BABES-BOLYAI UNIVERSITY CLUJ-NAPOCA

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

SPECIALIZATION COMPUTER SCIENCE

DIPLOMA THESIS

Music and Feelings: A deep learning approach to emotional composition

Supervisor:
Darabant Sergiu Adrian

Author:
Naiman Alexandru Nicolae

Cluj-Napoca
2020

Abstract

Year after year, the music industry sets new records on their revenue, earning billions of dollars through different mediums, from LP records to CDs, and with the rise of the Internet, through streaming platforms. This growth can be associated with the ease of producing and releasing new songs, processes that have been facilitated through different tools such as Digital Audio Workstations and Virtual Studio Technology plugins. Even with all these advancements, the approach one has when composing did not change in the last hundreds of years: composing based on inspiration with a combination of trial and error corrected by music theory.

This thesis introduces a tool designed for easing music composition, flattening the steep learning curve of music theory, by helping to compose songs based on feelings. The main goal of the application is providing the user with a way of generating songs based on some emotion given data. The user provides the input emotion values, and the software should respond with a composed song in which we can recognise the given emotion.

The application consists of two parts: the frontend web application with which the user interacts, and the backend on which our machine learning model is running. For the emotion data, we used the valence-arousal classification, which specifies that any feeling can be expressed through the two aforementioned values. For the composing algorithm, we used autoencoders and their capabilities of learning the internal structure of our data and used it for generating new songs based on new input.

This work is the result of my activity. I have neither given nor received unauthorized assistance on this work.

Contents

List of Figures

List of source codes

Chapter 1

Introduction

"Music is a moral law. It gives a soul to the Universe, wings to the mind, flight to the imagination, a charm to sadness, gaiety and life to everything. It is the essence of order, and leads to all that is good and just and beautiful." [36]

1.1 Motivation

The art of composing music is not one that any beginner can master due to the fact that music theory has a very steep learning curve. You can still compose music, without knowing anything about theory, but it will be very hard to know how to really express your thoughts since the notes do not have any meaning to you other than their sound. When you start learning, you will find out about scales, modes, chords progression and how they clearly relate to some feelings.

1.2 Solution Overview

In the latest years, with the technological advancements regarding artificial intelligence, dreaming about diagnosing a patient in seconds, facial recognition, instantaneous translating, and self-driving cars is no longer an idea in Asimov's mind. Nowadays, with a bit of work, one can create a neural network, train it on a given dataset and see the results about its objective, without needing to worry about the internal structure of each component inside the network.

I want to encourage and help all people that have difficulties composing music with the help of machine learning. I want to create an application that can compose, enhance or continue a piece of music based on some emotions that the user can provide. The music should have a structure and the app will generate a MIDI file containing a song with the given characteristics.

1.3 Thesis Outline

WIP: TODO: add thesis structure when done

Chapter 2

Machine Learning and Autoencoders

This chapter contains theoretical information about the paper's domain, deep learning, and the principal model used, autoencoders. Of course, before starting looking in its domain, we should start with the basics, namely, machine learning.

2.1 What is Machine Learning?

Machine learning is one of many subfields of artificial intelligence, the science of getting the computer to learn from experience without being explicitly programmed to do so, improving their ability to think, plan, decide, etc. [26].

Algorithms in this area are different in their approach, the type of data they input and output, and the type of problem they are learning to solve, but, despite all differences, they are based on the same idea that there are some generic algorithms that can discover patterns, without having to write code; instead, they build their own logic on the data fed to them.

2.2 Machine Learning Styles

When studying machine learning, one can identify a plethora of algorithms, each unique in its approach and purpose, but all of them can be classified based on the learning style. Therefore, there can be grouped into four main categories:

- **Supervised Learning**
- **Unsupervised Learning**
- **Semi-supervised Learning**
- **Reinforcement Learning**

2.2.1 Supervised Learning

In supervised learning, given a set of example input-output pairs, the job of the algorithms is to approximate a function that maps from input to output [32]. It is called supervised learning because human experts act as the teacher, where they serve the computer with training data containing the input and also provide the correct output, from which the computer should be able to observe patterns. As depicted in Figure *2.1*, the algorithm should be able to predict the output when given new inputs, based on the function that it approximates [15].

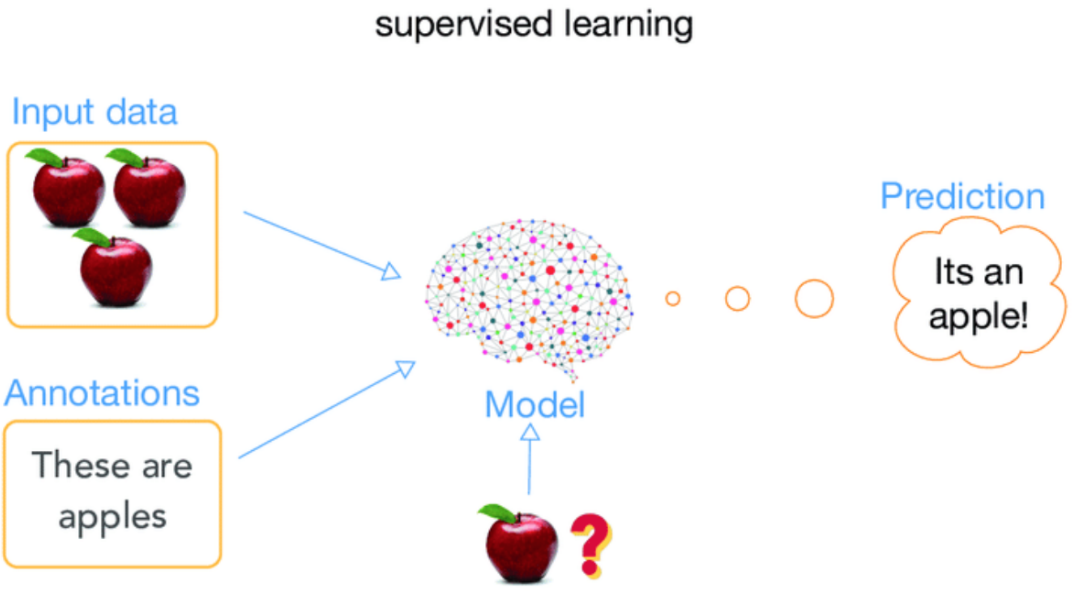

Figure 2.1: *Supervised algorithm predicting new output data given new input [37]*

Furthermore, supervised machine learning algorithms can also be grouped based on the type of problem they intend to solve. Having said that, there are classification and regression algorithms.

Classification algorithms solve problems in which the input data has been labelled in different classes or categories, meaning that the goal is to predict discrete values such as $\{0, 1\}$, $\{2, 4, 6, 8 \dots\}$, $\{cat, dog\}$, $\{spam, normal\}$, etc [15].

Regression algorithms, on the other hand, solve problems in which the output variable is a real value, meaning that the goal is to predict continuous values based on the input data, such as predicting house prices based on their size and location [15].

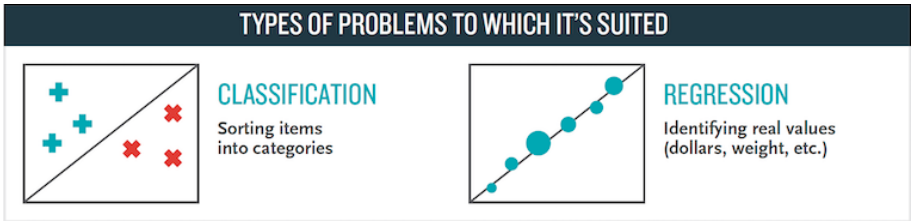

Figure 2.2: *Classification(left) and Regression(right) visualization [8]*

2.2.2 Unsupervised Learning

In unsupervised learning, the job of the algorithms is to observe patterns in the structure of the given data even though no explicit feedback is supplied [32]. There is no human factor implied in the learning process, hence the name of unsupervised learning. As shown in Figure *2.3*, they should be able to identify the underlying structure in the given data, to better understand it [8].

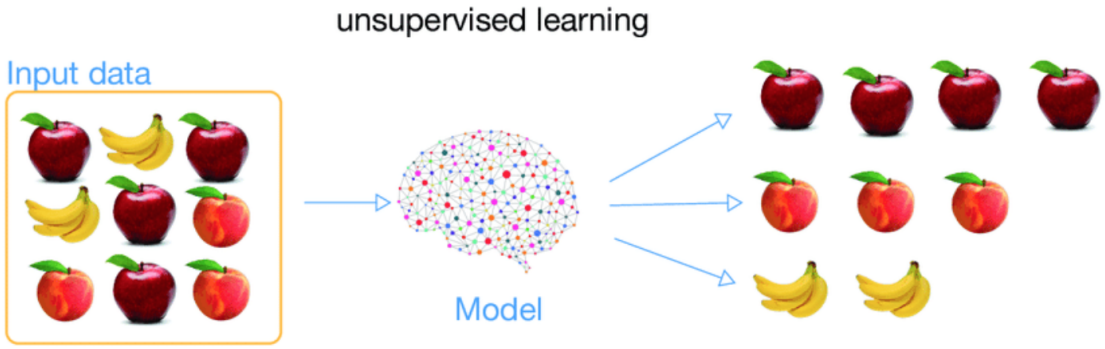

Figure 2.3: *Unsupervised algorithm grouping data by their observed features [37]*

The same as supervised learning, unsupervised algorithms can also be grouped based on the type of problem they intend to solve. Provided that, there are clustering and low-dimensional embedding algorithms [32].

Clustering algorithms try to solve problems by grouping the input data into different segments based on similarities learned, such as grouping customers by purchasing behaviour [6].

Low-dimensional embedding, also called dimensionality reduction, is a technique in which complex data, that is difficult to describe (data that needs more than two or three dimensions to represent) is reduced to a lower number of dimensions, wherein the machine learning algorithms can help since they can learn the internal structure inside our data, furthermore, transcribing it to an easier way of interpreting it [32].

2.2.3 Semi-supervised Learning

Semi-supervised learning algorithms lie between the aforementioned categories. They intend to solve problems where you have a large amount of data but only some labelled. A lot of machine learning problems are using this technique for the reason that labelled data can be expensive or time-consuming to obtain, in contrast to unlabelled data that is much easier to acquire [6].

As shown in Figure *2.4*, one can use an initial classifier on the labelled data, classify based on the features learned previously, the unlabelled data, retrain the classifier with the whole dataset, then use it to predict new outputs given new input data, ideally obtaining a better model [24]

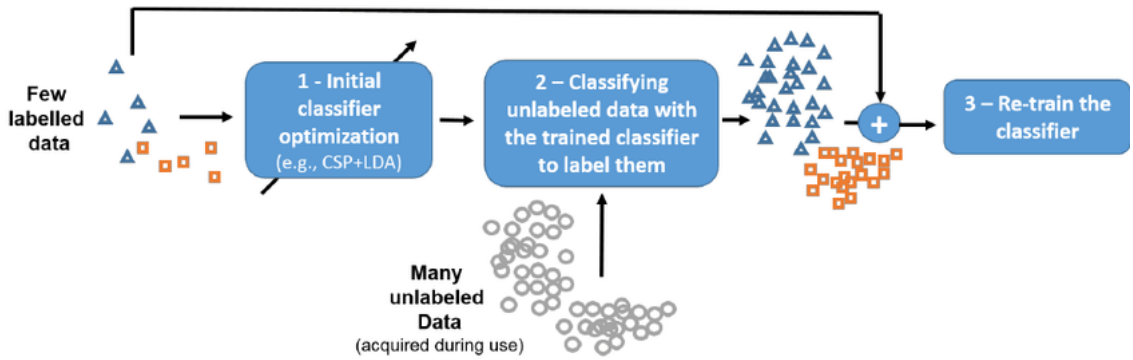

Figure 2.4: *Semi-supervised approach on data classification [24]*

2.2.4 Reinforcement Learning

Reinforcement learning algorithms are based on a system of type reward-punishment, where the job of the algorithm is to iteratively interact with its environment, making actions in such a manner that will maximize the rewards or minimize the punishments. To put it another way, in reinforcement learning, the software learns from its past experiences allowing it to develop an optimal behaviour within a specific setting, where it aims to increase its performance [15].

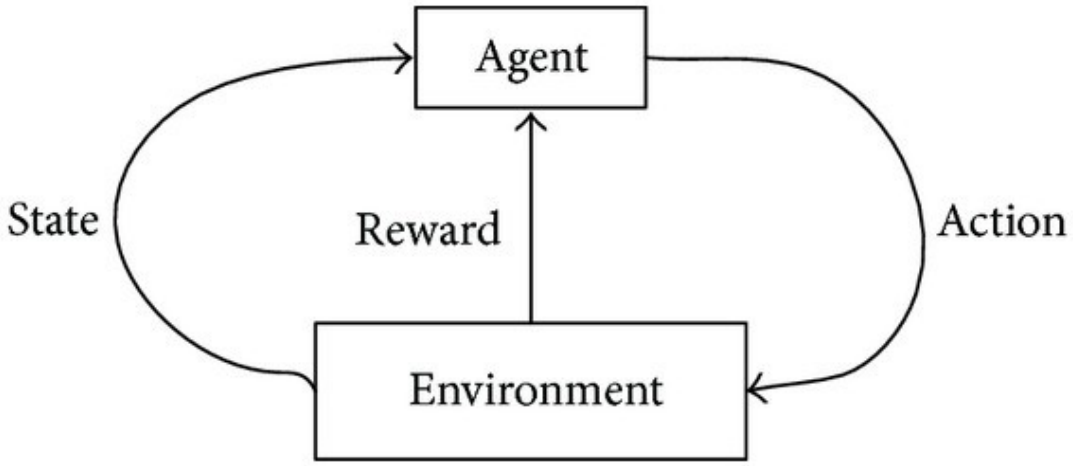

Figure 2.5: *Scheme of how reinforcement algorithms work [15]*

Figure *2.5* is a visual representation of how reinforcement learning works. The algorithm (called the "agent" in the picture), takes action in the environment, which will possibly trigger a reward. After that, the state is updated, and when deciding its next action, it will take into account its last state and the result of the previous action.

2.3 ANNs and how they work

One of the main types of machine learning algorithms that are vastly used is artificial neural networks or ANNs. They are structured as weighted graphs, modelled after the human brain where each node represents a neuron and each edge represents the

synapses of the neural network. The weight of each edge determines how powerful the synapse is. The neurons are distributed in groups named layers. Neurons can form synapses only with neurons from other layers [38].

2.3.1 Artificial neurons and activation function

The smallest unit of an artificial neural network is the **artificial neuron**, namely the nodes of the graph. It is the place where the computation happens: a node merges the input data by doing their weighted sum, to dampen or amplify the input. Further, the result will be given to a so-called **activation function** whose purpose is to determine whether or not the neural signal will be passed on to the next neurons or not [27]. A more detailed explanation of the subject can be seen in Figure *2.6*.

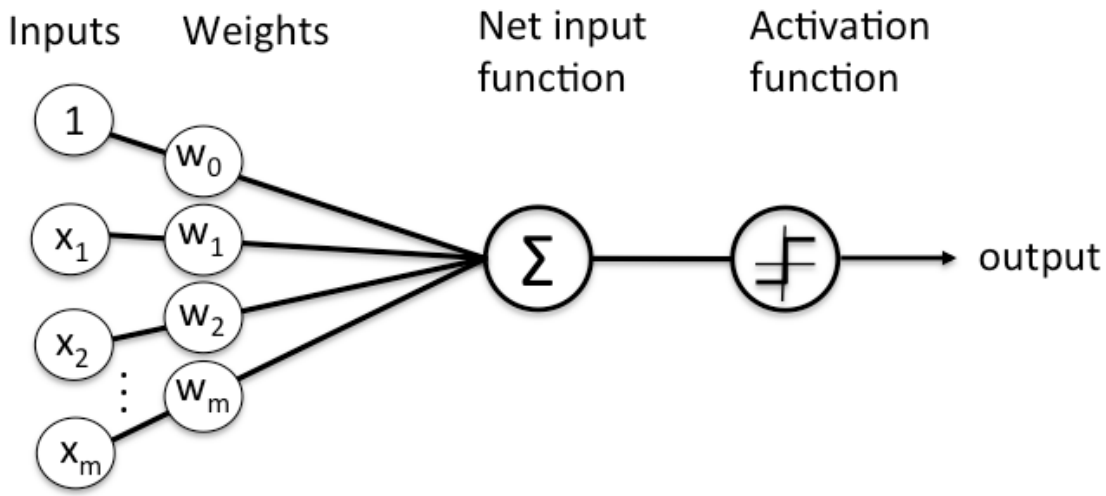

Figure 2.6: *Anatomy of an artificial neural network [27]*

The aforesaid **activation function** is a mathematical expression that helps the ANN to add non-linearity to itself. With more and bigger datasets, the patterns that need to be identified become more complex. With the help of these types of function, the input is much easier to analyze and interpret.

- **zero-centered**: output should be symmetrical at zero so that it does not move to a particular direction;
- **computationally inexpensive**: the function is called for every layer for each input data, meaning that it could be invoked thousands of times. Therefore, they should not be computationally expensive;
- **differentiable**: at the core of almost every machine algorithm lies an optimization algorithm. One of the most popular approaches is the gradient descent algorithm in which one of the most essential features is that the function used needs to be differentiable; this idea will be discussed in detail in a subsequent paragraph;

2.3.2 Backpropagation and gradient descent

Backpropagation describes how machine algorithms learn. As shown in Figure*2.7*, after the data traverses the neural network in a forward manner, propagating the signals through the neurons, the algorithm will go in reverse adjusting each weight for each node and its links [17]. This technique is called backpropagation of errors, or simply, backpropagation and is the learning engine of the most ANN's.

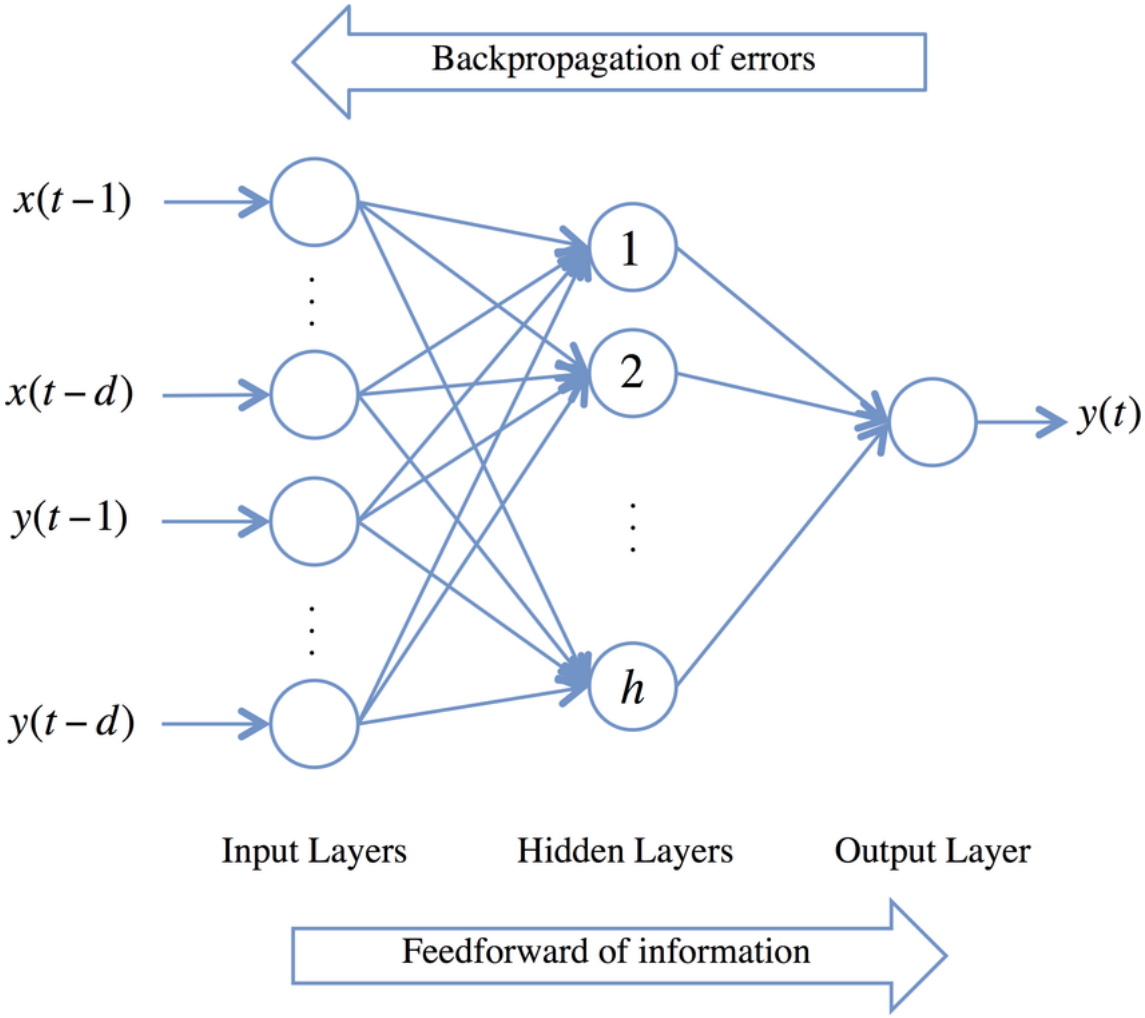

Figure 2.7: *Feedforward Backpropagation Neural Network architecture. [9]*

The **gradient descent**, as mentioned before, is an optimization algorithm best-used to find the (local) minimum value for a function [6]. Hence, it can be used to reduce the errors in our algorithm, since the training of a neural network is just the process of finding the set of weights and other parameters in such a fashion that the errors(differences) between our expected and actual results are minimum. Hence, the gradient descent is used to compute the errors of our algorithm, that will be later backpropagated through the ANN.

2.3.3 Layers of ANNs

Multiple artificial neurons are grouped in **layers**; the purpose of each one is to apply a non-linear transformation of the input from one vector space to another [13]. As

you can see in *2.8* there are three types of layers in an artificial neural network:

- Input layer

  This layer of a neural network is the very beginning of the ANN's workflow, bringing the initial data into the system for further processing by subsequent layers of artificial neurons [34].

- Hidden layer

  Any layer between the output and the input layer is considered to be a hidden layer. The number of neurons they contain and also their number can vary. Their role is to take in a set of weighted inputs and produce an output through an activation function [33].

- Output layer

  This layer of a neural network is the last layer of neurons that produces given outputs for the program. They usually are made much like any other artificial neuron, but they may be observed in a different way, the output layer coalesces and concretely produces the end result [35].

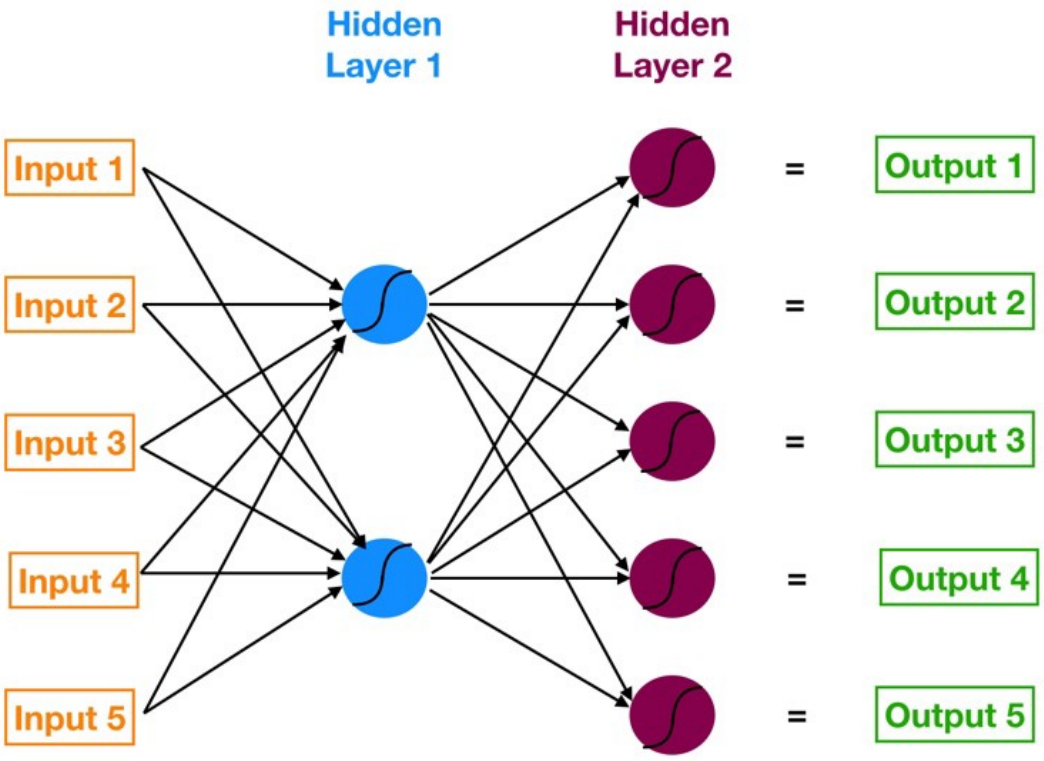

Figure 2.8: *Neural network with one hidden layer [38]*

2.4 What is Deep Learning?

Deep learning is a subfield of machine learning that contains a collection of ANNs that are known for their capability on learning unsupervised from data that is un-

structured or unlabeled, also known as deep neural learning or deep neural network.

A second classification we can make on ANNs is based on the relationships between their nodes. Then, neural networks can be recurrent or feedforward; the first one does not have any loops in its graph and can be organized in layers. A deep neural network is a feedforward ANN with many hidden layers.

A visual representation of the aforementioned is the Figure *2.9* wherein we could see the differences between a simple feedforward ANN with one hidden layer(left) and a deep neural network with three hidden layers.

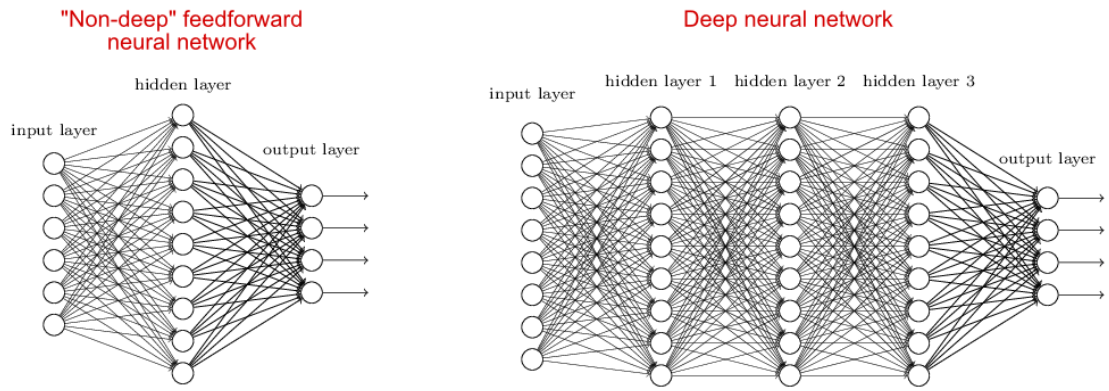

Figure 2.9: *Difference between a Non-deep ANN(left) and a deep ANN(right) [28]*

.

Deep learning is an area in machine learning that achieves state-of-the-art results by learning to represent the world "as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones".[17]

A visual representation of the aforementioned statement can be seen in Figure 2.3.

2.5 Autoencoders and how they learn

Autoencoders are neural networks usually used for dimensionality reduction. They learn in an unsupervised manner, their purpose being to obtain the internal representation of the data by compressing and then reconstructing the original input, making a copy of it.[28]

This type of neural network is composed of two parts:

- **encoder**
- **decoder**

The first one may be viewed as a network that aims to compress our data into a lower dimension code, named encoding. It could be represented as a function $\boldsymbol{h} = f(x)$.[28]

The second one intends to reconstruct the input based on that encoding, removing thus all the possible noise. It could be represented as a function $\boldsymbol{r} = g(h)$.[28]

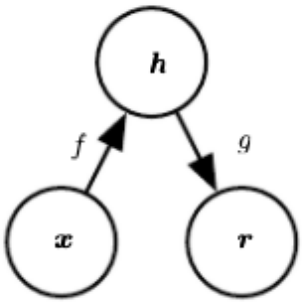

Figure 2.10: *Structure of an autoencoder [28]*

.

Therefore, as shown in Figure 2.10, the entire autoencoder can be seen as the following function: $g(f(x)) = x$. In other words, an autoencoder tries to mimic the identity function, but, due to the compression, is unable to do so. However, thanks to that same compression, it learns how to successfully identify the most important features of our data's internal structure, represented by the encoding.[28]

Chapter 3

Music and Machine Learning

Music, one of the oldest forms of art and entertainment humanity has ever had, has played a significant role in different cultures and civilizations worldwide since the beginning of history.

With the progress of artificial intelligence in many areas such as computer vision, speech recognition, and natural language processing, machine learning enthusiasts have tried to use all their knowledge in music-related fields. [21]

Artists tried different approaches to music composition, from free playing and improvisation to diving deep into the rhythm, notes, scales, modes, and other elements of music theory, for both of them existing various approaches in machine learning. This chapter explores several of these methods related to this paper's work, but before immersing in the machine learning aspects, we should discuss how they process data.

3.1 What is MIDI?

MIDI (**M**usical **I**nstrument **D**igital **I**nterface) specifies a standard for connecting, playing, and recording electronic instruments and a file format to store this information. It uses messages in an event-driven manner, meaning that each note is represented as an event that specifies its notation, pitch, velocity, and other useful characteristics. [5]

The MIDI file format is a popular choice among data scientist when working with audio and music due to several aspects:

- *ease of manipulation*: one can edit the parameters of a note without the need for rerecording; [16]
- *ability to change instruments*: a MIDI file only defines what notes should be played and in which order, not the instrument that should perform them; [16]
- *small size*: this file format is comparable to an electronic sheet; it does not hold any audio information, only instructions for playing the notes, saving a whole song in just several kilobytes of data. [16]

3.2 LakhNES and Transformer-XL

One of the latest music-generation approaches is Chris Donahue's *LakhNES: Improving multi-instrumental music generation with cross-domain pre-training*. The author uses language models and their ability to retain long-term sequences within the data, being able to generate multi-instrumental structured songs. [14]

Before discussing his approach, one must understand the insights of the used models. Transformer-XL is a neural network used for resolving natural language processing tasks, achieving state-of-the-art results on different language datasets. Other proposals for these problems are models based on recurrent neural networks and long short-term memory, which give great results on more straightforward data, but cannot establish longer dependencies between words due to their limitations. [19]

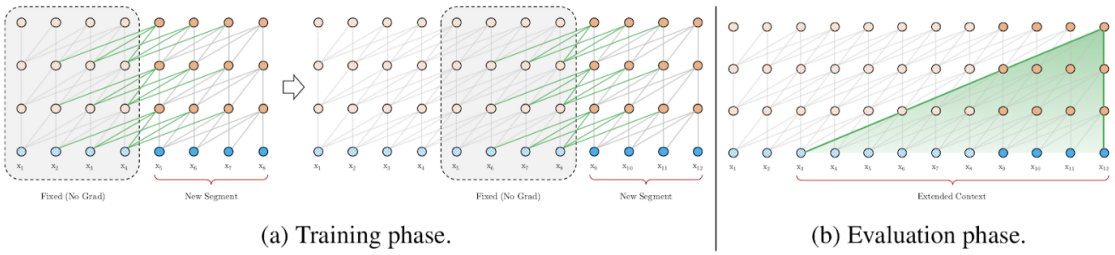

Figure 3.1: *Transformer-XL - training and evaluation [19]*

Vanilla Transformers tend to resolve this problem by introducing new attention modules. They receive a sequence of tokens (instead of processing tokens one by one) and determine the connections between them based on absolute positional encoding. The main obstacle with this network is that it can only receive one sequence of specified length at a time; hence, the data can suffer from context fragmentation. [19]

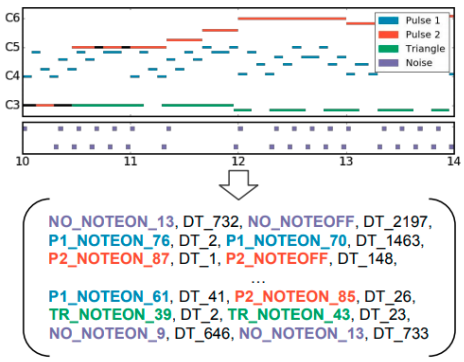

Figure 3.2: *Note conversion example used in [14]*

Transformers-XL solves the problems mentioned above by using relative positional encoding and a recurrence mechanism to extend their context for much better and faster performance. Figure *3.1* represents a small outline of the training and evaluation phase of the discussed network, in which one can observe how the extended context is created and used.

In [14], they used the Transformer-XL, by creating a grammar through which they converted the MIDI events into text that the network can process. An example of this conversion is in Figure *3.1*. In this manner, they can convert any MIDI to some text with which they train the model and then use it for generating new complex structured songs or completing existing ones.

Another noticeable aspect of their work is that they used transfer learning, their purpose being to generate NES music game, but, by using this approach of pre-training their model with general standard data, they increased the performance of the model by 10%. [14]

3.3 Composer and Autoencoders

The main inspiration for this work is HackerPoet's *Composer* [18], another point of view for music generation that uses deep autoencoders.

As explained in the previous chapters, this type of neural network learns in an unsupervised manner the internal representation of our data. Using only the decoding part, the author generates new MIDIs by feeding it new normalized encodings and predicting new songs. He trained his model with game music utilizing a dataset that he created, his main goal being game themes. [18]

This approach has excellent results regarding music theory, the model being able to compose songs with a well-defined structure, having popular chord progressions, bassline, multiple melodies, and also musical motifs. [18]

The good results and the ease of understanding and implementing autoencoders are the main reason they have been used in this application, one key difference in this paper's approach being represented by the structure of our autoencoder and how the encoding was enhanced with the emotional value from the dataset.

3.4 AIVA - Artificial Intelligence Virtual Artist

Another proposal worth mention is the algorithm behind the startup *AIVA* [1]. It is a deep learning model that uses reinforcement learning, initially and mainly trained on classical pieces based on compositions of great musicians like Mozart, Beethoven, and Bach. In the latest releases, it was also trained on different styles of music. [25]

It is the first artificial intelligence recognized as a composer by a copyright authority - SACEM (standing for Society of Authors, Composers and Publishers of Music). Therefore, its pieces are not public, their songs being copyrighted. [22]

AIVA's algorithm most essential features are creating new compositions or finishing existing ones based on a given genre, or even writing with influences of another song. In other words, it can learn the style of a piece and compose a song in the same fashion as the input one.[25]

Chapter 4

Proposed approach

4.1 Problem definition

With the advancements of technology, people tried digitalizing every aspect of their life. They renounced the traditional slow postal services in favour of the modern fast email. Digital photos gained popularity to the detriment of physical copies. In the same way, CDs and other physical mediums were forgotten with the arising of digital MP3s.

Over the past years, with the rise of the Internet, even the idea of digitalizing one's life became old, so people started to sync it with the cloud. Hence, a new way of storing and transmitting data emerged: streaming.

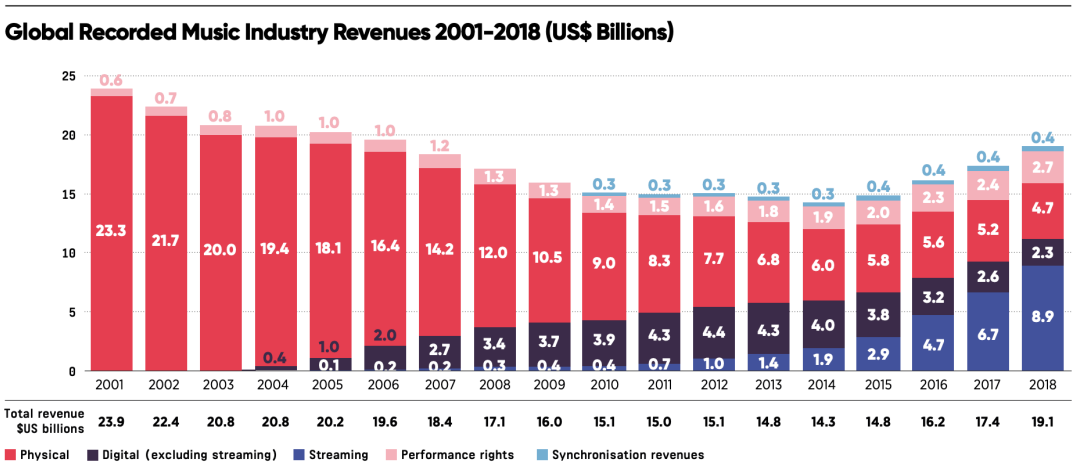

Figure 4.1: *Global Recorded Music Industry Revenues 2001-2018 [29]*

As shown in Figure *4.1*, in the last decade, the revenue of the music industry shifted, going from physical to digital; nowadays, the highest income is generated by streaming means.

With this change, one can easily compose and produce music from their studio or even bedroom, since it is not compulsory to be tied to a record label. There are lots

of successful independent musicians who create music this way and the numbers are growing more and more every year [11].

Even with all these developments, the act of composing music did not drastically change in the last few hundred years. If one wants to compose a song, they should learn how to use instruments, and how to put notes in such a manner that they sound good together. Technology eased the process of composing, but one must still learn chords, scales, and other elements of music theory to make the process less hazardous. Although music theory has a steep learning curve, learning it helps express oneself musically. Despite all these advancements, there is a lack of tools facilitating the creative process.

This paper tries to fill the aforementioned deficit; with the help of machine learning, it proposes a system whose goal is to aid the aspiring musician easily compose a song based on a given emotion. It provides a way of generating new pieces of music without the need to master music theory.

4.2 Orpheus's Sorrow's Approach

Orpheus's Sorrow is the name of the system mentioned before. It is a client-server web application where a musician can input emotion data and values to other parameters to visualize songs that this deep neural network generated on the given data. In the following sections, we will discuss the paper's approach related to music and emotion, dataset creation, and the internal structure of the model.

4.2.1 Music, emotions and machine learning

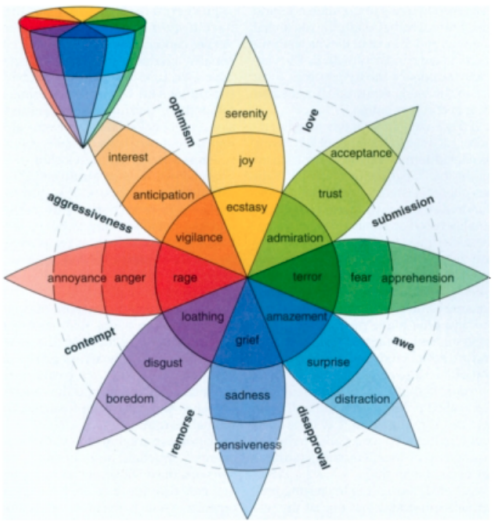

Figure 4.2: *Plutchik's wheel/cone of emotions [30]*

Music and emotion are highly connected, one of the main goals of a composer being expressing his feelings through notes, trying to reach his audience, and to induce them his mood and ideas.

When dealing with music classification based on the emotion it expresses, there are multiple strategies for managing the sentiment data, the most popular being tag-classification using some predefined emotions or a regression approach representing the feeling through some variables. Both of the approaches can be applied to the whole song or a subset of features extracted manually or automatically.

The first one usually is paired with an existing model for emotions, one of the most popular being Plutchik's wheel of emotions. As can be seen in Figure *4.2*,

Plutchik circularly arranged the feelings, proposing eight primary bipolar emotions: surprise - anticipation, fear - anger, trust - disgust, joy - sadness.

When choosing the second, one frequently uses Thayers's (also called Russell's) model. With this method, every emotion can be described using two variables: valence and arousal.

As seen in Figure *4.3*, in this way, feelings can be plotted on a cartesian system of coordinates obtaining four general quadrants of emotions: happy, angry, sad, and relaxed.

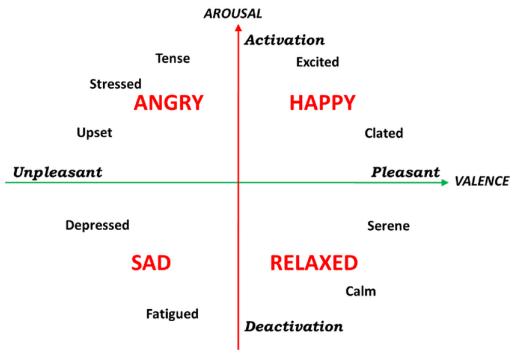

Figure 4.3: *Thayer's Model for Emotions (also named Russel's model) [30]*

4.2.2 MIDIs and Emotion Dataset

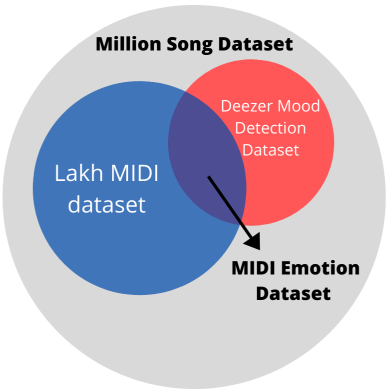

Figure 4.4: *Merging existing datasets to create a new one*

When discussing machine learning and neural networks, one must know what data is available for their needs, how to parse their data in a format they fully understand and maximizes the potential of their data.

The model proposed uses MIDIs to store and handle the music information data due to ease of manipulation, as mentioned before. During quick research, there were not any free/public datasets that satisfied the needs of the problem statement - a dataset containing MIDIs with the emotion value(arousal and valence) attached to each song. Hence there was a need to create a new dataset by merging some of the existing ones to fulfill the requirements. The merged datasets are the following:

- **Million Song Dataset** - a free compilation of metadata for a million contemporary common music records;[4]
- **Lakh Midi Dataset** - set of over 150,000 unique MIDI files, over 40,000 being matched to entries in the dataset mentioned above;[31]
- **Deezer Mood Detection Dataset** - valence and arousal dataset for over 10,000 songs also matched to the first-mentioned dataset.[12]

A visual representation of the sets is in Figure *4.4*. It represents how the merging was performed. The songs from the last two datasets were paired based on their ID in the Million Song Dataset and then joined together with their relevant information. The new dataset contains over 1800 MIDIs, with their respective valence and arousal, enough data to train the model to see if this approach can have good results.

The obtained dataset was split into three parts: 80% training data, 10% validation data, 10% testing data. The training data was also augmented by shifting the pitch with several notes higher and lower.

4.2.3 Model architecture

Now, with the data ready to be fed to the model, in the following paragraphs, it will be discussed about its sampling and also about the structure of the model.

Feeding the MIDI file directly in the model will not produce good results since there are too many irrelevant pieces of information beyond our purpose, and that will confuse the network. Before the actual training of the MIDI, the files were transformed into piano rolls, being sampled as a set of images. A piano roll here consists of an image of type $noteRange * timing$. The latter was chosen as 96, as used in [18], mainly because almost all standard time signatures are dividing. For the first one is chosen the same value since very few songs use the highest and lowest pitches and also for symmetry's sake.

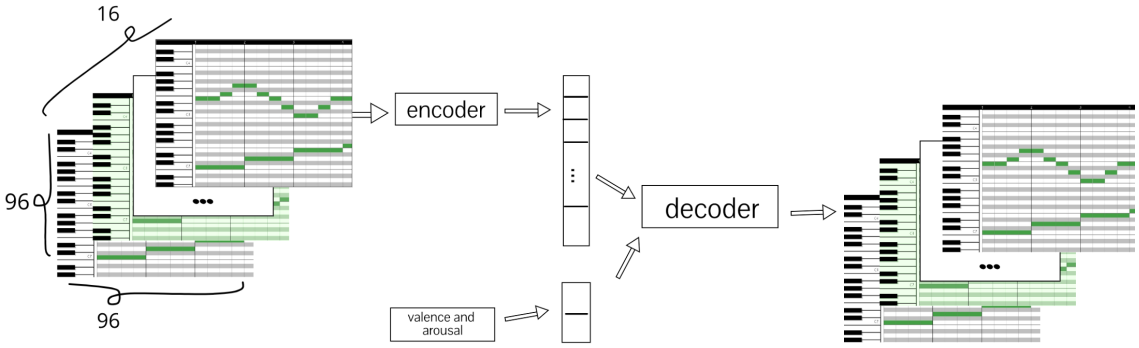

Figure 4.5: *Short description of paper's model*

As seen in Figure *4.5*, the deep neural network consists of a deep autoencoder used to find the encoding of a song; in other words, it finds the essential features of that piece of music. It works similarly with the [18], but the main highlight of this particular model is that to its encoding are concatenated the valence and arousal values for each song, this approach helping it to learn how to generate songs based on its input emotion data.

The composing works by creating an auxiliary model, namely a Keras backend function, that represents the decoder. Its input is represented by two arrays, the 48 values for the features the network identified, and the emotional values; the set of piano rolls represents its output, generating in this manner a new song.

Next, the main structure of the model will be presented, with its primary layers and other parameters configured, as depicted in Figure *4.6*.

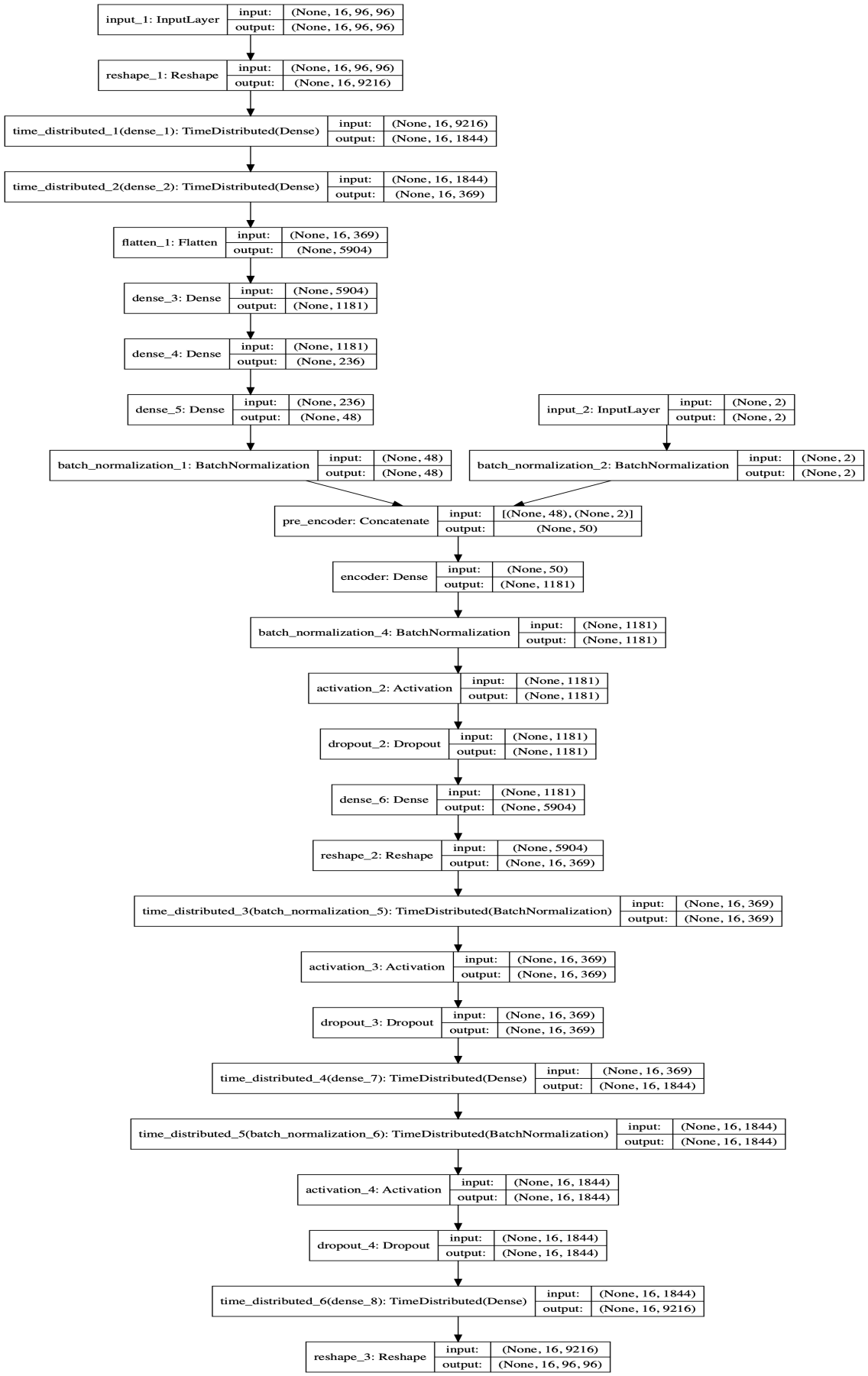

Figure 4.6: *Structure of the model*

For increasing its stability and speed, the model uses the batch-normalization layers. Their purpose is to normalize the output of the prior activation layer by subtracting the batch mean and dividing by its standard deviation, reducing in this manner generalization errors, providing some regularization. [20]

Another vital layer that helps to group the rolls in time-based sets is the time-distributed layer. It is usually used as a middleware for 3D input; given a timestep, it applies the same layer it wraps to each temporal slice of the same size as the timestep.[10]

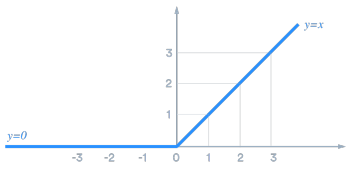

Figure 4.7: *ReLU visual representation [23]*

Another characteristic of our model is the activation function used - ReLU(Rectified Linear Unit). Figure *4.7* represents the function mentioned before, expressed mathematically as $y = max(0, x)$. It is not expensive to compute, the model taking less time to train and run. [23]

The loss function used is the binary cross-entropy since the model has to predict the probability of playing or not a note in its current context, the mentioned function being the default used in binary classification problems.[7]

4.2.4 Loss and Accuracy

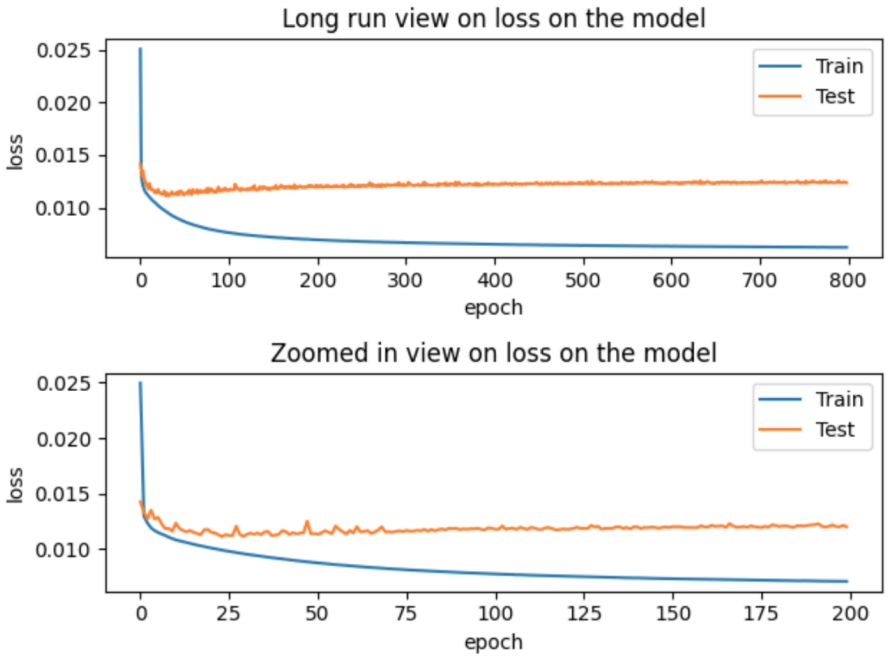

Figure 4.8: figure
Visual representation of loss

Researching other approaches for music composition influenced this paper, even regarding the number of epochs one should train the model. There are various empirical ways to estimate the most suitable number of epochs, but since our dataset has not been used in other machine learning algorithms, or since it does not have

a large number of MIDIs, when training, the model uses a callback that saves the model every n-th epoch. In this manner, the training period cannot be ruined by overfitting since the model can be reloaded at any saved moment.

From each model saved, there can be retrieved some metadata about our songs. This new information will be used when generating another song, normalizing the given input through singular value decomposition, reassuring that our features are in the normal range of our data.

As can be seen in Figure *4.8*, the top part, in the long run, the model starts overfitting around the 200-300th epoch, statement tested by the fact that every model above that period starts composing the same songs regardless the input data. On the zoomed-in view, the bottom part of the same Figure, the slope stats getting smaller around 175-200th epoch, the model that was empirically chosen the best being the 190th epoch due to its ability to compose the most exciting songs with a well-defined structure, and other essential characteristics such as constant rhythm, repeating structures (motifs). The accuracy curve, as seen in Figure *4.9*, starts flattening nearby the same interval for the train data and becomes constant almost for the validation data. When evaluating the chosen best model on the 10% songs from the dataset it was not trained with, the result obtained is 98% accuracy.

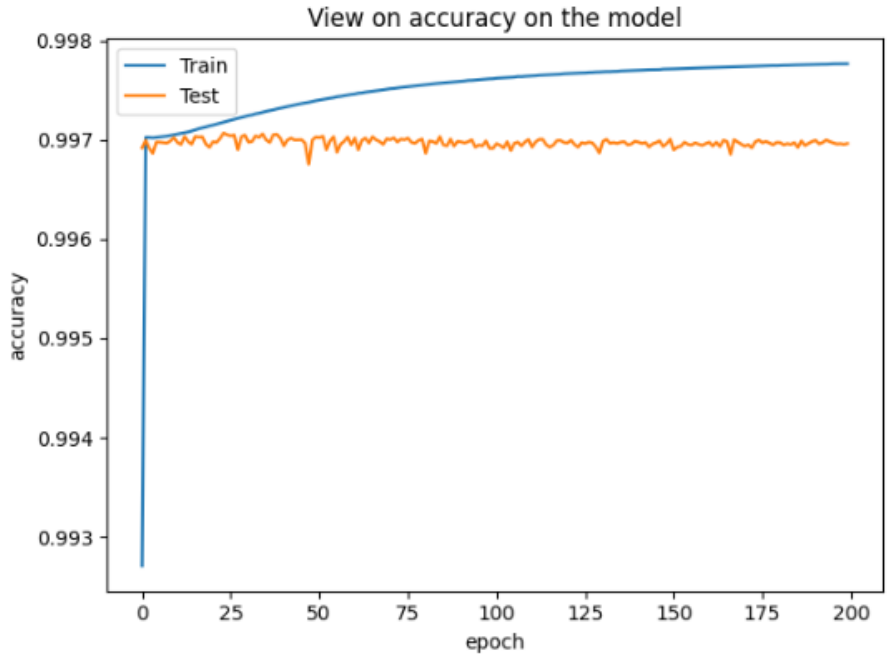

Figure 4.9: *Visual representation of accuracy*

4.3 Composing songs

The discussed model, trained on the created dataset, has excellent results regarding accuracy and loss, but the generated songs do not get the same expected results regarding the emotion part. Due to the smaller number of samples, the learned features and also the valence and arousal values are very dependent, meaning that

given a composed song, at first listen, the emotion denoted by the song is not clear, in the majority of cases.

Despite the vague emotion a composed song induces, there are some characteristics that the model learned for each different quadrant (see Figure *4.3*). For the first one (happy), most of the songs are using the major scale, the most common characteristic of a happy song. For the second one (angry), a preponderance of the songs is using minor scales, part of them also tritones/dissonant chords, the note density over time is much bigger, the rhythm being galloping; all of the above are features more frequently encountered in angry songs. The fact that most of the MIDIs in the dataset from the last-mentioned quadrant are metal or classical songs is also notable. For the third quadrant (sad), a part of songs uses minor scales, this being the quadrant with the most ambiguous results. For the last quadrant (relaxed), some songs also use the major scale, but the most notable fact is the low density of notes, which is characteristic of slow-paced, calm songs.

Chapter 5

Application

This chapter will present the client - server application in which the model is utilized, the technologies used, main use cases/features, and the app's design and implementation.

5.1 Technologies used

Before discussing other implementation and design details, this section will present the technologies and tools used in this paper's application and model.

5.1.1 Keras

Keras is an intuitive API for creating deep neural networks, whose aim is to reduce the cognitive load. It makes one able to focus on the data, but still being able to obtain state-of-the-art results, being used by NASA, CERN, and many more scientific organizations due to its flexibility, but also by the top-5 winning teams on Kaggle, due to its speed of iterations, letting one run experiments much more straightforward.[10]

5.1.2 Flask

Flask is a WSGI web application framework designed for an easier and quicker start, but can also scale up to complex applications, becoming one of the most popular Python web application frameworks.[2]

Flask does not demand any dependencies, leaving it up to the developer to choose, based on their needs, the tools and libraries they want to use. .[2]

5.1.3 React

React is a virtual-dom javascript library used for developing user interfaces. It is declarative, making it more manageable to create interactive UIs, designing simple views for each state of the application, React efficiently updating, and re-rendering the minimum number of components when the data changes. Being declarative views it makes the code more predictable and more comfortable to debug. It is

component-based, allowing encapsulating simpler components then compose them into complex UIs.

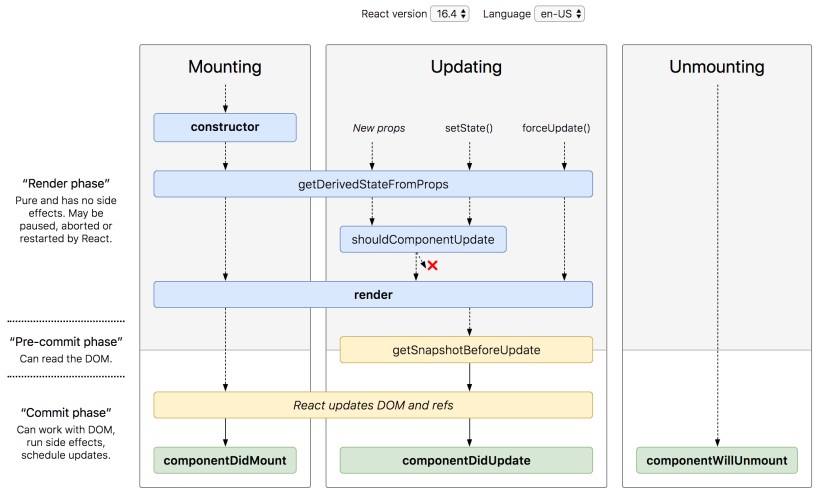

Figure 5.1: *Lifecycle of a React Component*

Each created component is extending the React Component class and should implement the render function. It takes the input data and returns what to display. Each component has two types of input data, which, when modified, trigger re-rendering: props or external properties, and state or internal properties. The rendering process is described as a lifecycle, with a series of predefined steps, some of them having a method that can be overridden for a better manipulating of the UI, as seen in Figure *5.1*.

5.1.4 MobX-state-tree

Mobx-state-tree(MST) is a state management library whose purpose is to combine the traceability of immutable data with the simplicity and ease of mutable data, also using the reactiveness and performance of observable data. The central concept of MST is the living tree. It is an observable container data structure that consists of mutable data, from inside the tree, using some defined methods, called actions, from which are automatically generated immutable objects, called snapshots. For developing the app, MST was chosen over other state management alternatives (Redux, Flux, Recoil) thanks to its ease of writing, lack of boiler code, and reactiveness of observable data.

5.2 Main use cases/features

In the following section, there will be presented the main use cases of our application, as depicted in Figure *5.2*. The main features/use cases of the app are the following:

- *Generate song*: the musician should be able to input values for valence and arousal, and the app should respond with a composed song, that induces the corresponding emotion.

  - actor: musician
  - precondition: inputs for valence, arousal, and other model's features
  - postcondition: a new song is generated and displayed to the user's browser
  - path: the user should open the application, insert the input data, and the app would render the song it generated
- *See live edits of a song*: the musician should be capable of modifying the input and see live edits of the displayed song
  - actor: musician
  - precondition: a song generated before
  - postcondition: UI is updated with the new form of the song
  - path: the user should first generate a song then modify the input variables, and the app would render the new updated song
- *Listen to songs*: the user should be able to listen to the generated song
  - actor: musician
  - precondition: a song generated before
  - postcondition: the application will start playing the generated song
  - path: the user should first generate a song then press space bar, and the app would play the song

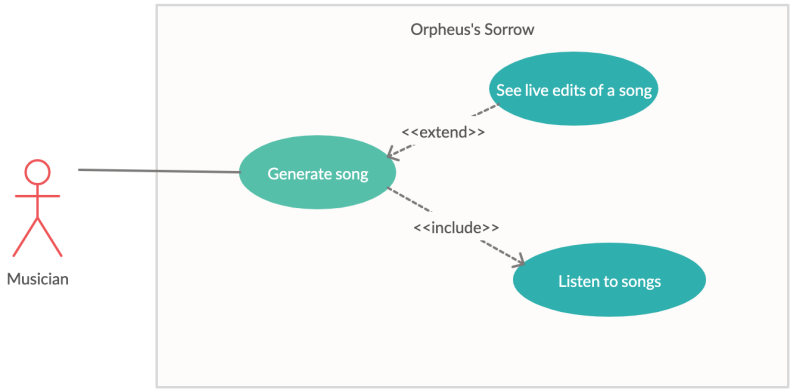

Figure 5.2: *UML use case diagram for the main features*

5.3 Design and implementation

The application is composed out of two servers, a React server for the frontend, and a Flask server for the backend, as shown in Figure *5.3*.

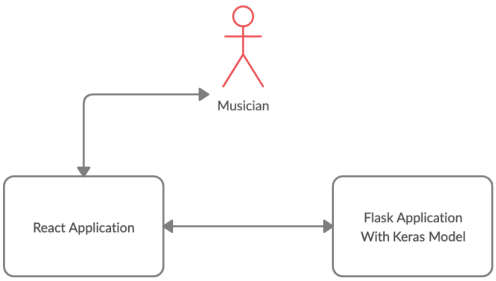

Figure 5.3: *Main structure of the application*

The main aspects of the servers' design and the interaction between the two of them or between them and the user are visible in the sequence diagram displayed in Figure 3. It explains how the application should react to user input, how the data should be processed and transmitted through the servers, considering our use cases. The following paragraphs will present the objects composing the diagram and the relationships between them.

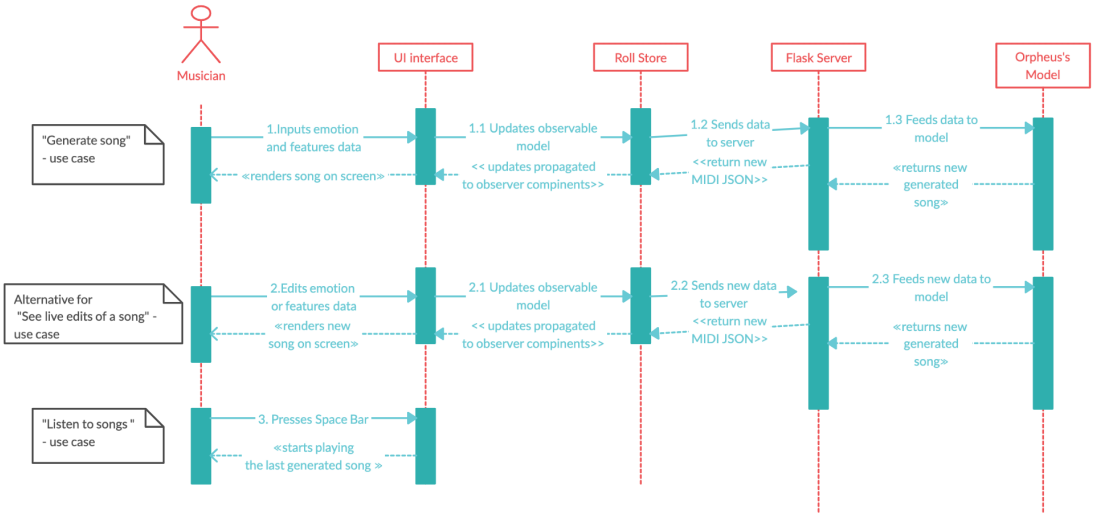

Figure 5.4: *UML sequence diagram containing the main use cases*

The UI interface represents the layer the end-user, the musician, should interact within all of the use cases. It is build using the React library, being composed out of function components and hooks due to their ease of managing logic, ease of sharing code between components, and readability.

The interface was designed using the concept of smart and presentational components [3]. The first type refers to components that should only be concerned about how things look (presentational) and how to render the data they receive, as seen in Listing *3*; they should not have any logic regarding the functionalities and only receive callbacks. The second type defines components that are concerned about how things work: they provide data and behavior that is passed down to presentational components and are linked to state management libraries, as seen in Listing *1*.

```
// Main smart component of our application in which we inject our store
const MidiContainer = () => {
  const { rollStore } = useStore();
  // other variables
  const memoisedSong = useMemo(() => {
    // mapping request data to our MIDI format
    const songRawBeats = rollStore.pianoRollsSnapshot.map(([roll, x, y]) => [
      roll * 96 + x,
      y + 16
    ]);
    // creating empty beats
    const songBeats = Array.from(Array((16 * 96) / 8), () => [
      [],
      [[INSTRUMENT, [], 1 / 16]]
    ]);
    // adding notes to corresponding beats
    songRawBeats.forEach(([beat, pitch]) => {
      try {
        songBeats[(beat / 8).toFixed(0)][1][0][1] = [
          ...songBeats[(beat / 8).toFixed(0)][1][0][1],
          pitch
        ];
      } catch {/* other error logic */}
    });
    return songBeats;
  }, [rollStore.pianoRollsSnapshot]);
  // Watching for `isPlaying` prop so we start playing the beat accordingly
  useEffect(() => {
    if (isPlaying) { // logic for starting the MIDI player }
    else { // logic for stopping the MIDI player }
  }, [isPlaying, memoisedSong, rollStore.volume]);
 // ... here we add other logic for our components
  return (
    <Wrapper>
      <PlayerContainer
        rolls={rollStore.pianoRollsSnapshot}
        beatIndex={midiRef.current?.beatIndex}
        ... injecting other props here
      />
     {/* other containers */}
    </Wrapper>
  );
};
```

Listing 1: *Example of a smart component*

```
/**
* Presentational component that renders all keys and positions them
* @param {number} scrollLeft -> value of the scrollLeft position
                                  of the Container
*                            -> used for dynamically changing the position of
                                  the keys onScroll
*/
const Keys = ({ scrollLeft }) => {
  return (
    <KeysWrapper scrollLeft={scrollLeft}>
      {NOTE_RANGE.map(item => (
        <Key key={item} isSharp={item.indexOf("#") > -1} />
      ))}
    </KeysWrapper>
  );
};
```

Listing 2: *Example of presentational component*

The Roll Store is our main state management container, being a MobX-state-tree model, customized for the application needs, using the Apisauce wrapper over Axios, a promise-based HTTP client. It defines the structure of our data through the Mobx-based models and the behavior/business logic through Mobx-based actions. It provides the observable data that is injected in the smart components, which, when updated, it triggers re-rendering, such that MobX alongside with React's reconciliation algorithms, reassures the minimum number of components that should update.

The Flask server is a small REST server that is mainly used as the middleware between the client application and the Keras model. It exposes a single REST API call, *generate_song*, that expects to receive the input parameters of our application; an example of the request is the Listing 4 each parameter being explained in the following:

- *threshold*, the amount above the value of a generated note should be to be considered valid;
- *valence and arousal*, the emotion data;
- *features*, the set of the rest of 48 features that can be tweaked in our model;
- *multiplier*, the variable used for accentuating the emotion data

Orpheus's Model represents the helper class used for creating, training, evaluating, and loading the Keras model. It also creates separated models for the encoder and decoder parts for later use, when generating a new song or normalizing and creating our metadata.

```
{
   "features":[
      45,
      23,
      35,
      // ...other features, in total 48
      23,
      50
   ],
   "threshold":50,
   "valence":30,
   "arousal":29,
   "multiplier":30
}
```

Listing 3: *Example of* ***get_song*** *request*

5.4 User interface

This section presents the steps for installing and running the Orpheus's Sorrow web application on a local machine. Then it will be discussed the interface of the app alongside the main features.

5.4.1 Installation instructions

Before downloading the application's dependencies, some prerequisites must be installed and configured on the local machine:

- `node`, a version above 10.x.x
- `yarn`, a version above 1.19.x, or `npm` above 6.x.x
- `python`, a version above 3.x with the minimum compatible `virtualenv`

After the mentioned requirements are satisfied, one can download the dependencies of the projects. For the server, one can create a virtual environment in the project's root, using the command `python3 -m venv env`, `cd model`, and then install all the project dependencies using the command `python3 -m pip install requirements.txt`. For the client, one should open a terminal in the project's root, `cd client`, run `yarn` (or `npm install`), and wait until all the dependencies are fetched.

To run the full application, one must run the backend first, meaning opening a terminal in the project root, `cd model`, `python3 server.py`. After that, the frontend can be run by also opening a terminal in the project's root, `cd client`, `yarn start` (or `npm start`)

5.4.2 Orpheus's Sorrow interface

The main UI of the paper's project can be seen in Figure *5.5*. It is composed of two parts: the player container (the upper part), and the input container (the bottom

part). Next, it will be discussed about each container separately.

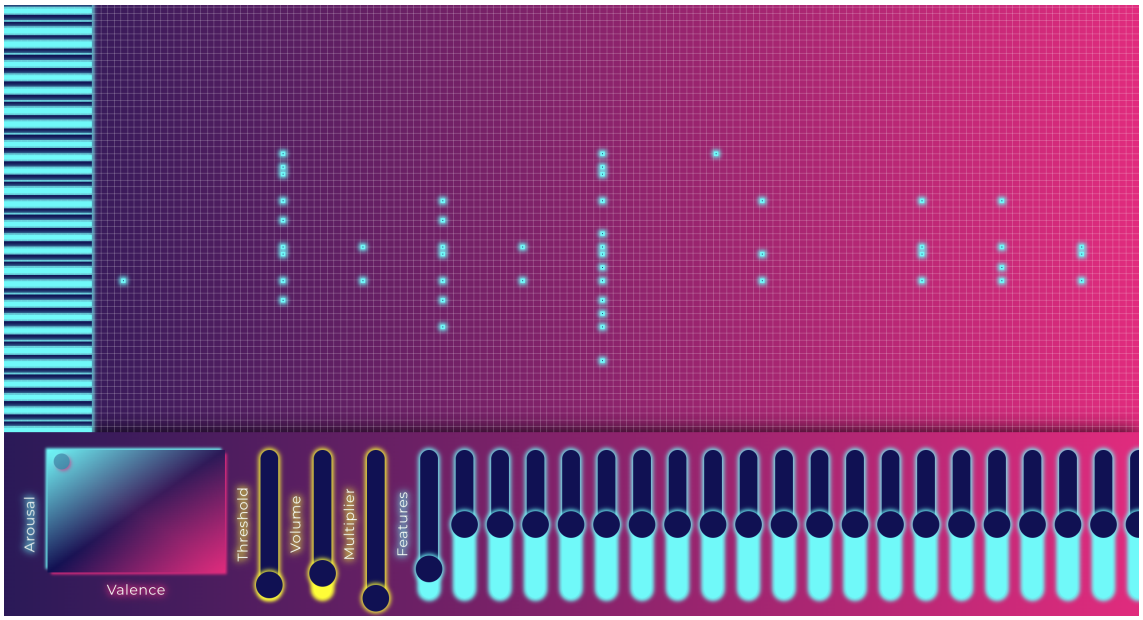

Figure 5.5: *Orpheus's Sorrow interface*

The first one, the input container, consists of three main parts: the keys, the notes, and the grid. The keys position matches the horizontal scroll due to the listener on the parent container. The notes and the grid are displayed as a piano roll, each note being positioned at its location in time and pitch, the main focus of the grid being to improve readability. When pressing the spacebar, the generated song starts playing alongside with the animation of the notes. They are translated from right to left, creating the illusion of the notes falling onto the keys and producing the sound.

The second one, the bottom container, also consists of three main parts: the emotion input, the meta-inputs, and the feature inputs. The emotion input is a 2d/cartesian slider with the x-axis representing the valence value and the y-axis the arousal value. The meta-inputs are the threshold, volume, and multiplier sliders. The first mentioned, as discussed before, represents the minimum value a note can have to be considered valid. The volume slider represents the volume of the MIDI player. When generating songs, one could desire to amplify the values of the emotion data so the model will be more influenced by them when composing. Prompted by this need, the multiplier slider was introduced; its primary purpose is to be multiplied with the valence and arousal values. The last part contains the feature sliders that can tweak the other variables of the features the model identified; in other words, it changes the structure of the song that is being composed. When modifying any of the sliders mentioned above, the song is being recomposed, hence, the possibility to edit the song almost instantaneously, easing the entire process, which is the principal purpose of the client app.

Chapter 6

Conclusion and future work

This paper tries to solve a problem that the aspiring musician has always been fighting, the music equivalent of "writer's block". It provides a tool whose primary purpose is to enhance its creational workflow by taking one of the most challenging tasks off his hands. Orpheus's Sorrow application offers an easy way of composing songs based on a given feeling, without the need for mastering music theory.

This trial represents only a proof of concept; this paper's work is merely scratching the surface of what musical composition implies, due to solely the complexity of the process, but thanks to the results, it has proved itself worthy of investing more time.

The model is capable of composing new songs with a well-defined structure, based on the input emotion data and features. Unfortunately, due to the small dataset the network was trained on, the actual emotions of the generated songs are not that clear. Despite that, general characteristics of a song composed based on the given emotions were found on the newly created compositions, meaning that the model was able to capture some information about the emotion data.

For future developments, one of the main improvements for this approach will be to enhance the dataset with more songs with normal-distributed emotion data; in this way, the prediction will be more accurate. Hence the songs will reflect the emotions better.

Another future improvement can be the ability to compose starting from an already written song, being able to enhance or continue that piece of music. This feature can be developed by using the encoder part of the model to predict the encoding of the given song, attach the emotion data or tweak the values of the encoding, and then plug the result in the decoder generating a new song.

The client app can also be improved with other features regarding the MIDI player. One of those could be the ability to edit the created song, change the playback instrument, and export the new piece as a MIDI file.

Bibliography

[1] *AIVA*.
https://www.aiva.ai/.

[2] *Flask*.
https://palletsprojects.com/p/flask/.

[3] Dan Abramov. *Presentational and Container Components*.
https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0, 2015.

[4] Thierry Bertin-mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. *The million song dataset*. In *In Proceedings of the 12th International Conference on Music Information Retrieval*, 2011.

[5] Sebastian Birch. *An Introduction to MIDI*.
http://www.personal.kent.edu/~sbirch/Music_Production/MP-II/MIDI/an_introduction_to_midi.htm.

[6] J. Brownlee. *Master Machine Learning Algorithms: Discover How They Work and Implement Them From Scratch*. Jason Brownlee, 2016.

[7] Jason Brownlee. *How to Choose Loss Functions When Training Deep Learning Neural Networks*.
https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/, 2017.

[8] Okan Bulut and Christopher Desjardins. *Exploring, Visualizing, and Modeling Big Data with R*.
https://okanbulut.github.io/bigdata/.

[9] Jacinta Chan Phooi M'ng and Mohammadali Mehralizadeh. *Forecasting East Asian Indices Futures via a Novel Hybrid of Wavelet-PCA Denoising and Artificial Neural Network Models*. *PLOS ONE*, 11:e0156338, 06 2016.

[10] François Chollet et al. *Keras*.
https://keras.io, 2015.

[11] M. Daniels. *Why Independent Musicians Are Becoming The Future Of The Music Industry*.
https://www.forbes.com/sites/melissamdaniels/2019/07/10/for-independent-musicians-goingyour-own-way-is-finally-starting-to-pay-off/#27f0c8a314f2.

[12] Rémi Delbouys, Romain Hennequin, Francesco Piccoli, Jimena Royo-Letelier, and Manuel Moussallam. *Music Mood Detection Based on Audio and Lyrics with Deep Neural Net*. In *ISMIR*, 2018.

[13] Arden Dertat. *Applied Deep Learning - Part 1: Artificial Neural Networks*. https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6.

[14] Chris Donahue, Huanru Henry Mao, Yiting Ethan Li, Garrison W. Cottrell, and Julian McAuley. *LakhNES: Improving multi-instrumental music generation with cross-domain pre-training*. In *Proceedings of the 20th International Society for Music Information Retrieval Conference*, pages 685–692, Delft, 2019. ISMIR.

[15] David Fumo. *Types of Machine Learning Algorithms You Should Know*. https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861.

[16] Amanda Ghassaei. *What Is MIDI?* https://www.instructables.com/id/What-is-MIDI/.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

[18] HackerPoet. *Composer*. https://github.com/HackerPoet/Composer.

[19] Rani Horev. *Transformer-XL Explained: Combining Transformers and RNNs into a State-of-the-art Language Model*. https://towardsdatascience.com/transformer-xl-explained-combining-transformers-and-rnns-into-a-state-of-the-art-language-model-c0cfe9e5a924.

[20] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[21] Sylvester Kaczmarek. *Machine Learning in Music Composition*. https://sylvesterkaczmarek.com/blog/machine-learning-music-composition/.

[22] Bartu Kaleagasi. *A New AI Can Write Music as Well as a Human Composer*. https://futurism.com/a-new-ai-can-write-music-as-well-as-a-human-composer.

[23] Danqing Liu. *A Practical Guide to ReLU*. https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7, 2017.

[24] Fabien Lotte. *Signal Processing Approaches to Minimize or Suppress Calibration Time in Oscillatory Activity-Based Brain–Computer Interfaces. Proceedings of the IEEE*, 103:871–890, 06 2015.

[25] Charles-Louis Machuron. *AIVA: The Artificial Intelligence Composing Classical Music*.

http://www.siliconluxembourg.lu/aiva-the-artificial-intelligence-composing-classical-music/.

[26] Samer Maini, V. Sabri. *Machine Learning for humans*. https://medium.com/machine-learning-for-humans/why-machine-learning-matters-6164faf1df12/.

[27] Chris Nicholson. *A Beginner's Guide to Neural Networks and Deep Learning*. https://pathmind.com/wiki/neural-network.

[28] M.A. Nielsen. *Neural Networks and Deep Learning*. http://neuralnetworksanddeeplearning.com/.

[29] International Federation of the Phonographic Industry. *Global Music Report 2019: State of the Industry Report*.

[30] Robert Plutchik. *The Nature of Emotions: Human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice. American Scientist*, 89(4):344–350, 2001.

[31] Colin Raffel. *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. 2016.

[32] Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall Press, USA, 3rd edition, 2009.

[33] Technopedia. *Hidden Layer*. https://www.techopedia.com/definition/33264/hidden-layer-neural-networks.

[34] Technopedia. *Input Layer*. https://www.techopedia.com/definition/33262/input-layer-neural-networks.

[35] Technopedia. *Output Layer*. https://www.techopedia.com/definition/33263/output-layer-neural-networks.

[36] Watson D. *THE WORDSWORTH DICTIONARY OF MUSICAL QUOTATIONS*. Wordsworth Reference, Edinburgh, 1991.

[37] Ma Yan, Kang Liu, Zhibin Guan, Xinkai Xu, Xu Qian, and Hong Bao. *Background Augmentation Generative Adversarial Networks (BAGANs): Effective Data Generation Based on GAN-Augmented 3D Synthesizing. Symmetry*, 10:734, 2018.

[38] Tony Yiu. *Understanding Neural Networks*. https://towardsdatascience.com/understanding-neural-networks-19020b758230.