the network potential knowledge. Here comes the core of the learning mechanism.
A differentiable *loss function* computes a distance measure between the predictions and

the ground truth: in other words, this is a way of tracking the quality of the network predictive power. The differentiable property of this measure is leveraged by a gradient-based *optimization algorithm* to update weights and biases in such a way that allows for a reduction of the loss value, meaning that predictions will hopefully be closer to the true targets. This optimization system generates an information flow that goes from the output back into the input and, for this reason, it is called *backpropagation* [9]. Then, new feed-forward and backpropagation iterations with updated weights and biases are performed until the knowledge gained by the network meets - when possible - the objectives. We refer to each of these iterations with the term *epoch*.
Figure 2.5 illustrates an example of a simple neural network and shows how each layer is made of several computing neurons.

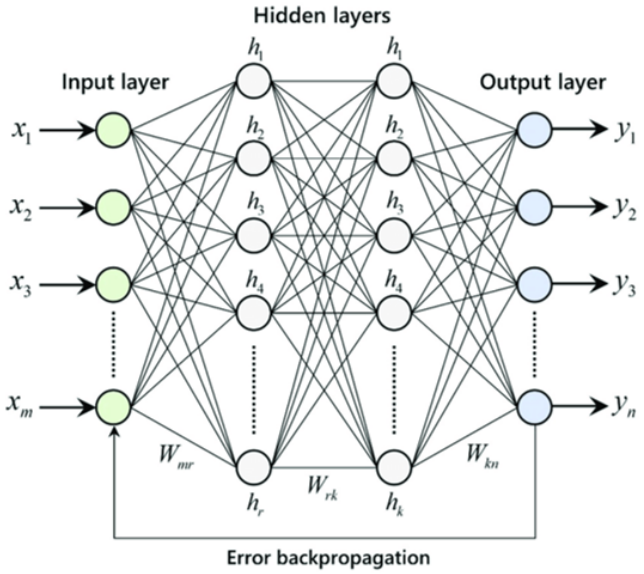

Figure 2.5: An example of neural architecture with two hidden layers.

Author: Fernandez-Caban et al. Original image by Fernandez-Caban et al. from *Predicting Roof Pressures on a Low-Rise Structure From Freestream Turbulence Using Artificial Neural Networks* [10]. Figure 4. Available at researchgate.net. Download date: 26 June 2020.

What we have discussed so far are the foundations of neural networks. Starting from this concepts, a large number of more sophisticated neural architecture have been proposed over the years. The differences among these, for instance, can be in the number of layers, the type of layers, the connections between layers, the optimization algorithm, the loss function, and so on. Some architectures have been found to be more effective for specific kinds of input data. Convolutional neural networks (CNN), for instance, are the state-of-the-art technology behind computer vision tasks, thus working with

images and videos. However, this project mainly deals with input data modelled as text sequences: here the most frequently used systems over the past few years have been recurrent neural networks (RNN). Therefore, the following subsections will expand on this kind of neural architecture, discussing both vanilla RNN and gated RNN.

2.2.2 Recurrent Neural Networks

Recurrent neural networks derive from a conceptual modification of the classic feed-forward networks - like the one illustrated in Figure 2.5 - to address its incapability of incorporating the time element into the learning process. In 1986, Jordan [11] formally defined the structural differences between a non-recurrent network and a recurrent network: while the first has no cycles in its architecture, a recurrent network allows information to flow not only from one layer to the other, but also along a feedback connection from one layer back to the layer itself. These recurrent layers transform the learning process of a typical feed-forward network from static, i.e. unaware of the chronological dimension, to dynamic, i.e. aware of the chronological dimension. This is a massive change in perspective: in fact, recurrent neural networks can build their own memory object that allows for learning future targets based not only on future inputs, but also on the knowledge gained from chronologically ordered past inputs.

How does this all translates into technical and structural modifications?
First of all, let us inspect the structure of a recurrent layer as illustrated in Figure 2.6.

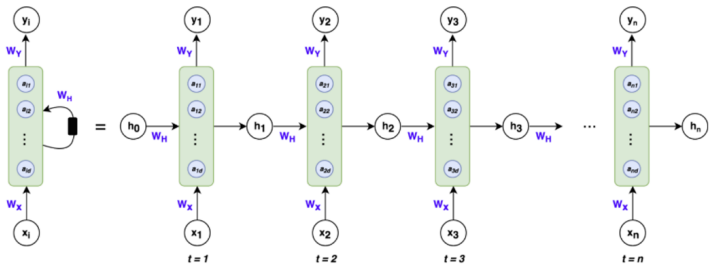

Figure 2.6: A recurrent layer in its compact (left) and unrolled (right) representations.
Author: Ben Khuong. Original image by Ben Khuong. from *The Basics of Recurrent Neural Networks (RNNs)*. Blog post.
Available at medium.com. Download date: 27 June 2020.

In its compact representation, a recurrent layer is simply a set of neurons with feedback connections that point to the layer itself. However, in order to fully understand the meaning this concept, it is necessary to discuss the actual structure of the recurrent layer, which is visualized by unrolling the compact notation. Here we can see that:

- A recurrent layer is actually a succession of multiple layers that share the same learning parameters and the same activation function. For this reason, it is better to use the term "recurrent cell" instead of recurrent layer.
- Each layer belonging to the recurrent cell ingests a piece of input that refers to a specific time frame. In other words, layer `t` processes input at time `t`, layer `t + 1` processes input at time `t + 1`, and so on.
- While in a feed-forward layer the output is a function of the input alone, in a recurrent layer the output is a function of the input and the *hidden state* computed so far in the layers chain. The hidden state ingested by a layer `t` is simply the weight matrix learnt from the previous layer `t − 1`. This is the reason for the term "recurrent": the output of a certain layer is computed using a recurrent relation, which is an equation that defines an array in such a way that a term of the sequence its computed as a function of the preceding terms.
- The learning process of a recurrent cell leads to the generation of multiple outputs, ordered in a chronological sequence that mirrors the input sequence. However, due to the recurrent relation, the last output of the sequence contains the knowledge of all the past layers.

Now that we know the structure of a recurrent cell, one may ask whether there are some modifications even in the optimization procedure for updating weights and, therefore, learning. Subsection 2.2.1 introduced the backpropagation system, which leverages gradient-based optimization algorithms to propagate weight updating back to the input layer of the network. In the case of neural networks with recurrent cell, a variation of backpropagation is used in order to deal with the time component: this algorithm is commonly called *backpropagation-through-time* [12][13][14] and the key idea behind is, as we have seen so far, unrolling the recurrent cell as a sequence of layers and computing the error for each time step. However, training a vanilla, or basic, recurrent neural network in an efficient and effective way can become a very hard task and several modifications of the standard recurrent cell have been proposed to overcome these issues. Next subsection expands on this topic, introducing the key concepts behind LSTM (Long-Short-Term-Memory) cells and GRU (Gated Recurrent Unit) cells.

2.2.3 Gated Recurrent Neural Networks

The reason why training vanilla recurrent neural networks can be tricky is that they are particularly exposed to the so called vanishing gradient problem. This is a concern that theoretically affects all kinds of neural networks, because it is related to the gradient-based optimization algorithms that all of them use. In few words, we know

that gradients of the loss function are used to update weights in such a way that the learning process hopefully goes in the right direction. We also know that this mechanism is called backpropagation because information flows from the last layer to the first layer of the network. Now, given that traditional activation functions shrink the output values of a layer to very small numbers, usually between 0 and 1 (`sigmoid` activation function), and given that gradients are computed using the chain rule, the earlier the layer the smaller the gradients associated with its weights will be because of the result of a multiplication of already small numbers. Therefore, the deeper the network, the more the gradient tends to vanish. But why is this a problem? Well, we know that the learning process of a neural network is all about updating weights in an effective way to find the optimal values for minimizing the loss function. Therefore, if the gradients are so small that the weights cannot be updated, the network stops learning.
What happens with recurrent networks is that this problem is significantly amplified because of the backpropagation-through-time, which consists of unfolding a recurrent cell as a very deep sequence of layers. In this context, it is often said that vanilla recurrent neural networks have short-term memory, meaning that the network find it hard to update the weights of neurons that are relatively far from the last neuron in the recurrent sequence.

Among the solutions that have been proposed over the years, two of them have become the state-of-the-art in recurrent neural networks for sequence processing. They consists of replacing a standard recurrent cell with a gated recurrent cell, where the gates are neural architectures that allows for the possibility of learning at each time step which past information should be kept for future neurons and which, instead, should be forgotten. These special cells are the Long-Short-Term-Memory (LSTM) cell [15] and the Gated Recurrent Unit (GRU) cell [16].

2.2.4 LSTM and GRU cells

Let us discuss the key concepts of LSTM and GRU with respect to their differences with a standard RNN cell from a qualitative perspective. The goal here is trying to grasp the ideas behind these novel approaches without going too much into the technical details.
We have seen in previous subsections that a recurrent cell can be seen as a very deep sequence of feed-forward layers that keep track of the state of the system. This is particularly valid for traditional recurrent cells and Figure 2.7 shows how each cell can be illustrated as a repeating module made of just one layer of neurons that uses `tanh` as activation function.

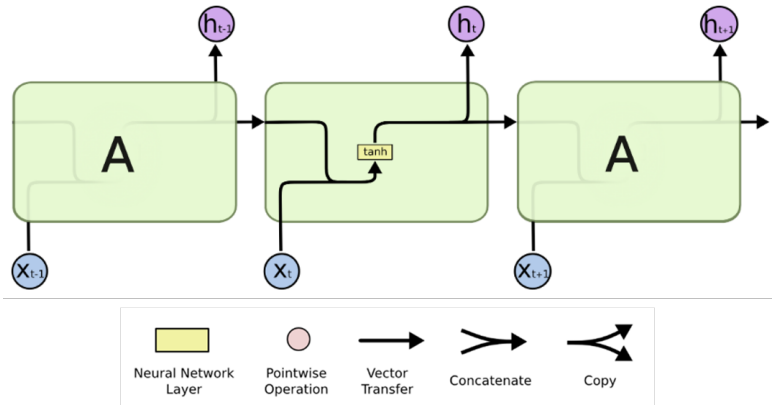

Figure 2.7: Structure of a module in a vanilla RNN cell

Author: Christopher Olah. Original image by Christopher Olah. from *Understanding LSTM Networks*. Blog post.
Available at colah.github.io. Download date: 28 June 2020.

On the contrary, the structure of a single LSTM module is much more complicated, as illustrated in Figure 2.8. In fact, each module is associated with four layers instead of a single one. The reason is that, other than the `tanh` layer, there are three information gates modelled as layers that use the `sigmoid` function for activation. Given that the `sigmoid` function outputs values between 0 and 1, the qualitative interpretation is that the the closer to zero the output is, the smaller the amount of information flows through the gate. In addition, there is a second memory object other than the hidden state: that is the *cell state*, which acts as a conveyor belt that carries information wherever the network needs it [8]. All three gates interact directly or indirectly with the cell state and each of them has a specific task. Simplifying: the *forget* gate filters the information coming from the previous cell state and learns what pieces of information should be kept at that step; the *input gate* filters the information coming from the `tanh` layer and learns what parts should be added to the cell state; the *output gate* filters the information coming from the previous hidden state and learns what should be the next hidden state.

These features significantly increase the complexity of the network, but allow to solve the short-term memory problem due to the vanishing gradient. As a consequence, a recurrent neural network with LSTM cells has great capability of learning longer and richer temporal patterns and it seems the way to go with respect to the implementation of this project.

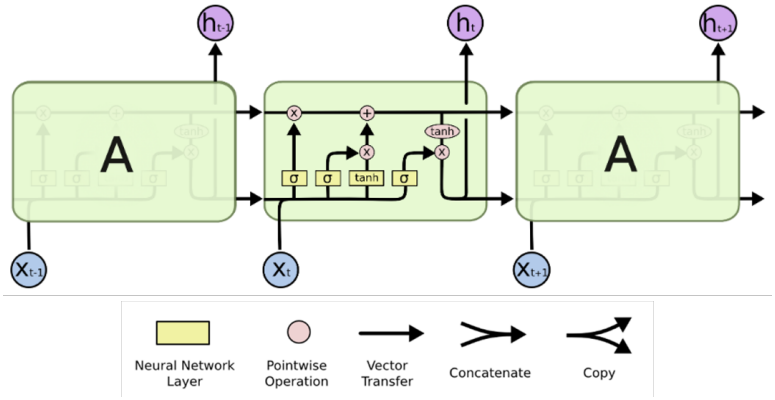

Figure 2.8: Structure of a module in an LSTM cell

Author: Christopher Olah. Original image by Christopher Olah. from *Understanding LSTM Networks*. Blog post.
Available at colah.github.io. Download date: 28 June 2020.

Given what has been discussed about LSTM-based networks, it should be easier to understand the structure of a Gated Recurrent Unit, or GRU. As Figure 2.9 shows, also GRU cells use additional layers of neurons acting as information gates. However, there are two key differences from LSTM cells:

1. GRU does not utilize a cell state because all the information flows through the hidden state.

2. There are only two gates: the *reset gate* and the *update gate*.

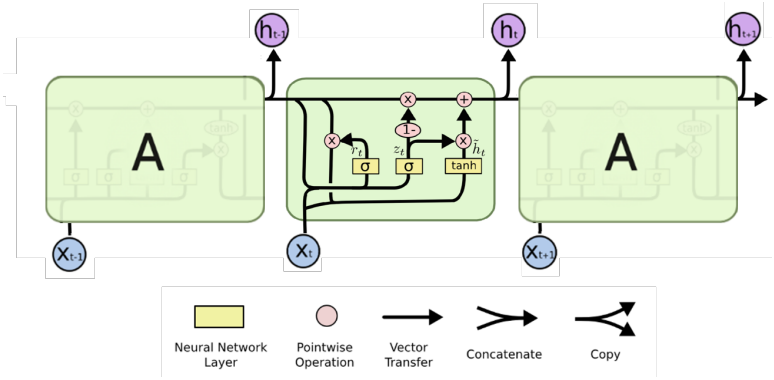

Figure 2.9: Structure of a module in a GRU cell

Author: Ludovico Lanni. Original images by
Rick Anderson. from *RNN, Talking about Gated Recurrent Unit*. Blog post.
Available at technopremium.com. Download date: 28 June 2020.
and by
Christopher Olah. from *Understanding LSTM Networks*. Blog post.
Available at colah.github.io. Download date: 28 June 2020.

As a result, the computational needs of a GRU cell are less demanding than what LSTM cells require. It is not a surprise that the GRU was introduced quite few years later than the LSTM unit, with the aim of simplifying the neural architecture and speed up the training process.
Overall, there are cases in which LSTMs, despite their longer training time, perform better than GRUs (e.g. [17]) and cases in which GRUs provide better results with less resources (e.g. [18]). Therefore, this project plans to try both gated recurrent neural architectures to empirically evaluate their performances, with the assumption that they would outperform a vanilla RNN.

2.3 Generative Deep Learning

At this point of the chapter, the fundamentals of music theory and MIDI standard have been presented and the gated recurrent neural networks have been selected as the underlying technology for the development of the project. However, we still need to know how to draw the neural architecture in such a way that the model can not only learn patterns in MIDI sequences, i.e. discriminate, but also output arbitrary patterns based on what it has learnt, i.e. generate. In other words, this section aims at briefly introducing the world of generative deep learning and focuses on a special kind of models used to teach machines how to write text in a certain language, which in our case consists of a special encoding of MIDI messages.

Generally speaking, machine learning is all about learning mathematical representation of data. Now, these representation can be seen as a latent space where the machine stores all the knowledge it has gained. For example, let us think about the task of trying to compress the information of an image with a total of `N` pixels into a set of `K` $<<$ `N` features. This is an unsupervised task, meaning that there are no external targets. For these reason, a good choice of simple architecture would be a feed-forward network that takes the `N` pixels as input, then compress it with a layer of `K` neurons, and finally send it to a layer of `N` neurons, where the goal is to learn the optimal combination of weights so that the output values of the network are as more similar as possible to the input data. This architecture where there is an encoder that squeezes data through a bottleneck and then tries to reconstruct the input using a decoder is technically called *autoencoder* (Figure 2.10).

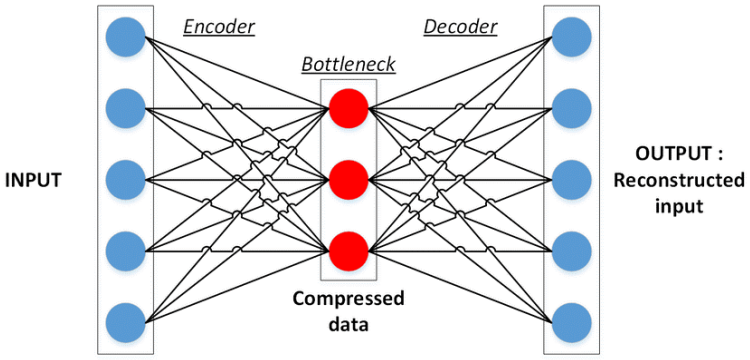

Figure 2.10: Example of a basic neural autoencoder

Author: Jeremie Sublime. Original image by Jeremie Sublime. from *Automatic Post-Disaster Damage Mapping Using Deep-Learning Techniques for Change Detection: Case Study of the Tohoku Tsunami* . See also [19].
Available at researchgate.net. Download date: 29 June 2020.

The idea behind autoencoders is that once the training process has successfully ended, the hidden bottleneck layer, i.e. the latent space, contains the encoding of the knowledge about the input. The advantage is that one may try to interpret these K-dimensional features and leverage the reduced dimensionality of the input for further computation.
This architecture, however, is not yet generative. The *variational autoencoders* (VAE) [20][21], in fact, utilize the autoencoder fundamental encoder-decoder idea with the novel concept of sampling from the probabilistic distribution of the hidden features in the latent space and generate output data that are sensible to their position in the latent space. Surprising results can be achieved with this technology, such as generating human faces that can change their facial expression according to the values of a feature vector, e.g. the smile vector (Figure 2.11).

Figure 2.11: Example of VAE-generated human faces that move along a smile vector.

Author: Francois Chollet. Original image by Chollet from *Deep Learning With Python* [8]. Figure 8.11. Available at faculty.neu.edu.cn. Download date: 29 June 2020.

Another remarkable generative deep architecture that has been particularly good at working with images consists of the generative adversarial networks (GANs) [22]. They combine a generative network with a discriminative network for letting the generator learn how to produce new data in such a way that the discriminator is not capable of distinguishing the nature of original and artificial data (see Figure 2.12).

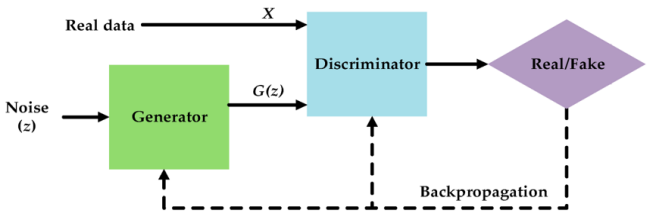

Figure 2.12: The basic structure of a GAN architecture.

Author: Jie Feng et al. Original image by Jie Feng et al. from *Generative Adversarial Networks Based on Collaborative Learning and Attention Mechanism for Hyperspectral Image Classification* [23]. Figure 1. Available at www.mdpi.com. Download date: 7 July 2020.

Having said that, these architectures have significantly high complexity and may lead to considerable computational risks. In addition, considering the objectives of this project and the knowledge and the resources that I currently have, it seems unfeasible to apply VAEs or GANs for the implementation of the neural music composer. As mentioned in Chapter 1, in fact, the idea is to model the music generation task as a text generation task by encoding MIDI messages in a language-alike sequence of characters. In this context, the generative architecture that learns how to progressively produce text from a certain language is called neural language model and most of the times uses gated recurrent neural networks.

2.3.1 Neural Language Models

A language model aims at capturing the statistical nature of a language [8] in order to generate sequential and meaningful text in that particular language. Once again, as we discussed in Section 2.3, the way to do it is learning the latent space of the mathematical representation and then sampling from it to generate data. A neural language model is a language model that leverages distributed representations or, in other words, neural networks [24].

How does a neural language model work? The architecture usually consists of a recurrent neural network that spends the training time learning to predict the next token - e.g. a word or a character - in a sequence using previous tokens as input. After it has computed a representation for the latent space, the network can ingest a seed sequence,

predict the next token by sampling from the latent space, then add the predicted output to the input sequence and iteratively generate a sequence of arbitrary length. Such model is said to be character-level when each token corresponds to a character while is word-level when each token corresponds to a word. Figure 2.13 illustrates this loop method for an example of character-based neural language model.

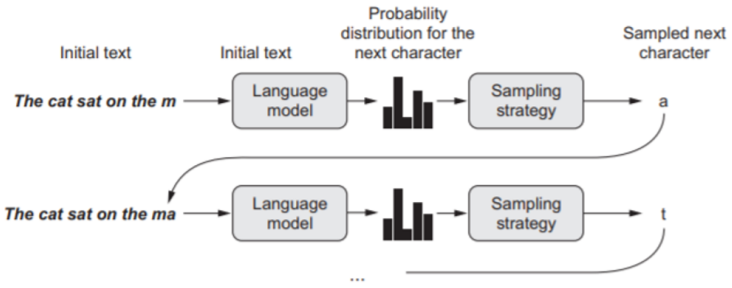

Figure 2.13: Example of a character-level neural language model.

Author: Francois Chollet. Original image by Chollet from *Deep Learning With Python* [8]. Figure 8.1. Available at faculty.neu.edu.cn. Download date: 29 June 2020.

The choice of using words or characters as tokens does not follow defined rules. In fact, as illustrated in Table 2.1, each of these methods has some advantages and disadvantages in terms of performance due to the different dimensional approach that they have.

Feature	**Char-Level**	**Word-Level**
Vocabulary size	Small	Large
Number of tokens in a sequence	Large	Small

Table 2.1: Dimensional trade-off in character-level and word-level language models

On the one hand, char-level models reduce the size of the vocabulary, which is the number of unique tokens in the dataset, when compared to word-level models. This is an advantage in terms of training time because the dimensionality of the input is smaller, thus less parameters have to be learnt. On the other hand, breaking down text sequences into characters instead of words leads to a larger number of time steps for representing the same sequence, which may force to increase the size of the network for effectively learning long-term dependencies.

At this stage, the plan for this project is to try to use both variants and evaluate which one generates text that can be converted to the most reliable music.

As a reminder, although neural language modeling techniques have been developed and applied mainly for natural language processing tasks, they can work with any type of text sequences. For this reason, once music is parsed as MIDI messages and these are converted into text sequences using a special encoding, neural language models can be implemented to generate new text, i.e. new music.

2.4 Conclusions

The Domain Analysis chapter has extensively discussed the theoretical background for this project. In particular:

- Music foundations have been explained in order to understand the MIDI language, that is the format in which files are extracted from the MAESTRO dataset and then processed in the experimental report.
- Deep learning theory has been covered as a gentle introduction to recurrent neural networks and their gated variants (LSTM, GRU), which allow for training neural architectures on data sequences with longer memory of the past.
- The world of generative deep learning has been introduced with a special focus on neural language models, which allows for predicting the next token in a sequence, usually using gated recurrent cells.

To summarise, this project aims at leveraging neural language models based on recurrent cells for working with text sequences of processed MIDI files in order to generate music pieces that sound like human composed.

While this chapter focused on the theoretical background, the next chapter focuses on the experimental background for this project, examining and analysing the existing practical solutions that have been proposed for solving a similar problem.

Chapter 3

Literature Review

This chapter provides an informative and critical discussion on some of the solutions that have been proposed over the years for solving problems that are similar to the object of this project, i.e. neural music generation.
In detail, the content is organized in three sections, each approaching related work from a different perspective:

- Section 3.1 focuses on the choice of the mathematical representation to use as input for the music neural language model.
- Section 3.2 examines and compares the proposals of different neural architectures for building the music language model.
- Section 3.3 expands on the cost aspect behind the above presented solutions in terms of the resources that need to be allocated for the training process, given the resources in time and tools that are available for this project.
- Section 3.4 provides the conclusion for this chapter.

3.1 Mathematical Representation of Music

The first choice to tackle in order to address the task of generating music with deep learning architectures is related to the possible ways of representing music files in mathematical terms. This is a crucial step, because the nature and the dimensionality of the input affect the whole development of the project, including the choice of a specific neural architecture.

A vast portion of research, specially in the early 21st century, has approached neural composition starting from symbolic kinds of music files, as opposed to raw audio. Boulanger-Lewandowski et al. (2012) [25], for instance, considered a variety of symbolic musical datasets, such as ABC notation files and MIDI files. However, these formats are not immediately ready to enter the neural architecture (see also Subsection 2.2.1): the authors, in fact, built a binary piano-roll (see also Figure 2.3) representation from the original files. Therefore, due to the binary nature of the matrix, although this representation can encode polyphony, i.e. harmony, it does not manage to keep track of the velocity information. Huang et al. (2016) [26] continued the work on symbolic music by using the piano-roll dataset constructed by Boulanger-Lewandowski et al. and by proposing an alternative representation, consisting in encoding both the raw MIDI messages and the piano-roll data as sequences of tokens. This approach can enrich the amount of information used in the experiments but may lead to dimensionality issues due to vocabulary size and sequences length. The authors encoded each unique combination of maximum three simultaneously played notes across all timesteps as a single token. As a consequence, the approach is strictly constrained to the unique combination of notes seen by the network and to three-dimensional harmonies. Agarwala et al. (2017) [27] chose to focus not on the MIDI format but instead on ABC notation. Although this is theoretically different from working with MIDI files, the authors represented the data in a similar manner to what Huang et al. did, by encoding music as a textual sequence. However, in this representation each note has a different symbol even if played in an harmonic chord: this is the basis for a much more creative approach to music generation, where the ability of the neural composer is not restricted to the chords it has previously seen. Hilscher et al. (2018) [28] also proposed a very similar textual representation starting from MIDI files instead of ABC notation files. Finally, Payne et al. (2019) [29] created MuseNet, a neural composer that is capable of emulating style features, starting from several MIDI collections, including the MAESTRO dataset (see also Subsection 2.1.4). The encoding that they proposed is once again a sequence of text, where the pitch, the volume, and the instrument information of a note is represented as a single token.

A parallel research path has been dedicating its effort to the development of neural

music generators by working with raw audio instead of MIDI or symbolic languages. For example, Nayebi et al. (2015) [30] and Kalingeri et al. (2016) [31] mathematically represented audio waveforms working in the frequency domain by using Discrete Fourier Transform. Also JukeboxAI, by far the best and most recent multi-style-aware neural composer introduced by OpenAI with a paper by Dhariwal et al. (2020) [32], ingests input music as raw audio.

On the one hand, the advantage of working with raw audio is that all kinds of music files can be used as training data, regardless of the genre, the number of instruments, the rhythm pattern and so on. This approach is totally unconstrained, but unleashes its higher potential when data are labelled with meta-information that allows to build a feature-based latent space, making technologies like JukeboxAI something futuristic and exceptional. On the other hand, working effectively with raw audio involves higher computational costs, specially in terms of memory, and much bigger models to deal with the highly unconstrained approach.
On the contrary, MIDI language can be seen as a much more structured and simple way of looking at music data. In addition, it is totally scalable in terms of complexity, meaning that the computational risk is significantly reduced. In other words, working with MIDI files allows for focusing on simplified music patterns, such as those produced by a single instrument, only considering the notes and their main features, such as pitch, velocity and duration.

Given what has been said, this project considers input data in MIDI format and aims at representing it as a sequence of tokens where notes can be discriminated one from each other.

3.2 Neural Architectures for Music Generation

After the inspection of the possible formats and mathematical representations for music files, the second crucial aspect to deal with is the choice of the generative neural architecture.

With respect to this topic, two main trends can clearly be identified across the research papers that we mentioned in the previous section, regardless of the differences in the input format that they consider.

1. The first trend is related to the use of recurrent neural networks and gated recurrent neural networks for building a many-to-one language model.

2. The second trend, much more recent than the first, is related to the use of transformers[1] that use sequence to sequence models.

The most remarkable results in terms of quality and usability of the artificially generated music were obtained in the past two years using transformers, after they had been introduced by Vaswani et al. [33] only in 2017. MuseNet (2019) and JukeboxAI (2020), for instance, are both based on complex transformer architectures and trained using networks with millions (sometimes billions) of parameters. One year after the introduction of transformers, Huang et al. (2018) [34] set the basis for the two above mentioned neural composers with their piano generation work using a music transformer, as part of Google's Magenta research project.

With that being said, transformers are out of the scope of this project. The honest reason is that, given what has been studied during the whole course and given the time boundaries for this project, working with this relatively new technology at this moment would mean hard-skipping a huge part of research on artificial intelligence and deep learning that is indeed worth studying and mastering. It is not a case that transformers themselves have been introduced after that the limitations and opportunities of the gated recurrent neural networks had been fully understood and put into practice.

Instead, this project can be collocated in the first trend, which considers neural architecture based on the use of recurrent cells in order to build music language models. Among the researchers that focused on this path, some compared the performances of different types of gated recurrent networks. Chung et al. (2014) [35] found the gated cells to be superior to the traditional recurrent cell, but were not able to determine whether the LSTM cell was better performing than the GRU cell or vice-versa. Their research was specifically conducted in the context of polyphonic music modeling using

[1] Transformers are neural architectures based only on the attention mechanism, without the recurrent pattern typical of LSTM-based and GRU-based recurrent neural network. See [33]

MIDI files. Differently, Nayebi et al. (2015) [30] compared the two types of gated recurrent cells in the task of generating music with audio waveforms. In addition to the fact that they found the LSTM-based network to consistently exhibit slightly lower training and validation loss than the GRU-based network, they observed that, while the generated output of the first was musically plausible, the output of the second primarily consisted of white noise.
It is important to note that although both of these experiments have been conducted as fairly as possible as the authors remark, the aim was evaluating the technology that was able to guarantee the most qualitative output, and not on the one that first converged to stable results.

Considering what has been discussed in this section and in Subsection 2.2.3, this project aims at using LSTM cells for music generation and possibly compare their performance to that of GRU cells.

3.3 Training and Feasibility

Section 3.1 and Section 3.2 have discussed the mathematical aspects of the problem from an effectiveness-oriented perspective. This section, instead, aims at briefly commenting on the efficiency aspect of the above presented solutions - focusing on end-to-end music generation from MIDI files using gated recurrent neural networks - by analysing their training cost in terms of availability and use of resources.

Huang et al. (2016) [26] built their character-level neural music language model using a 2-layered LSTM recurrent network and trained it using GPU instances. In detail, with respect to their experiments in treating raw MIDI messages as text, the training-time-per-epoch was about 1.47 hours for the full dataset containing 524 compositions, and 0.08 hours for a reduced version of the dataset. Instead, the textual sequence obtained from a piano-roll middle-step representation showed a training-time-per-epoch of about 0.00875 hours, which allowed for a training process that lasted for 800 epochs, i.e. 7 hours, on an AWS g2.2xlarge instance.

An alternative training option has been chosen by Hilscher et al. (2018) [28] who trained their character-based and LSTM-based generative recurrent network on Google Cloud. One of the best models according to the authors is relatively small in size, with only 1 LSTM layer made of 100 cells and trained on a dataset of less then 180 compositions for about 30 epochs.

Almost all the other researches that have been cited in the previous sections did not train the neural models on a CPU. This indicates that also this project must consider more powerful training tools than a CPU. Therefore, in order to minimize the cost

and the overall risk, freely available GPUs or TPUs should be considered as the first option[2].

3.4 Conclusions

This chapter has discussed the experimental background for this project, analysing and comparing the existing solutions for the neural music generation problem.

To summarize, the state-of-the-art technologies that in the past 10 years demonstrated the ability of artificially generating music are gated recurrent networks and transformer networks. Results have been remarkable both with music in MIDI format and with music in raw audio format. Particular attention should be payed also to the efficiency aspect: most of the solutions need more than a standard CPU for successfully running the experiments.

Given what has been discussed in this chapter, this project aims at working with MIDI files encoded in a text sequence fashion, using gated recurrent neural networks and experimenting GPUs or TPUs for training.

[2]This blog post gives an overview on how to train a Keras model on Google Colab using a TPU for free: www.kdnuggets.com/2019/03/train-keras-model-20x-faster-tpu-free

Chapter 4

Project Specification

This chapter examines the specifications of the project and is divided into five short sections as follows:

- Section 4.1 declares the aim of the project and details the objectives that have been established for eventually achieving the aim.
- Section 4.2 discusses the methods and the tools used for achieving each of the objectives listed in the previous section.
- Section 4.3 consists of the statement of the functional and non-functional requirements for this project.
- Section 4.4 discusses the potential issues from a legal, ethical, social and professional perspective.
- Section 4.5 contextualize the development of this work in the current social situation, making sure that all safety measures are satisfied.

4.1 Aims and Objectives

The aim for this project is to artificially generate music, using light end-to-end deep learning architectures, in such a way that the output sounds plausible to human listeners

In detail, the above stated aim is supported by the following objectives:

1. To collect simple and structured piano composition in MIDI format.
2. To select a sub-sample of the compositions so that the future processing and learning can match the constraints in resources and time.
3. To produce a mathematical and model-ready representation for MIDI files.
4. To use gated recurrent neural networks for building a music language model that learns long-term patterns in the musical sequence and iteratively generates new elements in the sequence.
5. To evaluate the plausibility of the generated music from the perspective of a sample of human listeners.

4.2 Methodology and Software Tools

This section expands the information given in Section 4.1 by detailing the methods and the tools used in order to achieve the objectives of this body of work.

1. The collection of MIDI files consists of the publicly available 2.0.0 version of the MAESTRO dataset (see Subsection 2.1.4). This is a set of piano compositions recorded from live performances which offers both audio waveforms and MIDI messages. Given that this project only deals with music in MIDI format, only the corresponding repository is downloaded.
2. The selection of a subset of the original MAESTRO dataset is done using the information provided by the metadata file that is attached to the MIDI repository when downloaded. Python programming language[1] is used in an interactive notebook environment, e.g. Jupyter Notebook or Google Colab, for processing and selecting data.
3. The selected collection of MIDI file is parsed and processed using Python. In detail, MIDI messages need to be converted to a piano-roll format and, ultimately, to a textual sequence of tokens. Finally, they are transformed into a network-readable format by encoding them in a numerical fashion.

[1] For more information, visit python.org

4. Gated recurrent cells like LSTMs and GRUs are used for constructing a neural network that learns to predict the next token in the sequence. Once trained, the model is used in a looping scheme that generates sequences of arbitrary length based on a seed set of tokens. The neural programming framework for these tasks consists of using Keras[2] in Python.

5. The generated sequence is converted back to a MIDI file in order to be played to a sample of human listeners that are in charge of evaluating its plausibility. To be specific, they are given an unlabelled set of audio extracts, some of which are artificially generated while some are human-composed, and they are asked to rate their perception on how likely is the song to be human made.

4.3 Requirements

Functional requirements directly determine the availability of the deep learning solution proposed in this project. In this context, the functional requirements are the following:

- The training neural architecture should be able to ingest MIDI files in an encoded format and learn the probabilistic structure of the musical sequences.
- The generative neural architecture should be able to leverage the knowledge of this probabilistic structure to produce a new musical sequence given a seed sequence of tokens.
- The generated sequences should be converted to MIDI format in order to be played and passed to the panel of listeners.

Non-functional requirements indirectly determine the availability of the deep learning solution proposed in this project by affecting its performance. In this context, the non-functional requirements are the following:

- The neural architectures should be trainable with a limited set of resources in time, computational power and volume of data.
- The accuracy of the final model should beat that of a very simple and naive baseline.
- The plausibility of generated music should not be perfectly distinguishable from that of original compositions by the panel of listeners.

[2]Keras is a Python deep learning API. For more information, visit keras.io

4.4 Legal, Ethical, Social and Professional Issues

This section formally clarifies the proportions of the potential issues that may be related to the development of this project.

From a legal point of view, no issues have been identified. In fact, all sources of information - e.g. pictures, books, articles, dataset, and so on - have been clearly referenced and the corresponding credits have been given to the authors to be compliant with the laws for intellectual property and with the good practices for publicly available material.

From the ethical perspective, no living human or animal is directly involved in the development of this project. The only interaction with humans is for evaluating the plausibility of the generated music, however no personal information is required. As a consequence, no ethical biases can be identified at this stage.

From a social point of view, it is important to note that this project does not aim at modifying people's perception of music and/or affecting the meaning that human-composed music has represented for us through the history. Instead, it can be seen as a simple contribution to the research that aims at exploring the possibilities of interaction between artificial intelligence and creativity.

From a professional point of view, this project is undertaken out of personal curiosity and interest, without any private company being involved in its development. As a consequence, there are no intentions of using the outcomes of this research for commercial purposes.

4.5 Risk and Safety

The investigation and the development of this project are undertaken in an unusual social and professional context due to the COVID-19 pandemic. As a consequence, the need to reduce the risk of infection and maximize everyone's safety is crucial. For this reason, this body of work is carried out remotely, without any physical contact with the university facilities and with the supervisor. All actions and experiments are taken in respect of social distancing.

Chapter 5

Summary

This chapter summarises the most important aspects of the project investigation report and, based on those, provides a graphical illustration of the working pipeline to be followed in the experimental report.

5.1 Investigation Summary

The project aims at generating mono-instrument polyphonic music from MIDI data using gated recurrent neural networks within an end-to-end music language model.
Chapter 2 discussed the theoretical background for music, MIDI language and deep learning, with a special focus on gated recurrent neural networks and neural language models.
Chapter 3 contextualized these theoretical concepts in the experimental framework consisting of the existing solutions that had been proposed over the years for solving the neural music generation problem.
Chapter 4 merged the information coming from both the theoretical and the experimental background for defining the content and the boundaries of this project, given the limitations in the available resources.

To sum up:

- MIDI format has been chosen for its light, structured and simple nature, compared to raw audio waveforms. The textual representation of MIDI files allows to include more information in training data, such as notes velocity, and fits perfectly in the context of a neural language model application.
- LSTMs and GRUs guarantees more capability of learning temporal dependencies than standard recurrent cells and, at the same time, they are a key technique to know and master before diving into the recently introduced and state-of-the-art transformer networks (which are out of the scope of this project).
- The implementation of the experiments will be conducted using Python programming language and Keras API for deep learning. Previous researches suggest the use of GPU instances for training the model.
- The qualitative evaluation of the generated music using blindfold tests is a common practice for assessing the plausibility of the music language model.

5.2 Experimental Working Pipeline

In conclusion, here is a diagram representing the working pipeline to be followed in the experimental report as a result of the considerations made in the investigation report.

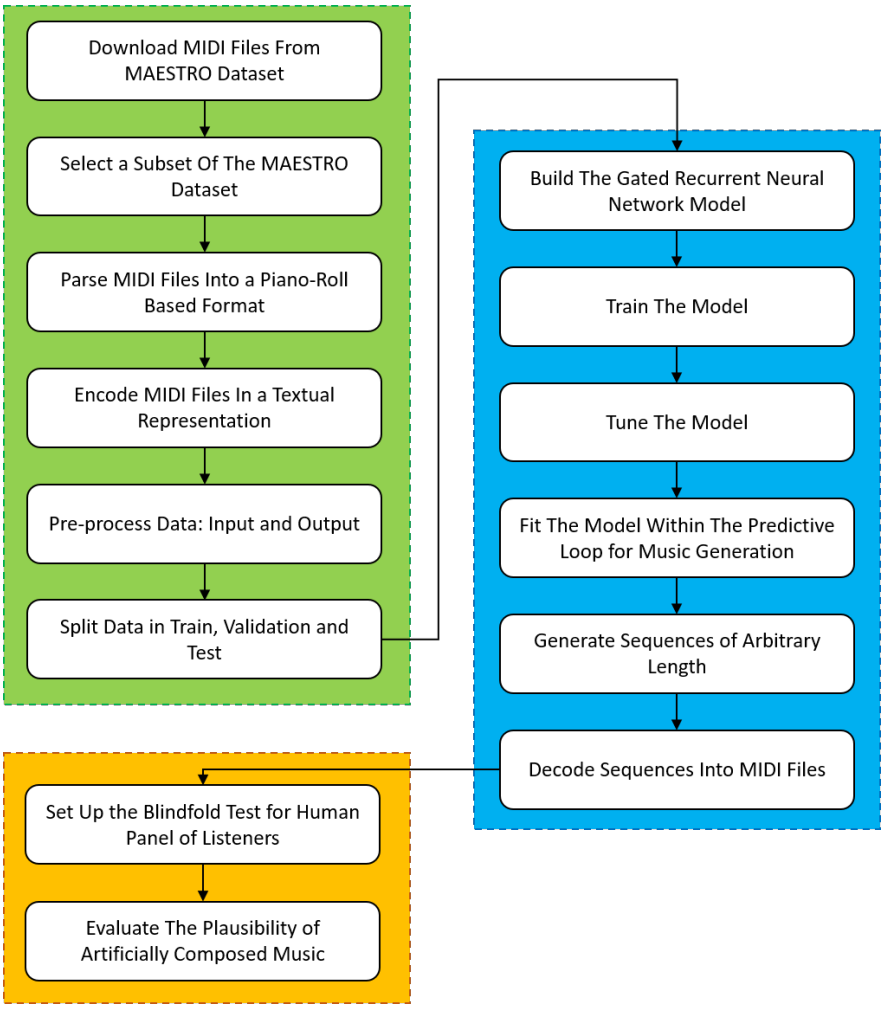

Figure 5.1: The general experimental pipeline as planned after the project investigation.

Author: Ludovico Lanni. Date: 10 July 2020.

Green Box: Data Pre-processing.

Blue Box: Neural Experiments.

Orange Box: Plausibility Assessment.

Bibliography

[1] Zaidel E SR Zaidel DW. Left and right intelligence: case studies of Raven's progressive matrices following brain bisection and hemidecortication. Cortex. 1981;17(2):167–185. Available from: https://pubmed.ncbi.nlm.nih.gov/7285591/.

[2] Nielsen JA, Zielinski BA, Ferguson MA, Lainhart JE, Anderson JS. An Evaluation of the Left-Brain vs. Right-Brain Hypothesis with Resting State Functional Connectivity Magnetic Resonance Imaging. PLOS ONE. 2013 08;8(8):1–11. Available from: https://doi.org/10.1371/journal.pone.0071275.

[3] Understanding Basic Music Theory. OpenStax CNX; 2013.

[4] Swift A. An introduction to MIDI; 2012. Available from: https://web.archive.org/web/20120830211425/http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol1/aps2/ [cited 24 June 2020].

[5] Walker J. Midicsv File Format; 2008. Available from: https://www.fourmilab.ch/webtools/midicsv/ [cited 24 June 2020].

[6] Briot JP, Hadjeres G, Pachet F. Deep Learning Techniques for Music Generation - A Survey. 2017 09;.

[7] Hawthorne C, Stasyuk A, Roberts A, Simon I, Huang CZA, Dieleman S, et al. Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset. In: International Conference on Learning Representations; 2019. Available from: https://openreview.net/forum?id=r1lYRjC9F7.

[8] Chollet F. Deep Learning with Python. Manning; 2017.

[9] Rumelhart D, Hinton G, Williams R. Learning representations by back-propagating errors. Nature. 1986;323. Available from: https://doi.org/10.1038/323533a0.

[10] Fernández-Cabán P, Masters F, Phillips B. Predicting Roof Pressures on a Low-Rise Structure From Freestream Turbulence Using Artificial Neural Networks. Frontiers in Built Environment. 2018 11;4.

[11] Jordan MI. Serial order: a parallel distributed processing approach. Technical report, June 1985-March 1986. 1986 5;Available from: http://cseweb.ucsd.edu/~gary/PAPER-SUGGESTIONS/Jordan-TR-8604-OCRed.pdf.

[12] Mozer M. A Focused Backpropagation Algorithm for Temporal Pattern Recognition. Complex Systems. 1995 01;3.

[13] Robinson AJ, Fallside F. The Utility Driven Dynamic Error Propagation Network. Cambridge, UK: Engineering Department, Cambridge University; 1987. CUED/F-INFENG/TR.1.

[14] Werbos PJ. Generalization of backpropagation with application to a recurrent gas market model. Zenodo; 1988. Available from: https://doi.org/10.1016/0893-6080(88)90007-x.

[15] Hochreiter S, Schmidhuber J. Long Short-Term Memory. Neural Computation. 1997;9(8):1735–1780. Available from: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[16] Cho K, van Merrienboer B, Gülçehre Ç, Bougares F, Schwenk H, Bengio Y. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. CoRR. 2014;abs/1406.1078. Available from: http://arxiv.org/abs/1406.1078.

[17] Shewalkar A, Nyavanandi D, Ludwig SA. Performance Evaluation of Deep Neural Networks Applied to Speech Recognition: RNN, LSTM and GRU. Journal of Artificial Intelligence and Soft Computing Research. 2019;9(4):235 – 245. Available from: https://content.sciendo.com/view/journals/jaiscr/9/4/article-p235.xml.

[18] Khandelwal S, Lecouteux B, Besacier L. Comparing GRU and LSTM For Automatic Speech Recognition. LIG; 2016. Available from: https://hal.archives-ouvertes.fr/hal-01633254.

[19] Sublime J, Kalinicheva E. Automatic Post-Disaster Damage Mapping Using Deep-Learning Techniques for Change Detection: Case Study of the Tohoku Tsunami. Remote Sensing. 2019 05;11:1123.

[20] Rezende DJ, Mohamed S, Wierstra D. Stochastic Backpropagation and Approximate Inference in Deep Generative Models; 2014.

[21] Kingma DP, Welling M. Auto-Encoding Variational Bayes; 2013.

[22] Goodfellow IJ, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, et al. Generative Adversarial Nets. In: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2. NIPS'14. Cambridge, MA, USA: MIT Press; 2014. p. 2672–2680.

[23] Feng J, Feng X, Chen J, Cao X, Zhang X, Jiao L, et al. Generative Adversarial Networks Based on Collaborative Learning and Attention Mechanism for Hyperspectral Image Classification. Remote Sensing. 2020;12(7). Available from: https://www.mdpi.com/2072-4292/12/7/1149.

[24] Bengio Y. Neural net language models. Scholarpedia. 2008;3(1):3881. Revision #140963.

[25] Boulanger-Lewandowski N, Bengio Y, Vincent P. Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription; 2012.

[26] Huang A, Wu R. Deep Learning for Music; 2016.

[27] Agarwala N, Inoue Y, Sly A. Music Composition using Recurrent Neural Networks; 2017. Available from: https://www.semanticscholar.org/paper/Music-Composition-using-Recurrent-Neural-Networks-Agarwala-Inoue/c93379a401dd159fc0c90eab44c43d07286b227e.

[28] Hilscher M, Shahroudi N. Music Generation from MIDI datasets; 2018. Available from: https://www.semanticscholar.org/paper/Music-Generation-from-MIDI-datasets-Hilscher-Shahroudi/6d5071756fe4c304981656cb1be9be7f5611ac53.

[29] Payne CM. MuseNet; 2019. Available from: openai.com/blog/musenet.

[30] Nayebi A, Vitelli M. GRUV : Algorithmic Music Generation using Recurrent Neural Networks; 2015. Available from: https://www.semanticscholar.org/paper/GRUV-%3A-Algorithmic-Music-Generation-using-Recurrent-Nayebi-Vitelli/392914e7a1eba8f64daeaa39da8563f6ec409a4c.

[31] Kalingeri V, Grandhe S. Music Generation with Deep Learning; 2016.

[32] Dhariwal P, Jun H, Payne C, Kim JW, Radford A, Sutskever I. Jukebox: A Generative Model for Music; 2020.

[33] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al.. Attention Is All You Need; 2017. Available from: https://arxiv.org/abs/1706.03762.

[34] Huang CZA, Vaswani A, Uszkoreit J, Shazeer N, Simon I, Hawthorne C, et al.. Music Transformer; 2018.

[35] Chung J, Gulcehre C, Cho K, Bengio Y. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling; 2014.

Appendix A

Early Project Specification

A.1 Project Title

The tile of the project is:

"**Generating piano melodies in MIDI format with Deep Learning**"

A.2 Description

The project consists of experimenting modern generative deep learning architectures in order to build a neural composer, i.e. an artificial intelligence that composes music melodies based on a number of examples it has been trained with.
In order to do so, the second and newest version of the MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization) [7] dataset will be considered. It contains mono-instrument and polyphonic melodies in MIDI (Musical Instrument Digital Interface) format, allowing for the development of a structured data science project.

A.3 Background and motivation

Generative deep learning is one of the most fascinating and recent applications in the spectrum of artificial intelligence. Here, in fact, the aim is teaching the machine not only how to learn by examples, but also how to produce new examples by leveraging the knowledge it has built during training stages. While the vast majority of research focuses on the generation of text or images, music has recently been an interesting application. Extraordinary results have been achieved over the past two years in neural music generation, specially considering the research outcomes [29][32] published by

OpenAI.
With that being said, producing music in MIDI format using a DAW (Digital Audio Workstation) is one of my favorite hobbies. Therefore, the opportunity of combining scientific research and personal interests is the core motivation behind this project. With respect to the development, the challenging aspects are mainly related to the aim of letting the machine learn how to generate music without any information on the structure (tempo, time-signature, ...), with limited computation resources and with a size-limited training set.

A.4 Key techniques

The key techniques and skills that are involved in the development of this project are:

- Advanced data engineering applied to unconventional data format.
- Advanced deep learning applied to data generation using recurrent neural networks
- Understanding of neural language models and Natural Language Processing techniques
- Evaluating generated output from qualitative and quantitative perspectives

A.5 Objectives

The objectives for the project are:

- Find an effective and efficient way to mathematically represent music files in MIDI format.
- Compare the performances of different generative deep learning architectures.
- Build a final model that has good generative power and could be scaled up by increasing and improving the computational resources.
- Measure the credibility of the generated music with a sample of human listeners.

A.6 Methodology and requirements

The methodology I have planned to implement for the practical parts of the project consists of the following steps:

1. Download MIDI files

2. Load MIDI files
3. Parse MIDI files
4. Encode MIDI files in machine learning format
5. Build a Generative Deep Learning architecture
6. Setup the experiments
7. Train and validate the model
8. Hyper-tune the model
9. Test the model Performance
10. Build final model
11. Generate machine-composed MIDI files
12. Decode MIDI files from ML format
13. Inverse-parse MIDI files
14. Store MIDI files
15. Play MIDI files
16. Optional: Deploy the Deep Composer as a web application

The requirements that are needed in order to develop the project are the following:

- The neural composer should generate music based on a seed sequence of chords.
- The neural composer should beat a naive baseline in terms of testing accuracy
- The neural generated music should be able to trick human listeners in terms of establishing whether it was artificially produced or not.

Appendix B

Project Management

This body of work has been carried out remotely due to the safety measures in response to the COVID-19 pandemic.

The meetings took place almost every Tuesday on Microsoft Teams, that was the central project management tool. In order to support the discussion, shared Google Docs were used for updating the supervisor about the daily progresses and for organizing the literature objects that were reviewed and cited in the report. Figures have been edited, saved and organized in Power Point.

Overall, I managed to structure the work in an optimal way together with my supervisor. As a result, the writing of the report has been fluid and with only minor obstacles.

Appendix C

Project Log

The following is a weekly summary of the work carried during the development of this body of work.

Week Ending: 26/5/2020

This week, prior to the first meeting with my supervisor Carlos, I spent time doing some general reading on existing attempts to generate music with deep learning. This led me to the identification of two possible macro-ways to develop the project: a) translate the midi files into 2D piano roll representation and then train VAE (Variational Autoencoders), Convnets or RNN (Recurrent Neural Networks); b) encode the midi files as text sequence and use RNN as a neural language model. In addition, I saved some useful links and web resources on the topic in order to support the work later on. Then I searched a suitable dataset and found two good sources: a) MAESTRO, a dataset introduced with a paper by GoogleAI; b) a website that stores Video Games theme songs in midi format. Lastly, I started studying the MIDI standard to get familiar with it before thinking about its best mathematical representation in our context.

In the meeting we discussed the above mentioned topics. For the next week, the focus should be on: a) getting deeper knowledge on MIDI and its possible representations, b) better scanning the two datasets, c) doing some reading about RNN and Generative Models.

Week Ending: 2/6/2020

Given the topics discussed in the previous meeting, this week I focused my work on three main tasks: a) inspecting in detail the MIDI language to understand its key hierarchical

components; b) studying introductions to RNNs, LSTMs, GRUs and generative deep learning; c) choosing a dataset and inspecting its meta information.

With respect to point a), I checked some online documentation on the MIDI standard. In detail, I focused on the midi-csv format, consisting of parsing the midi file into a csv editable format, much more flexible and modifiable. I then understood that MIDI messages can be grouped in three macro-containers depending on their usage: i) File structure records (they are always needed because they represent the structure of the file, i.e. start, resolution, end, ...), ii) File meta-events (they provide descriptive information on how to interpret the channel events messages, i.e. time signature, tempo, key-signature, ...), iii) Channel events (they are the most relevant data as they represent notes and controls progression over all the timesteps). Later on, I started experimenting some midi preprocessing in python, with the main objective of trying and comparing different modules. Probably the best option is using py-midicsv to parse the midi file in csv format.

With respect to point b), I read the respective chapters in "Deep Learning with Python" by F.Chollet and I studied a number of blog posts and Youtube tutorials on these deep learning architectures. In detail, I understood the conceptual difference between vanilla RNNs and their further evolution with the inclusion of LSTM or GRU cells in order to address the vanishing gradient problem. I came across RNN-based generative models applied to text sequences generation and I started investigating NLP fundamental concepts such as tokenization and embedding. In fact, the idea is formulating a midi sequence generation problem as a text sequence generation problem, thus building a neural language model.

With respect to point c), I compared different online sources of midi data and I selected the MAESTRO dataset to be the most reliable, extensive and well-structured one. MAESTRO is originally a collection of music files in midi and wav format with fine alignment, built by GoogleAI team in the context of the Magenta project. For my purposes, I will consider only midi files.

Other than these bullet points, I have done some literature review by going through 4 experimental research papers on neural music generation and summarising their content in the Literature Review dedicated doc.

In the meeting we discussed the above mentioned topics. For the next week, the focus should be on: a) studying NLP embedding concept and word2vec methods b) investigating some possible mathematical representation of midi files, c) deepening the knowledge on neural language models.

Week Ending: 9/6/2020

Given the topics discussed in the previous meeting, this week I did not have enough time to cover all of them in depth. In particular, I mainly focused on an introduction to NLP and to representation of words. I researched the limitations of traditional representations, such as n-grams, which can be summarised in the curse of dimensionality (non-scalability) and the word similarity ignorance. With the aim of overcoming those issues, distributed representations of words have been introduced. They provide compression, smoothing and densification in order to generate semantically meaningful word vectors. The idea of distributed representation is commonly referred to as word embedding. Furthermore, in a neural language model, word embeddings are learnt through the optimization algorithm during the training process. In Keras there is a layer class called Embedding, which can be used for this purpose.

Apart from this topic, I wanted to investigate a) possible and innovative mathematical representation of MIDI files and b) state-of-the-art neural language models, but I will dive into these points in the next week.

Having that said, I attended a 1-hour webinar promoted by Knime Analytics where I got useful information on RNNs with LSTM cells for text generation.

In the meeting we discussed the above mentioned topics, plus some project management tips. For the next week, the focus should be on: a) investigating some possible and innovative mathematical representation of midi files, b) learning how to work with resolution parameter in midi files, c) deepening the knowledge on state-of-the-art neural language models, d) filling the ethics form and write the project specifications document.

Week Ending: 16/6/2020

Given what has been discussed in the last meeting, this week I focused on:

Investigating possible mathematical representations for midi files. Piano roll is the easiest and most effective way of looking at a midi file. Limitations: it is not a loss-less transformation (control messages). I still need to figure out if it is possible to keep the velocity when going from piano roll to temporal sequence of tokens. Came up with two options for keeping it, but feasibility needs to be discussed with supervisor. Learning how to handle the resolution parameter for midi files. It is managed with pretty_midi library as part of the get_piano_roll() function. It is conveniently expressed in frames per second. 8 fs may be a good choice in the trade-off between detail-level and complexity. Filling the ethics form. Writing and compiling the project description

Extra: set up a project workflow visualization in PowerPoint.
Extra: coded a Jupyter notebook for subsetting the original maestro dataset by selecting only the 99 songs with duration between 2 and 3 minutes.
Extra: studied the dimensionality of recurrent layers in Keras (input, units, output).
Extra: studied embedding layer in Keras.
Extra: studied the concept of batch in training neural networks (Batch GD, mini-batch GD, stochastic GD).
Extra: followed a tutorial on how to train a Keras model on a TPU in Colab for free.
Extra: installed MikTex and Texmaker and tried to run the thesis template

In the meeting we discussed the above mentioned topics - specially the mathematical representation of midi files and the possible neural architectures - and we reviewed the files for the first two submissions, i.e. ethics form and project specifications. For the next week, the focus should be on: a) setting up the environment for the project investigation document (Texmaker, Overleaf, check references), b) planning the project investigation structure, c) starting to write the project investigation document and simultaneously creating the Bibtex database.

Week Ending: 23/6/2020

Given what has been discussed in the previous meeting, this week I focused on: Setting up the environment for the project investigation report. Following what my supervisor has suggested I moved from Texmaker to Overleaf, an online LaTex editor that allows for easy and quick compile, file-sharing and document versioning. Planning and implementing the structure of the project investigation report. Starting from the MSc Project Thesis template provided by the academic staff, I mounted it on Overleaf and I modified the chapter naming and progression to better suit my project. I decided to organize the writing in 5 main chapters: (1) introduction [to be completed by 23/05], (2) domain analysis [to be completed by 30/05], (3) literature review [to be completed by 07/06], (4) project specification and (5) summary [to be completed by 14/06]. Starting to write the report. As planned, I have completed the introduction chapter, which is articulated in the following sections: about this thesis, motivation, problem overview, chapter list. Simultaneously started populating the Bibtex database for efficient and compliant referencing.

Did not have a video-conferencing meeting this week. Next week the focus will be on starting and completing the domain analysis chapter. Obviously, the Bibtex database will be updated simultaneously.

Week Ending: 30/6/2020

As planned, this week I worked on the domain analysis chapter from start to finish. This chapter comes after the introduction chapter and expands on the theoretical background of the project. I structured it in four macro-sections:

1. Music and the MIDI standard
2. Deep Learning
3. Generative Deep Learning
4. Conclusions

Personally speaking I am very happy for what I managed to write and the way I did it. In addition, I am meeting my deadlines that allow me to finish the investigation report in two weeks, far before the official deadline.

Did not have a video-conferencing meeting this week but I kept in touch with my supervisor using Microsoft Teams chat space. Next week the focus will be on starting and completing the Literature Review chapter and hopefully starting the Problem Specification chapter.

Week Ending: 7/7/2020

As planned, this week I worked on the literature review chapter from start to finish. This chapter discusses the experimental background of the project by introducing, analysing and comparing the existing solutions for the neural music generation problem. I structured the chapter in four sections:

1. Mathematical representation of music
2. Neural architectures for music generation
3. Training and feasibility
4. Conclusions

In addition, I have started the Project Specification chapter and completed the first three sections:

1. Aims and objectives
2. Methodology and software tools
3. Requirements

To sum up, also this week I managed to meet the schedule and I am looking forward to finishing the investigation report by the end of the next week.

In the meeting we discussed the above mentioned topics, plus the adjustments to be made based on what has been written so far. For the next week, the plan is finishing to write the investigation report by completing the project specification chapter, the summary chapter and the side sections such as appendix, abstract, etc..

Week Ending: 14/7/2020

Given what has been discussed in the previous meeting, this week I completed the project investigation report by writing the project specification chapter, the appendices, the abstract and the acknowledgments. In addition, after making some corrections based on the supervisor's feedback from last week, I downloaded a PDF final draft of the report for receiving the ultimate feedback before submission.

In the meeting we discussed the above mentioned topics, plus final adjustments to be made based on what has been written so far. After making the ultimate corrections, the investigation report is completed and ready to be submitted. For the next week, the plan is setting up, organizing and the starting the experimental report.