A Deep Learning Approach to generate Beethoven's 10^{th} Symphony

Trabajo de Fin de Grado
Curso 2018–2019

Autor
Paula Muñoz Lago

Director
Gonzalo Méndez Pozo

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

A Deep Learning Approach to generate Beethoven's 10^{th} Symphony

Trabajo de Fin de Grado en Ingeniería Informática
Departamento de Ingeniería del Software e Inteligencia Artificial

Autor
Paula Muñoz Lago

Director
Gonzalo Méndez Pozo

Convocatoria: *Junio 2019*

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

31 de mayo de 2019

Dedicatoria

A José Lago, mi abuelo, a quien le hubiese encantado verme terminar esta etapa. Una vez me dijo: "Una carrera universitaria es difícil, pero la verdadera carrera es una vida sin estudios". *Non sei cándo nos veremos.*

A mi madre, por el ánimo y apoyo constante, a mi padre por la imaginación y a mi hermana por acompañarme en cada momento de mi vida.

A mis amigos, por ayudarme a levantarme del suelo cada vez que he caído a lo largo de estos cuatro años de carrera. Qué suerte que os hayáis cruzado en mi camino.

Agradecimientos

A lo largo de mi etapa universitaria he tenido la inmensa suerte de contar con profesores que me han motivado a alcanzar mis objetivos y animado a seguir aprendiendo y a dar todo de mí. Quiero dar un agradecimiento especial a Gonzalo Méndez por guiar mis pasos académicos desde que fue mi profesor hasta que ha dirigido este TFG. Gracias por aceptar esta idea tan ambiciosa, ponerle tanto empeño a que todo quedase perfecto y animarme a conseguirlo sin cortarme las alas. Sin todo esto no hubiese conseguido llegar hasta aquí.

Quiero agradecer al departamento de Ingeniería del Software e Inteligencia Artificial de la UCM por prestarme recursos informáticos de los que yo no disponía, a lo largo de este año, para conseguir los resultados reflejados en esta memoria.

Gracias a todas las personas con las que he coincidido en mi camino y han compartido su conocimiento conmigo, ayudándome a crecer y a ser mejor persona.

Abstract

Luidwig van Beethoven composed his symphonies between 1799 and 1825, when he was writing his Tenth symphony. As we dispose of a great amount of data belonging to his work, the purpose of this project is to work on the possibility of extracting patterns on his compositional model and generate what would have been his last symphony, the Tenth.

Computational creativity is an Artificial Intelligence field which is still being developed. One of its subfields is music generation, to which this project belongs. Also, there is an open discussion about the belonging of the creativity, to the machine or the programmer.

Firstly we have extracted all the symphonies' scores information, structuring them by instrument. Then we have used Deep Learning techniques to extract knowledge from the data and later generate new music. The neural network model is built based on the Long Short-Therm Memory (LSTM) neural networks, which are distinguished from others since these ones contain a memory module. After training the model and predict new scores, the generated music has been analyzed by comparing the input data with the results, and establishing differences between the generated outputs based on the training data used to obtain them. The result's structure depends on the symphonies used for training, so obtained music presents Beethoven's style characteristics.

Keywords

Music, Beethoven, artificial intelligence, machine learning, deep learning, music prediction, music generation, keras, neural network, LSTM.

Resumen

Luidwig van Beethoven compuso sus sinfonías entre 1799 y 1825, cuando estaba componiendo su décima sinfonía. Como disponemos de una gran cantidad de datos provenientes de su obra, el objetivo de este proyecto es investigar en la posibilidad de extraer patrones en su modelo composicional y generar lo que hubiese sido su última sinfonía, la Décima.

La creatividad computacional es un campo de la Inteligencia Artificial que se encuentra todavía en desarrollo, uno de los campos que abarca la creatividad computacional es la generación de música, en el que se encuentra este proyecto. Además, existe un debate abierto sobre si las máquinas pueden llegar a ser creativas, o el mérito debe atribuirse al programador.

En primer lugar hemos extraido la información de las partituras de todas sus sinfonías, estructurándolas por instrumento. A continuación hemos utilizado técnicas de aprendizaje automático para extraer conocimiento de los datos y posteriormente predecir música. El modelo neuronal se ha basado en las redes neuronales LSTM (Long Short-Therm Memory), que se diferencian del resto dado a que disponen de memoria. Tras entrenar el modelo y predecir nuevas partituras, la música generada ha sido estudiada comparando los datos de entrada con dichos resultados, y estableciendo diferencias entre los resultados obtenidos dependiendo de los datos usados para generarlos. La estructura de los resultados depende de las sinfonías que se han usado para aprender de ellas, por lo que la música obtenida presenta características reconocibles del estilo de Beethoven.

Palabras clave

Música, Beethoven, Inteligencia Artificial, aprendizaje automático, aprendizaje profundo, predicción de música, generación de música, keras, neural network, LSTM

Contents

List of figures

List of tables

Chapter 1

Introduction

1.1. Motivation

Music is an art, but also a global language present in every historical stage. Flutes have been found made with bones dated in prehistory. Although India has the oldest tradition regarding this art, it is in China where the largest prehistorical musical instrument collection has been found. The Ancient Egyptians acknowledged Thoth as the god that invented music, and there is evidence that they played harps, flutes and clarinets. Later on, music became an important part in the society-life of Ancient Greeks, creating choruses and performing at the theater, improving that way the cultural life in that moment, (Wikipedia, 2019a).

Medieval music was characteristic due to its monophonic chants. The musical notation was modified by the catholic church so chants' lyrics were written down along with the notes. Non religious music experienced a great development in the Renaissance, since with the invention of the printing press, music became less expensive due to the mass production of scores (Reese, 1959).

From 1600 to 1750 a new musical style was born in Europe which made music complexity increase exponentially: the Baroque. In this time, the first operas and polyphonic music (multiple melody lines playing at the same time) were written, and some music forms such as the Fugue, Sonata or Symphony appeared. Some significant baroque German composers are Bach, Handel, or the italian Vivaldi. Classical music covers from 1750 to 1820, and some baroque instruments evolutioned to the versions that are in use today, such as the Baroque violin or oboe, which transitioned to the common violin and oboe, while some other instruments fell into disuse, such as the Viola d'amore (Wikipedia, 2019b). Musical forms previously created where developed such as the Symphonies or the Sonata, while others where created (Trio, string quartet etc). Classical composer Wolfgang Amadeus Mozart wrote 23 string quartets for two violins, viola and violoncello. A significant

change in the Classical period was the increase of public concert, bringing composers the possibility to earn money without being hired by an aristocrat. This led to a growth in the number of orchestras, and the need of building auditoriums for them. The greatest Classical composers of this epoch are: Haydn, Mozart, Schubert or Beethoven. The two last ones are considered composers of the latest Classicism and beginnings of the Romanticism. This style was led by the individualism, emotions and the nature. The musical style became more expressive, dramatic and passionate. Some composers of this period are Wagner or Brahms. Musical motifs start to replace the melody, as it can be seen with the distinctive motive from Beethoven's Fifth Symphony, composed between 1804 and 1808.

The Impressionism is a musical movement emerged between the late 19^{th} and the early 20^{th} century, composers of this epoch such as Claude Debussy or Maurice Ravel tried new instrument combinations to obtain new sound effects.

20^{th} century music was characterized with the rhythms and sound exploration. The most meaningful genre born in this century is Jazz, and during the second half of the century, Rock music became the most popular. Electric technologies such as gramophones or radios enabled sound recordings to get to more people, causing the growth of popular music, as it allowed artists to be known worldwide by the succession of formats, form vinyl records to digital audio in the mid-1990s. Thanks to the technology evolution and the computational power obtained at this historical stage, the first Artificial Intelligence capable of generating music was created.

As Computational Creativity is a currently developing field, the goal of this project is to explore the possibilities that the Artificial Intelligence brings us in order to know how Beethoven's 10^{th} symphony would sound based on his published symphonies.

1.2. Project Aim

Given the music historical line, it is now possible to deeply explore what great composers created in the past in order to bring new music to the 21^{st} century, taking their work into account. The goal of this project is to compose a new Symphony based on Beethoven's work. We have also taken into account his life historical line, since the most brilliant symphonies were the last ones, when he was suffering from deafness and totally involved in a depression.

Romantic composer Ludwig van Beethoven wrote his Symphonies from 1799 to 1825, when he finished the No. 9. Although there is no constancy of the existence of the 10^{th} Symphony score, there exists some sheets found in Beethoven's house after his death that are thought to be part of the upcoming Symphony. Those sheets are kept in the museum dedicated to his

life in his natal city, Bonn, although they can be seen online [1]. The public manuscript is not easy to read and understand, so that existing data will not be used in this project.

The first movement of the 10^{th} Symphony [2] was composed in 1988 from 50 fragments by the musicologist Barry Cooper, who wrote a book relating Beethoven's life (Cooper, 2000). Since it cannot be proved that those found sketches were intended to be part of the 10^{th} symphony, Barry Cooper's work has caused a big controversy (Nieuwenhuizen, 2001).

A legend arises from this cause, called *"the* 10^{th} *Symphony curse"*. Following Beethoven's steps, several great composers were found dead before finishing their 10^{th} Symphony. This is the case of authors such as Franz Schubert (1797-1828), Anton Bruckner (1824-1896), Antonín Dvořák (1842-1904) or Gustav Mahler (1860-1911). The last one tried to avoid the curse by not assigning a number to his ninth Symphony, in order to be able to assign the number 9 to his tenth Symphony. Despite his effort in avoiding the curse, he was found dead while composing the last one.

This project is structured as follows. Previous work on using Artificial Intelligence in music generation and the most relevant techniques used for this purpose are exposed in the State of the art (chapter 3). After that, the work developed for this project is explained in detail, presenting the Deep Learning technique used, how the data is represented and the needed toolkits (chapter 4). Then, the Music Generation section is divided in: a introduction to the different possible musical data representations, and the music generation process. In this last section we explain the neural network design, the training and prediction and finally we have explained the three different approaches, its input datasets descriptions and results obtained with each of the experiments. The conclusions and future work are described in the last chapter, number 5.

[1] `https://da.beethoven.de/sixcms/detail.php?id=15241&template=dokseite_digitales_archiv_en&_eid=1510&_ug=Symphonies&_werkid=143&_dokid=wm188&_opus=Unv%203&_mid=Works&suchparameter=&_sucheinstieg=&_seite=1`

[2] `https://en.wikipedia.org/wiki/Symphony_No._10_(Beethoven/Cooper)`

Chapter 2

Introducción

2.1. Motivación

La música es un arte, pero también un lenguaje universal presente en todas las etapas de la historia. Se han encontrado flautas hechas con huesos datadas en la prehistoria. Aunque el país del que se conoce la tradición musical más antigua es la India, es en China donde se ha encontrado la colección de instrumentos musicales prehistóricos más amplia. En el antiguo egipto reconocieron a Thoth como el dios que inventó la música, y hay evidencias de que tocaban arpas, flautas y clarinetes. Después la música se convirtió en una parte importante de la vida social en la antigua grecia, creando coros y haciendo representaciones en los teatros, incrementando la vida cultural en aquel momento, , (Wikipedia, 2019a).

La música medieval estuvo caracterizada por cantos monofónicos. La notación musical fue modificada por la iglesia católica, para que la letra de dichos cantos pudiese ser escrita junto con las notas. La música no religiosa tuvo un gran desarrollo en el Renacimiento. Debido a la invención de la imprenta, la música se hizo más barata, dada la producción masiva de partituras (Reese, 1959).

De 1600 a 1750 se desarrolló un nuevo estilo en Europa, el cual hizo que la complejidad musical se incrementase exponencialmente: el Barroco. Durante este periodo se escribieron las primeras óperas y piezas de música polifónica (varias lineas musicales tocadas al mismo tiempo), y aparecieron algunas formas musicales como la fuga, la sonata o la sinfonía. Algunos compositores barrocos significantes fueron los alemanes Bach, Handel o el italiano Vivaldi. La música clásica abarca desde 1750 hasta 1820, y destaca la evolución de instrumentos barrocos hasta las versiones que conocemos actualmente, como el violín o el oboe, mientras que otros instrumentos cayeron en desuso, como la Viola de amor (Wikipedia, 2019b). Las formas musicales creadas previamente, como las sinfonías o las sonatas tuvieron un gran desarrollo, a la vez que surgían otras nuevas, tales como los tríos, los cuartetos de cuerda,

etc. El compositor clásico Wolfgang Amadeus Mozart escribió 23 cuartetos de cuerda para dos violines, viola y violoncello. Un cambio significativo en el periodo clásico fue el incremento de los conciertos públicos, aportando a los compositores la posibilidad de ganar dinero sin tener que ser contratados por un aristócrata. Esto conllevó a un crecimiento en el número de orquestas, y en la necesidad de construir auditorios para ellas. Los compositores clásicos más importantes son: Haydn, Mozart, Shubert o Beethoven. Los dos últimos son considerados compositores de finales del Clasicismo y principios del Romanticismo. Este movimiento estuvo conducido por el individualismo, las emociones y la naturaleza. El estilo musical se hizo más expresivo, dramático y apasionado. Algunos compositores son Wagner o Brahms. Los motivos musicales empezaron a sustituir a las melodías, como puede verse en el destacable motivo de la quinta sinfonía de Beethoven, compuesta entre 1804 y 1808.

El Impresionismo es un movimiento musical que emerge a finales del siglo XIX y principios del XX, los compositores pertenecientes a este movimiento como Claude Debussy o Maurice Ravel probaron combinaciones de instrumentos nuevas para obtener efectos sonoros diferentes.

La música del siglo XX se caracterizó por la exploración de ritmos y sonidos, el género más significativo nacido en este siglo fue el Jazz, y durante la segunda mitad del siglo, el Rock fue el estilo más popular. Las nuevas invenciones como el gramófono o la radio permitieron que las grabaciones de sonido llegasen a más personas, propiciando así el crecimiento de la música popular, ya que permitió a los artistas ser conocicos mundialmente a través de la sucesión de formatos, desde vinilos hasta el audio digital a mediadios de 1990. Gracias a la evolución de la tecnología y la capacidad computacional obtenida en esta etapa histórica, se creó la primera Inteligencia Artificial capaz de generar música.

Dado que la Creatividad Computacional es un campo en desarrollo actualmente, el objetivo de este proyecto es explorar las posibilidades que la Inteligencia Artificial nos aporta para conocer cómo hubiese sonado la décima sinfonía de Beethoven basándonos en sus sinfonías previamente publicadas.

2.2. Objetivo del proyecto

Dada la línea histórica musical, ahora es posible explorar en profundidad lo que los grandes compositores crearon a lo largo de la historia para conseguir nueva música en el siglo XXI en base a su trabajo. El objetivo de este proyecto es componer una nueva sinfonía basada en la obra de Beethoven. También hemos tenido en cuenta la línea histórica de su vida, dado que las sinfonías más brillantes fueron las últimas, cuando estaba sufriendo sordera y estaba totalmente inmerso en una depresión.

El compositor romántico Ludwig van Beethoven escribió sus sinfonías en-

tre 1799 y 1825, cuando terminó la número 9. Aunque no existe constancia de la existencia de la partitura de la décima sinfonía, existen algunas partituras encontradas en la casa del compositor tras su muerte el 26 de Marzo de 1827, que se cree que podrían pertenecer a la siguiente sinfonía. Estas partituras están conservadas en el museo dedicado a su vida en su ciudad natal, Bonn, aunque pueden verse online [1]. El manuscrito no es fácil de leer y entender, por lo que esa información no será utilizada en este proyecto. En el primer movimiento de la décima sinfonía [2] construyó en 1988 a partir de 50 fragmentos por el musicólogo Barry Cooper, el cual escribió un libro relatando la vida de Beetoven (Cooper, 2000). Dado que no se puede comprobar que esos manuscritos fuesen a formar parte de la décima sinfonía, el trabajo de Barry Cooper ha causado mucha controversia.

A raíz del fallecimiento de Beethoven surge una leyenda llamada *"la maldición de la décima sinfonía"*. Siguiendo los pasos de Beethoven, otros grandes compositores fallecieron antes de terminar de componer su décima sinfonía. Este es el caso de autores como Franz Shubert (1797-1828), Anton Bruckner (1824-1896), Antonín Dvořák (1842-1904) or Gustav Mahler (1860-1911). El último intentó evitar la maldición no escribiendo el número en la novena, y escribiendo el número 9 en la décima. A pesar del esfuerzo por evitar la maldición, falleció componiendo la última.

Este proyecto está organizado de la siguiente forma. El trabajo previo en generación de música utilizando Inteligencia Artificial y las técnicas más relevantes existentes se exponen y explican en el estado del arte (Capítulo 3). Después, el trabajo desarrollado para este proyecto se explica en detalle, presentando la técnica de aprendizaje profundo utilizada, cómo se representa la información y las herramientas utilizadas (Capítulo 4). A continuación, el apartado de generación de música está dividido en: una introducción a las diferentes formas de representar los datos musicales, y el proceso de generación musical. En este último apartado explicamos el diseño de la red neuronal, el entrenamiento y predicción y finalmente los tres diferentes acercamientos al objetivo del proyecto, sus datos de entrada y los resultados obtenidos con cada experimento. Las conclusiones y el trabajo futuro se desarrollan en el último capítulo, número 6.

[1] `https://da.beethoven.de/sixcms/detail.php?id=15241&template=dokseite_digitales_archiv_en&_eid=1510&_ug=Symphonies&_werkid=143&_dokid=wm188&_opus=Unv%203&_mid=Works&suchparameter=&_sucheinstieg=&_seite=1`

[2] `https://en.wikipedia.org/wiki/Symphony_No._10_(Beethoven/Cooper)`

Chapter 3

State of the art

In this chapter we describe the historical line that Artificial Intelligence has followed, more concretely the evolution of one of its sub fields, the computational creativity and the experiments that has been previously released. Artificial Intelligence techniques used for creativity purposes, Markov Chains and Machine Learning, will be also detailed. After exposing all the possibilities, in the conclusions section we will explain which technique of the ones previously exposed has been used for this project development.

3.1. Computational Creativity

Computational creativity appears from the intersection of Artificial Intelligence and Arts. Studied since the latter half of the 20^{th} century, it is considered the study of software development that presents the same creativity as humans, or enhances human creativity, without being creative itself. Progress in this field have raised many discussions, questioning the cultural value of the output. It has been hard for the society to assume that machines can have intelligence, so it is being even harder to assume that they may be endowed with creativity. (de Mántaras, 2017, p.99-123)

It is yet a novel field, so some things are not determined already, such as how to establish how good or creative the output given by the software is. Some experts declare that the value of the products obtained needs to be determined with a Turing Test (Pinar Saygin et al., 2000; Boden, 2010). If humans can't differentiate between machine or human generated products, then the output has resulted creative. Some others directly concludes that it is creative if it generates a certain impact on a human being. For instance, if a generated joke makes someone laugh. (Jordanous, 2012)

There has been relevant experiments on linguistic creativity, generating narrative (Gervás et al., 2005), getting to write the lines of a musical called *Beyond the fence* (Gardner, 2016), showed for the first time in London's Arts Theatre. But also there have been successful experiments generating poems

(Montfort et al., 2012; Gervás, 2001), as well as rimes, sarcasm or irony in jokes (Binsted and Ritchie, 1997; Ritchie, 2009).

Visual arts creativity's most famous program is AARON (Cohen, 1995), capable of painting on a canvas with a brush using a robotic arm. The Painting Fool, (Colton, 2012) emulates several painting styles. This field experimented a great growth due to the auctioned AI generated painting, *Portrait of Edmond de Belamy (2018)*, by the Christie's art gallery of New York, getting the price of 432.500$, whose algorithm was designed by *Obvious*[1]

3.1.1. Music Creativity

This Computational Creativity sub-field started in the early 50's, although the most relevant works are mainly focused on generating coherent sounds and scores for the human musicians use.

The starting point in music generation with Artificial Intelligence was the formalization of the Markov chains in the early 1900s, later used in the first music generation project (Hiller and Isaacson, 1957). This machine generated the *ILLIAC's suite*, a string quartet [2]. Generated notes were tested by heuristic compositional rules. In case that the rules were not violated, they were kept; otherwise, a backtracking process was followed. This project excluded any emotional or expressive generation, by just focusing on the notes.

Later on, a system called *CHORAL*, which produced the corresponding harmonization of a given Bach Choral, was developed creating rules and setting heuristics in a logic-programming language created by the author for this purpose (Ebcioglu, 1990).

While this probabilistic system can only produce subsequences that already exist in the original data, in 1989 a new technique is started to be used for this purpose: Recurrent Neural Networks (Todd and Loy, 1991). Since this deep learning method is limited by its short-term coherence, in 2002 Long Short Term Memory Networks started to be used for this purpose. Liu and Ramakrishnan (2014) brought baroque composer Johann Sebastian Bach back, by trying to create music that has both harmony and melody and is passable as music composed by humans. Melodies generated with LSTM networks in existing projects have resulted more musically plausible than with other models, such as Gated Recurrent Unit (GNR) (Nayebi and Vitelli, 2015). The first music generation project that used neural networks is MUSACT (Bharucha, 1992), which focuses on learning the harmonic model and generates expectations after listening to a certain chord. Some other projects that use variations of this networks models to generate new sounds

[1] `http://obvious-art.com/index.html`

[2] `https://www.youtube.com/watch?v=fojKZ1ymZlo`

have been developed through the last few years, using raw audio (Kalingeri and Grandhe, 2016). BachBot (Liang et al., 2017) composes and completes music in the style of Bach chorales using an LSTM generative model. They conducted a discrimination test to determine if the generated music was similar to Bach's chorales with 2336 participants, getting a rate of only a 1% of the people correctly determining which music was generated with BachBot.

SICIB (Morales-Manzanares et al., 2001) is another example of a system capable of generating music, but in this case, it works using corporal movements thanks to sensors in a dancer. Google's *Magenta*[3] have been developed since 2016. It explores the roll of Machine Learning in the artistic creation process. *DeepMusic*[4], is integrated in Amazon's assistant Alexa as a skill, so it plays AI generated music. Chinese company *Huawei* has published the unfinished part, third and fourth movements, from Shubert's symphony No. 8 (Mantilla, 2019), which the author left unfinished on purpose. Using neural networks, the system gave to the musicians some ideas to continue the music, and then musician and composer Lucas Cantor worked on them. The final version has been played on the 4^{th} of February, 2019, in a unique concert in London.

Currently best-work known on computer music composition is EMI, (Cope and Mayer, 1996). This system has successfully emulated Mozart, Brahams, Bach, Rachmaninoff or Chopin's music, generating new music [5]. It searches a pattern or signature as Cope's labeled, in at least two existing pieces of a concrete compositor. Using one of the artist scores, it locates the signatures to generate the new music, and in order to compose the music between signatures, it uses a rule analyzer.

The IAMUS (Quintana et al., 2013), named this way after the god of music, Apolos's son in the ancient Greece, is a computer system created at *Universidad de Málaga*. It is capable of composing a full score in 8 minutes, using genetic algorithms, whose music has been played by the London Symphony Orchestra. In this case, chromosomes including all the notes information are randomly generated, and fitness functions are applied to each of them. If a note is coded to be played by a violin and this instrument does not have the possibility to play that note, it is changed. After generating around 100 scores, a human composer chooses the best one as the final output.

Another challenging field relating Musical Creativity is Music Improvisation, since it has more difficulties from a creative point of view. Using Genetic Algorithms, GenJam (Biles, 1994) emulates a Jazz musician in his or her improvisation learning process, while the Continuator (Pachet, 2003) uses a Markov model to generate music in standalone mode, as continuations

[3] `https://magenta.tensorflow.org/`

[4] `https://amzn.to/2DBRwJc`

[5] `http://artsites.ucsc.edu/faculty/cope/5000.html`

of musician's input, or as interactive improvisation.

The most relevant workshop that combines music and Artificial Intelligence is the *International Computer Music Conference*, the first edition which took place in 1974.

3.2. Markov chains

A Markov chain is a stochastic model that, given a previous state, describes a sequence of possible events that can happen after it, generating the probability for each event to happen based on the last one, and storing them in a transition matrix. This is also called *sampling*, (Jean-Pierre Briot and Pachet, 2017, p. 52).

The so called *Markov property* is fulfilled if a future prediction of the process can be made based on the current state as accurate as if it knew the full history. This technique was the first approach to music prediction, proposed by (Hiller and Isaacson, 1979), characterizing each musical event in different states of the Markov chain. Although systems needs much more context to generate a more precise sequence of notes, it has been used since then for this purpose, as it can be seen by latest studies, such as (Tracy, 2013). There also exists an open source project called *MarkovGenerator* [6], that allows the user to simply generate music from chosen Midi files.

In the following example it can be seen the probability that a certain note or event appears, depending on the previous one.

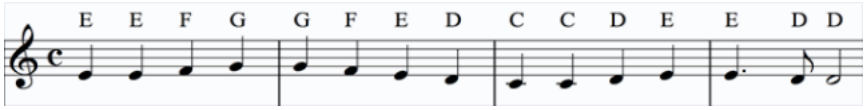

Figure 3.1: Beethoven's 9^{th} symphony's *Ode to Joy* main motive

	C	**D**	**E**	**F**	**G**
C	50%	50%	0%	0%	0%
D	33.3%	33.3%	33.3%	0%	0%
E	0%	40%	40%	20%	0%
F	0%	0%	50%	0%	50%
G	0%	0%	0%	50%	50%

Table 3.1: Transition matrix based on Figure 3.1

The current state is given by the row and the prediction by the column, so it can be seen that the probability that an E appears is a 33.3% if the

[6] `https://forum.pdpatchrepo.info/topic/10791/markovgenerator-a-music-generator-based-on-markov-chains-with-variable-length`

previous event was a D, 40% in case that the last note was another E or, most probably, it appears if the last note was an F. Having this information, a transition state diagram can be drawn. (Kemeny and Snell, 1976).

This technique is also used for text prediction in common phone applications such as instant messaging, but there exist some other projects developed using this model such as a Donald Trump tweet prediction [7].

3.2.1. Hidden Markov Models

In some cases, the relevant events that need to be taken into account are unseen, so the system is partially observable. The Hidden Markov Models (HMM) allow us to explore both types of events, the seen and the unobserved (hidden). Furthermore, this model differs from the Markov Chains as there is also a sequence of observations obtained from a vocabulary. The so called *emission probabilities* express the probability to obtain an observation from a certain state.

A first-order Hidden Markov Model establishes two different assumptions. First, the probability of an state is dependent only on the previous one. Second, the probability of an observation depends on the state that produced the previous observation. (Seymore et al., 1999)

This model is applied in pattern recognition problems such as speech handwriting, gesture or speech recognition (Rabiner, 1989), text abstraction (Conroy and O'leary, 2001), but also in bio-informatics (Krogh et al., 2001). This model has been used to generate music, although it has been proved that using this technique in a computational creativity music composition project brings some limitations such as the lack of global structure and melodic progression that the generated piece has (Yanchenko, 2017).

3.3. Machine Learning

This subfield of Artificial Intelligence has experimented a great evolution due to the data abundance and the growth of computational capacity. In machine learning, computers identify patterns by learning and predicting from a dataset, but it is also focused on researching in computational complexity, trying to improve the so called NP-hard problems, in which problems from NP difficulty, those that can be solved in a polynomial time.

In order to face a Machine Learning problem, we need a big amount of data. The datataset used in the creation of the model is divided into several sets, as shown in Figure 3.2. The model is initially fit on a training dataset, and learns from it. Once the model is trained with that set, it is able to predict new values over the test set, taken from the initial dataset, so it provides an evaluation of how the final model fits on the training dataset,

[7] `https://filiph.github.io/markov/`

as it has never been trained over this dataset. Evaluating the model on this set provides the possibility to regularize the process, as it returns an error measure that can determine if there has been overfitting on the training dataset, which means, learning too much about each examples that the model doesn't have the ability to generalize in order to find patterns (Bishop, 2006).

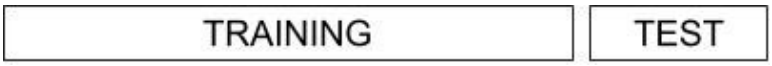

Figure 3.2: Simple Splitting

There are several ways to split the main dataset for the evaluation process. As it can be seen in Figure 3.2, the simple splitting consists on dividing the main set in 3/4 for training data and 1/4 for testing, in case the test data is representative and it has the same characteristics as the data shown in the training set. The second splitting method is the Cross Validation, which iterates over more subsets, so the accuracy is more precise since it constitutes the mean of all the parameters obtained in each iteration. An example of cross validation is the K-Fold Cross Validation, being K = 4 in Figure 3.3. It can be seen that, instead of taking the same set for testing as in the simple splitting (Figure 3.2), it is maintained the length of the testing set (1/4), but iterates over several tests sets with that size.

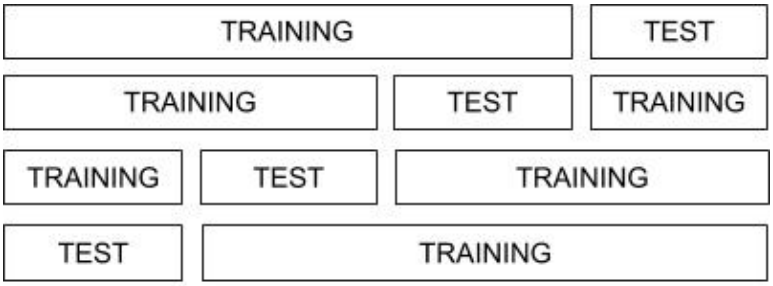

Figure 3.3: K-fold cross validation

Machine learning can be applied to a wide range of fields such as medical diagnosis (Kononenko, 2001), stock market analysis (Patel et al., 2015), speech recognition (Amodei et al., 2016), DNA sequence classification (Cho and Won, 2003), robotics or temporal series prediction such as weather, solar radiation (Voyant et al., 2017) or music prediction. There are several types of learning, such as supervised (Linear regression, Decision trees...), unsupervised (Clustering, K-means algorithm) or reinforcement learning, as described in the following subsections.

3.3.1. Supervised learning

All the examples in the dataset are constituted by attributes and the class to which they belong. The supervised learning's goal is to find classification rules in order to label new examples, i.e. assign each new example to its belonging class. For instance, if we had a dataset containing information about fruits, such as the diameter, length and shape, the supervised learning system should be able to classify a new fruit example given its diameter, length and shape.

There are several ways to determine the class of a given example. Some relevant supervised machine learning algorithms are:

- **Linear regression:** Mostly used for finding relationships between variables. In this type of analysis there always exists a linear relationship between them (Seber and Lee, 2012). Lets use an example to predict the Sales that our business will achieve depending on the amount of money we expend in advertising, given the data of the past 9 years.

Sales	**Advertising**
651	23
762	26
856	30
1063	24
1190	43
1298	48
1421	52
1440	57
1518	58

Table 3.2: Sales per amount expend in advertising in millions of Euros

  Table 3.2 is represented in Figure 3.4, where the red line shows the dependency between both axis, so the amount of money that should be spent in advertising can be estimated depending on how much we want to earn or vice-versa.

- **Decision trees:** Covering both classification and regression problems, it uses a tree-like model to decide on a certain data containing conditional control sequences. Decision trees are written from the root to the leaves. The internal nodes constitutes the condition, which based on an attribute, common to all the dataset's entries, will divide the set into two different subsets. The leafs corresponds to the final decision, i.e. the class to which the new example is classified.

  The music genre of a song can be classified using a decision tree (Bressan et al., 2017).

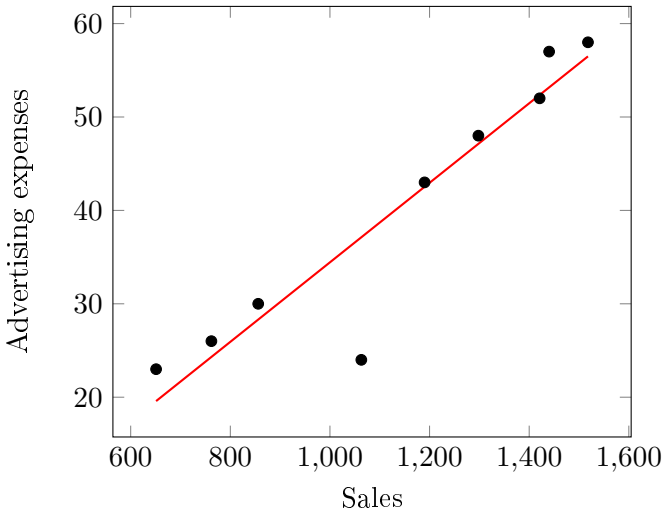

Figure 3.4: Sales per advertising expenses

- **Random forest:** Used for both kinds of machine learning problems, regression and classification. It works building several decision trees, given some observation and facts, so each tree will return its classification results for that class, and the forest will return the class having more *votes*, that will be the final prediction (Breiman, 2001). This method prevents the over-fitting problem, so the decision trees are not really concrete and the new examples can be labeled in a general way.

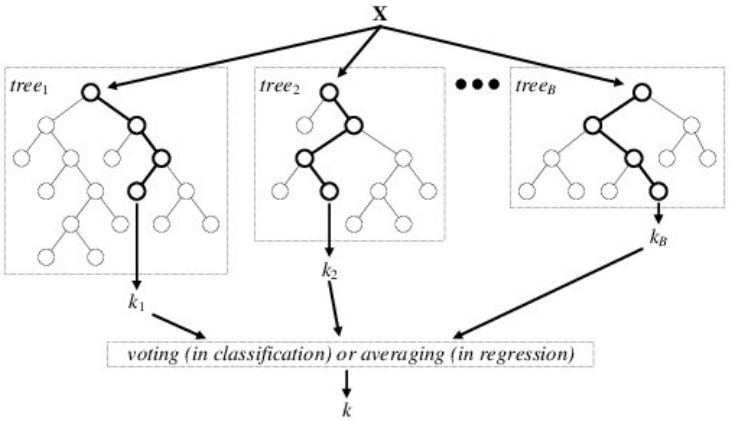

Figure 3.5: Random forest visual performance

- **Support vector machines (SVM):** Algorithm that can be used for both classification or regression problems, although it is more common

to use it for classification. Each dataset's instance is represented in an N-dimensional space, being N the number of attributes that each example has. After that, it finds the frontiers that separates better the different classes (Wang, 2005). Usually, to find the optimal hyperplane determining the different classes, we need to maximize the margin between the support vectors (nodes of the different classes closest to the frontier). As it can be seen in Figure 3.6, the blue and red nodes placed in coordinates (-8, 1) and (-4, 3) respectively are the so called support vectors, so the algorithm tries to maximize the margin between each hyperplane drawn over both support vectors and the final decision boundary, shown as a dashed line.

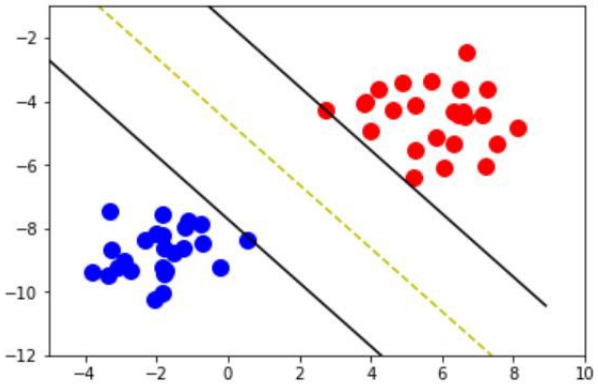

Figure 3.6: SVN visual performance

3.3.2. Unsupervised learning

Examples in the dataset used for training don't belong to a class, they don't correspond to any output variable, so the system finds patterns over the training set in order to classify them and label new ones. To find those structures, there exists several classification methods such as clustering or dimensionality reduction. Some popular non supervised machine learning algorithms are:

- **K-means:** Algorithm used for clustering problems. The goal is to find K groups or clusters in the input data, so each example is assigned to a clusters. For each one of them, a centroid is distinguished, which will be the most representative example of each label. In order to find patterns in the training set, the data is represented in the N-dimensional space (being N the number of attributes that each example has). In each iteration, a centroid is chosen, and then each instance is labeled as the closest centroid based on the Euclidean distance. After that, the

centroids are recomputed by performing the mean of each data points in the cluster. Both steps are repeated till no data points change its cluster (Hartigan and Wong, 1979).

- **Hierarchical Clustering:** Machine learning algorithm whose purpose is to build a clusters hierarchy. It begins with all the examples in the dataset assigned to a different cluster, so the algorithm will have as many clusters as the different examples in the input dataset. Then, two close clusters will join into the same one, and so one. Finally, all the examples will be joint into one single cluster. The output of this algorithm can be graphically represented with a dendrogram. It is built bottom-up, and it shows how the data has been joined into the different clusters, and the relationship between the initial examples. Using this technique and studying the evolution of stellar populations, White and Frenk (1991) researched on the Galaxy formation.

3.3.3. Reinforcement learning

The system receives some type of gratification depending on the level of accuracy, so it can improve its behavior. Typically, Reinforcement Learning algorithms use dynamic programming techniques, and is based on the Markov decision process.

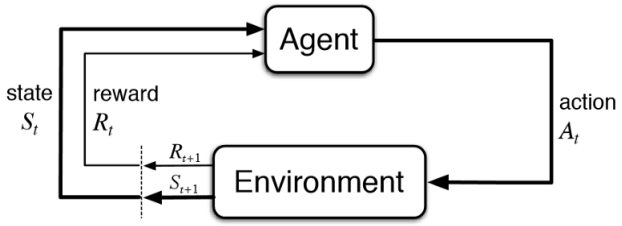

Figure 3.7: Reinforcement learning process

As it can be seen in the Figure 3.7 here are two relevant components, an agent (the RL algorithm) and an environment in which the agent is currently working. In each step of the interaction the environment sends its state to the agent, which chooses a certain action to be applied and sends it back to the environment, changing its state. The environment then sends back its new state and a reward to the agent. Finally, the Agent updates its knowledge based on the reward. This process ends when the environment evaluates the final state. (Kaelbling et al., 1996).

Some of the most popular reinforcement learning algorithms are Q-learning, State action reward state action (SARSA), Deep Q Network (DQN) or Deep Deterministic Policy Gradient (DDPG).

3.4. Deep Learning

Deep learning is included in the field of Machine learning, but the difference is that Deep learning models use neural networks. The first artificial network was developed between 1950 and 1960, and it is called a Perceptron (Rosenblatt, 1958). Rosenblatt proposed the model shown in Figure 3.8 in

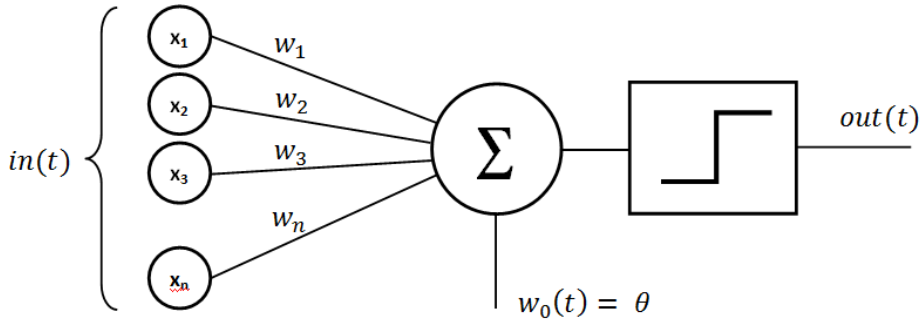

Figure 3.8: Perceptron model

which the neuron has n binary inputs, and produces one single output, 0 or 1, based on an activation function. In order to compute the output, we can assign weights (w) to each input, representing the importance of each entry, so if the addition of all those weights multiplied by the input is greater or equal than a threshold, the output is 1, otherwise it is 0. It is mathematically represented in Equation 3.1.

$$output = \begin{cases} 0 & if \quad \sum_i x_i w_i < threshold \\ 1 & if \quad \sum_i x_i w_i \geq threshold \end{cases} \tag{3.1}$$

As a single perceptron can not emulate the human process of making a decision, a network composed by several neurons is created. The so called Multilayer Perceptron can be seen in Figure 3.9, also called the Feedforward Neural Network. Each neuron has a set of inputs, as detailed in Figure 3.1, and a column of perceptrons compose a layer. The output of each neuron is forwarded to each perceptron in the second layer, as it is fully connected. The second layer, also called hidden, can produce more complex decisions than the first one, as it weighs up the results from the first layer, and repeats the same process. In case the problem is a binary classifier or a regression problem, the output layer will be composed of just one neuron, but if it is a multiple classification problem, it will have as many neurons as possible labels.

This model has evolved to the artificial neurons commonly used nowadays, the Sigmoid. This neurons are an improved version of the perceptron, since changes on the weights do not change the output in a considerable way.

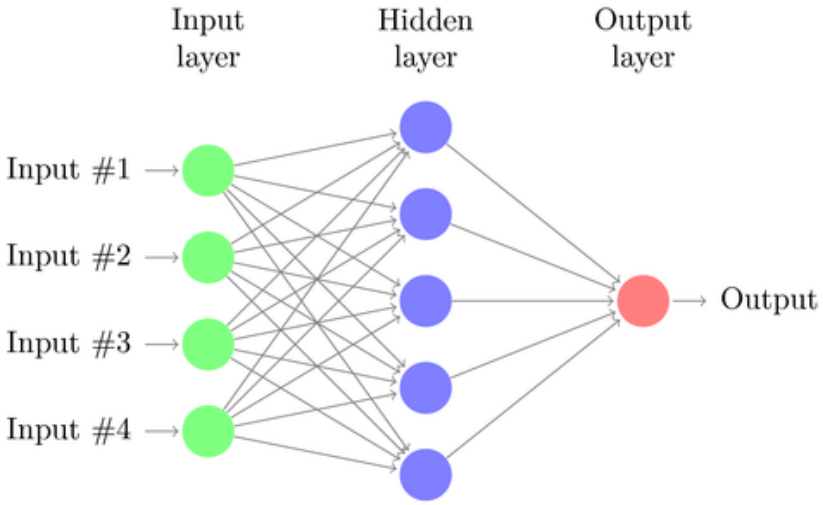

Figure 3.9: 2 layer Feedforward network

A new concept is introduced, the bias (b), which is a constant added to every output to normalize it. In this case, the input is not binary, it can take any value between 0 and 1, and the output is determined by a sigmoid function, whose Equation is shown in 3.2 and its shape in Figure 3.10, instead of an activation function based on a threshold. (Nielsen, 2015).

$$\begin{aligned} z &= w \cdot x + b \\ \sigma(z) &= \frac{1}{1 - e^{-z}} \end{aligned} \tag{3.2}$$

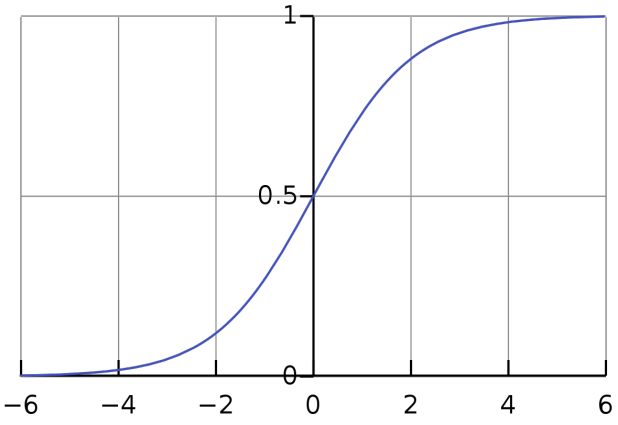

Figure 3.10: Sigmoid function shape

The deep learning process can be supervised, semi-supervised or unsupervised (Goodfellow et al., 2016). This architectures have been previously applied to fields such as Natural Language Processing (Collobert and Weston, 2008), Social Network Analysis (Perozzi et al., 2014) and to study the Computational Creativity (Nguyen et al., 2015).

There exist different types of neural networks. We can distinguish the previously seen Feedforward Neural Networks, Convolutional, commonly used for image recognition or Generative Adversarial, which consist on establishing one neural architecture against a second one, to learn from each other. Finally, we will explain the Recurrent Neural Networks, and we will get more deeply into the Long Short Term Memory Networks, which will be used for the development of this project.

3.4.1. Convolutional Neural Networks (CNNs)

The advancements in Computer Vision has been constructed over this type of neural networks. Highlighted in the image recognition field, they are used to identify faces (Lawrence et al., 1997), human actions (Ji et al., 2013) or traffic signs, so they are, among other things, the main engine that contribute to self driving cars to have vision (Bojarski et al., 2016).

The architecture of this kind of network assumes that the input is an image, it assigns weights to several objects in the picture, so the model turns to be more precise. Each layer has its neurons always arranged in 3 dimensions: width, height, and depth, as it can be seen in the third layer of the Figure 3.11. That Figure shows a 3-layer CNN, in which the first layer is the input, the second and the third are the hidden ones, and the last one corresponds to the output layer. Every layer transforms a 3D input into a 3D output of neuron activations. Since the network will reduce the image into a vector of class scores, the input of the first layer will be the image itself so it will have the same dimensions as the picture, and the output layer will be a vector of class scores.

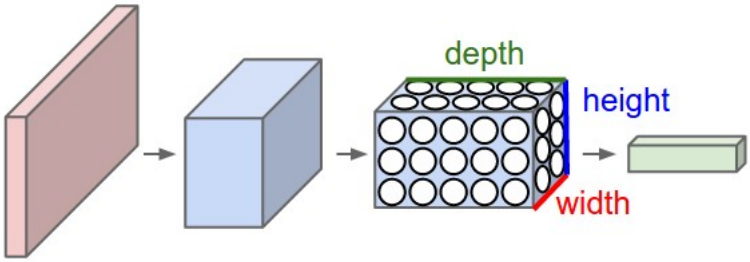

Figure 3.11: 3-layer Convolutional Neural Network

After some epochs or iterations, the model can differentiate between dominating and less important features in the image and label them using as final

layer the Softmax Classification, which returns as output a vector of probabilities for a list of outcomes.

3.4.2. Generative Adversarial Networks (GANs)

Proposed in 2014 by Ian Goodfellow, this type of Neural networks consists of two different architectures established one against the other (Goodfellow et al., 2014). The structure can be seen in Figure 3.12. Both networks are simultaneously trained, being the first architecture the one with the generative roll, G, and the second one the discriminative, D. This last network estimates the probability that an example came directly from the training data or from the generative architecture. The commit of the first network is to maximize the probability of the second architecture to make a mistake, generating samples similar to the ones in the training data.

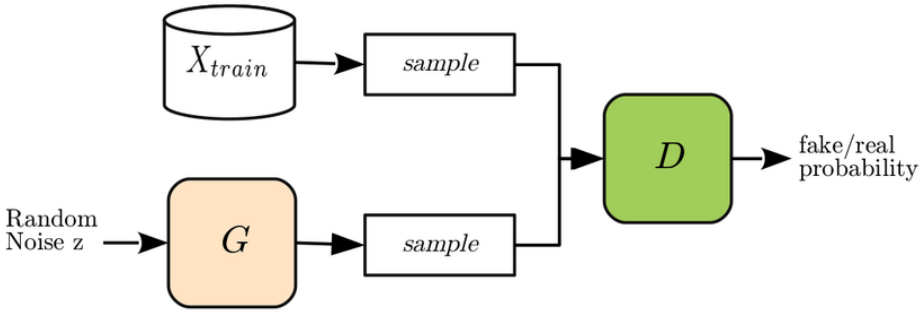

Figure 3.12: Generative Adversarial Networks (GANs)

3.4.3. Recurrent Neural Networks (RNN)

We can think about this kind of networks as a graph in which each neuron or node performs the same operation. Recurrent Neural Networks process sequences while retaining a memory of the previous elements. They are used for speech recognition (Graves et al., 2013b), or temperature measurement (Ku et al., 1992). It is useful when the element expected to be generated is dependent on the input previously received, such as the next word in a sentence or a note in a musical piece. Each cell in the network has some kind of memory, so it includes the dependency of the expected output with the previous events.

Figure 3.13 shows how each neuron in this model loops in them, so the information persists. First, the neuron will receive as input x_0, and it will generate h_0, which with x_1 will be the input of the next loop, and so on. So, it is shown that the model keeps remembering the context while training.

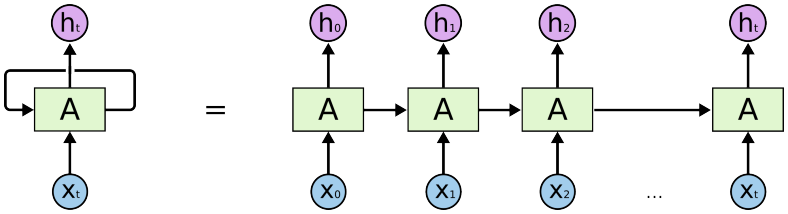

Figure 3.13: RNN: unrolled version

The use of this type of networks allows the system to take into account the context when making a prediction.

3.4.4. Long Short Term Memory Networks (LSTM)

Proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber (Hochreiter and Schmidhuber, 1997), they are a variant of RNN that can learn long-term dependencies just by changing, in the graph proposed by the RNN, each cell by improved ones. They can recognize long-term patterns, so they are useful when the network has to remember some information for a long period of time. That is the reason why this kind of networks are most accurate to be used in a music generation problem.

They have the ability to connect previous knowledge to the present task. Sometimes, for instance in speech recognition, we do not need to "remember" many things in order to guess which will be the next word, but sometimes phrases are more complex, so we need to have more context in order to predict the following word. As proved with Figure 3.14, we need the memory that this type of networks own. We have the sequence F - F - F, a predictor without memory would return another F. Although by learning from the notes before, it can extract that after three equal notes, it is probable that the upcoming note is two lines below the last one.

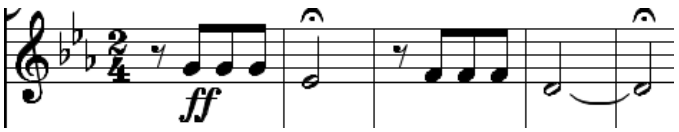

Figure 3.14: Beethoven's Fifth symphony snippet

Memory provided in each neuron can be seen as a cell that decides to store or forget the information received based on the priority assigned to that data. The algorithm learns through time which information is more relevant in order to set the importance of each data, represented as weights.

As it can be seen in Figure 3.15, the top line represents the flow of the cell state. Also, several layers are shown, such as the first sigmoid, which takes information from the previous state and determines if it is useful or

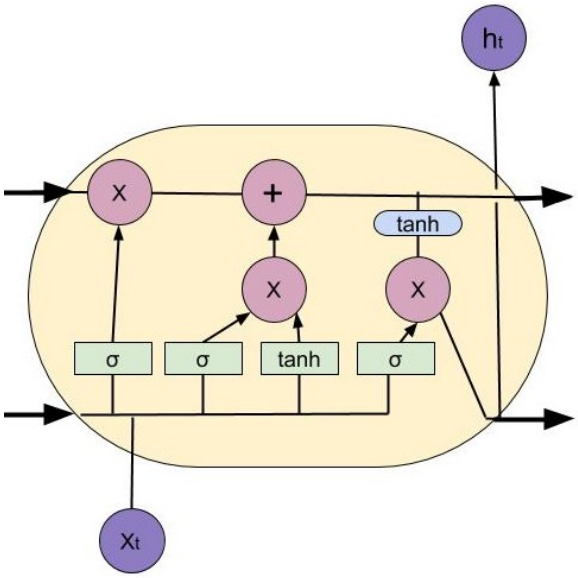

Figure 3.15: LSTM Neural Network cell

not, returning 0 or 1. As it is shown with the vertical arrow, it directly affects to the flow of the cell state. The second layer is composed of another sigmoid, which chooses the data to be updated from the previous state. The *tanh* component creates a vector of candidate values to be added to the state. The combination of both will be added to the current cell state. The final sigmoid layer decides which parts of the current state are more relevant. Those will be sent to a *tanh* function, which will convert the state into 1 or -1.

3.4.5. Toolkits

In this section we will detail the state of the art in different technologies that are useful to achieve this project's goal.

We will expose two types of tools, the first ones are intended to easily get into a Machine learning project, and the second ones are oriented to a music project, simplifying the cooperation between music and technology.

3.4.5.1. Deep learning toolkits

Given the accelerated advancement of Artificial Intelligence technologies and computation capacity, some open source tools have been developed to get into a Machine Learning project. Attending the wide range of possible programming languages with which a project of this characteristics could be developed, the most popular is Python, since its deep learning frameworks have been developed increasingly in the past 4 years (Raschka, 2015).

The most popular Python open source toolkits that can be used for a Machine learning project are the following.

- **TensorFlow**[8]**:** Released by Google in 2015, it is one of the most maintained and used frameworks for this kind of projects. It allows the user to develop neural networks using flowgraphs, suited for complex numerical computations of high volumes of data and and it is not only available for Python, but C++ or Java too. It was developed for supporting Google's research projects, although nowadays it is used by other big companies such as Intel or Twitter.

- **Theano**[9]**:** Developed in 2007, its main functionality lies on working with mathematical expressions, defining, optimizing and evaluating them, but it also allows to model several machine learning projects. This framework is capable of taking the desired structure and transform it into code, which can be integrated with other Python libraries. It is optimized for GPUs.

- **Keras**[10]**:** Released in 2015, constitutes a high level neural networks API developed in Python to simplify the interaction with different frameworks such as TensorFlow or Theano, making the creation of deep learning neural networks models easier. Supports both types of networks, convolutional and recurrent, and runs optimally on both CPUs and GPUs. Also, it contains implementations of other neural networks characteristics such as activation functions, optimizers, and several tools to make working with image and text data easier.

- **Caffe**[11]**:** Stands for Convolutional Architecture for Fast Feature Embedding, it was writen in C++ and released in 2017. This machine learning framework focuses on speed and modularity, and it is highlighted its fast performance, becoming an usual option for industrial development.

- **Torch**[12]**:** Written in the scripting language Lua and released in 2002, this machine learning library allows the project to use a wide range of deep learning algorithms. Providing a machine learning project with features such as linear algebra routines and GPU, iOS or Android support.

[8] `https://www.tensorflow.org/`
[9] `http://deeplearning.net/software/theano/`
[10] `https://keras.io/`
[11] `http://caffe.berkeleyvision.org/`
[12] `http://torch.ch/`

3.4.5.2. Musical toolkits

We can work with musical information thanks to the following toolkits, which makes possible the cooperation between technology and music.

- **Music21** [13]**:** Python toolkit used to study music, it provides an interface to acquire the scores and creating musical objects, so MIDI or XML files can be generated. Also, it can connect applications such as Musescore.
- **Magenta**[14]**:** Open source research project developed by Google exploring the role of machine learning as a tool in the creative process, distributed as a Python library and powered by TensorFlow.
- **Musica**[15]**:** Open source computational music toolkit written in Python covering several topics from music theory, audio analysis to high quality figure generation.
- **Musescore** [16]**:** Open source program to read scores, it has the possibility to import and export scores from MIDI format. It already has many options to handle music such as divide the input by instruments and process each of them separately.
- **Rosegarden** [17]**:** Music composition and editing environment based around a MIDI sequencer that features a rich understanding of music notation and includes basic support for digital audio.

3.5. Conclusions

Taking into account the project aim presented in Section 1.2, we have considered the following. We need to learn from Beethoven's music, being the most important factors the note information such as the name, duration, and where it is placed, but also the context of the note, as if it were a word in a novel. This last project's characteristic makes the RNN the wiser networks to develop this computational creativity project. Specifically, the LSTM networks constitutes a subtype of the previously mentioned, but they are provided with memory, allowing the system to understand the note in the context of the musical sentence or a musical motive, highlighted in Beethoven's work.

To work with LSTM Networks, we needed to establish the language and the libraries that we would use to develop the project. Python is the language

[13] `https://web.mit.edu/music21/`
[14] `https://magenta.tensorflow.org/`
[15] `https://pypi.org/project/musica-toolkit/`
[16] `https://musescore.org/es`
[17] `https://www.rosegardenmusic.com/`

that fits the most, due to its efficiency and the wide range of frameworks and libraries that we can include and work with to ease the development. The most suitable framework that fits with this project's aim is Keras, as it simplifies the TensorFlow usage and the creation of the deep learning model that we need to use. Also, in order to process the music information from the scores, we will use the python library Music21, as it provides a flexible interface to process several music formats.

Chapter 4

Deep Learning approach for music generation

In this chapter the development of the project will be explained, starting from an introduction to some musical definitions needed to fully understand the project, the different types of musical representation that we have used and the one that works better for this project's goal. After that, the music generation process will be followed from the neural network models used and the training and prediction processes to a deep explanation on the different approaches established to achieve the goal of this project, elaborating on the differences in the data representation, results obtained and limitations present on each of them, which generates the need of finding a new approach.

4.1. Musical definitions

In this section we introduce some musical concepts used throughout this chapter to fully understand the project development.

- Note: Musical event that describes a sound. It contains more information apart from the note name, but also the duration, or the pitch class.

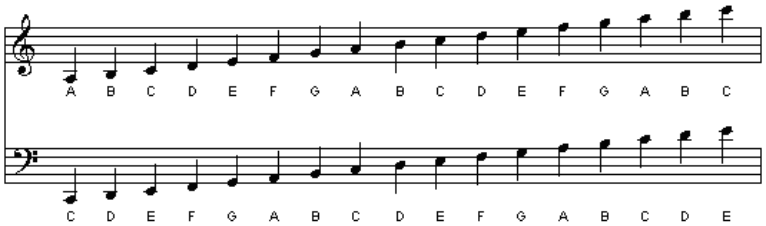

Figure 4.1: Note names of the chromatic scale

- Pitch: Property of sounds that allows a frequency scale ordering, distinguishing between "higher" or "lower" sounds.
- Clef: Musical symbol used to determine the name and pitch of the written notes, it is the first symbol that appears in the score. The tree types are: F (second stave from Figure 4.1), C and G (first stave from Figure 4.1).
- Key signature: Set of sharp or flat symbols placed after the cleff, it determines the notes that will be altered from their natural pitch during the score or till a new Key is written. A sharp raises one semitone the natural note, while the flat lowers it.
- Time signature: It determines how many beats are contained in each bar. It appears next to the key signature. As we can see in the example of Figure 4.6, the 2/4 time signature means that there are two crotchets or quarter notes per compass.

4.2. Input data

In order to use a neural network, it is needed to convert the music scores to readable data, so we need to choose the appropriate data representation. To get started, it is important to clarify the basic musical notation, shown in Figure 4.2.

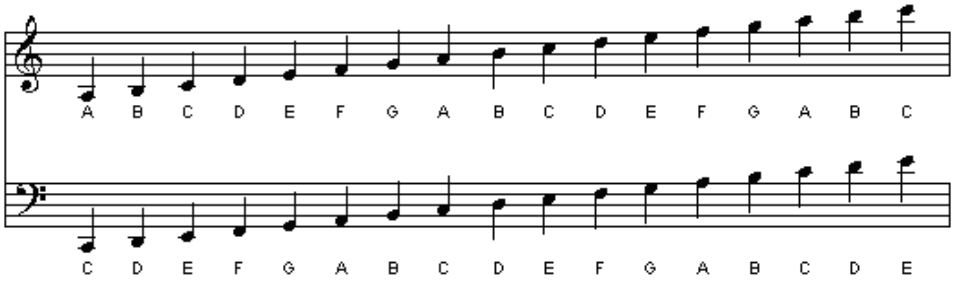

Figure 4.2: Notes with different pitches

The available data is represented in several formats, such as pfd, mp3, mp4, wav or midi. Obtaining the musical data from a pdf complicates the project, as we would need an image processing program to read every page, so it was discarded. Mp3, mp4 or wav formats were studied, although as they only store sounds, extracting more concrete information from them is more difficult than obtaining it from a midi file. This last format was also chosen due to the simplicity that music21 (section 3.4.5.2) gives us to process them.

4.2.1. MIDI

Musical Instrument Digital Interface (MIDI), is a format made to connect computers and musical instruments. A note is understood as an event with a *note_on* message, and each event carries information such as if the note is playing or not, volume, pitch, velocity and the note number, explained in Table 4.1.

Octave	**Note Numbers**											
	C	**C#**	**D**	**D#**	**E**	**F**	**F#**	**G**	**G#**	**A**	**A#**	**B**
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

Table 4.1: Midi notes table

Let's compare the most popular Beethoven motive from the Fifth Symphony, shown in Figure 4.3, with its correspondent MIDI file, whose first lines are shown in Table 4.2.

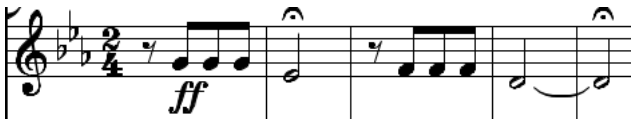

Figure 4.3: Beethoven's Fifth symphony score snippet

The first number specifies the time position of the event, the "note_on_c" label means that the note has to be played with the following characteristics, the 11 is the channel number, always an integer between 0 and 15, the next number is the note number [0 - 127] (it can be checked in Table 4.1), and finally, the last number means the velocity in which the note has to be played [0 - 127]. If the velocity is 0, it means that the note stopped playing, which could be also represented with the "note_off_c" label.

Firstly, as MIDI (.mid) files were popular in this research field, we used them as an input for our system, as it is a data file which contains information about the sounds: which note is played, when and how long or loud. As MIDI files store sound information, it does not differentiate between all the string

480	Note_on_c	11	67	108
512	Note_on_c	11	67	0
576	Note_on_c	11	67	106
608	Note_on_c	11	67	0
672	Note_on_c	11	67	106
704	Note_on_c	11	67	0
768	Note_on_c	11	63	105
1104	Note_on_c	11	63	0
1248	Note_on_c	11	65	103
1280	Note_on_c	11	65	0
1344	Note_on_c	11	65	113
1376	Note_on_c	11	65	0
1440	Note_on_c	11	65	106
1472	Note_on_c	11	65	0
1536	Note_on_c	11	62	105
2256	Note_on_c	11	62	0

Table 4.2: MIDI information

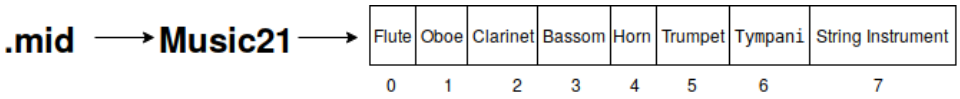

Figure 4.4: Music21 .mid files parsing

instruments in an orchestra, since their pitches are really similar, as it can be seen in Figure 4.4. That supposed a problem in the development of this project, since it prevented us for reaching our goal. Without being able to distinguish between string instruments, we would be loosing a big quantity of data. At this point, we converted all the input data to a MXL format, format that can be also processed by the previously chosen library music21 using musescore.

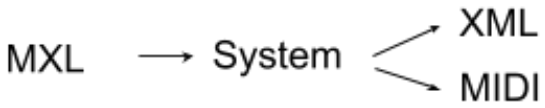

Figure 4.5: Final system's input and outputs formats

4.2.2. MusicXML

As explained in Section 1.2, this project's aim is to obtain all the different scores for every orchestra instrument, the input files format were changed to MXL. This extension refers to a compressed music score, which Music21 easily reads and processes. MXL files are the compressed format of the so called MusicXML, which is the standard XML format, although both extensions represent the same information. XML's file extension is used to store data in a legible way, by establishing several labels. For instance, in contrast with the MIDI format, the following information for Beethoven's Fifth symphony snippet represented in Figure 4.3 is stored. The key (Listing 4.1), the time signature (Listing 4.2) or the clef (Listing 4.3) is saved for each instrument in the score.

Listing 4.1: Key represented in XML

```
<key>
    <fifths> -3 </fifths>
</key>
```

Listing 4.2: Time signature represented in XML

```
<time>
    <beats> 2 </beats>
    <beat-type> 4 </beat-type>
</time>
```

Listing 4.3: Clef type represented in XML

```
<clef>
    <sign> G </sign>
    <line> 2 </line>
</clef>
```

Music21 allows us to generate the final output in any desired format, so we can obtain it in MIDI and XML, as it can be seen represented in Figure 4.5. After getting those files, Musescore can open both formats so the score can be visualized and played.

4.2.3. HDF5

The training process explained in section 4.3.2 generates a file with hdf5 extension, which stands for *Hierarchical Data Format* version 5, commonly used to store big quantities of data. It represents the weights of the different data possibilities. After training the system and generating this file, the prediction can easily load the hdf5 data into the model, so it has all the needed information to assign weights representing the possibility to appear to each data.

4.3. Music generation

Following the project's goal, explained in section 1.2, we intend to obtain a music piece based on Beethoven's symphonies, so we can imagine how the 10^{th} Symphony would have sounded like. The output is intended to be fully dependent of the system input, although the only characteristic that we have forced is the symphony's tempo and the key, as the sheets found in Beethoven's house after his death had 3 flats, and the tempo was a 6/8. He chose to write a large portion of his compositions in this key, as it is said that it represents a "stormy and heroic tonality", and it is used in works of unusual intensity, such as the Fifth Symphony (Figure 4.6).

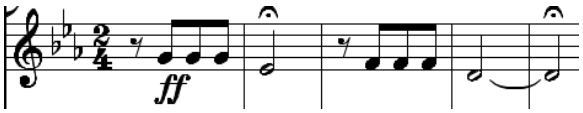

Figure 4.6: Snippet of Beethoven's Fifth Symphony in C minor

This project is divided in several phases. Firstly we need to extract a specific data from the scores. All the Beethoven symphonies have been converted to an mxl file using *musescore*, by loading the MIDI file of the representation of the symphonies obtained from the public repository from the Beethoven's museum in Bonn [1] and then exporting it to the desired format, in this case mxl, which constitutes the dataset that we have used to obtain the desired results, as explained in section 4.2.2.

After obtaining all the information and store it in a structured way, the second step is the Neural Network training. This way the system can learn from the given information, detailed in section 4.3.2. Finally the music prediction is explained in section 4.3.3, which generates the corresponding MusicXML and MIDI file as final output, both formats can be opened by *musescore*, as represented in Figure 4.5.

Three different approaches have been established in order to obtain the expected result, which is the new Beethoven's 10^{th} Symphony. We have also explored the data mining from the scores, as each approach uses slightly different information as input, since during the development of the project we have discovered new needs or lacks of information for some tasks.

In the following sections we will explain the LSTM Network design, as we have used several layer combinations in order to reach the best solution. Then we will elaborate on the training and prediction process. Also, we will deeply explain each approach with the purpose of comparing them and study the evolution of their input datasets and the results obtained with each of them and the reason why we implement each approach's changes. All the

[1] `https://da.beethoven.de/sixcms/detail.php?id=15241&template=untergruppe_digitales_archiv_en&_eid=1510&_ug=Symphonies&_mid=Works`

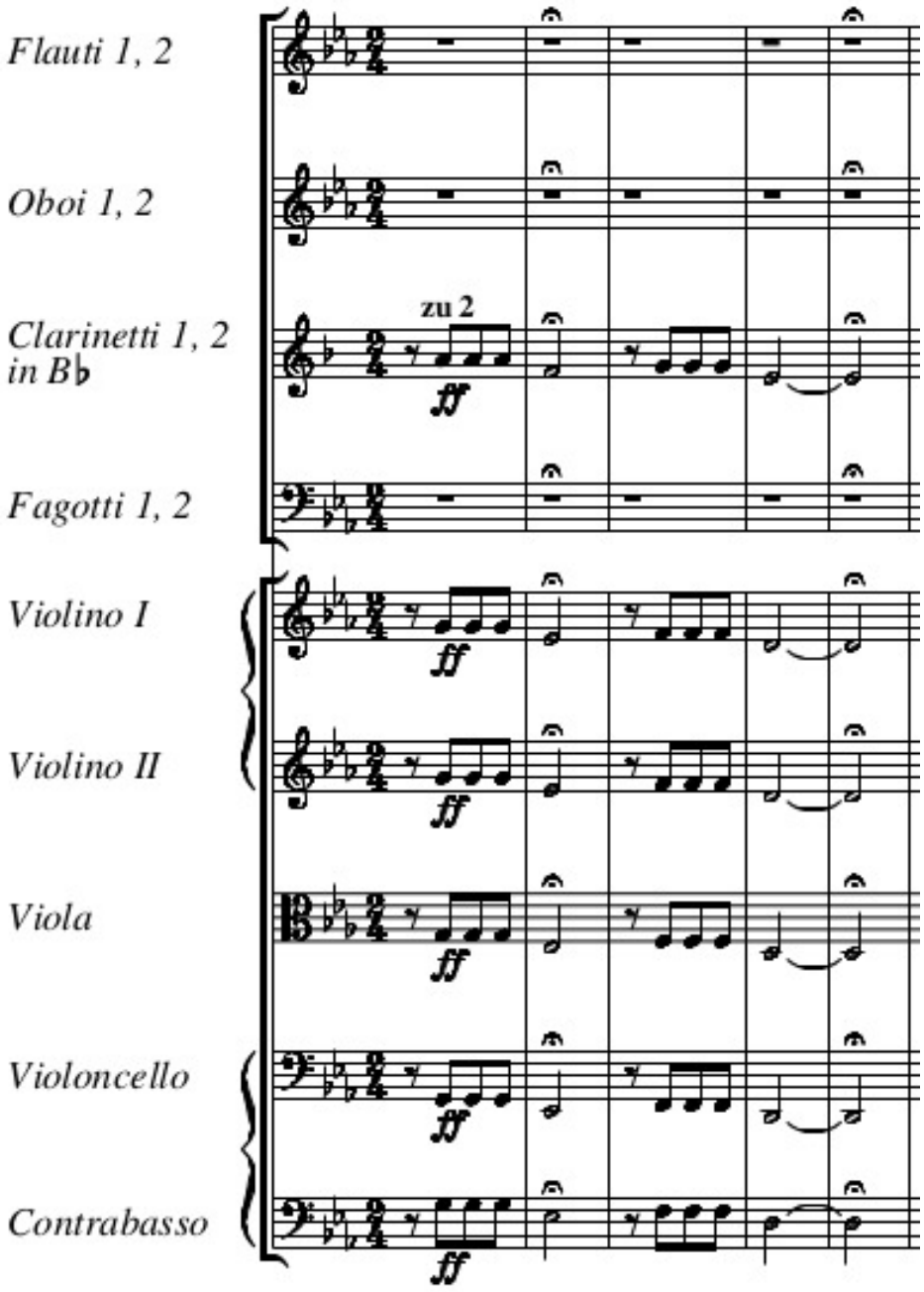

Figure 4.7: Snippet of Beethoven's Fifth Symphony in C minor

mp3 results are available in a Github repository [2]. We will explain the datasets obtained in each approach with the example shown in Figure 4.7, which represents the fifth first compasses of Beethoven's Fifth symphony.

[2] `https://github.com/paulamlago/Generated_Music`

4.3.1. LSTM Network design

The model that composes the neutral network is made-up by several layers, which constitutes an independent network whose basic units are the neurons, although each layer communicates with each others. This project's model follows a stacked LSTM architecture (see section 3.4.4) as the one represented in Figure 4.8, since the larger the depth, the less neurons per layer the network needs, and it is faster (Graves et al., 2013a). There is no formula established to determine how many layers the network should have, and how many neurons would work better for each layer, so one of the tasks during the development of this project has been to obtain that information empirically.

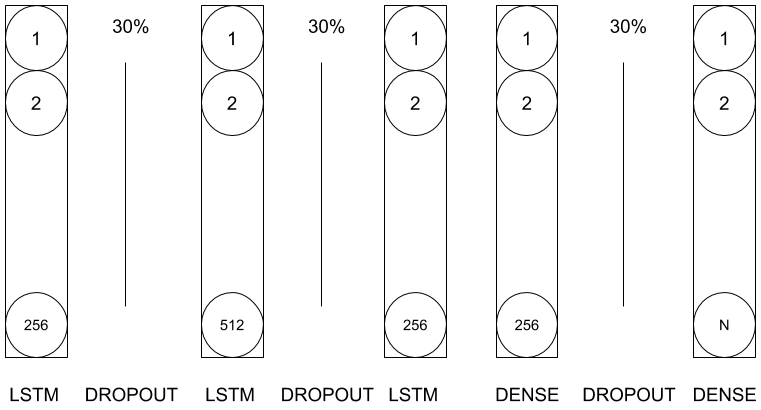

Figure 4.8: Model, being N the number of different tuples of information

The first model, represented in Figure 4.8, is composed by 3 different types of layers: LSTM, Dropout and Dense. The most relevant ones are the LSTM layers, which take the input sequences and return new ones. Then, the Dropout layers prevent overfitting, ignoring randomly selected neurons during the training, setting those inputs to 0. This layer has been set to 30% as several public projects on this field uses that value (Skúli, 2017). The Dense (Density) layer serves as a full connection mechanism. This layer is the last one, so the system returns the same number of outputs as the different numbers of tuples (note name, note duration) the input data had. Finally, the activation function used for every layer is set, determining how each node's output is represented. In this case, a linear activation is used, allowing the output to be interpreted as a probability between 0 and 1.

In order to compare results during the development of the project, several models have been tried from the first model, represented in Figure 4.8. The second one (Figure 4.9) constitutes a small modification by erasing the 256 nodes dense layer and the drop out layer that comes after it.

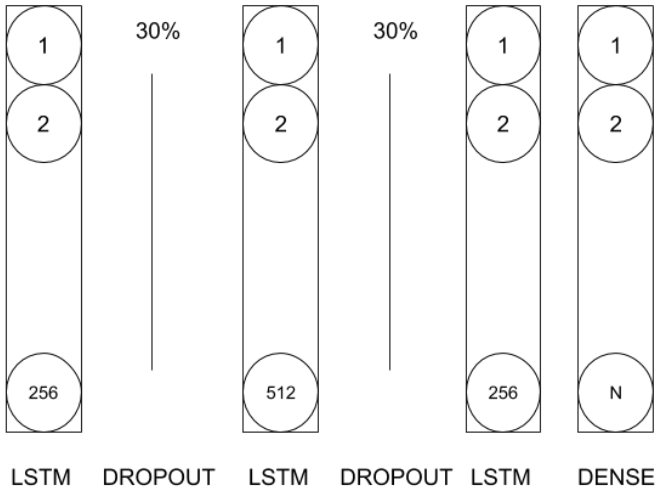

Figure 4.9: Second model, being N the number of different tuples of information

In this project, since it is all about creativity, we do not have the final validation step on a not trained group of data, present in the majority of machine learning problems, due to the nonexistence of a correct solution. Instead, a validation by an expert has been carried out.

4.3.2. Training the Neural Network

The first step when starting the training process is to read the input data and convert it into a understandable format for the neural network. As it is represented with tuples or lists, we need to create a dictionary to convert from each data to a number, so the neural network can work with it. This dictionary has the data as key and a unique number as value, so its length is the number of different data in the input. The conversion dictionary obtained from working whit the score present in Figure 4.10 will be the represented in Table 4.3.

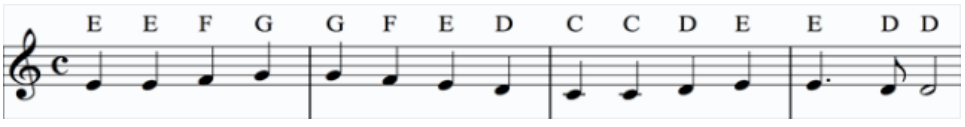

Figure 4.10: Violin's *Ode To Joy* snippet

As a result of reading the data and convert it with the previous dictionary in Table 4.3, the input data in a number representation would have the form showed in Table 4.4.

Tuple	Conversion
(E, quarter)	0
(F, quarter)	1
(G, quarter)	2
(D, quarter)	3
(C, quarter)	4
(E, dotted quarter)	5
(D, quaver)	6
(D, half)	7

Table 4.3: Conversion matrix from data to number

0	0	1	2	2	1	0	3	4	4	3	0	5	6	7

Table 4.4: Final violin's Ode to Joy number representation

Finally, we can generate the network input and output data. By establishing a certain sequence length, the output for each input sequence will be the first note that comes after the notes sequence in the input. This sequence length will be remembered by the Neural Network, as a pattern. It is important to take into account that in case of establishing a big sequence length, the network may generalize, while setting a small sequence length, the system may over learn.

For example, setting a sequence length equal to two, the first stages of the system's work flow for the Figure 4.10 input would be the shown in Table 4.5, which may lead to an overlearn problem, as the length is really small and there are many different and concrete cases.

sequence_in	sequence_out
[0, 0]	[1]
[0, 1]	[2]
[1, 2]	[2]
[2, 2]	[1]
[2, 1]	[0]
[1, 0]	[3]
[0, 3]	[4]
[3, 4]	[4]

Table 4.5: First compasses extraction of input and output sequences with sequence length = 2

To solve that problem, we set the length to a greater number, such as four, as the compass in this case is a 4/4. The first compasses input and output sequences for that sequence length can be studied in Table 4.6.

sequence_in	sequence_out
[0, 0, 1, 2]	[2]
[0, 1, 2, 2]	[1]
[1, 2, 2, 1]	[0]
[2, 2, 1, 0]	[3]
[2, 1, 0, 3]	[4]
[1, 0, 3, 4]	[4]
[0, 3, 4, 4]	[3]
[3, 4, 4, 3]	[0]

Table 4.6: First compasses extraction of input and output sequences with sequence length = 4

In case of the input, reshaping into a 3 dimension matrix is needed so it is compatible with the LSTM layers, using Python's numpy module. The first dimension or shape of the network is the number of different tuples or lists. In case of the last approach, in the dataset, the second one is the previously established sequence length and finally the last dimension is forced to be 1, so it has just one input information per sequence length. After that, the software normalizes the input into sequential values, from 0 to 1, to work with a regression model. In case of the output, it is converted into a categorical model.

Once the model is built and the input and output data are ready, it gets trained, generating a hdf5 file containing the weights, or priorities, for the input notes. A graph summing up the steps followed in the training process can be seen in Figure 4.11.

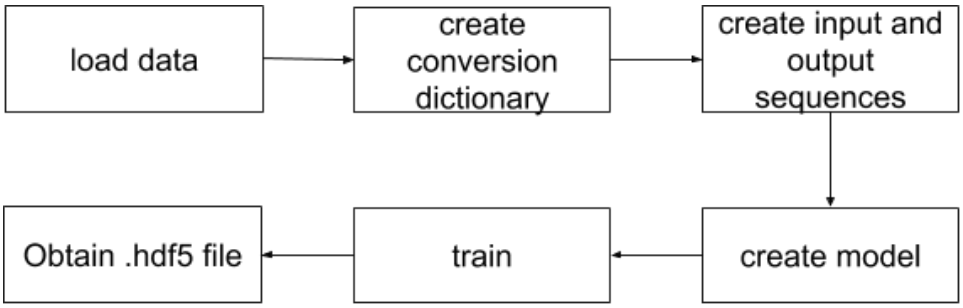

Figure 4.11: Training schema

4.3.3. Predicting new music

For this task, the network input is generated again, as in the previous process (see Table 4.5 or 4.6). Since it needs to work over the same model, it is created again, with the same parameters, but now, instead of training the

model, it loads the generated weights (hdf5 file) from the previous process. It is important at this point that the network input shapes and the loaded weights have the same dimensions.

Once the model is ready, the opposite dictionary to which is created in the training process, seen in Table 4.3 is created to convert from the network output's number to an observation from the original dataset. If we are trying to generate a single instrument score, the tuple is composed of (note name, note duration), while if we want to generate a conductor's score composed of several instruments, the instrument identification will have to be included in the tuple.

The prediction starts working after getting the information of the desired music duration, information that we have to manually give to the system. Then, a random sequence from the input is used as the starting point of the new score. As in the training, this sequence has to be reshaped into a 3 dimension matrix. The first dimension corresponds to the number of sequences, which is always 1, the second to the length of the sequence and the third, as in the training, is forced to be 1. After that, all the sequence values are converted into sequential ones (between 0 and 1), so the model can return a prediction given those input values.

The output of the prediction is an array with a probability for each tuple. Then, the system sorts the values from the greatest probability to the lowest. As the system is not working with tonal music, it gives priority to the notes belonging to the key scale used in the new score, present in Figure 4.12, but it allows non tonal music to appear in the new score, as we think that manually erasing sharps and flats would reduce the system's creativity. In case of a chord, it checks only the root note, so chords and single notes has the same probability of appearance.

Once it has the indexes of the most interesting note, the system can work on the predicted information accessing to the conversion dictionary.

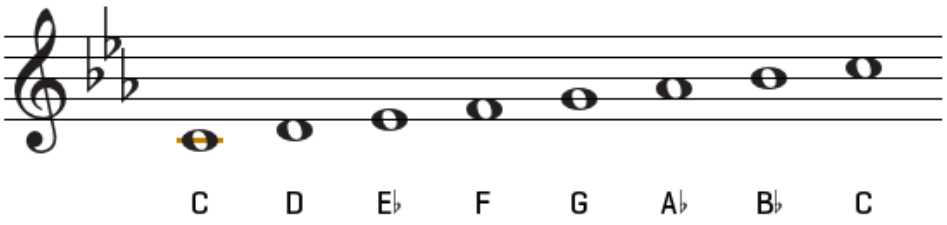

Figure 4.12: Key scale

Some other manually made restrictions is that if the predicted note differs more than one octave from the last one, it is transposed in order to get closer to the previous one, since it will not be easy to play for a musician. The transposition process consists in getting the octave in which both notes are

located and if the difference is more than one, we approximate the new note to the previous one by reducing or extending its octave. Another change made at this point is that if the predicted note and the previous one are rests, the lengths are added. This can only be applied to rests since we need to have several identical notes following (see Figure 4.7). Finally, it is checked that chords do not have repeated notes. In that case, only the unique notes are kept.

After finishing this manual improvements function, the predicted information is added to the pattern, which serves as an input for the next prediction, and it will have to be reshaped again to continue predicting notes with the new incorporation to the score. The system predicts notes till the addition of the duration of the notes predicted gets to the desired duration manually established when starting the prediction process.

Once the system has all the required predicted information (notes, chords, rests, and all the needed information such as their durations or the instrument that plays them) it is processed and .xml and .mid files are created using music21.

A graph summing up the prediction process is shown in Figure 4.14.

4.3.4. First approach: Music generation for individual instruments

The first approach is based on generating all the different orchestra instruments scores individually, as seen in Figure 4.13, by training each instrument with a concrete existing set of symphonies. After that, we have manually joined all the different obtained instrument's scores to study if the overall symphony was musically valid. Since each instrument was trained without information of the other instruments, the obtained conductor score had a lack of coordination between them.

4.3.4.1. Dataset description

This approach's goal is to obtain each instrument's score individually, the note names and durations are stored in an independent file, being the different tuples of note names and durations which constitutes the final dataset, and will be the input data to the training. This way after repeating the process for all the desired instruments and obtaining all the individual scores, we can put them together to obtain the orchestra's final score.

If we were working with this approach with the aim of generating new music for Clarinet from the example in Figure 4.7, the data that we would use is shown in Table 4.7. The final dataset is composed of multiple tuples containing note name and note duration, this tuples are sorted by the appearance order in the score.

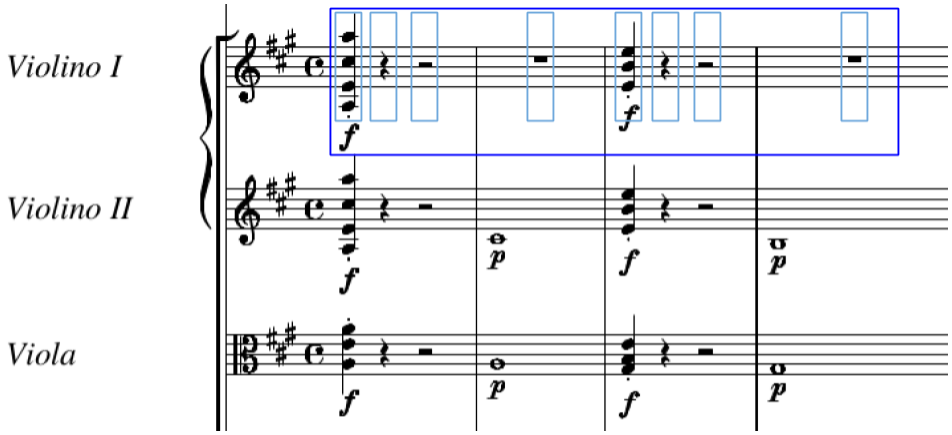

Figure 4.13: Snippet of Beethoven's Seventh Symphony in C minor representing the first approach's way of storing the musical information

{(Rest, 1/8), (A, 1/8), (A, 1/8), (A, 1/8), (F, 2/4), (Rest, 1/8), (G, 1/8), (G, 1/8), (G, 1/8), (E, 2/4)}

Table 4.7: First approach for clarinet

4.3.4.2. Results

The first experiment was to train the system with all the Beethoven's Fifth Symphony's movements. The output obtained is shown in Figure 4.15. We established the hypothesis that the output would have some similarities with the input, and being the most famous symphony, we could distinguish it easier rather than using any other as input.

It can be seen that different measures showed up, such as quarter, eight, sixteenth or half notes but also thirty-second notes, and some patterns show up. For instance, in the first two staves a half note appears tied to an eight and a sixteenth note. However, as we only considered notes, we introduced the possibility to use rests in the melody, so the next step at this point was to retrain the system, again with the most famous symphony, but allowing rests to appear. The results can be seen in Figure 4.16. Again, although a different score is generated, we can distinguish some patterns in the composition, marked in green in Figure 4.17.

At this point, the time measure is 4/4 as a first approach, although after discovering that Beethoven's house sketches belonging to the upcoming symphony had measure 6/8, it was set to that one, as explained in Section 1.2. The last generated score shows consecutive notes being the first one really high and the second one low in comparison with the previous note, having a difference of more than one octave between them. The empirical restrictions applied during the prediction are implemented at this point after realizing

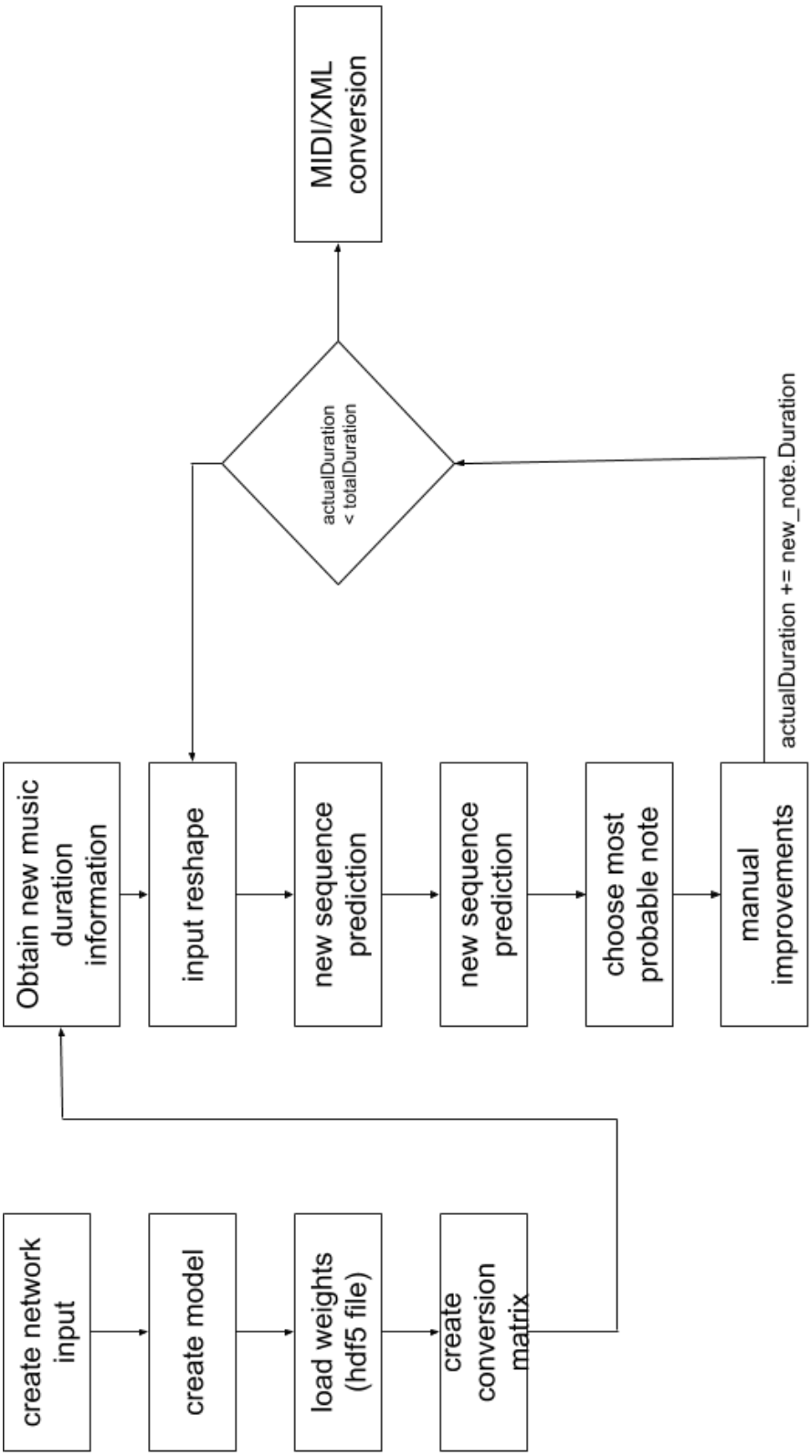

Figure 4.14: Prediction scheme

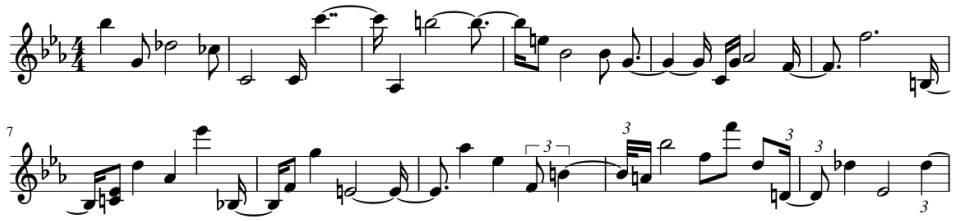

Figure 4.15: Results from training with the Fifth Symphony

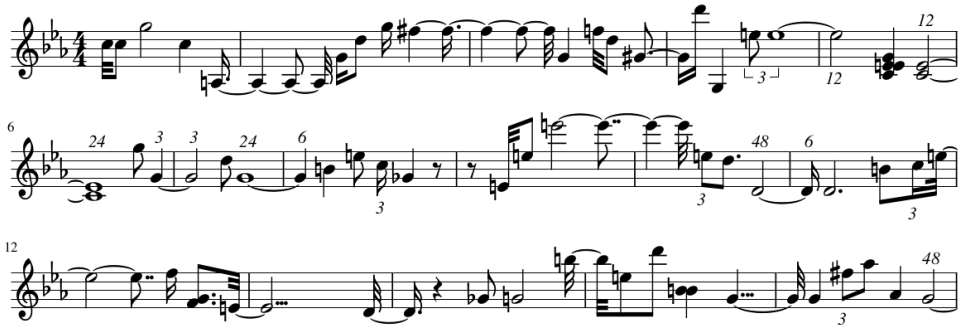

Figure 4.16: Results from training with the Fifth Symphony allowing rests

with the last score all the possible improvements marked with orange squares in Figure 4.17. The first orange square shows consecutive notes that have a difference of more than one octave between them, the second one in the next compass shows a chord with repeated notes, and finally the two last orange squares shows three dotted notes, which is not a common musical notation. All the experiments from this point include these manual improvements.

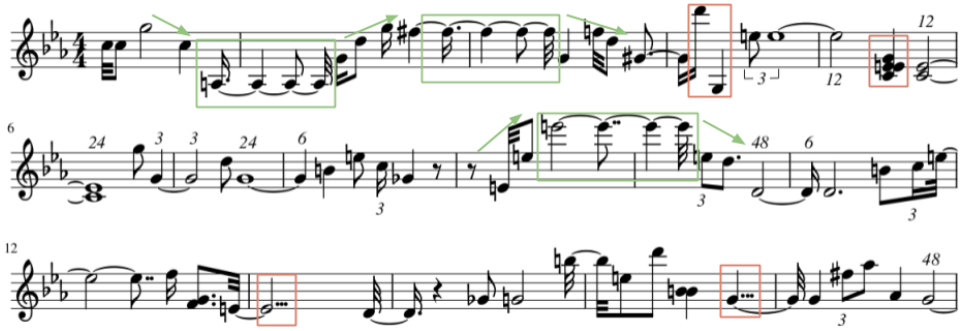

Figure 4.17: Patterns and keys to apply manual changes. Green squares denote the patterns and orange squares the items that highlights the need of manual improvements

Using the same weights as before, the first three staves of the outcoming score is shown in Figure 4.18. These improvements include reducing the

octaves in case two notes differ more than one, or give preference to show up to a note belonging to the key scale, as explained in section 4.3.3. The result differs from Figure 4.16, being the new one clearer but maintaining the motifs, such as the half note tied to two quarter notes, quality that characterizes Beethoven's Fifth Symphony.

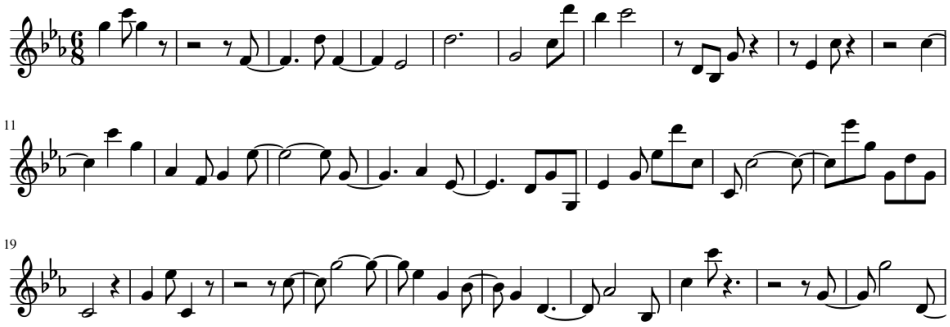

Figure 4.18: Results from training with the Fifth Symphony with manual improvements

Keeping the system state, we train it with the Seventh symphony, and generate the violins as before. The first three staves of this result can be seen in Figure 4.19, in which there is not an easy-to-recognize motive such as in the previous experiments. That may be because this symphony does not have a clear motive such as the Fifth's. The result looks similar, although it is remarkable the increment in the number of rests showing in the score. This may be due to the remarkable amount of silent compasses in the second movement of this symphony. Violins start playing in compass number 50, which is not a common characteristic of the violin scores in any symphony, being usually the instrument playing the main melody.

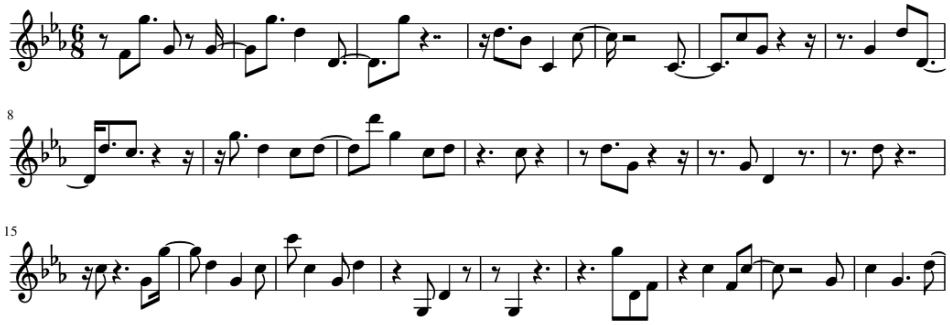

Figure 4.19: Results from training with the Seventh Symphony with manual improvements

Now that we have concluded the experiment with the Fifth and Seventh symphonies, the next step is to train the system with both of them. The

output is shown in Figure 4.20. It can be seen that the amount of rest notes is increased from other results that does not use the Seventh symphony violin's as input, but the motives present in the output obtained from training with the Fifth keeps showing. The same happens in Figure 4.21, obtained from training the Fifth, Seventh and Ninth Symphonies Violins.

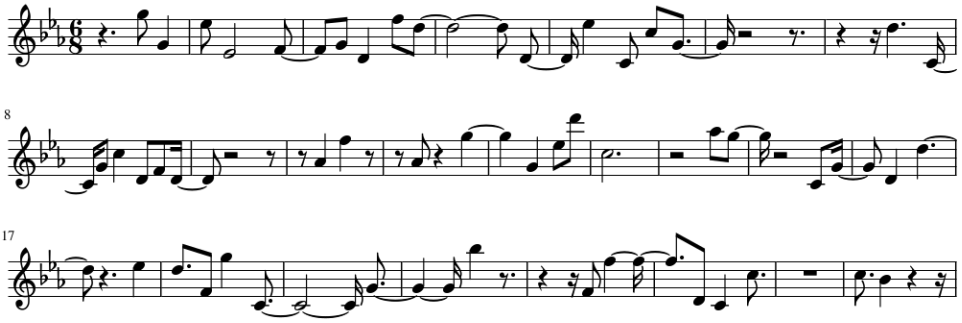

Figure 4.20: Results from training with the Fifth and Seventh Symphony

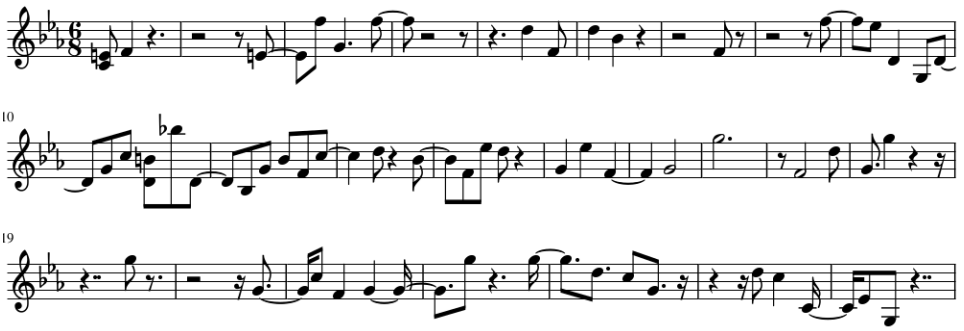

Figure 4.21: Results from training with the Fifth, Seventh and Ninth Symphony

After completing all the experiments previously described, the system was trained with some of the orchestra's instruments. Figure 4.22 shows the prediction result for Violin, Violas, Violoncellos, Contrabass, Flutes, Oboes and Clarinets, training with the Seventh symphony. This result has been obtained from training each instrument individually and putting them together manually by joining the XML generated scores for each instrument with musescore. A lack of coordination between each instrument is distinguishable, since each melody has been generated without having knowledge on any other instrument's voices. That has caused that each musical phrase from the different instruments does not coordinate with the others to generate a group sound.

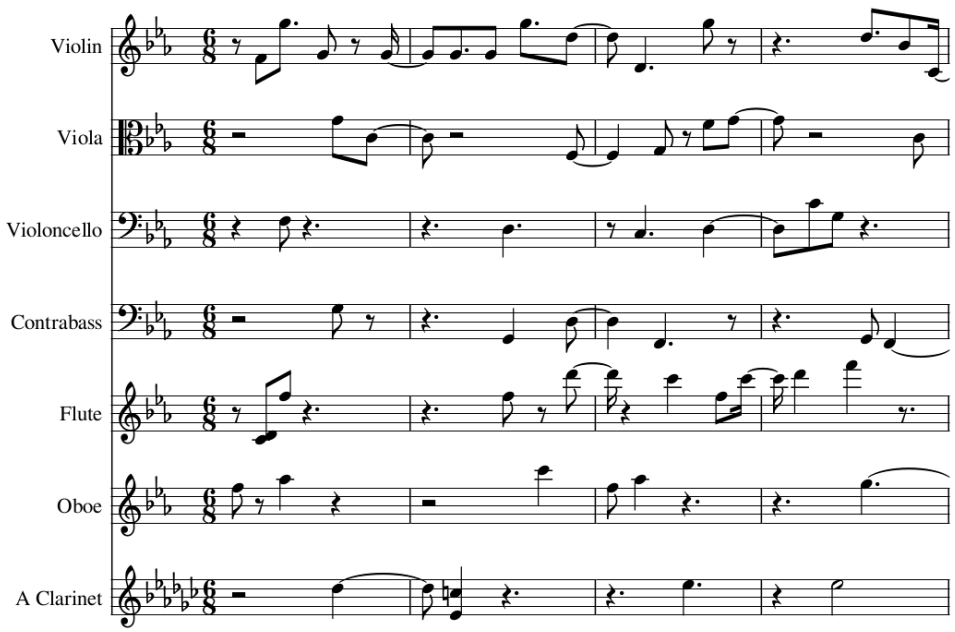

Figure 4.22: Results from training separately 7 different instruments with the Seventh symphony

4.3.4.3. Limitations

After this first approximation to the music generation, we conclude that each instrument should have information about other's instruments while training. Otherwise the melody would sound incoherent and although maybe each instrument's music would make sense individually, while putting all of them together it does not acquire a coherent sound, as the different instruments does not respect each others melodies.

As the goal of this project is to generate a group score, a second approximation is established, now adding more information into the input dataset.

4.3.5. Second approach: Music generation for coordinated instruments

The second approach was intended to increase the coordination between each instrument, so we have changed the dataset structure by adding more information to the tuples so the system is able to train a set of instruments at the same time from a concrete set of symphonies, as seen in Figure 4.23. This way, the generated scores present a considerable increment of coordination and it is easier to differentiate each musical phrase.

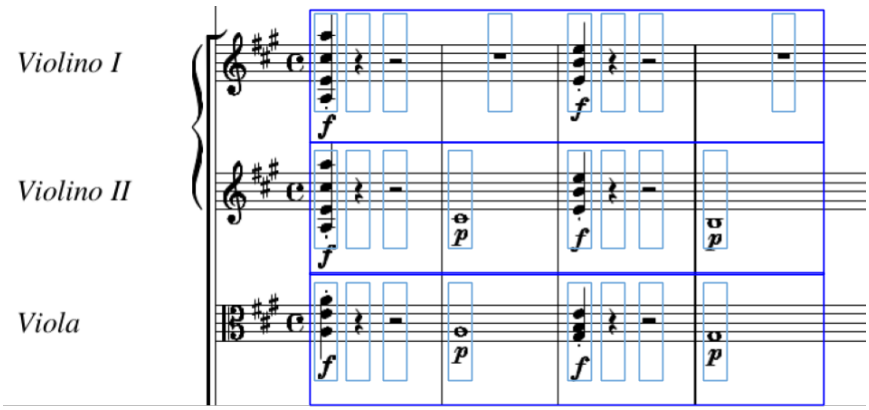

Figure 4.23: Snippet of Beethoven's Seventh Symphony in C minor representing the second approach's way of storing the musical information

4.3.5.1. Dataset description

The second approach trains with the chosen instruments at the same time, so we need to store, besides the note name and duration, the offset and instrument that plays it. The offset will be used to sort the data, but after making sure that the events are sorted as they are in the original score, it can be removed from the dataset. This way, the training data will be composed of the different tuples of note names, note durations and the instrument that plays it.

The input data for Clarinets would be the represented in Table 4.8. It can be appreciated all the different tuples containing the note name, duration, offset where it is located and the instrument name. As it can be seen, the data is sorted by offset, as the information has been obtained in that order.

{(Rest, 1/8, 0.125, Clarinet), (A, 1/8, 0.25, Clarinet), (A, 1/8, 0.375, Clarinet), (A, 1/8, 0.5, Clarinet), (F, 2/4, 1, Clarinet), (Rest, 1/8, 1.125, Clarinet), (G, 1/8, 1.25, Clarinet), (G, 1/8, 1.375, Clarinet), (G, 1/8, 1.5,Clarinet), (E, 2/4, 1.625, Clarinet)}

Table 4.8: Second approach for clarinet

Let's see an example where we train the system with the Clarinets and Violoncellos, using the same approach. Firstly, the system will obtain the tuples as it can be seen in Table 4.9.

After that, the system will sort them by offset as shown in Table 4.10.

Later, as the offset is not necessary any more, it was ment to be used to order the events, it can be removed from each tuple. The final dataset is shown in Table 4.11.

{(Rest, 1/8, 0.125, Clarinet), (A, 1/8, 0.25, Clarinet), (A, 1/8, 0.375, Clarinet), (A, 1/8, 0.5, Clarinet), (F, 2/4, 1, Clarinet), (Rest, 1/8, 1.125, Clarinet), (G, 1/8, 1.25, Clarinet), (G, 1/8, 1.375, Clarinet), (G, 1/8, 1.5,Clarinet), (E, 2/4, 1.625, Clarinet), (Rest, 1/8, 0.125, Violoncello), (G, 1/8, 0.25, Violoncello), (G, 1/8, 0.375, Violoncello), (G, 1/8, 0.5, Violoncello), (E, 2/4, 1, Violoncello), (Rest, 1/8, 1.125, Violoncello), (F, 1/8, 1.25, Violoncello), (F, 1/8, 1.375, Violoncello), (F, 1/8, 1.5„Violoncello), (D, 2/4, 1.625, Violoncello)}

Table 4.9: Second approach for clarinet and violoncello

{(Rest, 1/8, 0.125, Clarinet), (Rest, 1/8, 0.125, Violoncello), (A, 1/8, 0.25, Clarinet), (G, 1/8, 0.25, Violoncello), (A, 1/8, 0.375, Clarinet), (G, 1/8, 0.375, Violoncello), (A, 1/8, 0.5, Clarinet), (G, 1/8, 0.5, Violoncello), (F, 2/4, 1, Clarinet), (E, 2/4, 1, Violoncello), (Rest, 1/8, 1.125, Clarinet), (Rest, 1/8, 1.125, Violoncello), (G, 1/8, 1.25, Clarinet), (F, 1/8, 1.25, Violoncello), (G, 1/8, 1.375, Clarinet), (F, 1/8, 1.375, Violoncello), (G, 1/8, 1.5, Clarinet), (F, 1/8, 1.5, Violoncello), (E, 2/4, 1.625, Clarinet), (D, 2/4, 1.625, Violoncello)}

Table 4.10: Second approach for clarinet and violoncello sorted by offset

4.3.5.2. Results

To avoid the musical disorder obtained in the previous result, this second approach was used. As explained before, in this case the system is trained with a set of desired instruments, getting this way scores such as the one shown in Figure 4.24, a duo for Flute and Violin. This result shows how each instrument compliments the others, having the violin the main melody at the beginning, but respecting the Flute's main appearance in compasses seventh and eight.

In the second experiment we tried to increment the coordination complexity by training with one more instrument. The same behavior can be seen in the score in Figure 4.25, which shows how Violins, Violas and Violoncellos, while being trained only with the Seventh symphony, assumes a trio music by respecting the other instrument's melodies and complementing each other. Differences between this score and the one belonging to the first approach, shown in Figure 4.22, can be appreciated. In that case, in order to compare them, the corresponding Violin, Viola and Violoncello staves are the first, second and third. As it can be seen, the coherence of the differ-

{(Rest, 1/8, Clarinet), (Rest, 1/8, Violoncello), (A, 1/8, Clarinet), (G, 1/8, Violoncello), (A, 1/8, Clarinet), (G, 1/8, Violoncello), (A, 1/8, Clarinet), (G, 1/8, Violoncello), (F, 2/4, Clarinet), (E, 2/4, Violoncello), (Rest, 1/8, Clarinet), (Rest, 1/8, Violoncello), (G, 1/8, Clarinet), (F, 1/8, Violoncello), (G, 1/8, Clarinet), (F, 1/8, Violoncello), (G, 1/8, Clarinet), (F, 1/8, Violoncello), (E, 2/4, Clarinet), (D, 2/4, Violoncello)}

Table 4.11: Final dataset: Second approach for clarinet and violoncello

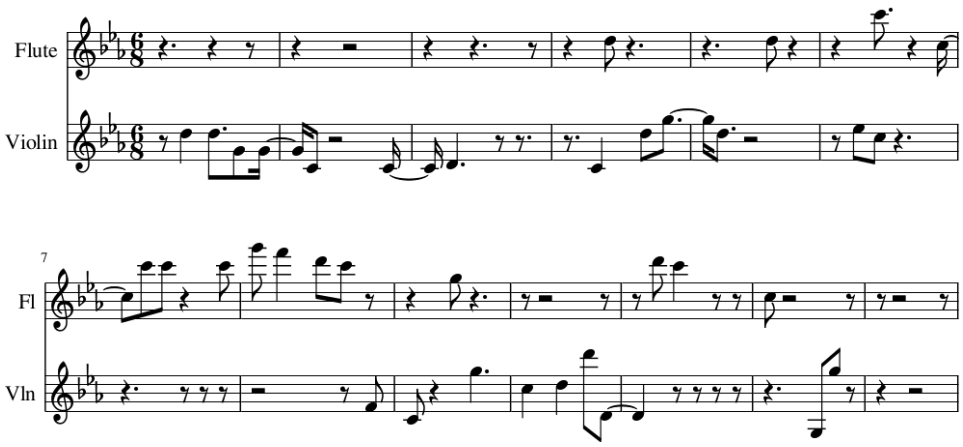

Figure 4.24: Second approach trained with the Seventh symphony for Flutes and Violins

ent instruments is enhanced in the second approach. Nevertheless, we can observe a lack of information in the experiment seen in Figure 4.25, as the Violas stave appears with F clef, while this instrument, as shows the result in the previous experiment (Figure 4.24), uses C clef.

After discovering that using the same approach in two cases we obtain different results in constant items such as the clef, we started to deeply explore into de data to find out where this disinformation came from, which finally lead into a homogenization plan and a new approach.

4.3.5.3. Limitations

In this approximation, the increment of the data and the lack of clearness in it lead to a decompensation in the results. Regardless the need of cleaning the data an homogenize it before training, the results showed more coordination between all the different parts of the orchestra.

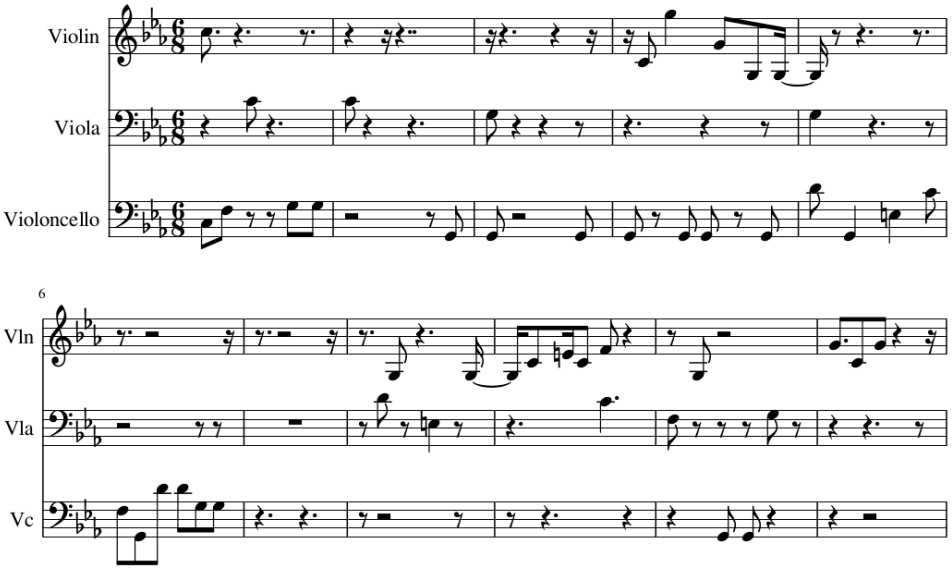

Figure 4.25: Score obtained from training Violins, Violas and Violoncellos with the Seventh symphony

4.3.6. Third approach: Music generation for coordinated instruments with data homogenization

After exploring in detail the input data that we were giving to the system, we found out that it needed some homogenization. The most relevant change is the storage of the Violin's information, as Beethoven's symphonies are composed for two Violin's groups, Violin I and Violin II, music21 read both as "Violin", without the possibility of distinguishing. As the aim of the project is to generate the orchestra score, and this musical group always have two violin voices as minimum, we forced the differentiation, which had to be made by exploring the data and manually change the first violin data by "Violin I", and the second by "Violin II". Previous aproaches did not distinguish between both voices, and the generated violins scores constituted a prediction on the Violins I and Violins II scores together.

In this approach a new way of extracting the data from the scores was proposed, so the data was obtained vertically, as it is shown in Figure 4.26 in which each offset is represented by a vertical rectangle, creating lists of tuples storing the information of each instrument for every offset.

The changes made after the homogenization and the new way of extracting the information from the scores constitutes the third approach.

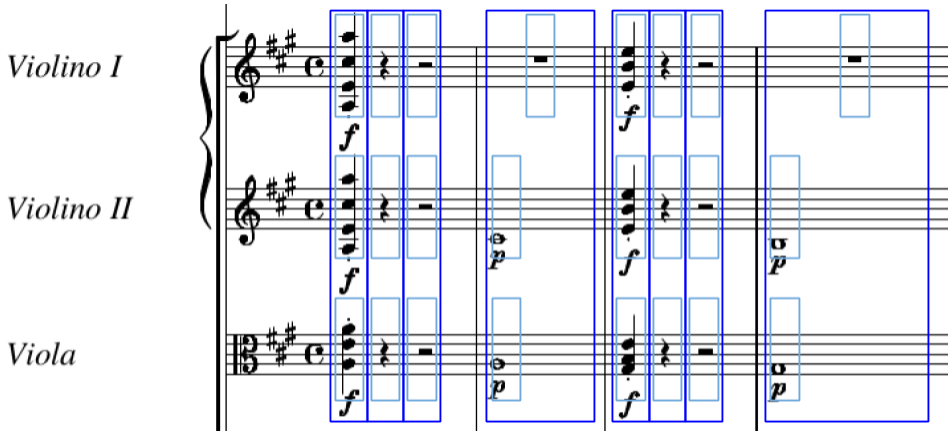

Figure 4.26: Snippet of Beethoven's Seventh Symphony in C minor representing the third approach's way of storing the musical information

4.3.6.1. Dataset description

The last proposed experiment consists on reading all the instruments by offset. That could be visually understood by reading the score vertically, storing all the tuples in a list which belongs to the value of a dictionary, whose key is the offset.

If we want to generate a score for Clarinet, Violins and Violoncello, again based on Figure 4.7, the input would be the shown in Table 4.12. It can be seen that lists of tuples belonging to each instrument are stored ordered by offset.

After some weeks of research another problem showed up while training with several symphonies, since the generated note durations were imprecise. We found out that, as in Figure 4.7 the half note played in offset 1 and 2, in this case has duration 2/4, since the compass in this Symphony is 2/4, and that note occupies the whole compass. Nevertheless not all the symphonies are written in a binary subdivision of the compass, or even each movement have a different compass defined. For example a half note in a 6/8 compass, which has a ternary subdivision, would be stored with a duration of 4/6. This leads to the conclusion that in case of binary or ternary subdivision, which are the most common, the notes will not be represented in the same way. A general representation has been found by storing the name of the note, rather than its numerical duration, as it can be seen in Table 4.13.

As we forced the new symphony key to have 3 flats, there are some music combinations used in other symphonies that generates a correct harmony for the key in which they are written, but maybe not the C minor key that we are setting. To solve that, we implement a new manual change in which every note is transposed from its key to the C minor key, to maintain the melody but generate a correct harmony for our tonality.

0.125 >>[(Rest, 1/8, Clarinet), (Rest, 1/8, Violin I), (Rest, 1/8, Violin II), (Rest, 1/8, Violoncello)]
0.25 >>[(A, 1/8, Clarinet), (G, 1/8, Violin I), (G, 1/8, Violin II), (G, 1/8, Violoncello)]
0.375 >>[(A, 1/8, Clarinet), (G, 1/8, Violin I), (G, 1/8, Violin II),(G, 1/8, Violoncello)]
0.5 >>[(A, 1/8, Clarinet), (G, 1/8, Violin I), (G, 1/8, Violin II),(G, 1/8, Violoncello)]
1 >>[(F, 2/4, Clarinet), (E, 2/4, Violin I), (E, 2/4, Violin II),(E, 2/4, Violoncello)]
1.125 >>[(Rest, 1/8, Clarinet), (Rest, 1/8, Violin I), (Rest, 1/8, Violin II), (Rest, 1/8, Violoncello)]
1.25 >>[(G, 1/8, Clarinet), (F, 1/8, Violin I), (F, 1/8, Violin II),(F, 1/8, Violoncello)]
1.375 >>[(G, 1/8, Clarinet), (F, 1/8, Violin I), (F, 1/8, Violin II),(F, 1/8, Violoncello)]
1.5 >>[(G, 1/8, Clarinet), (F, 1/8, Violin I), (F, 1/8, Violin II),(F, 1/8, Violoncello)]
2 >>[(E, 2/4, Clarinet), (D, 2/4, Violin I), (D, 2/4, Violin II),(D, 2/4, Violoncello)]

Table 4.12: Third approach for clarinet violin and violoncello sorted by offset

4.3.6.2. Results

This approach arises from the necessity of clarifying the input data, because studying it we found out that some elements were missing so the instrument grouping in the last approach was not successful in every case. The most significant change is the Violin I and Violin II naming. As while exploring the input data we discovered that music21 was labeling both voices as "Violin" and both were mixed in the same stave. The second and most relevant change was to read the information vertically, elaborating lists of tuples with each instrument's information for each offset. This change will be visually distinguishable in this approach's output.

The first experiments can be seen in Figures 4.27, trained with the seventh symphony and 4.28, trained with the fifth, seventh and ninth symphonies. After exploring the data and change the violin's names, two differentiated staves shows up, being the first one the corresponding to the first Violins and the second one to Violin II, as in a common conductor's score. Figure 4.28 highlights the vertical way of extracting the information, as it can be seen that every instrument is playing the same rhythms in this first two compasses, being the violins, flute and clarinets the voices playing quavers, while the rest of the group is playing mostly semiquavers, generating a time subdivision. Nevertheless, at the end of the last compass, it is observed

0.125 >>[(Rest, quarter, Clarinet), (Rest, quarter, Violin I), (Rest, quarter, Violin II), (Rest, quarter, Violoncello)]
0.25 >>[(A, quarter, Clarinet), (G, quarter, Violin I), (G, quarter, Violin II), (G, quarter, Violoncello)]
0.375 >>[(A, quarter, Clarinet), (G, quarter, Violin I), (G, quarter, Violin II),(G, quarter, Violoncello)]
0.5 >>[(A, quarter, Clarinet), (G,quarter, Violin I), (G, quarter, Violin II),(G, quarter, Violoncello)]
1 >>[(F, half, Clarinet), (E, half, Violin I), (E, half, Violin II),(E, half, Violoncello)]
1.125 >>[(Rest, quarter, Clarinet), (Rest, quarter, Violin I), (Rest, quarter, Violin II), (Rest, quarter, Violoncello)]
1.25 >>[(G, quarter, Clarinet), (F, quarter, Violin I), (F, quarter, Violin II),(F, quarter, Violoncello)]
1.375 >>[(G, quarter, Clarinet), (F, quarter, Violin I), (F, quarter, Violin II),(F, quarter, Violoncello)]
1.5 >>[(G, quarter, Clarinet), (F, quarter, Violin I), (F, quarter, Violin II),(F, quarter, Violoncello)]
2 >>[(E, half, Clarinet), (D, half, Violin I), (D, half, Violin II),(D, half, Violoncello)]

Table 4.13: Third approach for clarinet violin and violoncello sorted by offset with generalized note durations

how Violin I changes the rhythm and starts with the semiquaver subdivision. These firsts experiments highlights that this new way of extracting the information seems to generate more meaningful and coordinated scores, although the previously explained clef confusion keeps present. In this case, in the result represented in Figure 4.28, every instrument assumes a G clef, which in case of instruments such as Violas and Violoncellos is not correct, while the result in Figure 4.27 has a deviation in the Viola's clef, as it should be C instead of G. To fix that, we manually forced the keys for each instrument to be the original.

The example shown in Figure 4.29 is the one with more similarities to the input data, as it starts with a four notes motive as the Symphony that has been used for training (see Figure 4.7).

At this point we explored the possibility of modifying the neural network model previously used carrying out the fist experiment using the second model (see Figure 4.9). By using the same Symphony as before as input data and the same number of epochs and sequence length, the result for Violin I (first stave) and Violin II (second stave) can be seen in Figure 4.30. In this example, we can also see the small motive in the first two compass, in which the First violin's two notes are repeated.

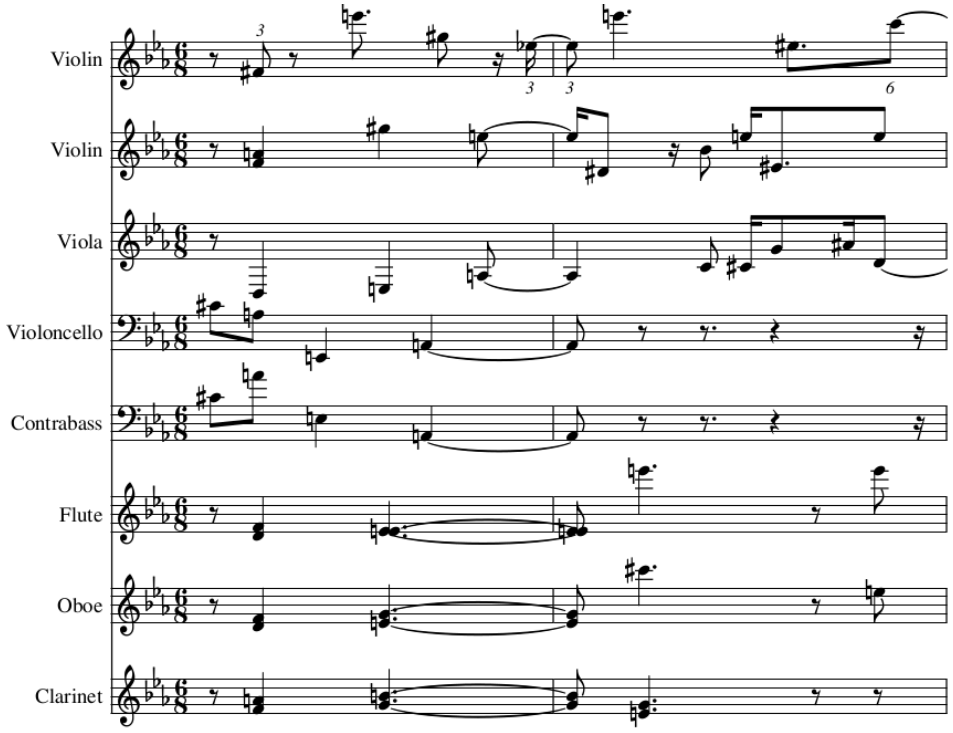

Figure 4.27: Score obtained from training with the Seventh symphony for Violins, Violas, Violoncellos, Contrabasses, Flute, Oboe and Clarinet

4.3.6.3. Limitations

This last approach results show an increment on the music coordination between instruments in comparison with last approaches. Also as the data predicted are "vertical" lists of tuples that has previously appeared on any symphony, the harmony created in every offset is correct as it has been previously used by the compositor. That eliminates the need of elaborating large musical rules to establish a correct harmony between instruments.

Nevertheless, as this approach generate coordinated notes and melodies, the biggest limitation is the music expressiveness, as we have only worked with the notes but not with the dynamics. In the music files predicted every instrument plays at the same volume all the time, without the expressiveness that a musician will contribute to the piece. This last problem is deeply explained in Chapter 5.

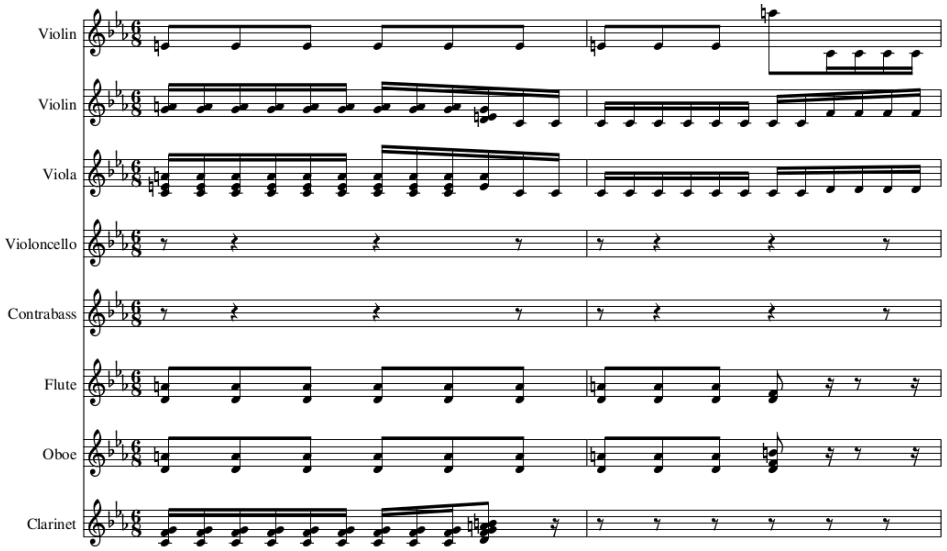

Figure 4.28: Score obtained from training with the Fifth, Seventh and Ninth symphony for Violins, Violas, Violoncellos, Contrabasses, Flute, Oboe and Clarinet

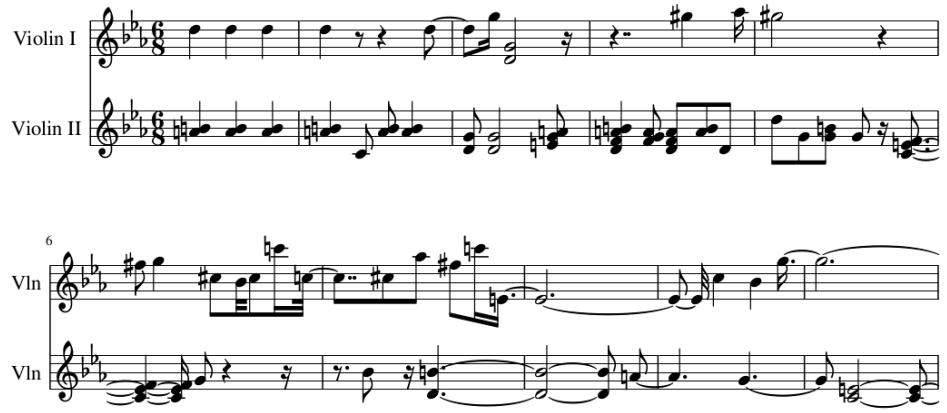

Figure 4.29: Score obtained from training Violins with the Fifth symphony

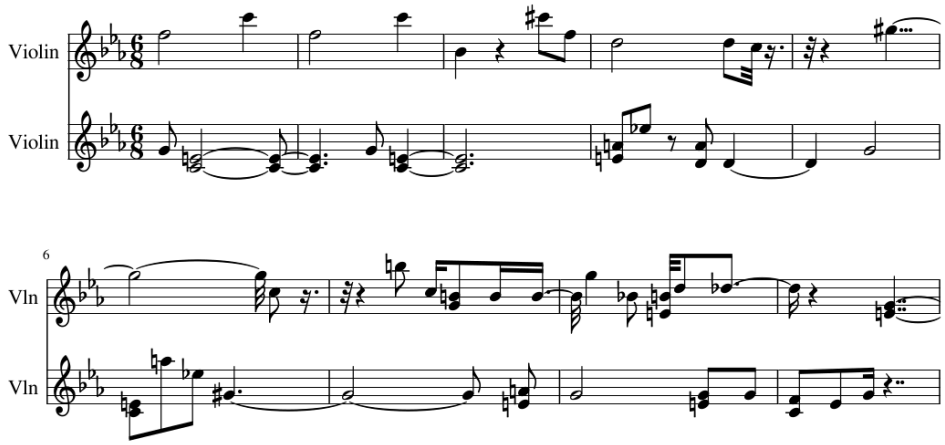

Figure 4.30: Score obtained from training Violins with the Fifth symphony using the second model

Chapter 5

Conclusions and Future Work

5.1. Conclusions

This project explores the possibility of generating new music based on Beethoven's style by a system doted with Artificial Intelligence, using LSTM neural networks, which learn and remember musical phrases of a concrete length, finally showing that it is possible to obtain music that imitates this composer's style for several instruments.

During the specification of the problem that we were facing, we established three ways of approximating to the new symphony. The first one was to train and generate separately each instrument scores, and manually creating the conductor's score. The results obtained were satisfactory for each single instrument separately, getting to generate music in a recognizable style. However, when joining all the different scores, the sound was not coordinated and the musical phrases belonging to the different instruments were not respected by the other voices. We concluded that with this first approach we could generate solo scores, but not group music. The second approach was intended to solve the main problem that the first one presented, that the instruments were not sufficiently coordinated since each instrument was trained separately, without any information on the music that the others were playing, which is crucial in an orchestra. The solution proposed was to train and generate music belonging to different instruments at the same time. This way the results obtained were more coordinated and we could see that each instrument respected each other, having rests or accompanying the main melody when they did not have the leading voice. Finally, the third approach intended to elaborate on the coordination an leave no data misunderstanding, by distinguishing between the two Violin's voices, for instance. It presents an improvement in the way of extracting information, generating lists for each offset containing information of every instrument.

The results obtained can be seen in Table 5.1, as we have progressively studied the output generated with the three approaches, by first working

on the generation of single instruments score, and checking that way if they were musically correct, to finally generate a conductor score. The system can return solo scores, but also duos, trios, quartets and also, an orchestra score, although we have not get to generate the score trained with all the existing symphonies, which should be the 10th, since it is needed a big computation capacity.

The human interpreter is always the source of emotions, so it is remarkable the lack of dynamics in the generated music, being played all the notes at the same volume during the whole piece. In this project we have focused on the notes production and instruments coordination, so the generated scores do not have notation of the dynamics.

5.2. Future work

Following the problem exposed in the conclusion, the next steps should be to research in music expressiveness, in order to transmit it to the system, to obtain music similar to what a human composer would create. An option to start in this task could be to obtain the score's dynamics, and train a Deep Learning model with the expressiveness of the work, in order to generate a *template*, which would be the equivalent to the composer's way to capturing his or her feelings. After generating the dynamics, the *"most human"* or sentimental part, a system like the one generated for this work would generate the notes and they would be fitted in the dynamic's template.

Another improvement that could be made to the developed system is to establish more elaborated musical rules to generate notes. For instance, taking First violin's melody as the main motive, while generating new instrument notes, it should be taken into account the harmony created between the notes, so a nice and clear sound is composed. For that purpose, some musical research about harmony effects and how it contributes to the perception of a musical phrase (Palmer and Krumhansl, 1987) should be considered.

Artificial Intelligence generates a big controversy nowadays. The social awareness and unconcern should be progressively made, by calming down the latent discussion around Artificial Intelligence and the possibility of *stealing human jobs*. In case of this project, the most affected community are the music composers, worried of being substituted by machines. This last fact should be contradicted by clarifying that Artificial Intelligence will work as a tool to enhance their production, contributing with new ideas when the composer needs it, but, at this point, it will not generate any score without a composer's help.

Approach	Instruments	Symphony trained	Details	Figure
First approach	Violins	5th	Without rests	4.15
			With rests	4.16
			With manual improvements	4.18
		7th		4.19
		5th + 7th		4.20
		5th + 7th + 9th		4.21
	Violins, Violas, Violoncellos, Contrabass, Flute, Oboe, A Clarinet	7th		4.22
Second approach	Violins, Flute	7th		4.24
	Violins, Violas, Violoncellos			4.25
Third approach	Violins, Violas, Violoncellos, Contrabass, Flute, Oboe, A Clarinet	7th		4.27
		5th + 7th + 9th		4.28
	Violins	5th		4.29
			Model 2	4.30

Table 5.1: Experiments summary

Chapter 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

Este proyecto explora la posibilidad de generar nueva música basada en el estilo de Beethoven con un sistema dotado con Inteligencia Artificial, utilizando redes neuronales LSTM, las cuales aprenden y recuerdan frases musicales de una longitud concreta, mostrando finalmente que es posible obtener música que imita el estilo del compositor para varios instrumentos.

Durante la especificación del problema al que nos enfrentamos, establecimos tres formas de aproximarnos a la nueva sinfonía. La primera fue entrenar y generar la partitura de cada instrumento por separado, para después unirlas manualmente creando la partitura del director. Los resultados obtenidos fueron satisfactorios para cada instrumento de forma individual, consiguiendo generar música en un estilo reconocible. Sin embargo, al unir cada partitura, el sonido no era coordinado y las frases musicales de los diferentes instrumentos no eran respetadas por las otras voces. Concluimos que con este primer acercamiento podemos generar partituras para solista, pero no para grupos musicales. El segundo acercamiento estaba orientado a resolver el problema principal que el primero presentó, que los instrumentos no estaban lo suficientemente coordinados puesto que cada uno había sido entrenado por separado, sin información de la música que las otras voces estaban produciendo, lo cual es crucial en una orquesta o en cualquier agrupación musical. La solución propuesta fue entrenar y generar música perteneciente a cada instrumento a la vez. De esta forma los resultados obtenidos fueron más coordinados y pudimos ver cómo cada instrumento respetaba a los otros, teniendo silencios o acompañando la melodía principal. Finalmente, el objetivo del tercer acercamiento era mejorar la coordinación y eliminar los malentendidos musicales, por ejemplo distinguiendo las dos voces de los violines. Este tercer acercamiento presenta una mejora en la forma de extraer la información, generando listas por cada evento temporal, conteniendo información de todos los instrumentos.

Los resultados obtenidos pueden verse en la Tabla 6.1, puesto que hemos estudiado los resultados progresivamente con los tres acercamientos, trabajando en la generación de instrumentos individuales, comprobando que son musicalmente correctos, hasta finalmente generar una partitura de orquesta.

El sistema puede devolver partituras para solista, pero también duos, trios, cuartetos y música para orquesta, aunque no hemos llegado a generar una partitura entrenada con todas las sinfonías existentes, la que sería la décima sinfonía, puesto que se necesita una gran capacidad de computación.

El interprete humano es siempre la fuente de emociones, por lo que es destacable la falta de expresividad en la música generada, siendo reproducidas todas las notas con el mismo volumen durante toda la pieza. En éste proyecto nos hemos centrado en la producción de notas y la coordinación entre instrumentos, por lo que las partituras generadas no tienen información de la expresividad.

6.2. Trabajo Futuro

Siguiendo con el problema expuesto en las conclusiones, los próximos pasos tendrían que ser en la exploración de la expresividad musical, para transmitirla al sistema y así obtener música similar a lo que un compositor humano crearía. Una opción para comenzar podría ser obtener la expresividad de la partitura y entrenar un modelo de aprendizaje profundo con dicha información, para generar una plantilla, que se correspondería con la forma del compositor de capturar sus sentimientos. Tras generar la expresividad de la obra, que es la parte más sentimental o humana, un sistema como el desarrollado para este proyecto generaría las notas para rellenar la plantilla previamente obtenida. Otra mejora que podría realizarse al sistema desarrollado en este proyecto sería establecer reglas musicales más elaboradas para generar notas. Por ejemplo, tomar como referencia la melodía del primer violín, mientras genera las notas de los instrumentos que la van a acompañar, para así tener en cuenta la armonía generada entre las notas, por lo que se generaría un sonido más claro y agradable. Para este propósito se deberían tener en cuenta investigaciones sobre los efectos de la armonía y cómo contribuye a la percepción de una frase musical, (Palmer and Krumhansl, 1987).

La Inteligencia Artificial genera mucha controversia en la actualidad. La concienciacion y despreocupación social debería producirse progresivamente, calmando la discursión latente alrededor de la Inteligencia Artificial y la posibilidad de "terminar con muchos puestos de trabajo". En el caso de este proyecto, la comunidad más afectada son los compositores musicales. Este último hecho debería ser rebatido aclarando que la Inteligencia Artificial funcionaría como una herramienta para mejorar su trabajo, pero a día de hoy no va a generar una partitura sin la ayuda de un compositor humano.

Approach	Instruments	Symphony trained	Details	Figure
First approach	Violins	5th	Without rests	4.15
			With rests	4.16
			With manual improvements	4.18
		7th		4.19
		5th + 7th		4.20
		5th + 7th + 9th		4.21
	Violins, Violas, Violoncellos, Contrabass, Flute, Oboe, A Clarinet	7th		4.22
Second approach	Violins, Flute	7th		4.24
	Violins, Violas, Violoncellos			4.25
Third approach	Violins, Violas, Violoncellos, Contrabass, Flute, Oboe, A Clarinet	7th		4.27
		5th + 7th + 9th		4.28
	Violins	5th		4.29
			Model 2	4.30

Table 6.1: Resumen de los experimentos

Appendix A

An approach to Beethoven's 10th Symphony

In the first half of the development of the project, we wrote an article explaining the music generation process that we were following for the computational creativity congress ICCC19. Nevertheless, it was not accepted.

An approach to Beethoven's 10th Symphony

Paula Muñoz Lago[1], **Gonzalo Méndez**[1,2]
[1]Facultad de Informtica
[2]Instituto de Tecnologa del Conocimiento
Universidad Complutense de Madrid
Madrid, España
pmunoz06@ucm.es, gmendez@fdi.ucm.es

Abstract

Luidwig van Beethoven composed his symphonies between 1799 and 1825, when he was writing his Tenth symphony. As we dispose of a great amount of data belonging to his work, the purpose of this paper is to investigate the possibility of extracting patterns on his compositional model and generate what would have been his last symphony, the Tenth. A neural network model has been built based on the Long Short-Therm Memory (LSTM) neural networks. After training the model, the generated music has been analyzed by comparing the input data with the results, and establishing differences between the generated outputs based on the training data used to obtain them. The structure of the outputs strongly depends on the symphonies used to train the network, so the music obtained presents characteristics recognisable as a Beethoven-like style.

Introduction

Music is an art, but also a global language present in every historical stage. From the prehistory, to Medieval, Baroque or Classical, each stage has it is own social, politics, and also artistic characteristics, present in paintings, literature, and music.

Romantic composer Ludwig van Beethoven wrote his Symphonies from 1799 to 1825, when he finished the No. 9. Although there's no constancy of the existence of the 10th Symphony score, there exists some sheets found in Beethoven's house after his death that are thought to be part of the upcoming Symphony. Those sheets are kept in the museum dedicated to his life in his natal city, Bonn, although they can be seen online [1]. The public manuscript is not easy to read and understand, so that existing data will not be used in this paper.

In 1988 Barry Cooper, a musicologist who wrote a book relating Beethoven's life (Cooper 2000), built from 50 fragments, the first movement of the Symphony [2]. Since it can't be proved that those found sketches were intended to be part of the 10th symphony, Barry Cooper's work has caused a big controversy.

A legend arises from this cause, called *"the 10th Symphony curse"*. Following Beethoven's steps, several great composers were found death before finishing it is 10th Symphony. This is the case of authors such as Franz Schubert (1797-1828), Anton Bruckner (1824-1896), Antonín Dvořák (1842-1904) or Gustav Mahler (1860-1911). The last one tried to avoid the curse by not assigning a number to his ninth Symphony, in order to be able to assign the number 9 to his tenth Symphony. Despite his effort in avoiding the curse, he was found death while composing the last one.

The goal of this work is to generate music, based on Beethoven's compositional model, obtaining all the orchestra instrument's scores. The first approach has been to train the system with each instrument individually, to generate all the different scores, and then put them all together in a conductor's score. In contrast, the second approach has consisted in training the system with information from different instruments at the same time. The output is intended to be fully dependent of the system prediction, although the only characteristic that we have forced is the symphony's tempo and the key, as the sheets found in Beethoven's house after his death had 3 flats, and the tempo was a 6/8. He chose to write a large portion of his compositions in this key, as it is said that it represent a "stormy and heroic tonality", and it is used in works of unusual intensity, such as the Fifth Symphony (Fig. 1).

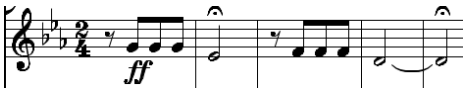

Figure 1: Snippet of Beethoven's Fifth Symphony in C minor

Regarding the C minor key, it was considered appropriate for masonic music to have that key signature, due to the importance that the number 3 and letter b had in the freemasonry. Relevant composers in the history such as Wolfgang Amadeus Mozart wrote music for masonic use, (Henry and Massin 2006). Although Beethoven is not documented as a mason, there are strong grounds for believing in that, since it is known that several of his compositions were played in the Masonic Lodge. This information is documented in *The age of Mozart and Beethoven* (Pestelli 1984).

Some musical definitions important to fully understand the paper development are:

[1] https://bit.ly/2BKPAOx
[2] https://bit.ly/2T5KBBH

- Note: Musical event that describes a sound. It contains more information apart from the note number, but also the duration, or the pitch class.

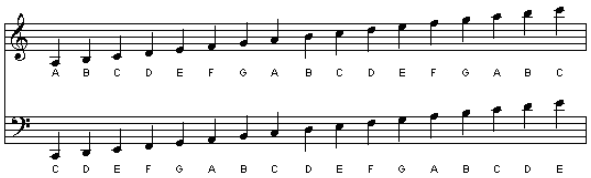

Figure 2: Notes names of the chromatic scale

- Pitch: Property of sounds that allows a frequency scale ordering, distinguishing between "higher" or "lower" sounds.
- Clef: Musical symbol used to determine the name and pitch of the written notes, it is the first symbol that appears in the score. The tree types are: F (second stave from Figure 2), C and G (first stave from Figure 2).
- Key signature: Set of sharp or flat symbols placed after the cleff, it determines the notes that will be altered from their natural pitch. A sharp raises one semitone the natural note, while the flat lowers it.
- Time signature: It determines how many beats are contained in each bar. It appears next to the key signature. As we can see in the example of Figure 1, the 2/4 time signature means that there are two crotchets or quarter notes per compass.

This paper is structured as follows. Previous work on using Artificial Intelligence in music generation is exposed in the State of the art section. After that, the work developed for this paper is explained in detail, presenting the Deep Learning technique used, the needed toolkits and how the data is represented. Then, the Music Generation subsection is divided in: Dataset description, training and prediction. The results section is focused on explaining the reason why the system returns a certain output when trained with a specific set of symphonies. The conclusions and future work are described in the last sections.

State of the art

Studied since the latter half of the 20th century, Computational Creativity can still be considered as a novel field. There has been relevant experiments on Linguistic creativity, such as narrative generation (Gervás et al. 2005), getting to write the lines of a musical called *Beyond the fence* (Gardner 2016), showed for the first time in London's Arts Theatre, but also poems (Montfort et al. 2012) or jokes generation (Ritchie 2009). Visual Arts creativity has been a recurrent conversation topic these last months due to the auctioned AI generated painting, *Portrait of Edmond de Belamy (2018)*, by the Christie's art gallery of New York, getting the price of 432.500$, whose algorithm was designed by *Obvious*[3]. Another relevant work on this field is AARON, (Cohen 1995), a robot capable of taking a brush with it is robotic arm and paint. The Painting Fool, (Colton 2012) emulates several styles.

Music creativity

This Computational Creativity sub-field started in the early 50's, although the most relevant works are mainly focused on generating coherent sounds and scores for the human musicians use. The first used Artificial Intelligence technique for this purpose was the Markov chains. This model defines the probability for an event to happen based on previous ones, storing them in a transition matrix. An example of the application of the Markov Chains is *ILLIAC* (Hiller and Isaacson 1958). This machine generated the *ILLIAC's suite*, a string quartet [4]. Generated notes were tested by heuristic compositional rules. In case that the rules weren't violated, they were kept, otherwise a backtracking process was followed. This project excluded any emotional or expressive generation, by just focusing on the notes. Later on, a system called *CHORAL*, which produced the corresponding harmonization of a given Bach Choral, was developed creating rules and setting heuristics in a logic-programming language created by the author for this purpose (Ebcioglu 1990).

In 1989 a new technique was implemented for this purpose, Recurrent Neural Networks. Since that method turned out to be limited by it is short-term coherence, Long Short Term Memory (LSTM) neural networks started to be used for this purpose. This method incorporated the ability to learn long-term dependencies. Since music is built on themes and motifs repeating over time, it makes LSTM networks a reasonable option for computer music creativity. Melodies generated with LSTM networks in existing projects have resulted more musically plausible than with other models, such as Gated Recurrent Unit (GNR) (Nayebi and Vitelli 2015). The first music generation project that used neural networks is MUSACT (Bharucha 1992), which focuses on learning the harmonic model and generates expectations after listening to a certain chord. Some other projects that use various of this networks models to generate new sounds have been developed through the last few years, using raw audio (Kalingeri and Grandhe 2016). BachBot (Liang et al. 2017) composes and completes music in the style of Bach chorales using an LSTM generative model. They conducted a discrimination test to determine if the generated music was similar to Bach's chorales with 2336 participants, getting a rate of only a 1% of the people correctly determining which music was generated with BachBot.

Some other examples of music generation projects are *DeepMusic*[5], which is integrated in Amazon's assistant Alexa as a skill, so it plays AI generated music. Chinese company Huawei recently published the unfinished part, third and fourth movements, from Shubert's symphony No. 8 (Mantilla 2019), which the author left unfinished on purpose. Using neural networks, the system gave to the mu-

[3] http://obvious-art.com/index.html

[4] https://www.youtube.com/watch?v=fojKZ1ymZlo

[5] https://amzn.to/2DBRwJc

sicians some ideas to continue the music, and then musician and composer Lucas Cantor worked on them. The final version has been played on the 4^{th} of February, 2019, in a unique concert in London.

Currently best-work known on computer music composition is EMI, (Cope and Mayer 1996). This system has successfully emulated Mozart, Brahams, Bach, Rachmaninoff or Chopin's music, generating new music [6]. It searches a pattern or signature as Cope's labeled, in at least two existing pieces of a concrete compositor. Using one of the artist scores, it locates the signatures to generate the new music, and in order to compose the music between signatures, it uses a rule analyzer. The IAMUS (Quintana et al. 2013), named this way after the god of music, Apolos's son in the ancient Greece, is a computer system created at *Universidad de Málaga*. It is capable of composing a full score in 8 minutes, using genetic algorithms, whose music has been played by the London Symphony Orchestra. In this case, chromosomes including all the notes information are randomly generated, and fitness functions are applied to each of them. If a note is codified to be played by a violin and this instrument doesn't have the possibility to play that note, it is changed. After generating around 100 scores, a human composer chooses the best one as the final output.

Another challenging field relating Musical Creativity is Music Improvisation, since it has more difficulties from a creative point of view. Using Genetic algorithms, GenJam (Biles 1994) emulates a Jazz musician in his or her improvisation learning process, while the Continuator (Pachet 2003) uses a Markov model to generate music in standalone mode, as continuations of musicians input, or as interactive improvisation.

Work description

Technical background

Deep learning: LSTM Networks Included in the field of Machine Learning, Deep Learning involves the use of neural networks in the task of providing *knowledge* to the machines. There exists several types of neural networks, such as Deep Neural, Deep Brief and Recurrent Neural Networks (RNN). In this paper we work with the last ones, since we need to process sequential data, assuming each event depends on previous ones. The most accurate RNN variant is LSTM. As proved with Figure 1, we need the memory that this type of networks own. We have the sequence F - F - F, a predictor without memory would return another F, although by learning from the notes before, it can extract that after three equal notes, it is probable that the upcoming note is two lines below the last one.

Proposed in 1997, those neural networks can learn long-term dependencies, improving the cells or neurons in the RNN graph. They have the ability to connect previous knowledge to a present task. Each cell has a certain memory, and it decides to store or forget a certain data based on a given priority, assigned by the algorithm after a certain time learning and represented as weights.

[6] https://bit.ly/2DDVKjF

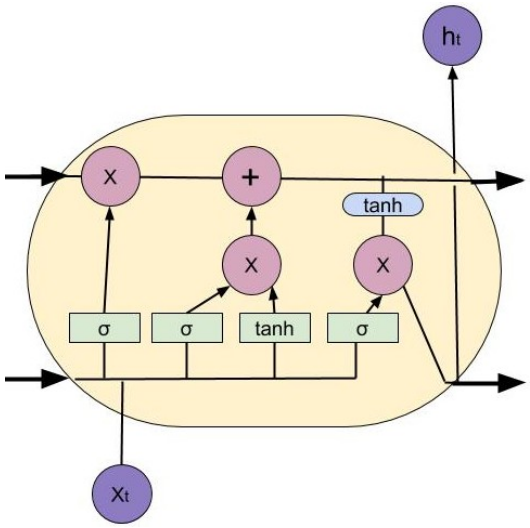

Figure 3: LSTM Neural Network cell

As it can be seen in Figure 3, the top line represents the flow of the cell state. Also, several layers are shown, such as the first sigmoid, which takes information from the previous state and determines if it is useful or not, returning 0 or 1. As it is shown with the vertical arrow, it directly affects to the flow of the cell state. The second layer is composed of another sigmoid, which chooses the data to be updated from the previous state. The tanh component creates a vector of candidate values to be added to the state. The combination of both will be added to the current cell state. The final sigmoid layer decides which parts of the current state are more relevant. Those will be sent to a tanh function, which will convert the state into 1 or -1.

Toolkits

This project has been developed in Python, and the most relevant library used is Music21 [7], which allows parsing and generating scores in different formats. Also, every musical action and representation that we needed to perform, was made possible using that library. For the Deep Learning engine we have used Keras [8], simplifying that way the use of TensorFlow. Finally, in order to manage the score formats, Musescore [9] brought us the possibility to import and export the symphonies, so we could see the score and listen to it at the same time. It has many musical functionalities and it is an open source program available for every platform.

Data representation Several ways of representing Beethoven Symphonies scores have been studied for this paper.

Firstly, as MIDI (.mid) files were popular in this research field, we used them as an input for our system, as it is a data file which contains information about the sounds: what

[7] http://web.mit.edu/music21/
[8] https://keras.io/
[9] https://musescore.com/

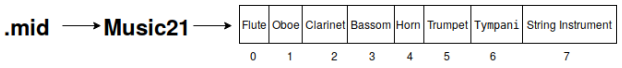

Figure 4: Music21 .mid files parsing

note is played, when and how long or loud. As MIDI files store sound information, it doesn't differentiate between all the string instruments in an orchestra, since their pitches are really similar, as it can be seen in Figure 4.

Since the main goal of this paper is to obtain all the different scores for every orchestra instrument, the input files format were changed to MXL. This extension refers to a compressed music score, which Music21 easily processes. MXL files are the compressed format of the so called MusicXML, which is the standard XML format. Music21 allows us to generate the final output in any desired format, so we can obtain it in MIDI and XML. After getting those files, Musescore can open both formats so the score can be visualized and played.

In order to represent the output of the training, i.e. the weights of the different notes and durations, the model also returns an HDF5 file, *Hierarchical Data Format* version 5, commonly used to store big quantities of data.

Music Generation

In this paper, we have established two different approaches in order to obtain the expected result, which is the new Beethoven's Tenth Symphony. The first approach is based on generating all the different orchestra instruments scores individually. By training each instrument with a concrete existing set of symphonies, we have obtained each score. After that, we have manually joined all the different scores to study if the overall symphony was musically valid. Since each instrument was trained without information of the other instruments, the obtained conductor score had a lack of coordination between them.

The second approach was intended to increase the coordination between each instrument, so we have trained a set of instruments at the same time from a concrete set of symphonies. This way, the generated scores present a considerable increment of coordination and it is easier to differentiate each musical phrase.

Dataset description All the Beethoven symphonies have been converted to an mxl file, which constitutes the dataset that we have used to obtain the desired results. Also, the instrument or instruments with which the system works has to be established, so the Python module music21 can divide the mxl score into all the present instruments, and take only the choosed ones. Then, in the first approach, where the goal is to obtain each instrument's score individually, the note names and durations are stored in an independent file, being the different tuples of note names and durations the training data. The second approach trains with the chosen instruments at the same time, so we need to store, besides the note name and duration, the offset and instrument that plays it. The offset will be used to sort the data, but after making sure that the events are sorted as they are in the original score, it can be removed from the dataset. This way, the training data will be composed of the different tuples of note names, note durations and instrument.

At this point, we create a dictionary to convert from each data tuple to a number, so the neural network can work with it.

Training Finally, we can generate the network input and output data. By establishing a certain sequence length, the output for each input sequence will be the first note that comes after the notes sequence in the input. It is important to take into account that in case of establishing a big sequence length, the machine may generalize, while setting a small sequence length, the system may over learn.

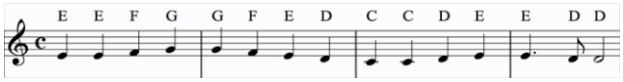

Figure 5: Violin's *Ode To Joy* snippet

For example, setting a sequence length equal to two, the first stages of the system's work flow for the Figure 5 input would be the shown in Table 1.

sequence_in	**sequence_out**
[(E, 1), (E, 1)]	[(F, 1)]
[(E, 1), (F, 1)]	[(G, 1)]
[(F, 1), (G, 1)]	[(G, 1)]
[(G, 1), (G, 1)]	[(F, 1)]

Table 1: Figure's 5 sequences

In case of the input, reshaping into a 3 dimension matrix is needed so it is compatible with the LSTM layers, using Python's numpy module. The first dimension or shape of the network is the number of different patterns obtained in the last step, the second one is the previously established sequence length and finally the last dimension is forced to be 1, so it has just one input information per sequence length. After that, the software normalizes the input into sequential values, from 0 to 1, to work with a regression model. In case of the output, it is converted into a categorical model.

The next step is to create the model, which follows a stacked LSTM architecture, since the larger the depth, the less neurons per layer the network needs, and it is faster (Graves, Mohamed, and Hinton 2013). There's no formula established to determine how many layers the network should have, and how many neurons would work better for each layer, so one of the tasks during the development of this project has been to obtain that information empirically.

The network is composed 3 different types of layers. The most relevant ones are the LSTM layers, which take the sequences and return new ones. Then, the Dropout layers prevent overfitting, ignoring randomly selected neurons during the training, setting those inputs to 0. The Dense (Density) layer serves as a full connection mechanism. This layer is the last one, so the system returns the same number of outputs as the different numbers of tuples (note name, note duration) the input data had. Finally, the activation function

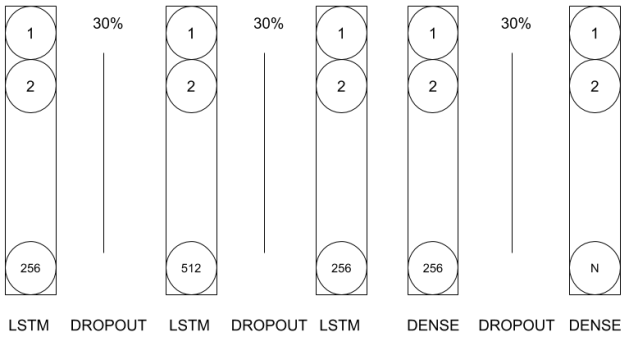

Figure 6: Model, being N the number of different tuples (note name, duration)

used for every layer is set, and it determines how each node's output is be represented. In this case, a linear activation is used, the softmax function, valid for multi-classification tasks, allowing the output to be interpreted as a probability between 0 and 1.

In this problem, since it is all about creativity, we do not have the final validation step on a not trained group of data, present in the majority of machine learning problems, due to the nonexistence of a correct solution.

Once the model is built and the input and output data are ready, it gets trained, generating an .hdf5 file containing the weights, or priorities, for the input notes.

Prediction For this task, the network input is generated again, as in the previous process (see Table 1). Since it needs to work over the same model, it is created again, with the same parameters, but now, instead of training the model, it loads the generated weights (hdf5 file) from the previous process. It is important at this point that the network input shapes and the loaded weights have the same dimensions. Once the model is ready, a matrix is created to convert from the network output to a tuple. If we are trying to generate a single instrument score, the tuple is composed of (note name, note duration), while if we want to generate a conductor's score composed of several instruments, the instrument identification will have to be included in the tuple. Then, a random sequence from the input is extracted and started to predict a fixed number of notes. As in the training, this random sequence has to be reshaped into a 3 dimension matrix. The first dimension corresponds to the number of sequences, which is always 1, the second to the length of the sequence and the third, as in the training, is forced to be 1. After that, all the sequence values are converted into sequential ones (between 0 and 1), so the model can return a prediction given those input values. The output of the prediction is an array with a probability for each tuple. Then, the system sorts the values from the greatest probability to the lowest. Once it has the indexes of the most interesting notes, the system can work on the given tuples accessing to the conversion matrix. It forces the predicted notes to have a duration greater or equal to 0.5 (quaver), for the score's simplicity. Another important restriction is to give priority to notes that belong to the key scale used in the new score, present in Figure 7.

Some other restrictions manually made is that if the note

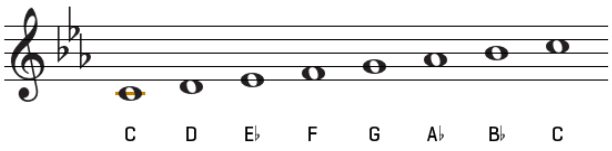

Figure 7: Key scale

with highest prediction differs more than one octave from the last one, it is transposed in order to get closer to the previous one, but not modifying the note predicted, since it will not be easy to play for a musician. Another change made at this point is that if the predicted note and the previous one are rests, the lengths are added. This can only be applied to rests since we need to have several identical notes following (see Figure 1).

After choosing the most appropriate note, the index of the selected note is added to the pattern, which serves as an input for the next prediction.

Once the system has all the required predicted information (notes, chords, rests, and all the needed information such as their durations or the instrument that plays them) it is processed and .xml and .mid files are created using music21.

Results

The system output differs from the information given to the training, although once with the same trained data, the system predicts the same score, which denotes a lack of variability.

Approach 1: Generating individual instruments melodies Training with all the Fifth Symphony's movements, the output obtained is shown in Figure 8.

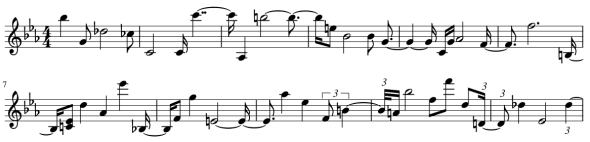

Figure 8: Results from training with the Fifth Symphony

At this point, the time measure is 4/4 as a first approach, although after discovering that Beethoven's house sketches belonging to the upcoming symphony had measure 6/8, it was set to that one.

It can be seen that different measures showed up, such as quarter, eight, sixteenth or half notes but also thirty-second notes, and a motif shows up. In the first two staves, a half note appears tied to an eight and a sixteenth note in several compass. However, there are no rests, so the next step at this point was to retrain the system, again with the most famous symphony, but allowing rests to appear. The results can be seen in Figure 9. Again, although a different score is generated, we can distinguish some patterns in the composition.

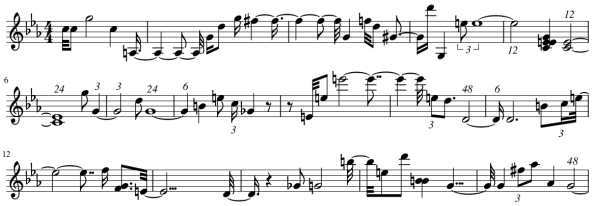

Figure 9: Results from training with the Fifth Symphony allowing rests

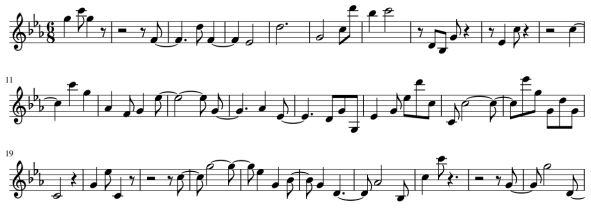

Figure 10: Results from training with the Fifth Symphony with manual improvements

At this point, the empirical restrictions explained above during the prediction are implemented, and all the experiments from this point include these manual improvements. For instance, using the same weights as before, the first three staves of the outcoming score is shown below in Figure 10.

The result differs from Figure 9, being the new one clearer but maintaining the motifs, such as the half note tied to two quarter notes, quality that characterizes Beethoven's Fifth Symphony.

Keeping the system state, we train it with the Seventh symphony, and generate the violins as before. The result looks similar, although it is remarkable the increment in the number of rests showing in the score. This may be due to the amount of silent compasses in the second movement of this symphony. Violins start playing in compass number 50, which is not a common characteristic of the violin scores in any symphony, being usually the instrument playing the main melody.

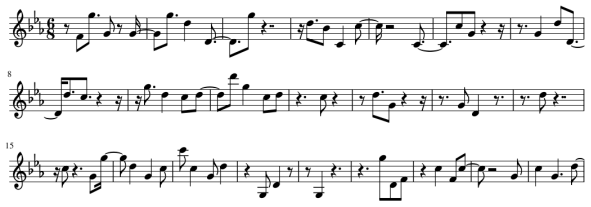

Figure 11: Results from training with the Seventh Symphony with manual improvements

Figure 11 shows only the first three staves, in which there's not an easy-to-recognize motive such as in the previous experiments. That may be because this symphony doesn't have a clear motive such as the Fifth's.

Now that we have concluded the experiment with the Fifth and Seventh symphonies, the next step is to train the system with both of them. The output is shown in Figure 12.

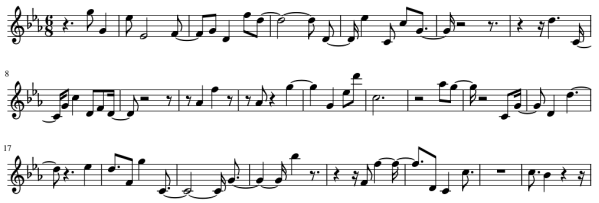

Figure 12: Results from training with the Fifth and Seventh Symphony

It can be seen that the amount of rest notes is increased from other results that doesn't use the Seventh symphony violin's as input, but the motives present in the output obtained from training with the Fifth keeps showing. The same happens in Figure 13, obtained from training the Fifth, Seventh and Ninth Symphonies Violins.

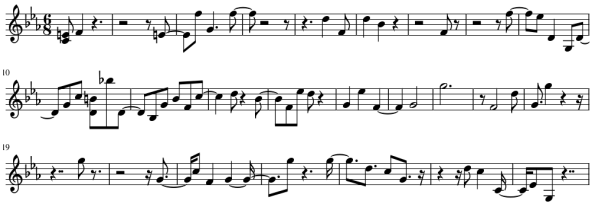

Figure 13: Results from training with the Fifth, Seventh and Ninth Symphony

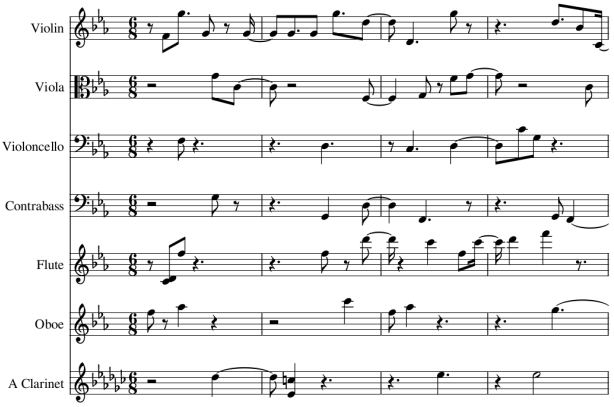

Figure 14: Results from training separately 7 different instruments with the Seventh symphony

Approach 1: Generating music for several instruments After completing all the experiments previously described, the system was trained with some of the orchestra's instruments. Figure 14 shows the prediction result for Violin, Violas, Violoncellos, Contrabass, Flutes, Oboes and Clarinets, training with the Seventh symphony. Although this result has been obtained from training each instrument individu-

ally and putting them together manually, it is distinguishable a lack of coordination between each instrument, since each melody has been generated without having knowledge on any other instrument's melody. That has caused that each musical phrase from the different instruments doesn't coordinate with the others to generate a group sound.

Approach 2: Generating several instruments at the same time To avoid the musical disorder obtained in the previous results, the second approach was used. As explained before, in this case the system is trained with a set of desired instruments, getting this way scores such as the one shown in Figure 15. This result shows how each instrument compliments the others, having the violin the main melody at the beginning, but respecting the Flute's main appearance in compasses seventh and eight.

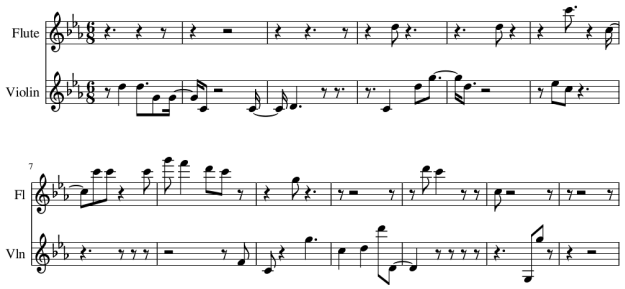

Figure 15: Second approach trained with the Seventh symphony for Flutes and Violins

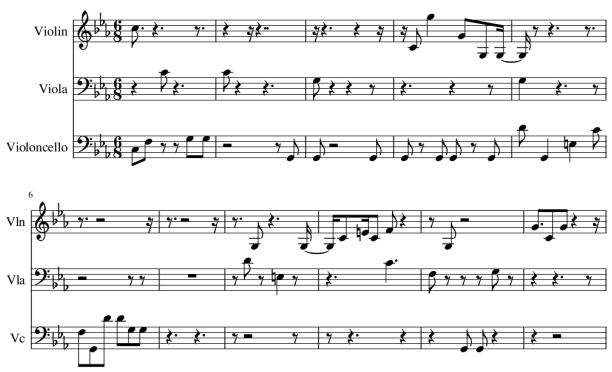

Figure 16: Score obtained from training Violins, Violas and Violoncellos with the Seventh symphony

The same behavior can be seen in the result shown in Figure 16, which shows how Violins, Violas and Violoncellos, while being trained only with the Seventh symphony, assumes a trio music by respecting the other instrument's melodies and complementing each other. It can be appreciated the differences between this score and the one shown in Figure 14. In that case, the corresponding lines are the first, second and third (Violin, Viola and Violoncello). As it can be seen, the coherence of the different instruments is enhanced in the second approach.

All the mp3 results are available in Github repository [10].

Conclusions

This paper explores the possibility of generating new music based on the Beethoven's style by a system doted with Artificial Intelligence, using LSTM neural networks, which learn and remember musical phrases of a concrete length, finally showing that it is possible to obtain music that imitates this composer's style for several instruments.

During the specification of the problem, we established two ways of approximating to the new symphony. The first one was to train and generate separately each instrument scores, and manually creating the conductor's score. The results obtained were satisfactory for each single instrument separately, getting to generate music in a recognizable style. However, when joining all the different scores, the sound was not coordinated and the musical phrases belonging to the different instruments were not respected by the others. We concluded that with this first approach we could generate solo scores, but not group music. The second approach was intended to solve the main problem that the first one presented, that the instruments were not sufficiently coordinated since each instrument was trained separately, without any information on the music that the others were playing, which is crucial in an orchestra. The solution proposed was to train and generate music belonging to different instruments at the same time. This way the results obtained were more coordinated and we could see that each instrument respected each other, having rests or accompanying the main melody when they did not have the leading voice.

The amount of results obtained can be seen in Table 2, as we have progressively studied the output generated with both approaches, by first working on the generation of single instruments score, and checking that way if they were musically correct, to finally generate a conductor score. The system can return solo scores, but also duos, trios, quartets and an orchestra score, although we have not got to generate the score trained with all the existing symphonies.

The human interpreter is always the source of emotions, so it is remarkable the lack of dynamics in the generated music, being played all the notes at the same volume during the whole piece. In this paper we have focused in the notes production and instruments coordination, so generated scores have not notation of the dynamics.

Future work

Following the problem exposed in the conclusion, the next step is to research in music expressiveness, in order to transmit it to the system, to obtain music similar to what a human composer would create. An option to start in this task could be to obtain the score's dynamics, and train a Deep Learning model with the expressiveness of the work, in order to generate a *template*, which would be the equivalent to the composer's way to capturing his or her feelings. After generating the dynamics, the *"most human"* or sentimental part, a system like the one created for this work would generate

[10] `https://bit.ly/2tzuHBb`

Approach	Instruments	Symphony trained	Details	Figure
First approach	Violins	5th	Without rests	8
			With rests	9
			With rests and manual improvements	10
		7th	With rests and manual improvements	11
		5th + 7th	With rests and manual improvements	12
		5th + 7th + 9th	With rests and manual improvements	13
	Violins, Violas, Violoncellos, Contrabass, Flute, Oboe, A Clarinet	7th	With rests and manual improvements	14
Second approach	Violins, Flute	7th	With rests and manual improvements	15
	Violins, Violas, Violoncellos	7th	With rests and manual improvements	16

Table 2: Results

the notes and they would be fitted in the dynamic's template. Another improvement that could be made to the developed system is to establish more elaborated musical rules to generate notes. For instance, taking First violin's melody as the main motive, while generating new instrument notes, it should be taken into account the harmony created between the notes, so a nice and clear sound is composed. For that purpose, some musical research about harmony effects and how it contributes to the perception of a musical phrase, (Palmer and Krumhansl 1987), should be considered.

The social awareness and unconcern should be progressively made, by calming down the latent discussion around Artificial Intelligence and the possibility of *stealing* human jobs. In case of this paper, the most affected community are the music composers, worried of being substituted by machines. This last fact should be contradicted by clarifying that Artificial Intelligence will work as a tool to enhance their production, but, at this point, it will not generate any score without a composer's help.

References

[Bharucha 1992] Bharucha, J. J. 1992. Musact: A connectionist model of musical harmony. In *Machine Models of Music*, 497–509. MIT Press.

[Biles 1994] Biles, J. 1994. Genjam: A genetic algorithm for generating jazz solos. In *International Computer Music Conference*.

[Cohen 1995] Cohen, H. 1995. The further exploits of aaron, painter. *Stanford Hum. Rev.* 4(2):141–158.

[Colton 2012] Colton, S. 2012. The painting fool: Stories from building an automated painter. In *Computers and creativity*. Springer. 3–38.

[Cooper 2000] Cooper, B. 2000. *Beethoven*. Oxford University Press, USA.

[Cope and Mayer 1996] Cope, D., and Mayer, M. J. 1996. *Experiments in musical intelligence*, volume 12. AR editions Madison.

[Ebcioglu 1990] Ebcioglu, K. 1990. An expert system for harmonizing chorales in the style of j.s. bach. *The Journal of Logic Programming* 8(1):145 – 185. Special Issue: Logic Programming Applications.

[Gardner 2016] Gardner, L. 2016. Beyond the fence review computer-created show is sweetly bland. *The Guardian*.

[Gervás et al. 2005] Gervás, P.; Daz-Agudo, B.; Peinado, F.; and Hervs, R. 2005. Story plot generation based on cbr. *Knowledge-Based Systems* 18(4):235 – 242. AI-2004, Cambridge, England, 13th-15th December 2004.

[Graves, Mohamed, and Hinton 2013] Graves, A.; Mohamed, A.; and Hinton, G. E. 2013. Speech recognition with deep recurrent neural networks. *CoRR* abs/1303.5778.

[Henry and Massin 2006] Henry, J., and Massin, B. 2006. *Mozart the freemason: the masonic influence on his musical genius*. Inner Traditions.

[Hiller and Isaacson 1958] Hiller, Jr., L. A., and Isaacson, L. M. 1958. Musical composition with a high-speed digital computer. *J. Audio Eng. Soc* 6(3):154–160.

[Kalingeri and Grandhe 2016] Kalingeri, V., and Grandhe, S. 2016. Music generation with deep learning. *CoRR* abs/1612.04928.

[Liang et al. 2017] Liang, F. T.; Gotham, M.; Johnson, M.; and Shotton, J. 2017. Automatic stylistic composition of bach chorales with deep lstm. In *ISMIR*, 449–456.

[Mantilla 2019] Mantilla, J. R. 2019. Um algoritmo completa a misteriosa sinfonia inacabada de schubert. *El País*.

[Montfort et al. 2012] Montfort, N.; Baudoin, P.; Bell, J.; Bogost, I.; Douglass, J.; Marino, M. C.; Mateas, M.; Reas, C.; Sample, M.; and Vawter, N. 2012. *10 PRINT CHR (205.5+ RND (1));: GOTO 10*. mit Press.

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

[Pachet 2003] Pachet, F. 2003. The continuator: Musical interaction with style. *Journal of New Music Research* 32(3):333–341.

[Palmer and Krumhansl 1987] Palmer, C., and Krumhansl, C. L. 1987. Pitch and temporal contributions to musical phrase perception: Effects of harmony, performance timing, and familiarity. *Perception & Psychophysics* 41(6):505–518.

[Pestelli 1984] Pestelli, G. 1984. *The age of Mozart and Beethoven*. Cambridge University Press.

[Quintana et al. 2013] Quintana, C. S.; Arcas, F. M.; Molina, D. A.; Fernández, J. D.; and Vico, F. J. 2013. Melomics: A case-study of ai in spain. *AI Magazine* 34(3):99–103.

[Ritchie 2009] Ritchie, G. 2009. Can computers create humor? *AI Magazine* 30(3):71.

Bibliography

Y así, del mucho leer y del poco dormir,
se le secó el celebro de manera que vino
a perder el juicio.

Miguel de Cervantes Saavedra

Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G. et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, 173–182. 2016.

Bharucha, J. J. Musact: A connectionist model of musical harmony. In *Machine Models of Music*, 497–509. MIT Press, 1992.

Biles, J. Genjam: A genetic algorithm for generating jazz solos. In *International Computer Music Conference*. 1994.

Binsted, K. and Ritchie, G. Computational rules for generating punning riddles. *HUMOR-International Journal of Humor Research*, Vol. 10(1), 25–76, 1997.

Bishop, C. M. *Pattern recognition and machine learning*. springer, 2006.

Boden, M. The turing test and artistic creativity. *Kybernetes*, Vol. 39, 409–413, 2010.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J. et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Breiman, L. Random forests. *Machine Learning*, Vol. 45(1), 5–32, 2001. ISSN 1573-0565.

Bressan, G. M., de Azevedo, B. and ElisangelaAp, S. A decision tree approach for the musical genres classification. *Applied Mathematics*, Vol. 11(6), 1703–1713, 2017.

Cho, S.-B. and Won, H.-H. Machine learning in dna microarray analysis for cancer classification. In *Proceedings of the First Asia-Pacific bioinformatics conference on Bioinformatics 2003-Volume 19*, 189–198. Australian Computer Society, Inc., 2003.

Cohen, H. The further exploits of aaron, painter. *Stanford Humanities Review*, Vol. 4(2), 141–158, 1995.

Collobert, R. and Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, 160–167. ACM, 2008.

Colton, S. The painting fool: Stories from building an automated painter. In *Computers and creativity*, 3–38. Springer, 2012.

Conroy, J. M. and O'leary, D. P. Text summarization via hidden markov models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, 406–407. ACM, New York, NY, USA, 2001. ISBN 1-58113-331-6.

Cooper, B. *Beethoven*. Oxford University Press, USA, 2000.

Cope, D. and Mayer, M. J. *Experiments in musical intelligence*, Vol. 12. AR editions Madison, 1996.

Ebcioglu, K. An expert system for harmonizing chorales in the style of j.s. bach. *The Journal of Logic Programming*, Vol. 8(1), 145 – 185, 1990. ISSN 0743-1066. Special Issue: Logic Programming Applications.

Gardner, L. Beyond the fence review – computer-created show is sweetly bland. *The Guardian*, 2016.

Gervás, P., Díaz-Agudo, B., Peinado, F. and Hervás, R. Story plot generation based on cbr. *Knowledge-Based Systems*, Vol. 18(4), 235 – 242, 2005. ISSN 0950-7051. AI-2004, Cambridge, England, 13th-15th December 2004.

Gervás, P. Generating poetry from a prose text: Creativity versus faithfulness. *AISB'01 Symposium on Artificial Intelligence and Creativity in Arts and Science*, 2001.

Goodfellow, I., Bengio, Y. and Courville, A. *Deep learning*. MIT press, 2016.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. Generative

adversarial nets. In *Advances in neural information processing systems*, 2672–2680. 2014.

Graves, A., Mohamed, A. and Hinton, G. E. Speech recognition with deep recurrent neural networks. *CoRR*, Vol. abs/1303.5778, 2013a.

Graves, A., Mohamed, A.-r. and Hinton, G. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, 6645–6649. IEEE, 2013b.

Hartigan, J. A. and Wong, M. A. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, Vol. 28(1), 100–108, 1979. ISSN 00359254, 14679876.

Hiller, L. A. and Isaacson, L. M. *Experimental Music; Composition with an Electronic Computer*. Greenwood Publishing Group Inc., Westport, CT, USA, 1979. ISBN 0313221588.

Hiller, L. A., Jr. and Isaacson, L. M. Musical composition with a high speed digital computer. *Audio Engineering Society Convention 9*, 1957.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, Vol. 9(8), 1735–1780, 1997.

Jean-Pierre Briot, G. H. and Pachet, F. Deep learning techniques for music generation - A survey. *CoRR*, 2017.

Ji, S., Xu, W., Yang, M. and Yu, K. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, Vol. 35(1), 221–231, 2013.

Jordanous, A. A standardised procedure for evaluating creative systems: Computational creativity evaluation based on what it is to be creative. *Cognitive Computation*, Vol. 4(3), 246–279, 2012. ISSN 1866-9964.

Kaelbling, L. P., Littman, M. L. and Moore, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research*, Vol. 4, 237–285, 1996.

Kalingeri, V. and Grandhe, S. Music generation with deep learning. *CoRR*, Vol. abs/1612.04928, 2016.

Kemeny, J. G. and Snell, J. L. *Markov Chains*. Springer-Verlag, New York, 1976.

Kononenko, I. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in medicine*, Vol. 23(1), 89–109, 2001.

Krogh, A., Larsson, B., Von Heijne, G. and Sonnhammer, E. L. Predicting transmembrane protein topology with a hidden markov model: application to complete genomes. *Journal of molecular biology*, Vol. 305(3), 567–580, 2001.

Ku, C.-C., Lee, K. Y. and Edwards, R. Improved nuclear reactor temperature control using diagonal recurrent neural networks. *IEEE Transactions on Nuclear Science*, Vol. 39(6), 2298–2308, 1992.

Lawrence, S., Giles, C. L., Tsoi, A. C. and Back, A. D. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, Vol. 8(1), 98–113, 1997.

Liang, F. T., Gotham, M., Johnson, M. and Shotton, J. Automatic stylistic composition of bach chorales with deep lstm. In *ISMIR*, 449–456. 2017.

Liu, I. and Ramakrishnan, B. Bach in 2014: Music composition with recurrent neural network. *CoRR*, Vol. abs/1412.3191, 2014.

de Mántaras, R. L. *The next step: Exponential life*. Turner, 2017.

Mantilla, J. R. Um algoritmo completa a misteriosa 'sinfonia inacabada' de schubert. *El País*, 2019.

Montfort, N., Baudoin, P., Bell, J., Bogost, I., Douglass, J., Marino, M. C., Mateas, M., Reas, C., Sample, M. and Vawter, N. *10 PRINT CHR (205.5+ RND (1));: GOTO 10*. mit Press, 2012.

Morales-Manzanares, R., Morales, E. F., Dannenberg, R. and Berger, J. Sicib: An interactive music composition system using body movements. *Computer Music Journal*, 2001.

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

Nguyen, A. M., Yosinski, J. and Clune, J. Innovation engines: Automated creativity and improved stochastic optimization via deep learning. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, 959–966. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3472-3.

Nielsen, M. A. *Neural networks and deep learning*, Vol. 25. Determination press San Francisco, CA, USA:, 2015.

Nieuwenhuizen, C. Barry cooper speaks for beethoven 10th symphony. 2001.

Pachet, F. The continuator: Musical interaction with style. *Journal of New Music Research*, Vol. 32(3), 333–341, 2003.

Palmer, C. and Krumhansl, C. L. Pitch and temporal contributions to musical phrase perception: Effects of harmony, performance timing, and familiarity. *Perception & Psychophysics*, Vol. 41(6), 505–518, 1987.

Patel, J., Shah, S., Thakkar, P. and Kotecha, K. Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. *Expert Systems with Applications*, Vol. 42(1), 259–268, 2015.

Perozzi, B., Al-Rfou, R. and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, 701–710. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2956-9.

Pinar Saygin, A., Cicekli, I. and Akman, V. Turing test: 50 years later. *Minds and Machines*, Vol. 10(4), 463–518, 2000. ISSN 1572-8641.

Quintana, C. S., Arcas, F. M., Molina, D. A., Fernández, J. D. and Vico, F. J. Melomics: A case-study of ai in spain. *AI Magazine*, Vol. 34(3), 99–103, 2013.

Rabiner, L. R. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, Vol. 77(2), 257–286, 1989.

Raschka, S. *Python machine learning*. Packt Publishing Ltd, 2015.

Reese, G. *Music in the Renaissance*. WW Norton New York, 1959.

Ritchie, G. Can computers create humor? *AI Magazine*, Vol. 30(3), 71, 2009.

Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, Vol. 65(6), 386, 1958.

Seber, G. A. and Lee, A. J. *Linear regression analysis*, Vol. 329. John Wiley & Sons, 2012.

Seymore, K., McCallum, A. and Rosenfeld, R. Learning hidden markov model structure for information extraction. In *AAAI-99 workshop on machine learning for information extraction*, 37–42. 1999.

Skúli, S. How to generate music using a lstm neural network in keras. 2017.

Todd, P. M. and Loy, D. G. *Music and connectionism*. Mit Press, 1991.

Tracy, M. S. *Bach in Beta: Modeling Bach chorales with Markov Chains*. PhD thesis, Harvard University, 2013.

Voyant, C., Notton, G., Kalogirou, S., Nivet, M.-L., Paoli, C., Motte, F. and Fouilloy, A. Machine learning methods for solar radiation forecasting: A review. *Renewable Energy*, Vol. 105, 569–582, 2017.

Wang, L. *Support vector machines: theory and applications*, Vol. 177. Springer Science & Business Media, 2005.

White, S. D. and Frenk, C. S. Galaxy formation through hierarchical clustering. *The Astrophysical Journal*, Vol. 379, 52–79, 1991.

Wikipedia. Music history. 2019a.

Wikipedia. Music history. 2019b.

Yanchenko, A. *Classical Music Composition Using Hidden Markov Models*. PhD thesis, Duke University, 2017.