

M.Sc. PROJECT FINAL 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

August 2020

Supervisor Dr. Carlos Moreno-García

Abstract

This body of work consists of the final 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 leveraging what has been discussed and research in the investigation report for developing and evaluating a neural music composer based on gated recurrent neural networks that is able to generate plausible music extracts in MIDI format. First of all, the methodology is justified by summarising the theoretical and the experimental background. After that, the design and the implementation of the neural architecture are discussed step by step. In detail, two models have been created and experimented: the first using a single block of LSTM (Long-Short-Term-Memory) cells, the second using a single block of GRU (Gated Recurrent Unit) cells. These models are trained on MIDI sequences after they have been converted into character-level symbolic text format, keeping track of all the information contained in piano-roll representations. The most valuable generated extracts from each model are used in a listening game, together with some human-composed extracts, with the aim of letting a panel of players assessing the plausibility of each composition based on their subjective perception and without knowing the group each extract belongs to. Consequently, results are presented and interpreted. In conclusion, both models managed to produce extracts that have been able to trick a significant number of players, with the GRU model outperforming the LSTM one.

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*.....
Ludovico Lanni

Date ... 19-August-2020 ...

Contents

Abstract	ii
Acknowledgements	iii
Declaration	iv
1 Introduction	1
1.1 About this Thesis	2
1.2 Project Overview	2
1.3 Theoretical Background	2
1.3.1 The Music Principles and The MIDI Standard	3
1.3.2 Deep Learning and Recurrent Neural Networks	5
1.3.3 Generative Deep Learning and Neural Language Models	9
1.4 Experimental Background	11
1.5 Investigation Report Outcomes	13
1.6 Chapter List	14
2 Implementation	15
2.1 Implementation Plan	16
2.2 Data Pre-Processing	17
2.2.1 The Starting Point: The MAESTRO Dataset	18
2.2.2 From MIDI To Piano-roll	19
2.2.3 From Piano-roll To Text	20
2.2.4 From Text To Binary Data	22
2.2.5 Train-Validation-Test Split	25
2.3 Neural Experiments	25
2.3.1 Environment And Resources	26
2.3.2 Models Design	27
2.3.3 Training And Tuning	28
2.3.4 Predictive Loop And Sequences Generation	30

2.3.5	From Text To MIDI	31
2.4	Plausibility Assessment	34
2.4.1	The Listening Game	35
2.5	Conclusions	36
3	Evaluation	38
3.1	Results	39
3.1.1	Listening Game Scores	39
3.1.2	Traditional Metrics vs Human Perception: GRU or LSTM? . . .	40
3.2	Objectives-Achievements Comparison	41
3.3	Requirements-Achievements Comparison	42
3.4	Reflections on Potential Issues, Risk and Safety	43
4	Conclusions	45
4.1	Summary	46
4.2	Limitations and Future Work	47
	Bibliography	49
A	Project Management	51
B	Project Log	52

List of Tables

1.1	Comparison between LSTM gates and GRU gates.	9
1.2	Dimensional trade-off in character-level and word-level language models	11
2.1	Details on the conversion from piano-roll to text	20
2.2	Training, validation and test scores for the final models.	30
3.1	Aggregated results of the listening game with 53 players.	39

List of Figures

1.1	Two sequential octaves side by side on a keyboard	3
1.2	Piano-roll representation of a music extract.	5
1.3	Visualization of a simple Neural Network: the Multi-Layer Perceptron. .	6
1.4	A recurrent layer in its compact (left) and unrolled (right) representations.	8
1.5	Structure of a Long-Short-Term-Memory (LSTM) cell.	9
1.6	Structure of a Gated Recurrent Unit (GRU) cell.	9
1.7	Example of a character-level neural language model.	10
2.1	The experimental pipeline as planned after the project investigation. . .	16
2.2	Data Pre-processing Pipeline	17
2.3	Duration of compositions in the reduced version of the dataset with time resolution set to 8 frames-per-second	19
2.4	Example of conversion from piano-roll matrix to text	22
2.5	Second reduction of the size of the original dataset.	23
2.6	Neural Experiments Pipeline	26
2.7	Structure of the LSTM-based neural network.	27
2.8	Structure of the GRU-based neural network.	28
2.9	Piano-roll representation: a qualitative extract (Top) and a poor extract (Bottom)	34
2.10	Assessment Pipeline	34
2.11	Example of a question from the listening game	36
3.1	Visualization of the answers for the GRU-generated extract named midi_11.	40

Listings

1.1	Example of a midi-csv file.	4
2.1	Function for converting a piano-roll repository into a symbolic text repository	20
2.2	Function for converting a symbolic text repository into a binary and model-ready repository	24
2.3	List of callbacks used when training the models.	28
2.4	Function for converting a symbolic text string into a piano-roll matrix .	32

Chapter 1

Introduction

This chapter provides an introduction to the topics covered in this report and summarises what has been discussed in the project investigation report [\[1\]](#). First, a short section will state the author, the context and the purpose of this body of work. After that, an overview of the project will inform the reader about the objectives of this project and the content of this report. The third section will briefly discuss the theoretical background, summing up the relevant concepts. Then, a brief recap of the experimental background will be provided. After that, The outcomes of the investigation report will be presented as a starting point for the practical work at the center of this report. 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.

In particular, this is the experimental report that follows what has been analyzed and discussed in the investigation report [1].

1.2 Project Overview

Music is at the core of this project. It is an expression of the human soul and is able to make our mind escape from reality by creating imaginary contexts and by giving us the capability of dreaming with the eyes open. To a certain extent, the world of artificial intelligence share some of this objectives with music. The race towards the making of an Artificial General Intelligence - a computer program that not only solves the problems that humans pose to them, but is also able to find new problems autonomously for eventually solving them without any human interaction - demonstrates how magical and still relatively unexplored this field can be.

This project wants to be on the bridge between rationality and creativity, sciences and arts, computing and music. Indeed, the goal is to engineer a neural composer that is able to learn from a small collection of classical piano compositions in MIDI format in order to generate short music extracts that are as more plausible as possible for human listeners.

1.3 Theoretical Background

Since this project aims at using deep learning techniques for processing and generating non-tabular data that consists in music files in MIDI format, its theoretical background mainly involves the understanding of the following features: music and the MIDI standard, time-aware deep learning techniques, generative neural architectures.

All these topics have been extensively discussed in the investigation report [1] and the three following subsections will summarize the content.

1.3.1 The Music Principles and The MIDI Standard

This section focuses on introducing the most relevant concepts regarding music theory with respect to the development of this project, where the world of artificial intelligence meets the world of music.

First of all, the musical notes are the atomic elements in a composition: no song can exist without them. Notes are simply a convention for representing combinations of features in sound waves, that are the physical background of music. A musical note can be identified by four main attributes: pitch, velocity, duration and timbre.

The pitch gives us information about the frequency of the wave that produces the sound of a certain note. Each key in a keyboard represent a different pitch, for instance. Conventionally, the set of possible pitches follows a discrete and modular progression: in fact, as Figure 1.1 shows, they are organized in pitch-increasing octaves, each being a set of twelve different semitones ($A, A\sharp \text{ or } B\flat, B, C, C\sharp \text{ or } D\flat, D, D\sharp \text{ or } E\flat, E, F, F\sharp \text{ or } G\flat, G, G\sharp \text{ or } A\flat$).

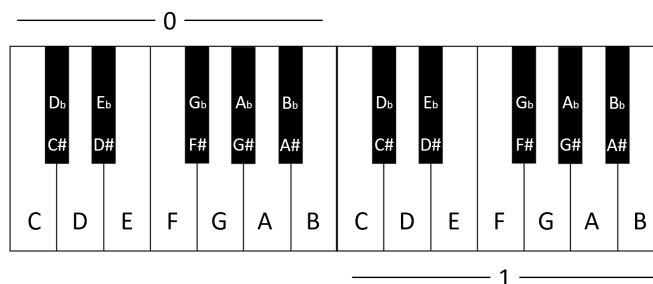


Figure 1.1: Two sequential octaves side by side on a keyboard

Author: Ludovico Lanni. Original Vector by Clker-Free-Vector-Images from *Pixabay*.

Available at pixabay.com. Download date: 25 June 2020

The velocity of a note is the volume or intensity of the sound that that note generates. The greater the velocity, the louder the sound.

The duration of a note indicates the time the respective sound starts to be generated and the time it stops. It can be expressed in classical time units, such as seconds, or music-specific measures, such as beats. The important thing to note, however, is that is always possible to convert the length of a note into seconds.

Finally, the timbre of a note is associated with the color of the sound, which usually depends on the instrument that note is played with. In other words, two notes played with same pitch at the same velocity can still sound different.

Given this definitions, a simple way of working with music files is to consider the notes and the attributes that describe their characteristics. This is why the MIDI format has been chosen as the nature of the data for this project. MIDI (Musical Instrument Digital Interface), in fact, consists in a standard protocol for representing and connecting music files among electronic instruments and devices: its hierarchical and symbolic structure allows for isolating a defined set of features such as, in our case, musical notes and their attributes. Furthermore, for the same reasons, the use of MIDI format reduces the computational risk of this project, in the sense that its scalable nature always offer a simpler and easier option for the development of our neural music composer.

Having said that, MIDI format is a low-level language that must be encapsulated for more solid and meaningful information. Therefore, this project plans to use csv format as a higher level parser for MIDI files: the result is what is called midi-csv format. An example is illustrated in Listing 1.1.

```
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 1.1: Example of a midi-csv file.

Author: John Walker. Original example available at fourmilab.ch.

Extraction date: 25 June 2020. See also [2].

MIDI interfaces are often associated to the piano-roll representation: this consists of a matrix that expresses the velocity at which a note with a certain pitch is played at a certain time frame. In other words, a piano-roll matrix is a simplification of a MIDI file that only focuses on musical notes and their key attributes: note that this perfectly matches the plan for this project.

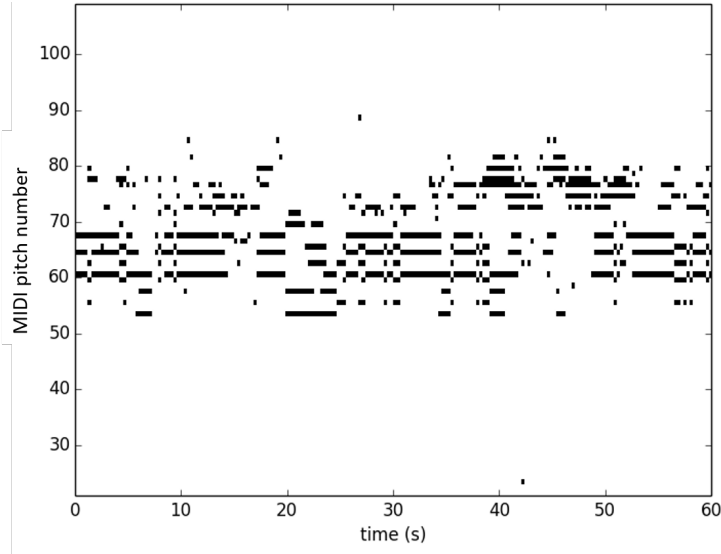


Figure 1.2: Piano-roll representation of a music extract.

Author: Ludovico Lanni. Original Image by Dr. Nick Kelly from *Transpose midi with python for computational creativity in the domain of music*.

Available at nickkellyresearch.com. Download date: 8 August 2020

As Figure 1.2 shows, the range of pitch values in MIDI files usually go from 0 to 127, for a total of 128 different pitches. The black boxes in the picture mean that the note with that particular pitch is being played. In the mathematical version of the piano-roll, which is basically a matrix, they correspond to integer values greater than zero. This is the velocity of the note, that can vary across the time frames and that, once again, ranges between 0 and 127.

1.3.2 Deep Learning and Recurrent Neural Networks

Now that the key principles of music that are relevant for this project have been introduced, let us move to the presentation of the fundamental computational techniques that will be used for working with music files.

These techniques belong to the field of deep learning, which is in turn a peculiar field in the scope of machine learning. In fact, just like other machine learning tool-sets,

deep learning consists of finding the optimal mathematical representation of the input data that best describes them with respect to pre-defined target data, and then using this knowledge, i.e. the representation learnt, for estimating the target for new data where this is unknown. What is special about deep learning is the nature of the mathematical representation, which is distributed, or layered [3]. Generally speaking, this kind of computational architecture is called neural network, meaning that it is made of successive layers of neurons that functionally maps the input to the output of the network. Figure 1.3 shows a basic neural network, often referred to as feed-forward network, or multi-layer perceptron.

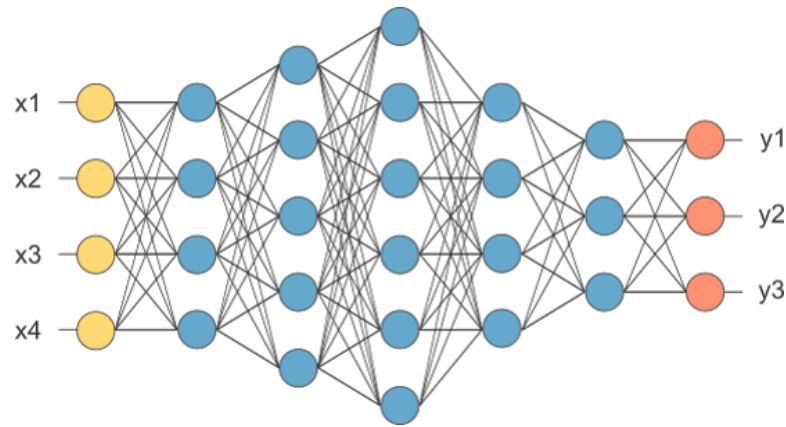


Figure 1.3: Visualization of a simple Neural Network: the Multi-Layer Perceptron.

Author: Jai Pancholi. Original Image by Jai Pancholi from *Picasso Me - Neural Style Transfer*.

Available at jaipancholi.com. Download date: 8 August 2020

Each neuron is a simple computational unit that applies one or more mathematical functions to the input data flowing through its inner connections and let the output of this operation flow through its outer connections. These computations transform data by using numerical parameters, i.e. weights and biases. To be precise, the whole purpose of a neural network is to learn the optimal combination of these values for better estimating the true targets. Therefore, the mathematical representation we mentioned above is all about the network architecture and its numerical parameters. So, how does a neural network, given its structure, learns the optimal combination of weights and biases? Thanks to a routine which alternates feed-forward computation with an optimization algorithm called backpropagation [4]. The process consists of the following steps:

1. Initialize the weight matrices and the bias vectors with random values.
2. Perform a feed-forward computation in the network, transforming the input into

estimated targets.

3. Use a loss function to compute a distance measure between the estimated targets and the true targets.
4. Compute the gradients of the loss function with respect to weights and biases.
5. Update weights and biases decreasing their value by the respective gradient smoothed using a learning rate parameter.
6. Go to step [2](#)

The purpose of the backpropagation algorithm is to update the parameters in a direction that allows for the minimization of the loss function. When the estimated targets are close to the true targets, in fact, the network has managed to scan the input data and predict their label by using the parameters it has ultimately learnt. This means that, if the network has not over-fitted input data, it is capable of correctly estimating the target of new unlabelled data.

Now, depending on the type of input data, the problem to be solved and the available resources, one can choose among a considerable number of different types of neural networks. Given the context of this project, where the neural architecture has to deal with temporal sequences, and given the limited resources available, gated recurrent neural networks have been chosen as the most appropriate tool.

As better detailed in the project investigation report [\[1\]](#), recurrent neural networks (RNNs) consist in a conceptual modification of the classic multi-layer perceptron in order to address its incapability of understanding the time element during the learning process. In 1986, Jordan [\[5\]](#) defined the structural differences between a non-recurrent network and a recurrent network: while the architecture of the first has no cycles, 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 blocks transform the learning process of a typical feed-forward network from static, which means unaware of the chronological dimension, to dynamic.

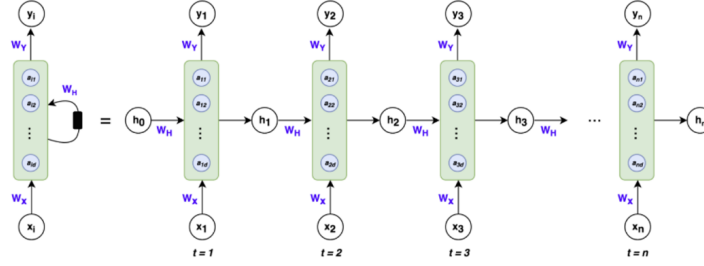


Figure 1.4: A recurrent layer in its compact (left) and unrolled (right) representations.

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.

As Figure 1.4 shows, a recurrent block in a neural network can be seen as a sequence of layers that shares the same weights and where the output of each layer is a function of two components: not only the input data, but also the output of the previous recurrent layer in the sequence. Therefore, due to the considerable depth of recurrent neural networks in terms of number of effective layers, they significantly suffer from the vanishing gradient problem. This means that the earlier layers of the network are likely to stop learning prematurely because the correspondent gradient becomes so small that weights are not being updated anymore. This is why standard RNNs, sometimes called vanilla RNNs, are said to have short-term memory. Since many applications that deal with temporal sequences require the model to learn long-term patterns, the vanishing-gradient problem led to modifications in the neural architecture with the aim of introducing a mechanism for carrying relevant information wherever it is needed in the recurrent sequence of layers. These can be achieved by adding information gates to each layer of a recurrent block: their function is to decide at every step what information should be used, what should be stored and what should be forgot. A simplified interpretation of the gates consists of thinking of them as binary ports which accept either all information or no information. In reality, each gate is an actual additional layer that can accept partial amounts of information.

The two most popular gated recurrent blocks are based on the Long-Short-Term-Memory (LSTM) cell [6] and the Gated Recurrent Unit (GRU) cell [7]. Figures 1.5 and 1.6 illustrates their structure.

They mainly differ in the number of gates, with the LSTM cell having one more than the GRU cell. Table 1.1 draws a brief comparison between the gates of the two recurrent cell types by summarising their role. Furthermore, the first stores information with the help of a cell state, in addition to the traditional state - or hidden state - of the

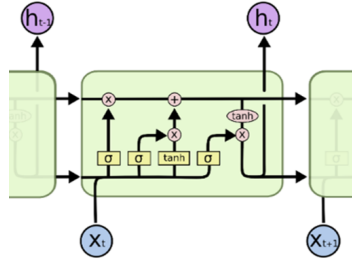


Figure 1.5: Structure of a Long-Short-Term-Memory (LSTM) cell.

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

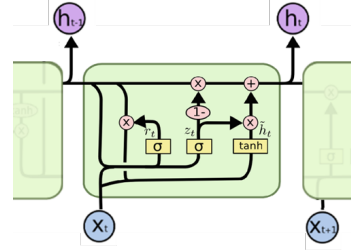


Figure 1.6: Structure of a Gated Recurrent Unit (GRU) cell.

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.

recurrent layers, while the second only use one memory object, i.e. the hidden state of the network. The lighter structure of GRU cells often results in networks with less parameters and, for that reason, more straight-forward training process. However, this does not imply that they are always more effective than LSTM cells. For that reasons, this project aims at experimenting with both types of gated recurrent cell.

Gates	LSTM	GRU
Forget Gate	What to forget from cell-state	-
Input Gate	What to add to cell-state	-
Output Gate	What to send to hidden-state	-
Update Gate	-	What to keep in memory
Reset Gate	-	What to forget from memory

Table 1.1: Comparison between LSTM gates and GRU gates.

1.3.3 Generative Deep Learning and Neural Language Models

A recurrent neural network needs to be encapsulated in a higher-level process that leverages its natural capability of discriminating, or predicting, in order to give it the ability of generating new data based on what has been learnt during training. This kind of techniques belong to the field of generative deep learning, which aims at

understanding the underlying structure of a data collection using distributed learning, with the purpose of being able to replicate it as more plausibly as possible.

The key idea behind this kind of processes relies in the so called latent space of a neural model: this term refers to the distributed representation, i.e. the hierarchical set of weight matrices and bias vectors, that is learnt by any kind of neural network. How to generate new data given the latent space of a trained model? As simple as sampling from its distribution.

A lot of deep generative architectures have been proposed over the years and they usually differ depending on the task they have been engineered for. Focusing on text generation, which is the key technique involved in the implementation of this project, the so called neural language models aim at capturing the statistical nature of a language [3] in order to generate sequential and meaningful text in that particular language by using neural networks [8].

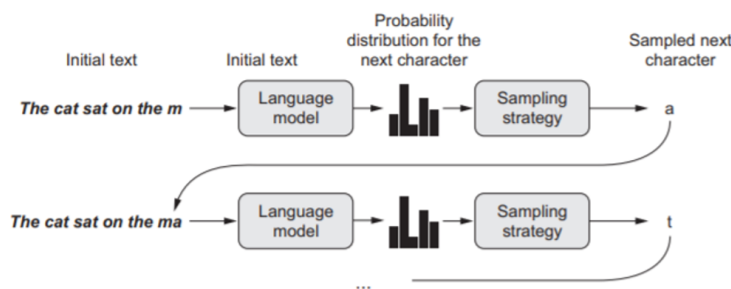


Figure 1.7: Example of a character-level neural language model.

Author: Francois Chollet. Original image by Chollet from *Deep Learning With Python* [3].

Figure 8.1. Available at faculty.neu.edu.cn. Download date: 29 June 2020.

The architecture of a neural language model (Figure 1.7) usually consists of a recurrent neural network that is trained on existing textual data in order to predict the next token - e.g. a word or a character - in the sequence using previous tokens as input. Then, after it has modelled the latent space, the network can ingest a seed sequence, predict the next token by sampling from the latent space itself, then add the predicted output to the input sequence and iteratively generate a sequence of arbitrary length. In the case that each token corresponds to a character in the text, the model is said to be character-level. On the contrary, if each token corresponds to a word in the text, the model is said to be word-level. What are the advantages and disadvantages of these two options? As Table 1.2 summarizes, technically speaking the main advantage of char-level models is that the size of the vocabulary, which is the number of unique tokens in the dataset, is smaller than in word-level models. In fact, this helps in terms of training

time because the dimensionality of the input is smaller, thus less parameters have to be learnt. However, 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, thus the number of parameters, for effectively learning long-term dependencies.

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

Table 1.2: Dimensional trade-off in character-level and word-level language models

Given the objectives and the available resources for this project, only one type of neural language model can be implemented. So, how to deal with this hard trade-off and manage to choose? Well, keeping the technical aspects aside for a moment, it is important to note that there is a huge conceptual difference between character-level and word-level text generation, that is creativity. In fact, while word-level models can only generate text made of words that already exist in the vocabulary, char-level models can form completely new words using the same set of characters. Therefore, in the case that the goal of a text generation project is to teach the machine to speak in a natural language, such as English or Italian, and the available vocabulary is comprehensive of all the existing words, the use of a word-level model is not discouraged by lack of creativity. However, since the goal of this project is to generate symbolic text that encodes music patterns by using a limited-in-size dataset, the use of a character-level model can lead to original notes that were not part of the dataset. For these reasons, this project focuses on character-level neural language models based on gated recurrent neural networks for symbolic text generation.

1.4 Experimental Background

While Section 1.3 briefly presented the theoretical background of this project, introducing the relevant concepts techniques for neural music generation, this section summarises the experimental background, discussing and analysing how researches have used these tools in their works. A more comprehensive and detailed discussion is provided in the project investigation report [1].

First of all, the task of generating music with deep learning architectures can take very different paths depending on the choice of the way of representing music files in mathematical terms. Speaking of this, there have been two main trends in working with

music files in the context of deep learning.

The first consists of working with raw audio. For instance, Nayebe et al. (2015) [9] and Kalingeri et al. (2016) [10] 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) [11], ingests input music as raw audio. The advantage of following this path is that the approach is totally unconstrained and can be applied to any kind of music files, regardless of genre, instrument, key-signature and so on. However, significantly greater computational resources are needed and meta-label must be provided for allowing the model to move along the latent vectors representing all the possible genres and styles in the dataset, for instance.

The second possible path for mathematically representing music files is starting from symbolic formats, such as MIDI. This allows for more focused and less risky experiments, which is a crucial need in the context of this project. Boulanger-Lewandowski et al. (2012) [12], for instance, considered a variety of symbolic musical datasets, including MIDI files, for encoding them in a binary piano-roll format, which, unfortunately, cannot keep track of the velocity information. Huang et al. (2016) [13] proposed an alternative representation for MIDI files, consisting of the transformation in sequences of tokens. Despite the greater capability of this approach, the authors encoded each unique combination of maximum three simultaneously played notes across all timesteps as a single token, consequently restricting the composer to the combination of notes already seen by the network and to three-dimensional harmonies. Few years later, Hilscher et al. (2018) [14] proposed to encode each note as a single token for relaxing the constraint of the neural composer. Their model, however, could not track the velocity information. Finally, Payne et al. (2019) [15] created MuseNet, a neural composer that is capable of emulating style features, starting from several MIDI collections. 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. Considering the incredible results of their approach, enhanced by the availability of company-level resources, this project aims at building a much lighter neural composer that still manages to keep track of both notes and velocity. Therefore, only using a small fraction of the dataset they also used, the word-level, multi-instrument and multi-style approach will be replaced by a character-level and mono-instrument approach.

The other crucial aspect to tackle when planning the design and the implementation of a neural music composer is the choice of the neural architecture. Once again, two main trends can be identified: the first consists in the use of gated recurrent neural networks, the second in the use of transformer networks¹.

¹Transformers are neural architectures based only on the attention mechanism, without the recurrent

Despite the most remarkable results in terms of quality and usability of the artificially generated music were obtained in the past two years using transformers - MuseNet (2019) and JukeboxAI (2020), for instance, are both based on complex transformer architectures and trained using networks with millions (sometimes billions) of parameters -, they are out of the scope of this project [1].

Indeed, the focus will be on gated recurrent neural networks, considering both LSTM and GRU cells. Among the researchers that focused on this path, some compared the performances of these two different types of structures [17] [9]. However, overall, it was not possible to demonstrate the superiority of a single type over the other.

Lastly, a significant consideration about all these models that have been cited is that the vast majority of them required experiments to run on GPUs (Graphic Processing Units) or TPUs (Tensor Processing Units).

This suggest that also this project must consider the option of using similar processors².

1.5 Investigation Report Outcomes

Now that the investigation phase of this project has been summarised in Sections 1.3 and 1.4, let us reconsider and list the final outcomes of the investigation report [1] before moving to the actual design and implementation of the neural music composer:

- MIDI format has been chosen for its light, structured and simple nature, compared to raw audio waveforms. The textual representation of MIDI files, consequent to a piano-roll conversion, allows to include more information in training data, such as notes velocity, and fits perfectly in the context of a character-level 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). Both types of recurrent cells will be experimented.
- The implementation of the experiments will be conducted using Python programming language and deep-learning-specialized environments such as Tensorflow and Keras. Previous researches suggest the use of GPU or TPU instances for training the model.

pattern typical of LSTM-based and GRU-based recurrent neural network. See [16]

²This 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

- The qualitative evaluation of the generated music using blindfold tests is a common practice for assessing the plausibility of the music language model. In other words, a panel of human listeners is in charge of evaluating their perception on the nature of both original and artificial music extracts without knowing it in advance.

1.6 Chapter List

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

Chapter 2 Implementation. This chapter describes the design and the building of the neural models and follows the whole data transformation pipeline for training these models, testing them and using them for generating new MIDI extracts.

Chapter 3 Evaluation. This chapter provides and interprets the results of the neural experiments, then discusses the achievements by comparing those results to the objectives and the requirements of the project itself, and, finally, reconsiders legal, social, professional and ethical issues.

Chapter 4 Conclusions. This chapter draws the conclusions of this research report by providing a summary of its content and by discussing some final considerations related to limitations and future improvements.

Chapter 2

Implementation

This chapter leverages the research that has been detailed in the project investigation report [1] and summarised in Chapter 1 as the starting point for describing the design and the practical implementation of the neural music composer.

In detail:

- Section 2.1 illustrates the implementation plan, or design, for the experimental pipeline.
- Section 2.2 expands on the extraction and the transformation of MIDI files with the aim of preparing them for the neural models.
- Section 2.3 details the design of the neural architectures used for learning long-term patterns in text sequences and generating new ones. Furthermore, it also describes the process of converting text files back to MIDI playable extracts.
- Section 2.4 explains the procedures that have been used for assessing the plausibility of the generated music extracts.
- Section 2.5 provides the conclusions for this chapter.

2.1 Implementation Plan

The project investigation report [1] ends with the illustration of the experimental pipeline as planned based on the research that has been detailed in that work. This experimental report confirms the design showed in Figure 2.1.

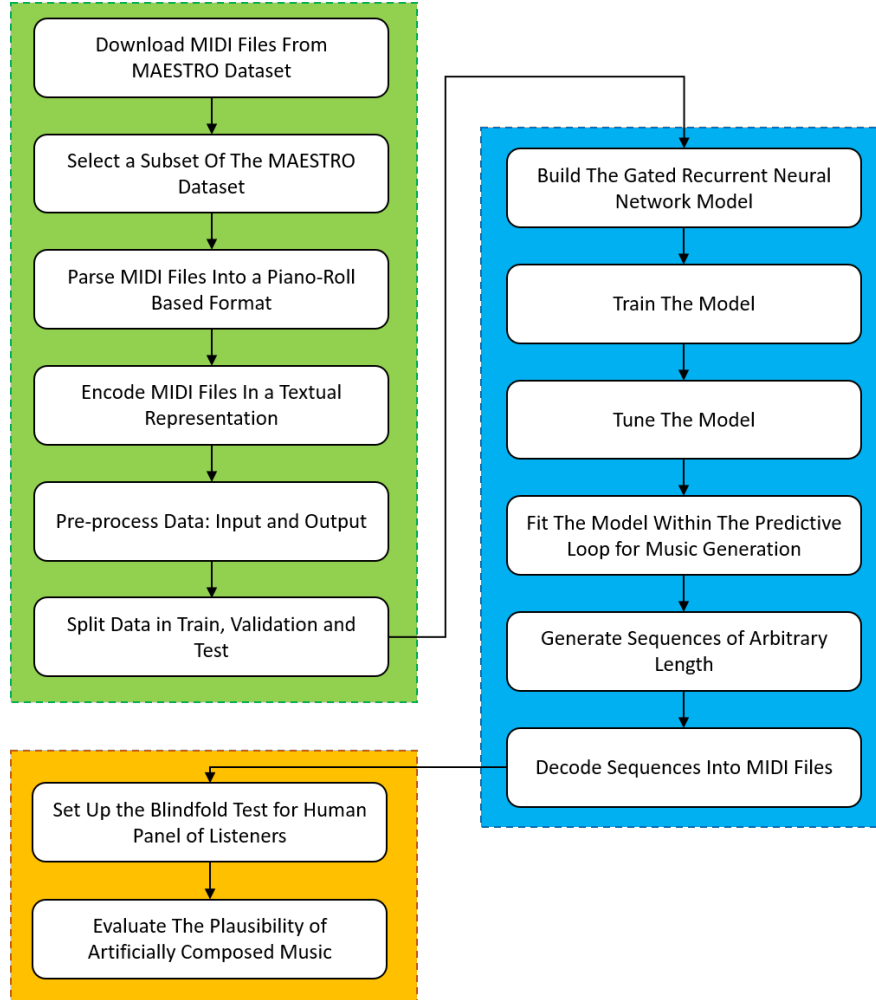


Figure 2.1: The experimental pipeline as planned after the project investigation.

Author: Ludovico Lanni. Date: 10 July 2020.

Green Box: Data Pre-processing.

Blue Box: Neural Experiments.

Orange Box: Plausibility Assessment.

The following sections, each focusing on a different block, will explain in detail the execution of the entire pipeline, which is entirely coded in Python programming language paired with interactive notebook environments, both local, i.e. Jupyter Notebook¹, and

¹Find information about Jupyter Notebook at: jupyter.org

cloud-based, i.e. Google Colab². Indeed, all the I-Python notebooks created for this project have been upload on Github in a repository called neural-music-composer³.

The only task which does not involve any code is the plausibility assessment one: in fact, the final listening game has been developed using Google Form software.

2.2 Data Pre-Processing

This section focuses on the first phase of the practical work, which consists of all the operations that allow to go from the MIDI dataset, in the size and the format it is originally provided with, to the input data and the labels that are used in the following neural architectures.

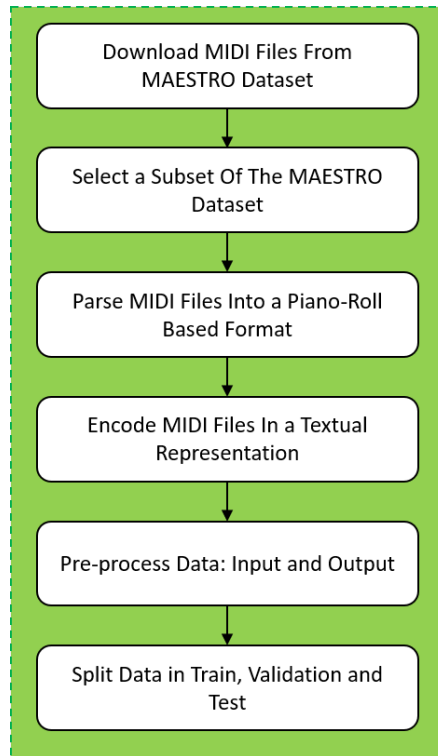


Figure 2.2: Data Pre-processing Pipeline

Author: Ludovico Lanni. Date: 10 July 2020.

This process, as illustrated in Figure 2.2, is mainly made of extraction and transformation operations. The following subsections will provide the details for each of

²Find information about Google Colab at: colab.research.google.com

³Find the neural-music-composer repository at: github.com/ludovicolanni/neural-music-composer

these by describing the work that has been developed in the notebooks *DataExtractionTransformation.ipynb*⁴ and *DataPreparation.ipynb*⁵, both available in the neural-music-composer Github repository.

2.2.1 The Starting Point: The MAESTRO Dataset

The dataset that has been selected as the starting point for this project is called MAESTRO v2.0.0 [18], which stands for the second version of the MIDI and Audio Edited for Synchronous TRacks and Organization.

Starting from this very large collection of classical music piano performances captured with fine alignment between note labels and audio wave forms, the first step was to download only the MIDI repository, since this project do not want to focus on raw audio music files.

After that, the size of the dataset has been significantly reduced to a total of about 4 hours of compositions by only selecting the songs with duration between 2 and 3 minutes. The reason is that, given the necessity of having a relatively small dataset due to the limited resources available for this project, the choice of keeping short full songs instead of extracts of long compositions allows the model to learn more natural and structured patterns. The straight-forward discrimination in duration has been possible thanks to a meta-data tabular file provided together with the MAESTRO dataset that contains side information about all the music files in the collection, included the duration and the suggested split group - whether train, test or validation.

In order to keep the code clean and the work easy to follow from a logical point of view, all the relevant data at this stage have been stored in a dictionary called `music`, which stores the actual MIDI composition, the suggested split and the file name for each sample in the reduced collection.

MIDI files have been read using the *pretty-midi*⁶ library, which among the few available for manipulating this kind of files has revealed itself to be the most efficient and effective for the needs of this project.

⁴Find the notebook at: [neural-music-composer/blob/master/DataExtractionTransformation.ipynb](https://github.com/neural-music-composer/blob/master/DataExtractionTransformation.ipynb)

⁵Find the notebook at: [neural-music-composer/blob/master/DataPreparation.ipynb](https://github.com/neural-music-composer/blob/master/DataPreparation.ipynb)

⁶Find info about pretty-midi at: craffel.github.io/pretty-midi

2.2.2 From MIDI To Piano-roll

The first relevant conversion in the pipeline consists of building the piano-roll representation for each song in the `music` dictionary. Fortunately, the *pretty-midi* library offers a built-in function for obtaining this two-dimensional representation. Therefore, another key was added to the dictionary for holding the piano-roll repository.

When converting MIDI files to piano-roll matrices, the time granularity, or resolution, must be explicitly chosen. In fact, each time-step in the matrix is actually a fraction of a second, called frame, and the density of this frames can be set as a parameter of the conversion function. In this project, the frames-per-second parameter has been set to 8, meaning that the total number of time-steps in the two-dimensional representation of a song in the dataset is eight times its duration in seconds. This resolution is a good trade-off between the need to keep the size of the dataset reasonable, which tries to push the value down, and the need to keep an acceptable quality of the compositions, which tries to push it up.

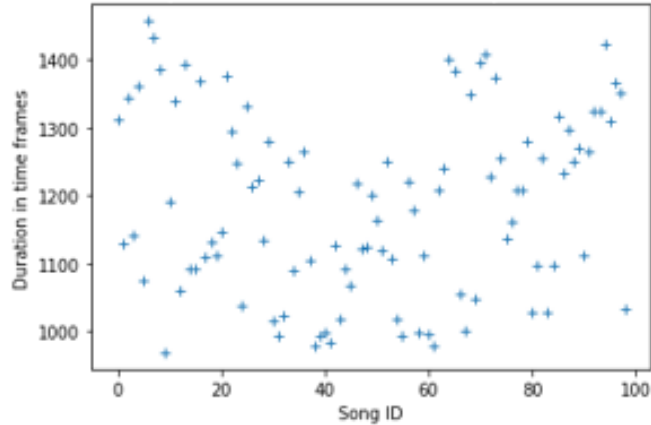


Figure 2.3: Duration of compositions in the reduced version of the dataset with time resolution set to 8 frames-per-second

Author: Ludovico Lanni. Date: 12 August 2020.

Indeed, Figure 2.3 shows that the horizontal dimension of the piano-roll matrices in the `music` dictionary varies between 960 time-frames, i.e. 120 seconds, and 1440 time-frames, i.e. 180 seconds.

2.2.3 From Piano-roll To Text

Now that the repository of piano-roll matrices has been constructed and added to the `music` dictionary, the next step is to perform a second major conversion that takes this repository as input and outputs a symbolic text repository. But what does symbolic text means in this context? The word symbolic indicates that each piano-roll-shaped composition in the dataset is converted into a textual sequence of characters that is not a natural language expression. Instead, the string representing each composition is an ordered set of symbols that simply encode all the information contained in the respective piano-roll, which consists of: pitch, velocity, chords, time-steps.

Table 2.1 illustrates the main concepts behind the conversion process, for which the code is fully detailed in Listing 2.1.

Feature	Piano-roll Matrix	Text String
Pitch Velocity	First dimension index Cell numerical value	Note = "PitchName-NumericalValue"
Chord	Multiple notes at the same time	Chord(t) = "NoteA NoteB ...NoteN"
Time-step	Second dimension index	"Chord(t_0) Chord(t_1) ... Chord(t_n)"

Table 2.1: Details on the conversion from piano-roll to text

```
def from_pianoroll_to_text(data):

    '''This function converts a MIDI pianoroll dataset (in the form of a ↵
        list of 2D numpy arrays) into a text representation (in the ↵
        form of a list of strings). It returns a list of n strings of ↵
        variable length m, where n is the number of songs and m is ↵
        proportional to the number of timesteps in a specific song.
    Each timestep in a song string is separated by a blank space and the ↵
        code for a single timestep is of type 'NOTE_NAME-VELOCITY| ↵
        NOTE_NAME-VELOCITY|...'. Where no notes are played in a certain ↵
        timestep, the corresponding empty character is encoded as e. '''

    l = len(data)
    p = []

    for x in range(l):    # loop over the pianoroll matrices

        test_pr = data[x]    # test piano roll matrix
```

```

n = test_pr.shape[1]

t = []

for c in range(n):      # loop over the columns of the matrix

    test_column = test_pr[:,c]    # test column c

    pitches = np.argwhere(test_column!=0).flatten()
    velocities = test_column[np.argwhere(test_column!=0)].flatten()

    if len(pitches)==0:

        s='e'

    else:

        s=''
        i=0

        while i < len(pitches):

            if i != len(pitches)-1:
                s = s + str(pretty_midi.note_number_to_name(
                    pitches[i])) + '-' + str(int(velocities[i])) +
                    '|'

            else:
                s = s + str(pretty_midi.note_number_to_name(
                    pitches[i])) + '-' + str(int(velocities[i]))

            i = i+1

        t.append(s)

    p.append(t)

p = [" ".join(song) for song in p]

return p

```

Listing 2.1: Function for converting a piano-roll repository into a symbolic text repository

Author: Ludovico Lanni. Full code available at [DataExtractionTransformation.ipynb](#).

Extraction date: 12 August 2020.

In other words, the conventional name of a pitch value and the numerical value for the velocity, or volume, separated by a dash “-” constitute the expression of a single note. Then, multiple notes can be played at the same time forming a chord. This is represented by chaining the different notes using a pipe symbol “|”. Finally, for expressing the temporal dimension made of successive time-steps, or time-frames, the encoded representation of two adjacent frames consists of separating their individual representations using a blank space. Note that the function shown in Listing 2.1 also contains a support mechanism for empty time-steps, i.e. those time-frames where no note is played. This frequently happens when the resolution is more than 1 frame-per-second, therefore it is something that the function needs to deal with. The solution is encoding this type of time-steps using the character “e”, which stands for *empty*.

To make the explanation of the process more tangible, Figure 2.4 shows a practical example.

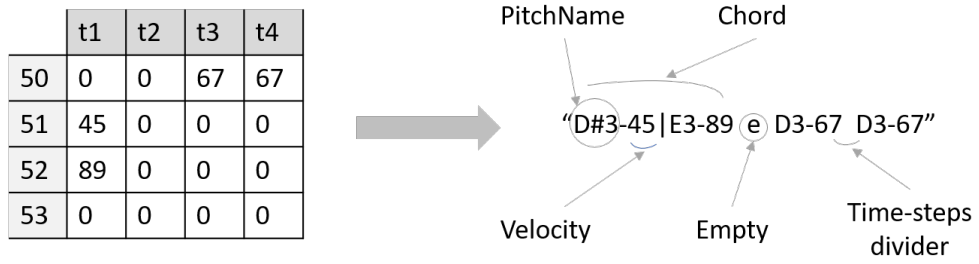


Figure 2.4: Example of conversion from piano-roll matrix to text

Author: Ludovico Lanni. Date: 12 August 2020.

Thanks to the application of the conversion function, the new textual repository is stored with a dedicated key in the `music` dictionary.

2.2.4 From Text To Binary Data

At this point in the transformation pipeline the goal is to one-hot encode the textual representation of music files in such a way that the input can be used by a neural network.

As mentioned in Subsection 1.3.3, the neural composer will be based on a character-level representation: therefore, each character, i.e. each token, must be converted to a binary vector. While engineering and testing the code for this task, the Google Colab environment did not manage to support the whole textual representation of the initially reduced version of the original dataset. In fact, due to these memory concerns, the size of the dataset was once again reduced from 99 to 34 compositions by selecting only

those where the textual representation was made of less than 20000 characters. Figure 2.5 shows the reduction in the number and the character-wise length of the text strings from the initially reduced dataset and the final text repository.

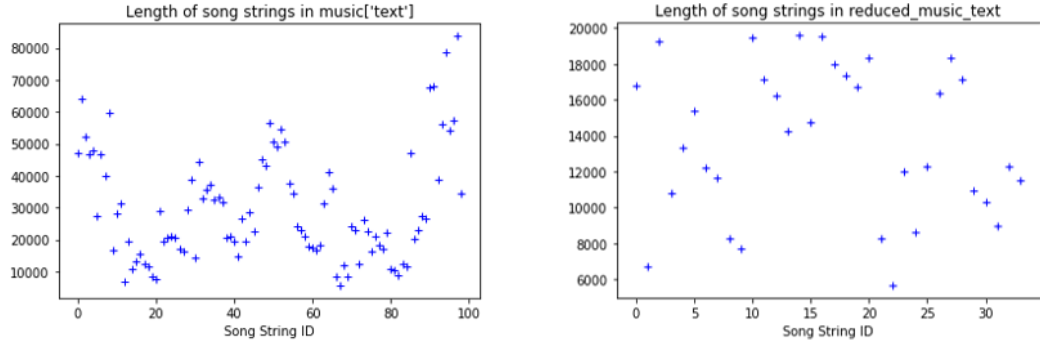


Figure 2.5: Second reduction of the size of the original dataset.

Author: Ludovico Lanni. Full code available at [DataPreparation.ipynb](#).

Extraction date: 12 August 2020.

Once the final text repository is obtained, it is possible to extract the vocabulary of the collection, which is the set of all the distinct characters considering the whole repository. In our context, the vocabulary contains 22 unique characters.

Now, the challenge is to code a function that is able to:

- Take the text repository as input.
- Split each text string, i.e. each composition, into overlapping sub-strings of a certain length using a step parameter for sliding the window over the full string. In this project the length of each sample has been set to 512 for optimizing the trade-off between the need to include as much information as possible in a single sequence and the need of keeping the number of recurrent cells reasonable in terms of computational costs. The step parameter has been set to 3.
- Identify the next character for each sub-string.
- One-hot-encode each sub-string as a 2-dimensional binary array of shape `(substring_length, vocabulary_size)` and collect all of them in a 3-dimensional array.
- One-hot-encode the next character of each sub-string as a 1-dimensional binary array of shape `(vocabulary_size)` and collect all of them in a 2-dimensional array.

This function is reported in Listing 2.2.

```
def char_one_hot_encode(data, vocabulary, maxlen=512, step=3):

    ''' This function divides each element in a list of strings into ↵
        sequences of length <maxlen> with sample step of size <step>, ↵
        identifying the next character for each of these sequences.
        Using the <vocabulary> dictionary, it returns two objects:
        x: the 3D binary array of sequences where each character has been one- ↵
            hot encoded;
        y: the 2D binary array of the next one-hot encoded character for each ↵
            sequence in x'''

    # Length of extracted character sequences
    maxlen = 512

    # We sample a new sequence every 'step' characters
    step = 3

    # This holds our extracted sequences
    sentences = []

    # This holds the targets (the follow-up characters)
    next_chars = []

    for text in data:

        for i in range(0, len(text) - maxlen, step):
            sentences.append(text[i: i + maxlen])
            next_chars.append(text[i + maxlen])

    print('Number of sequences:', len(sentences))

    # Next, one-hot encode the characters into binary arrays.
    print('Vectorization...')
    x = np.zeros((len(sentences), maxlen, len(vocabulary)), dtype=np.bool)
    y = np.zeros((len(sentences), len(vocabulary)), dtype=np.bool)
    for i, sentence in enumerate(sentences):
        for t, char in enumerate(sentence):
            x[i, t, vocabulary[char]] = 1
            y[i, vocabulary[next_chars[i]]] = 1

    return x,y
```

Listing 2.2: Function for converting a symbolic text repository into a binary and model-ready repository

Author: Ludovico Lanni. Full code available at [DataPreparation.ipynb](#).
Extraction date: 12 August 2020.

To sum up, at this point the function for performing the final conversion of this phase has been coded and is ready to be applied to symbolic text repositories, using the previously stored vocabulary.

2.2.5 Train-Validation-Test Split

Instead of first applying the conversion function to the whole text repository and then splitting the obtained arrays into train, test and validation data, the strategy that has been implemented consists of splitting first and converting after. In addition to the way the conversion function has been coded, the reason is the possibility of easily leveraging the suggested split of the original MAESTRO dataset, which already fights the risk of having the same composition, played by different people, in different split-groups. This would create a trivial bubble in the model, leading to non-trustable performance measures.

The result of the splitting process assigned 26 compositions to the training set, 4 to the validation set and 4 to the test set. Then, as mentioned before, each of these three text repositories is plugged into the function of Listing 2.2, which generates the input data and the labels that can be used for training and testing the neural models.

2.3 Neural Experiments

This section focuses on the second major phase of the practical work and describes the whole process of building and applying the generative neural architecture and converting the output of the models into MIDI playable format. Figure 2.6 illustrates the key passages of this working pipeline.

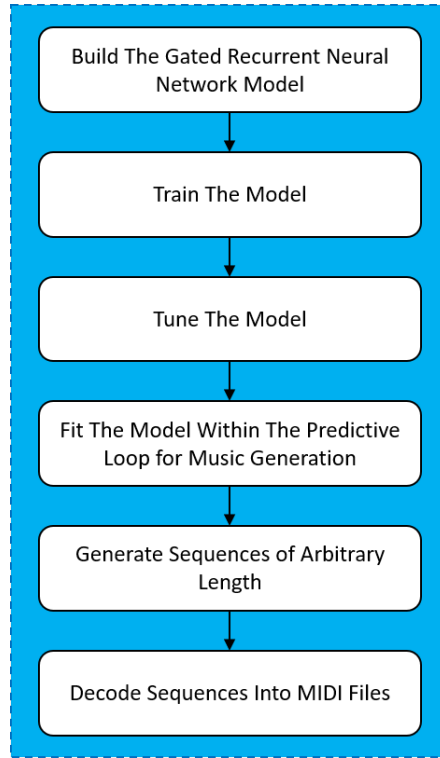


Figure 2.6: Neural Experiments Pipeline

Author: Ludovico Lanni. Date: 10 July 2020.

The following subsections discuss the process in depth by referring to the code that has been produced in these three notebooks, always available in the neural-music-composer repository on Github: *GatedRnnModel.ipynb*⁷, *SequencesGeneration.ipynb*⁸, *Text2Midi.ipynb*⁹.

2.3.1 Environment And Resources

First of all, it is crucial to provide details about the environment and the resources used for the following experiments. As mentioned before, the work has been conducted using Python programming language in the cloud-application called Google Colab. The default setting on this platform does not involve the use of any hardware accelerator: the code is normally running on CPU and memory consists of about 12 GB. This setting has been used for almost every notebook, apart from *GatedRnnModel.ipynb*, where the neural models are built and trained.

In fact, the considerable size of the models would have required them to train for more

⁷Find the notebook at: [neural-music-composer/blob/master/GatedRnnModel.ipynb](https://github.com/neural-music-composer/blob/master/GatedRnnModel.ipynb)

⁸Find the notebook at: [neural-music-composer/blob/master/SequencesGeneration.ipynb](https://github.com/neural-music-composer/blob/master/SequencesGeneration.ipynb)

⁹Find the notebook at: [neural-music-composer/blob/master/Text2Midi.ipynb](https://github.com/neural-music-composer/blob/master/Text2Midi.ipynb)

than 15 hours each if executed with the default setting. Consequently, both the LSTM-based and the GRU-based neural models are built and trained using Google Colab’s free TPU hardware accelerator, which consists of 8 TPU cores working in parallel. This allowed to save an incredible amount of time and deliver the project before the deadline.

Although the monetary cost of using the TPU accelerator was absent, some extra time was spent on optimizing the code for this setup. Given that Tensorflow 2.2.0 and Keras 2.4.3 are the deep learning frameworks used in this project, the optimization procedure also consisted of converting the input data and the labels from numpy arrays into tensorflow datasets. In addition, these datasets have been batched in 128 samples per TPU device, for a total of 1024 samples per batch.

Finally, with respect to the actual building the training of the models, the code was adjusted to be encapsulated in a `strategy.scope()` object which associated it to the TPU hardware accelerator.

2.3.2 Models Design

Two gated recurrent neural networks have been built in the development of this project: the first, namely `lstm_1`, uses a 512-deep LSTM block, and the second, `gru_1`, uses a 512-deep GRU block. As explained in Subsection 2.2.4, the number of cells in a gated recurrent block, i.e. 512, equals the length of the input sequences of characters. This was suggested by Hilscher and Shahroudi (2018) [14] that empirically found this relation to provide better results.

Layer (type)	Output Shape	Param #
Input (InputLayer)	[(None, 512, 22)]	0
LSTM (LSTM)	(None, 512)	1095680
Output (Dense)	(None, 22)	11286
Total params: 1,106,966		
Trainable params: 1,106,966		
Non-trainable params: 0		

Figure 2.7: Structure of the LSTM-based neural network.

Author: Ludovico Lanni. Date: 13 August 2020.

Layer (type)	Output Shape	Param #
Input (InputLayer)	[(None, 512, 22)]	0
GRU (GRU)	(None, 512)	823296
Output (Dense)	(None, 22)	11286
Total params: 834,582		
Trainable params: 834,582		
Non-trainable params: 0		

Figure 2.8: Structure of the GRU-based neural network.

Author: Ludovico Lanni. Date: 13 August 2020.

As Figures 2.7 and 2.8 show, the LSTM-based model has a larger number of trainable parameters than the GRU-based model: this is due to the fact that the architecture of a GRU cell is less complex than that of an LSTM cell, with the first having one gate less than the second (see also Subsection 1.3.2).

The last layer of both models is a *softmax* layer, meaning that the problem is formulated as a classification task where the network outputs the probability of being the next character of the input sequence for all the characters in the vocabulary.

In addition, for fair comparison reasons, both models are compiled using the same loss function, i.e. Keras’s `categorical_crossentropy`, and the same optimizer, i.e. `RMSprop` with a starting learning rate of 0.01.

2.3.3 Training And Tuning

The training process have been refined using Keras’s callbacks, which allow for smarter procedures, better monitoring and in-training hyper-tuning. Listing 2.3 provide details about the type of callbacks that have been implemented by printing the actual code used for setting them up with respect to the LSTM-based model.

```
callbacks_list_lstm = [
    tf.keras.callbacks.EarlyStopping(
        monitor='acc',
        patience=5,
    ),
    tf.keras.callbacks.ModelCheckpoint(
        filepath='/content/drive/My Drive/RNN_models/lstm_1_100.h5' ←
        ,
        monitor='val_loss',
        save_best_only=True,
    ),
]
```

```
tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.1,
    patience=2,
),
tf.keras.callbacks.CSVLogger(
    filename = '/content/drive/My Drive/RNN_models/ ↵
        log_lstm_1_100.csv',
    separator=',',
    append=False
),
tf.keras.callbacks.TerminateOnNaN()]
```

Listing 2.3: List of callbacks used when training the models.

Author: Ludovico Lanni. Full code available at [GatedRnnModel.ipynb](#).

Extraction date: 13 August 2020.

In detail:

- **EarlyStopping** ensures that the training process stops when the accuracy score is not improving for more than 5 epochs in a row.
- **ModelCheckpoint** saves the current best model after each epoch based on the validation loss score.
- **ReduceLROnPlateau** dynamically tune the learning rate parameter by multiplying it using a factor of 0.1 when the validation loss is not improving for more than 2 epochs in a row.
- **CSVLogger** saves the history of the training procedure in a csv file.
- **TerminateOnNaN** stops the training procedure when the loss becomes undefined.

Once defined the callbacks, both models have been trained for an initial target of 100 epochs using the sequences originated by the 36 compositions in the training dataset for computing the training metrics and those coming from the 4 compositions in the validation set for calculating the validation metrics.

The LSTM-based network stopped training after 45 epochs and was evaluated on the test set of sequences achieving a test accuracy of 52,57%. The GRU-based network trained for 40 epochs achieving a test accuracy of 54,25%. Table 2.2 summarises the performance of the two final models.

Model	E	Tr Acc	Tr Loss	Val Acc	Val Loss	Te Acc	Te Loss
lstm_1	30	0,6762	0,8760	0,5476	1,2300	0,5257	1,2765
gru_1	31	0,6578	0,9313	0.5528	1.1678	0,5425	1,2059

Table 2.2: Training, validation and test scores for the final models.

E=epoch, Tr=training, Val=validation, Te=test, Acc=accuracy

As Table 2.2 indicates, the LSTM-based model performs slightly better in terms of training measures, but the GRU-based model appear to be more accurate when tested on new data. However, both models show some signs of over-fitting training data, but in the context of music generation this is not a major issue as long as it is not severe.

2.3.4 Predictive Loop And Sequences Generation

The two models that have been trained and tested so far are not capable of generating sequences of arbitrary length yet. In fact, they have learnt the distribution of characters in the symbolic text sequences representing music compositions and this knowledge must be now exploited for generating new data.

The strategy here consists of using the trained model for predicting the distribution of the next character of an input seed sequence, then sampling from this distribution and repeating this process by adding the predicted character at the end of the seed sequence and dropping the first one. Few things need to be specified in order to implement this predictive loop for sequences generation.

The sampling function, for instance, determines how the next character of a sequence is picked given the distribution that the model outputs. The easiest implementation here would be selecting the character with the highest probability. Although this is not technically wrong, it may not give the result that are expected in a text generation task. In fact, the generated sequence can easily fall in a boring loop or even contain just one character repeated from start to finish. In other words, this solution has zero creativity. So, what could be the solution? The answer relies in what is called sampling with temperature, which stands for the process of recomputing the probability distribution initially calculated by the model using an entropy factor, i.e. the temperature.

$$q_i(temp) = \frac{e^{\frac{\ln p_i}{temp}}}{\sum_i e^{\frac{\ln p_i}{temp}}} \quad (2.1)$$

Equation 2.1 shows how the probability of a character being the next one in the input sequence is recomputed using temperature. Indicatively, the more the temperature

value is closer to zero, the more the generated sequence will be naive and rational. In contrast, the higher the temperature the greater the creativity of the model. With respect to this project, four different values - 0.01, 0.2, 0.5 and 0.8 - of temperature have been tested and 0.5 was judged to be the optimal value in terms of plausibility-creativity trade-off. Therefore, after generating initial sequences using all four different values, the focus then shifted only on the implementation of the predictive loop with temperature equal to 0.5.

Another crucial parameter to be specified before applying the predictive loop for sequences generation is the length of the generated sequences. This is a completely arbitrary parameter: the set up that has been used in this project consists of generating 1500 new characters. Note that this does not imply that the generated music extracts, once converted from text, will all have the same duration in seconds. This is due to the fact that the number of characters in a time-step can vary a lot depending on the notes it contains.

The last important aspect to cover before running the predictive loop is deciding what sequences to feed to the network. Since the goal of this project is to generate creative music and since the trained networks appeared to slightly over-fit training data, the most reasonable thing to do is feed them using never-seen-before seed sequences. The easiest way to do that is to randomly pick a new one in the test set at the start of each generation process.

At this point the predictive loop can be coded and implemented in order to generate brand new sequences. The full code can be found in the notebook called *Sequences-Generation.ipynb*, always belonging to the neural-music-composer Github repository. A useful feature that has been added to the code is that the output of the predictive loop is already in symbolic text format thanks to an inner binary-to-text conversion mechanism.

To sum up, more than 10 new sequences of 1500 characters have been generated by each of the two models and they are already in symbolic text format.

2.3.5 From Text To MIDI

The last step before being able to play the artificially generated music files consists of converting the generated sequences from symbolic text to MIDI format. The challenge here is coding the inverse process of what has been done when converting MIDI files into piano-roll matrices (Subsection 2.2.2) and, then, these ones into symbolic text (Subsection 2.2.3).

The pipeline is organized in three main steps:

- Convert symbolic text to piano-roll matrix. This step is performed by the function shown in Listing 2.4 that focuses on reversing the operations performed by the function in Listing 2.1 for decomposing the syntax of the text sequences and reconstructing their 2D mathematical representation.

```
def text_to_pianoroll(test):

    ''' This function converts a symbolic music textual sequence, ↵
        namely test, into a piano-roll matrix of shape (pitches, ↵
        timesteps) '''

    # string to list of strings (each element will be a column of ↵
    # the matrix)
    list_s = test.split(" ")[1:-1]

    # initialise the matrix
    mat = np.zeros(shape=(128,len(list_s)))

    # loop over each string of the main string

    for c in range(len(list_s)):

        test_element = list_s[c]

        if test_element=='e':
            mat=mat

        elif test_element=='':
            mat=mat

        else:

            # get the pitches and the velocities

            # decompose the string
            test_element_decomposed = test_element.split('|')

            # fill the matrix with velocities for the right pitch values

            for i in test_element_decomposed:

                pit_vel = i.split('-')

                pitch = pretty_midi.note_name_to_number(pit_vel[0])
                velocity = float(pit_vel[1])
```

```

    if pitch in range(mat.shape[0]):

        mat[pitch, c] = velocity

    else:
        mat=mat

    return mat

```

Listing 2.4: Function for converting a symbolic text string into a piano-roll matrix

Author: Ludovico Lanni. Full code available at [Text2Midi.ipynb](#).

Extraction date: 14 August 2020.

- Plot piano-roll matrix. Here the material available on the official *pretty-midi* Github repository offered the code for a simple function that plots the piano-roll matrix as it appears in a digital-audio-workstation. This step is useful for assessing both the duration in seconds of the generated music extract and its quality. In fact, not all the generated samples will be converted to MIDI files, but also those that show a qualitative and plausible piano-roll representation. Figure 2.9 shows a comparison between an extract with a good piano-roll representation and one that is likely to be discarded. It has been empirically found that, given the experimental set up for this project, the best extracts last between 7 and 12 seconds: in fact, these samples have a good balance between presence of notes and silent pauses. In contrast, shorter samples are not usable for plausibility assessment and longer samples are mainly made of silent pauses, as clearly visible in Figure 2.9.
- Convert piano-roll matrix to MIDI file. This step has been performed thanks to slight modifications operated to the `piano_roll_to_pretty_midi` function, available in the official *pretty-midi* repository on Github. As said before, only the extracts whose piano-roll was judged qualitative were converted into MIDI files in order to proceed to the plausibility assessment test.

At the end of this last conversion pipeline, it has been possible to play the MIDI files and the 4 extracts, 2 per model, that appeared to be more plausible were selected for taking part in the listening game, which will be discussed in the following section.

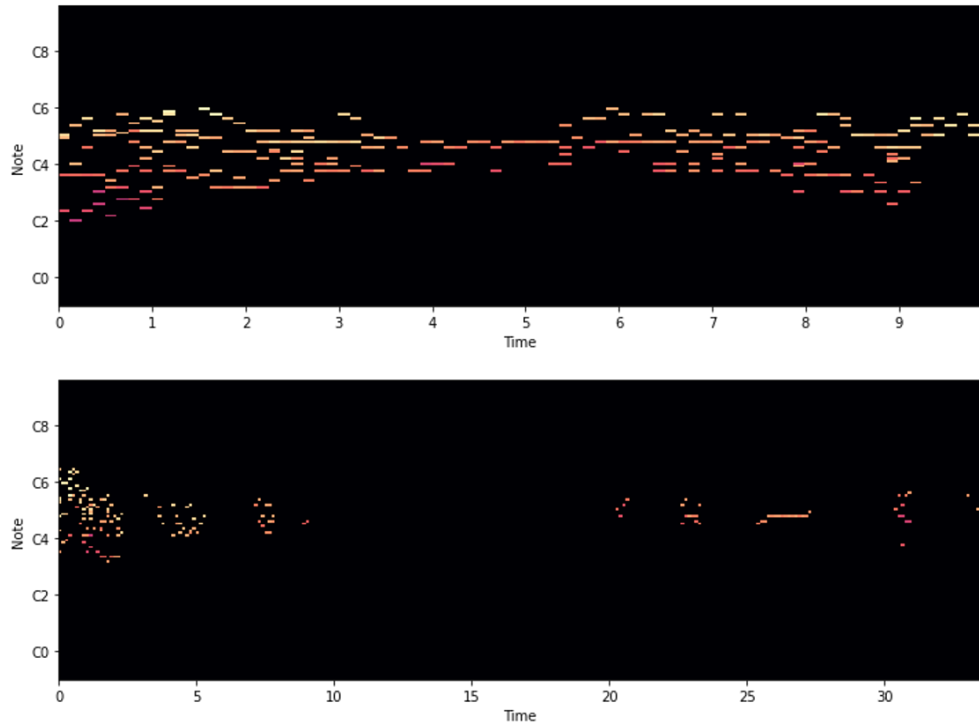


Figure 2.9: Piano-roll representation: a qualitative extract (Top) and a poor extract (Bottom)

Author: Ludovico Lanni. Date: 14 August 2020.

2.4 Plausibility Assessment

This section focuses on the last bit of the experimental pipeline of this project (Figure 2.10, i.e. on the methods used for assessing the plausibility of the artificially generated music extracts).

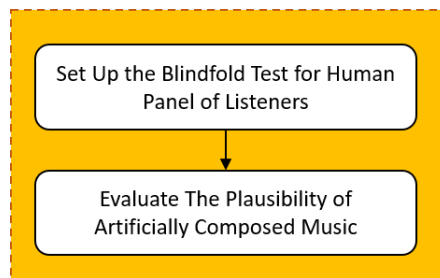


Figure 2.10: Assessment Pipeline

Author: Ludovico Lanni. Date: 10 July 2020.

The main task to satisfy this need consists of engineering a proper evaluation test, which

aims at leveraging people’s perception of what sounds like composed by a machine learning model and what sounds like composed by a human. Therefore, the test should include not only the selected qualitative music extracts that were generated by the models, but also comparable-in-duration extracts from the original human-composed songs. In addition, naturally, the panel of listeners should not know whether the sample they are trying to assess in terms of plausibility is from one group or the other.

These initial considerations led to the creation of a listening game where players are in charge of assigning a score to a composition they are told nothing about at listening time, where the score reflects their confidence on whether the extract they are listening to was composed by a human or by a machine learning model.

The details of this method are covered in the next subsection, while the results are presented and discussed in Chapter 3.

2.4.1 The Listening Game

The listening game¹⁰ has been implemented in Google Form, which offers an easy way for creating file-based questions, for sharing the test among potential players and, finally, for looking at the aggregated results.

The key part of the instructions reported in the introduction to the game consists of the following:

“Here you will listen to a total of 8 short music extracts played with piano. Some of them are extracted from original songs composed and played by humans, others are generated using artificial intelligence architectures.

You are asked to rate each extract from 1 to 5 , where 1 means you are sure the music was composed by a human and 5 means you are super confident that the music is the output of the artificial intelligence model.”

The four extracts coming from the artificial generation experiments consist of two compositions generated by the LSTM-based model and two generated by the GRU-based model. The four extracts coming from the original human-made compositions, instead, are selected by randomly trimming a comparable-in-duration part of four randomly picked songs from the original MAESTRO dataset.

As mentioned before, the samples last about 10 seconds each and their source is hidden to the player of the listening game, which therefore consists of eight questions, one for each music extract. The players are asked to try to answer each questions individually

¹⁰The listening game can be found at the following link: forms.gle/zA6PevncUQRa8JhQ8

before moving to the next one, but they have the possibility of listening to the extracts multiple times during the test. In order to fight the bias generated by keeping the same order in the list of questions displayed to the players, this list is always shuffled before being displayed.

Finally, in order to avoid people from making calculations about the number of the samples they think are human-composed and those they think are generated by the deep learning model, they are not told how many samples are actually from one group or the other.

Figure 2.11 illustrates an example of a question as it appears to players.

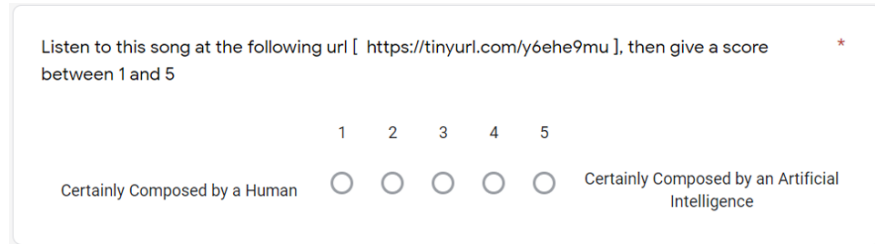
The image shows a user interface for a listening game. At the top, it says "Listen to this song at the following url [https://tinyurl.com/y6ehe9mu], then give a score between 1 and 5" followed by a red asterisk. Below this, there are five radio buttons labeled 1, 2, 3, 4, and 5. Under the radio buttons, the text "Certainly Composed by a Human" is on the left and "Certainly Composed by an Artificial Intelligence" is on the right. The radio buttons are currently unselected.

Figure 2.11: Example of a question from the listening game

Author: Ludovico Lanni. Date: 14 August 2020.

The way this plausibility assessment test has been engineered offers not only the possibility of evaluating the quality of the artificially generated music by comparing their in-game performance to the human-composed extracts, but also the possibility of comparing the effectiveness of the LSTM-based model to the effectiveness of the GRU-based model.

2.5 Conclusions

This chapter focused on the practical phase of the project.

First of all, the implementation pipeline diagram has been illustrated in order to provide a general view of the experiments design.

Then, the process has been described in detail for each of its three main phases:

- Data pre-processing. Here the focus was on the extraction, the transformation and the preparation of data, going from the original MAESTRO dataset in MIDI format to binary mathematical representations that could be fed to the neural networks as input and labels.

- Neural experiments. Here the main tasks were: building two character-level gated recurrent neural networks, one LSTM-based and one GRU-based; hyper-training them and selecting their best configuration; including them in a predictive loop for generating sequences of 1500 characters; finally converting those sequences back into MIDI format and selecting the most qualitative ones for the plausibility test.
- Plausibility assessment. Here a listening game was engineered and implemented in Google Form for testing people’s opinion on the plausibility of the generated music extracts. The game consisted of eight blindfold questions where, for each of these, the player was asked to listen to an extract and assigning a score from 1, if sure that it was human composed, to 5, if sure it was artificially generated. The set of music extracts was made of four artificial samples and four original samples.

Chapter 3

Evaluation

This chapter introduces and discusses the results of the project implementation and analyses them in a broader perspective by comparing the achievements with the aim and the objectives of this work, as declared in the investigation report [\[1\]](#). Moreover, it contains a potential reconsideration of legal, social, ethical and professional issues, including some reflections on risks and safety.

3.1 Results

This section focuses on presenting the aggregated results of the plausibility test that was conducted in the form of a listening game, as described in Subsection 2.4.1, and discusses the performance of the gated recurrent neural models by comparing the use of GRU and LSTM structures.

3.1.1 Listening Game Scores

The listening game consisted of a blindfold test where people were asked to assign a score to each of the eight music extracts in the test. The score, going from 1 to 5, reflected their perception on whether the music had been human-composed or, otherwise, made by an AI. Among the eight compositions, four came from the MAESTRO dataset, two were generated by the LSTM-based model and two by the GRU-based model.

The Google Form containing the game has been shared among a network consisting of family and academic connections. Without restricting the number of potential players, a total of 53 people responded to the quiz. Table 3.1 illustrates the aggregated results.

		Scores				
file	origin	1	2	3	4	5
midi_9	LSTM-1	5,7%	7,5%	22,6%	26,4%	37,7%
midi_10	LSTM-2	9,4%	13,2%	7,5%	43,4%	26,4%
midi_11	GRU-1	17%	13,2%	22,6%	24,5%	22,6%
midi_12	GRU-2	15,1%	17%	17%	30,2%	20,8%
midi_5	ORG-1	22,6%	35,8%	20,8%	13,2%	7,5%
midi_6	ORG-2	15,1%	22,6%	7,5%	30,2%	24,5%
midi_7	ORG-3	47,2%	26,4%	7,5%	13,2%	5,7%
midi_8	ORG-4	49,1%	26,4%	9,4%	13,2%	1,9%

Table 3.1: Aggregated results of the listening game with 53 players.

1=surely human-composed, 5=surely artificially-composed

Generally speaking, listeners managed to guess the right answers for 7 out of 8 extracts, with the exception of an original music extract (*midi_6*) that was judged to be artificially composed. This means that the majority of the players correctly spotted the artificial samples, having a clear idea on how human-composed music should sound like.

Going into detail, however, it is evident that spotting model-shaped extracts was not an easy task. For example, a total percentage of 35,8% of players were not able to

correctly mark *midi_9* sample as artificial, while 30,1% of them did the same mistake with the other LSTM-generated output, *midi_10*. These figures become even more significant when looking at the GRU-based extracts: the majority of people (52,8%) were incapable of marking *midi_11* as artificially-generated, while 49% of them did the same with *midi_12*.

In other words, artificially-generated samples were difficult to spot, with GRU-based ones - and in particular *midi_11* - being by far the trickiest examples. Figure 3.1 shows the visualization of the answers for *midi_11* as available on Google Form.

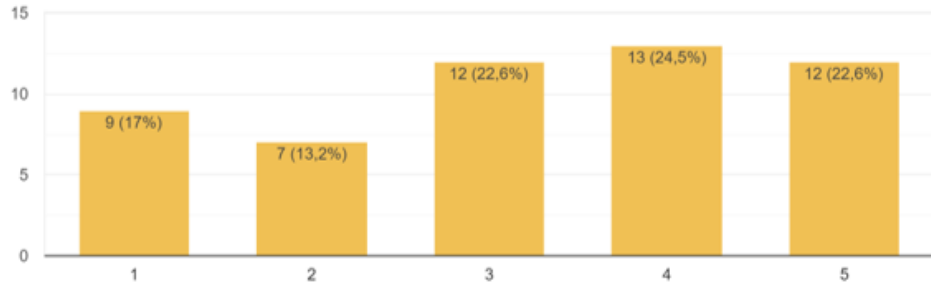


Figure 3.1: Visualization of the answers for the GRU-generated extract named *midi_11*.

Author: Ludovico Lanni. Date: 17 August 2020.

It is important to note that the players scores are totally subjective and are affected by their knowledge and/or perception on what artificial intelligence is and how artificially-composed music should sound like. For this reason, it may happen that some biases arise: for example, one may think that artificially-composed music should sound perfect and emotionless and may be pushed to unintentionally give the wrong score based on this belief. In order to reduce the misleading effect of these potential cases, the panel of listeners have been freed from any constraint on their education, age, sex, job title and so on.

3.1.2 Traditional Metrics vs Human Perception: GRU or LSTM?

The previous subsection mentioned the fact that the plausibility of the extracts generated by the GRU-based model was significantly higher than the plausibility of those generated by the LSTM-based model.

This subsection aims at cross-analysing this result with the training, validation and testing scores of the two models, which were presented and discussed in Subsection 2.3.3, Table 2.2.

On the one hand, traditional and quantitative performance metrics show that the LSTM-based model better fit training data but is slightly worse than the GRU-based model at understanding the statistical structure of new music sequences. On the other hand, human perception, expressed by the players of the listening game, show that the extracts generated by the LSTM-based model are significantly easier to spot, meaning that those generated by the GRU-based model are far more capable of tricking humans in terms of plausibility. Therefore, it seems that the model that over-fits the most training data is the one associated with less human-like compositions. From a machine-learning-specific point of view, this is not a surprise: over-fitting is always considered as an issue for the model and needs to be addressed during training due to its consequences that lead to lower capability of generalization. However, in the context of music-composition, over-fitting is something natural that may not lead to worse results in terms of music plausibility. In fact, the history of music itself shows signs of patterns repetition across songs of different ages, genres or artists.

Given the set up of the experiments in this project, the reason for the better performance of GRU-related extracts does not seem to be related to over-fitting issues of the LSTM-based model. Moreover, even if it shows better scores with unseen data, also the GRU-based one over-fits training data.

In other words, in order to be able to conclude something backed-up by theory or evidence on the difference in performance between LSTM and GRU in this project, the traditional metrics and the human perception assessment may not be enough. As a suggestion for researchers who want to take this project further, the listening game could be modified by asking players to write a short motivation on the elements that affected the score they assigned to each composition. That way, interesting and unexpected insights in this evaluation sphere may arise.

3.2 Objectives-Achievements Comparison

This section focuses on discussing the objectives and the aim of this project, as stated in the investigation report [1], by reflecting on the comparison with the achievements of the experimental sections in this report.

In order to carry this type of analysis, each objective in the list from the investigation report will be displayed and commented based on whether it has been achieved or not.

1. *To collect simple and structured piano composition in MIDI format.* The objective has been achieved: the MAESTRO dataset has been selected as the starting point of the experiments.

2. *To select a sub-sample of the compositions so that the future processing and learning can match the constraints in resources and time.* The objective has been achieved: the MAESTRO dataset has been reduced in size twice due to limitations in resources.
3. *To produce a mathematical and model-ready representation for MIDI files.* The objective has been achieved: MIDI samples have been converted first into piano-roll, then into symbolic text and finally into one-hot-encoded tensors.
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.* The objective has been achieved: both LSTM and GRU cells have been experimented and included in a predictive loop for generating new sequences in symbolic text format.
5. *To evaluate the plausibility of the generated music from the perspective of a sample of human listeners.* The objective has been achieved: the test has been modelled as a listening game where 53 players expressed their perception on the music extracts.

Finally, also the aim of the project has been achieved: *to artificially generate music, using light end-to-end deep learning architectures, in such a way that the output sounds plausible to human listeners.*

3.3 Requirements-Achievements Comparison

The focus of this section is on discussing the functional and non-functional requirements for this project, as stated in the investigation report [1], by reflecting on whether they have been met or not.

In order to carry this type of analysis, each requirement in the list from the investigation report will be displayed and commented.

With respect to the functional requirements:

- *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 requirement has been met: the neural models ingested MIDI compositions encoded in binary sequential format and demonstrated the capability of understanding the syntax of the music files.
- *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 requirement has been met: the networks demonstrated the ability of generating sequences of arbitrary length given a seed sequence by predicting one character for each iteration.

- *The generated sequences should be converted to MIDI format in order to be played and passed to the panel of listeners.* The requirement has been met: The vast majority of the generated sequences in symbolic text format could be converted into MIDI files thanks to their perfect syntax.

With respect to the non-functional requirements:

- *The neural architectures should be trainable with a limited set of resources in time, computational power and volume of data.* The requirement has been met: both models were built with a single recurrent layer of 512 cells and were trained in less than 30 minutes thanks to the use of freely-available TPU units and to the limited size of the training datasets.
- *The accuracy of the final model should beat that of a very simple and naive baseline.* The requirement has not been practically demonstrated: a naive baseline, such as a model that predicts the most frequent chord in the song, has not been tested. However, the plausibility of such model would be absolutely null, while the models that have been implemented demonstrated the ability of confusing the panel of listeners.
- *The plausibility of generated music should not be perfectly distinguishable from that of original compositions by the panel of listeners.* The requirement has been met: no generated extract was perfectly classified as artificial by the players of the listening game.

3.4 Reflections on Potential Issues, Risk and Safety

This section formally reconsiders the potential issues and risks related to the development of this project, which have been already addressed in the investigation report [1].

From a legal point of view, this report confirms that no issues have been identified. 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, this report confirms that no living human or animal is

directly involved in the development of this project. Moreover, the only interaction with humans is for evaluating the plausibility of the generated music as part of the listening game, but no personal information is saved. In addition, the game has been shared among family and academic networks without any ethical bias.

From a social point of view, this report confirms 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 report confirms that the 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.

With respect to risks 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. This report confirms that all actions and experiments have been taken in respect of social distancing and safety measures.

Chapter 4

Conclusions

This chapter provides the conclusions for this body of work by first summarising the purpose and the outcomes of this research and then discussing the possibilities of extending the work by identifying its limitations in order to suggest future research paths in this field.

4.1 Summary

This report described the design and the building of a neural music composer that uses gated recurrent neural networks in order to learn patterns from MIDI sequences encoded in symbolic textual format, generate sequences of characters and, finally, play the artificially composed music for testing its plausibility against human-composed extracts.

The background and the specifications of this project have been extensively discussed in the investigation report [1], but the most relevant concepts have been summarised also in Chapter 1. Here the theoretical background highlighted the principles of music theory, the essence of the MIDI standard, the structure of gated recurrent cells and their role within generative deep learning architectures such as neural language models. Moreover, the experimental background compared several solutions already existing in literature for building a neural music model: here the use of MIDI format has been justified for its simplicity and scalability, the character-level neural music models based on gated recurrent cells like LSTM and GRU have been adopted as the key methodology for the development of this project and, finally, the need for specific kind of resources - such as hardware accelerators - has been quantified.

The practical implementation of the neural music composer has been described in detail in Chapter 2. The whole transformation, learning and testing process has been accompanied with some of the code from the neural-music-composer public repository¹ on GitHub, where all the work has been executed and saved. A subset of the MIDI files coming from Google’s MAESTRO dataset have been parsed and converted first into 2D piano-roll format, with a resolution of 8 frames-per-second, and then into textual format, keeping the information about both chords and notes velocity. Text sequences have been considered at character-level and then encoded in binary format for being ingested by a neural model. After that, two simple gated recurrent neural networks have been built: the first using a block of 512 LSTM cells and the second using a block of 512 GRU cells. These have been trained on text sequences with the aim of predicting the next character, and then they have been evaluated on never-seen-before data: the LSTM model showed better training performance but the GRU model demonstrated greater capability of generalization. These models have been then included in a predictive loop for generating sequences of 1500 characters based on a seed sequence of 512 characters. The generated data, already in symbolic text format, have been converted back into MIDI files in order to be played.

Chapter 3 discussed the plausibility assessment test for the generated extracts. The

¹Find the repository at: github.com/ludovicolanni/neural-music-composer

two most plausible extracts from each model have been used to set up a listening game on Google Form together with four extracts coming from the original MAESTRO dataset. Players have been asked to quantify their perception on whether the music they were listening to was human-composed or artificially-composed. A total of 53 people responded to the quiz and the outcomes highlighted that all artificial extracts managed to confuse at least 30% of players, with those coming from the GRU-based model outperforming the extracts generated by the LSTM model. These figures are even more valuable considering that players were able to correctly spot almost all the human-composed samples. The achievements are totally in line with the aims, the objectives and the requirements stated within the specifications of the project.

4.2 Limitations and Future Work

Generating music using deep learning techniques is a task that can be approached in many different ways. Each of the potential solutions comes with advantages and disadvantages.

With respect to the dataset, the use of mono-instrument MIDI files from the same genre, i.e. classical music, allowed for simpler mathematical representation but significantly restricted the diversity of the generated music. On the other hand, working with raw audio allows the use of any kind of music files and can lead to more expressive music generation at the cost of increased complexity and reduced interpretability. Suggesting future developments for this project, it would be interesting to use MIDI datasets from different genres and see whether the output reflects a nice blend between these styles.

With respect to the mathematical modelling of MIDI files, the character-level textual representation has the advantage of being an unconstrained approach that potentially generates chords that were never seen during training. However, since the number of characters in a time-step varies a lot depending on the notes being played, this may lead to the generation of extracts of significantly different duration. In this project, for instance, a lot of generated music were too short, meaning that they were unusable for human testing, or too long, meaning that they were mainly made of silence. Only the extracts with duration of about 10 seconds were effectively usable and plausible, because the ratio between number of timesteps and number of characters was optimal. Suggesting future improvements, this issue could be addressed by considering note-level sequences and by fixing an upper-bound for the number of notes in a single chord. The cost of doing this, however, would be inevitable limitation in creativity.

With respect to the neural models, the networks that have been built only have one gated recurrent block each due to the limited computational and temporal resources

available for this project. However, it would be interesting to see if trying to change the shape of the network, for example by adding an additional recurrent block, leads to improvements in pattern-learning performances and in the plausibility of the generated music.

With respect to the plausibility assessment test, the use of Google Form has allowed to quickly build a listening game without worrying about building a custom application. On the other hand, it lightly constrained the quiz space: for instance, the absence of a question-specific option for randomizing the order in which the questions appear to the user, which is absolutely necessary for reducing the bias, made it impossible to ask players to provide a brief comment on the factors that affected their choices. In contrast, the option of only including scoring questions and randomizing their order has been adopted. As future work suggestions, it would be interesting to build a custom application for the listening game, where the descriptive users feedback can be implemented. This could allow for deeper interpretation of the results.

Bibliography

- [1] Lanni L. Generating Piano Melodies In MIDI Format With Deep Learning: Project Investigation Report. Robert Gordon University of Aberdeen: School of Computing. United Kingdom; 2020.
- [2] Walker J. Midicsv File Format; 2008. Available from: <https://www.fourmilab.ch/webtools/midicsv/> [cited 24 June 2020].
- [3] Chollet F. Deep Learning with Python. Manning; 2017.
- [4] Rumelhart D, Hinton G, Williams R. Learning representations by back-propagating errors. *Nature*. 1986;323. Available from: <https://doi.org/10.1038/323533a0>.
- [5] 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-8604-0CRed.pdf>.
- [6] 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>.
- [7] Cho K, van Merriënboer 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>.
- [8] Bengio Y. Neural net language models. *Scholarpedia*. 2008;3(1):3881. Revision #140963.
- [9] 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/392914e7a1eba8f64daaaa39da8563f6ec409a4c>.
- [10] Kalingeri V, Grandhe S. Music Generation with Deep Learning; 2016.
- [11] Dhariwal P, Jun H, Payne C, Kim JW, Radford A, Sutskever I. Jukebox: A Generative Model for Music; 2020.
- [12] Boulanger-Lewandowski N, Bengio Y, Vincent P. Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription; 2012.
- [13] Huang A, Wu R. Deep Learning for Music; 2016.
- [14] 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>.
- [15] Payne CM. MuseNet; 2019. Available from: openai.com/blog/musenet.

- [16] 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>.
- [17] Chung J, Gulcehre C, Cho K, Bengio Y. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling; 2014.
- [18] 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=r11YRjC9F7>.

Appendix A

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.

The code has been developed on Jupyter Notebook and, mainly, on Google Colab. The listening game has been created using Google Form.

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 B

Project Log

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

Week Ending: 21/7/2020

This week I focused on:

Setting up the environment for the writing of the final project report on Overleaf;

Defining a provisional structure of the document

Starting coding: to be precise, I finished the data preparation phase using Jupyter Notebook and Google Colab.

In the meeting we discussed the above mentioned topics, focusing on the necessity of powerful machines for building and running the neural models in the next phase of the experiments. The free option of using TPUs in Google Colab is most likely the best one. Next week, the focus should be on building the models, training them and make predictions.

Week Ending: 28/7/2020

This week I focused on:

Building the recurrent neural networks for training purposes: one has a 512 deep LSTM layer, the other has a 512 deep GRU layer.

Training the models with Tensorflow and Keras, using TPU environment provided for free by Google Colab.

Coding the predictive loop for the neural music model in order to generate sequences

of 1000 characters. Different temperatures were experimented for adjusting the complexity of the predictions. Low values were associated with naive models that easily got stuck in the repetition of the same element/s. High values generated more colorful and unpredictable sequences.

Surprisingly, the models were capable of understanding the syntax of music converted into text.

In the meeting we discussed the above mentioned topics, focusing on the options for evaluating the generated sequences (e.g. Google Poll). Next week, the focus is on coding the transformation from generated sequences into midi files, going through the piano roll format. That way, the output of the generative neural models can be played and human evaluated from a quality-oriented perspective.

Week Ending: 4/8/2020

This week I focused on:

Coding the functions for converting generated text sequences into piano roll format first and midi file later.

Applied the functions to the sequences generated last week

Played the generated midi files. This way I noticed that the best value for temperature parameter is 0.5 for both models. In addition, I was keeping the seed sequence in the generated midi file, thus I needed to change the process. Therefore, I generated 10 new sequences - with 1500 new characters instead of 1000 - for each model, not including the seed sequences and with fixed temperature at value 0.5. The majority of these samples were syntactically perfect and could be converted to piano roll and midi format.

I have selected the 2 most musically plausible samples for each generative model and used them to create a blindfold game on Google Form that works as a user evaluation on how artificial they think each music is. For this purpose, the 4 artificial samples partner with 4 randomly selected extracts from original human-composed songs. Players do not know whether the extract they are listening to is artificial or not, and the questions are always shuffled for reducing bias.

In the meeting we discussed the above mentioned topics, focusing on the test of the Google Form, adding some features and updating others. Next week, the focus is on sending the form to an undefined group of players (collecting the aggregated scores for each extract) and starting to write the project report.

Week Ending: 11/8/2020

This week I focused on:

Sharing the listening game form, reaching about 50 actual players.

Uploading the I-Python Notebook on Github for properly and effectively referencing them in the experimental report.

Starting writing the experimental report. I have improved the skeleton structure that I had already done few weeks ago and I have written the introduction chapter, which also contains a comprehensive summary of the project investigation report.

In the meeting we discussed the above mentioned topics, focusing on the reflections to be made based on the answers we got for the listening game on Google Form. Next week, the focus is on accelerating in the writing of the experimental report by almost completing it before Tuesday.

Week Ending: 18/8/2020

This week I totally focused on finishing writing the final report by completing the implementation, evaluation and conclusions chapters. The last step in order to do so was to freeze and collect the results of the listening game, where, at the end, 53 people participated.

In the meeting we discussed final modifications and adjustments to be made to the report and we planned the making of the report. In addition, we spoke about the presentation and demonstration, highlighting the optimal strategy to be followed. Next week, the only task is finishing the project by reviewing and submitting the final report, making the poster and the presentation/demo.