NOVA IMS Information Management School

MAA

Master Program in Advanced Analytics

Algorithmic Composer, an Unconventional Music Classification System

Carolina Almeida Duarte

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master in Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão da Informação
Universidade Nova de Lisboa

Algorithmic Composer, an Unconventional Music Classification System

To all of those who supported me throughout this journey.
Every second I shared with you was timeless. Your company
has inspired me and made me a better person.

Acknowledgements

I would like to express my gratitude to my advisor Professor Leonardo Vanneschi, for his support in the development of this thesis and, specially, for his joy and dedication as a teacher which sparked my own passion for the Machine Learning world.
I would also like to thank Bea Babiak and András Kolbert, my dearest friends and colleagues during the first year of the Masters and from whom I have learned so much.
Also, this thesis would not be possible without Miguel Domingos - you introduced me to Music. You have inspired me with your knowledge and with your skills.
And finally, I would like to thank my dearest work friends, with whom I have shared so much during the last year. I feel blessed to have met you all. Márcia Pinheiro, Sara Lopes, Joana Loureiro, Alexandra Jesus, Tiago Rodrigues... And especially you, André Marques! You gave me the strengh and energy to finish this document.

Abstract

Music is an inherent part of the human existence. As an art, it has mirrored its evolution and captured its thinking and creative process over the years. Given its importance and complexity, machine learning has long embraced the challenge of analyzing music, mainly through recommendation systems, classification and composition tasks.

Current classification systems work on the base of feature extraction and analysis. The same applies for music classification algorithms, which require the formulation of characteristics of the songs. Such characteristics can be of varying degrees of complexity, from spectrogram analysis to simpler rhythmic and melodic features. However, finding characteristics to faithfully describe music is not only conceptually hard, but mainly too simplistic and restrictive given its complex nature.

A new methodology for music classification systems is proposed in this thesis, which aims to show that the knowledge learned by state of the art composition systems can be used for classification, without need for direct feature extraction.

Using an architecture of recurrent neural networks (RNN) and long-short term memory cells (LSTMs) for the composition systems and implementing a voting scheme between them, the classification accuracy of the experiments between classes of the Nottingham dataset ranged between 60% and 95%. These results provide strong evidence that composition systems do indeed possess valuable information to distinguish between classes of music. They also prove that an alternative method to standard classification is possible, as classification targets are not directly used for training.

Finally, the extent to which these results can be used for other applications is discussed, namely its added value to more complex classification systems, as well as to recommendation systems.

Keywords: Music Classification; Composition Systems; Recurrent Neural Networks; Long Short Term Memory Cells

Resumo

A Música é uma componente inerente à existência humana. Enquanto arte, tem refletido a sua evolução e captado o seu processo cognitivo e criativo ao longo dos tempos. Tendo em conta a sua importância e complexidade, a área do Machine Learning desde há muito abraçou este desafio, sobretudo através de sistemas de recomendação, classificação e composição musical.

Os sistemas de recomendação atuais funcionam na base de extração de features e respetiva análise. O mesmo se aplica a algoritmos de classificação musical, que requerem a formulação de características musicais. Estas podem ter diferentes graus de complexidade, desde análise de espectros a simples features melódicas e rítmicas. Contudo, formular caracteríticas musicais não só é conceptualmente difícil, como sobretudo simplista e restritivo dada a sua natureza complexa.

Uma nova metodologia para sistemas de classificação musical é proposta nesta tese, com o objectivo de demonstrar que o conhecimento aprendido por sistemas de composição pode ser utilizado para classificação, sem que haja necessidade de um processo de conceptualização e extração de características.

Utilizando uma arquitectura de redes neuronais recorrentes e células de memória longa e curta para os sistemas de composição e implementando um sistema de votação entre eles, a precisão para classificações binárias entre as classes do Nottingham dataset variou entre 60% e 95%. Estes resultados demonstram uma forte evidência de que os algoritmos de composição podem ser utilizados para tarefas de classificação e provam ainda que um método alternativo à classificação convencional é possível.

Finalmente, a aplicabilidade destes resultados para outros projetos é discutida, nomeadamente o valor acrescentado que pode trazer para sistemas de classificação mais complexos, assim como a sistemas de recomendação.

Palavras-chave: Classificação Musical; Sistemas de Composição; Redes Neurais Recorrentes; Células de Memória Longa e Curta

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning techniques have opened new possibilities in a world where most seemed already explored. Wherever there is data, there is potential for discovery. There has come a time when it is possible to stare again at an immense ocean and wonder what is beyond. An ocean of data waiting to be uncovered.

Reality is data waiting to be processed and comprehended – it is why it is such a plentiful resource – and yet, some things are still hardly recognized as such because of their unstructured nature. Music, for instance. Music is the information the human mind extracts from a collection of sounds. However, thinking of music as data is still somehow surprising. The first goal of this thesis is therefore to explore a subject outside the beaten path and increase the sensibility and awareness of what data is and is not and how it can be treated in order to extract information from it.

With the current machine learning solutions, the number of tasks that only the human mind can do in comparison with intelligent systems is rapidly decreasing and in many cases the performance of the later has been higher than the former – otherwise, such systems would not be in use. Music, however, is still a task reserved to the talented mind. Intelligent systems are still far from composing at a high-quality level or even understanding music, mostly because of our inability to properly describe it so that the knowledge behind its creation can be learnt. This brings the second goal of this thesis – to explore the current work made on the field regarding music composition and classification and understand its fundamental ideas, as well as its weaknesses and strengths.

An investigation of the current state of music classification algorithms reveals an interesting paradox. Classification systems work based on characteristics of the data, from which several algorithms can be applied to determine a function that performs a better separation between classes. In the case of music classification systems, those characteristics are musical features of the songs. However, given the inherent difficulty of describing something as complex as music, it makes little sense to base a classification system on characteristics that are unable to capture the nature of music itself. This factor brings motivation and inspiration to the theme of this thesis, which is to

develop a new concept of a musical classification system that avoids the direct use of characteristics of the songs, requiring no extraction of the same.

Such system will be based on the performance of the composition systems on the songs to be classified. Each composer will be specialized in a certain class of music, which can theoretically be achieved by training it only with songs that belong to that class. It is then expected that, given a song, the composition system which better succeeds at predicting it will determine its class.

With this methodology, no direct training on the target variable is performed for the classification and no need for direct extraction of features of the songs is necessary, which represents a new paradigm in classification algorithms. In order explore this concept, the current work is organized as follows. In section 2 the literature of current composition and musical classification algorithms is reviewed. Section 3 provides basic knowledge on the mathematical foundations of the algorithms which will be used ahead. Section 4 explores a variety of musical representations and its weaknesses and strengths regarding its use for composition. In section 5, the structure and technical details of the classification system proposed in this thesis are explained in detail and in section 6 its results on several different experiments are presented and discussed. Finally, section 7 provides an overlook of future improvements to the current work, as well as perspective of its value and scope of application.

Chapter 2

Previous and Related Work

Music has long captivated and inspired the human mind, with millions of songs being composed over the course of history. What might be surprising at first sight is that music is also data – much like everything that surrounds us. To be aware of reality is to be cognizant of the surrounding events and this is only possible when receiving data from those elements – when the lights go off, awareness of the surrounding space is lost. Music is the brain's interpretation of the information present in sound waves and so, in its simplest form, data waiting to be processed.

With music being such a plentiful and challenging type of data, and considering its importance to society, the research on the field has been intensive. The main fields of study include recommendation systems, classification and algorithmic composition. In this thesis, the focus will be on the latter two.

2.1 Algorithmic Composition

For years, music composition has been a task reserved only to the most talented. Time has brought new players into action and, since the 1950s, different computational techniques related to Artificial Intelligence have been used for composition. Hiller and Isaacson introduced the first computer-generated musical piece in 1955 [1]. The "Iliac Suite" was a composition for string quartet divided into four experiments - the first three where generated with pre-defined musical rules and the fourth using variable order Markov chains. This work was particularly interesting as it contained the two main approaches later followed in the field: rule-based and learning systems.

Rule-based systems are the most intuitive means of creating an artificial composition and rely on a set of rules which the computer is obliged to follow either in producing or assessing music. The complexity of such rules may vary, but a simple example may be found in the Musical Dice Games developed by Mozart in which pre-composed phrases were combined using a table of rules and the outcome of a dice roll. Those rules can also embody complex knowledge about specific musical styles, as in [18] where 270 production rules, constraints and heuristics where found from the

empirical observation of J. S. Bach chorales.

The systems previously described use rules to evaluate randomly generated sequences of music and their connection with the previously defined sequence. However, more complex systems can be built based on rules, as is the case of evolutionary algorithms. Such algorithms rely on a fitness function to evaluate sequences and then evolve them through a set of operators that simulate the effects of crossover and mutation on a population of musical sequences. In theory, human evaluation could be used to assess the quality of those sequences, however it is not a scalable solution. This fitness bottleneck can be eliminated by creating automated evaluators, which are commonly defined as a weighted sum of a set of features of the composition [9]. Relevant works can be found in [1], [7] and [21], where several rules concerning melody, harmony and rhythm are incorporated into a fitness function and in [11], where a statistical approach is taken to find mean values for certain characteristics, with fitness being then defined as the distance between the individual and the mean of the characteristic.

Two main issues arise with the rule-based systems described: not only it is difficult to determine rules concerning what is musically pleasant or not - whenever a set of rules is nailed down, exceptions and extensions are always discovered that necessitate more rules [3] - but those rules can also restrict the system. With music being an art, creativity and unpredictability are both critical. Rule-based systems, by definition, lack exactly this higher-level knowledge.

Learning systems are the other main approach to music composition - rather than requiring the development of a set of musical rules, they are trained on a set of musical examples. As such, and unlike rule-based systems, the richness of the final compositions is only limited by the power of the algorithm and, naturally, by the data itself. The first notable works following this approach used Markov chains [1] [2].

Although Markov-process music has proven to be successful over the short term, it has failed to show structure over the long term. In other words, the events at a given time have a small influence range over the following events, which for time dependent problems such as music is a critical downside. Theoretically this issue could be overcome by using higher order chains, for they would allow to increase the influence range along the time axis. However, this would imply an exponential growth in the state space, making it increasingly difficult to adequately populate the corresponding transition matrix and, ultimately, to train the model [4]. For this reason, recurrent neural networks (RNNs) have taken place as the number one choice for learning composition systems. In theory, their architecture should allow an easier exploration of long term dependencies on time series problems.

However, the first works on music composition with RNNs revealed difficulties in the training procedure, as stated by Mozer (1994) [5]. In his work, melodies were generated by a system named CONCERT, which was trained on sets of 10 Bach pieces to generate melodies by note-wise prediction. The results exposed a lack of ability

to learn the global structure of the music, similarly to what had been reported with Markov chains. But the issue was identified - the difficulty in training the system was due to the vanishing gradient problem.

The vanishing gradient problem, as described by Bengio et al. (1994) [32] causes difficulty in learning long term dependencies in the sequence. It is inherent not only to RNNs, but to all gradient based methods in general and may be avoided to some extent using different techniques. Regarding RNNs, the problem can be solved using modified cells – Long-Short Term Memory cells (LSTMs). As the name suggests, each of these cells contains an internal memory that allows them to keep track of the information considered relevant, even if it refers to events from a distant past.

Several studies have been made in the recent years using this architecture (RNN + LSTM), for instance [24] and [31]. Mostly, they differ by the type of music representation used and the extent to which the different components of the music are used for prediction – if more than one instrument is used, if only chords or only melodies are predicted, etc. Regarding the representation of music, various alternatives have been studied, with focus on text and binary matrix representations. The first corresponds to music written in abc notation and can be viewed as a natural language processing (NPL) problem, as explored in [29] and [24]. This approach is certainly richer than a binary matrix representation, however it is also considerably slower to successfully train such systems. For this reason, and as composition is not the final goal of this thesis, a binary matrix representation is used.

Works following this approach can be found in [35], [31] and [15], with focus on the former two, for they successfully analyze both the melodic and the harmonic structure of music and thus serve as the main source of inspiration for the composition system used in this thesis.

2.2 Classification

Reality is a continuous flow of information – processing such an amount of data is a task the human mind has successfully accomplished through its ability of finding patterns and organize them into groups.

Algorithmic classification aims at automating this task and ultimately apply it to problems that are still unreachable. This process usually implies the identification of characteristics that are as unique as possible to the subject of study. Music classification follows the same direction – classification in genre / artist / epoch, etc. is usually accomplished by determining a set of characteristics and feeding them into an algorithm. Through the years, the differences between studies in the field have mainly been around feature extraction and the algorithms used.

Inspiring works can be found in [33], where 109 musical features where extracted from symbolic recordings and trained with an ensemble of feedforward neural networks and k-nearest neighbor classifiers to classify the recordings by genre with success

rates of up to 90%, and also in [10], where a convolutional recurrent neural network (CRNN) was developed for the purpose of music tagging. Adopting a RNN architecture allowed to consider the global structure of the music by taking into consideration the information extracted from log-amplitude mel-spectrograms at each time interval analyzed. Other works have been important to the field of music classification, as in [12], [13] and [17], with success rates ranging between 63% and 84%.

All the classification systems mentioned before share a common ground – they all rely on a set of pre-defined musical features. The challenge is to find which of those features can successfully describe something as complex as music, assuming it is possible. The similarity with rule-based systems is undeniable and so current music classification systems suffer from the same issues – good rules or features are hard to find and, mainly, they are usually restrictive.

In this thesis, a classification system is proposed in order to overcome this issue. With inspiration on the mentioned works, an ensemble of recurrent neural networks will be used to classify different types of music. However, no set of extracted features will be used. Instead, the analysis of the songs similarity will be left for a composition system to determine, allowing to consider the time series nature of the problem. Such architecture defies the nature of classification algorithms, as the information of the target classes is not directly provided for training and no predefined set of features is used. Rather, n composition systems are trained for each of the n target classes defined, only with songs from the corresponding class. A new music will then be labeled according to the target class of the most successful composer.

Chapter 3

Neural Networks

This chapter aims to cover the principles of neural network architectures needed to develop a successful music composition system.

3.1 The Inspiration Behind Neural Networks

The study of artificial neural networks has been motivated by their similarity to working biological systems, which consist of very simple but numerous neurons that work massively in parallel and have the capability to learn [17].

A neuron is a nerve cell composed by:

- Dendrites: these structures receive information passed on from other neurons by means of an electrical or chemical process called synapse;
- Nucleus: when the accumulated signal collected from different dendrites exceeds a certain threshold, an electrical pulse is activated;
- Axon: the electrical pulse generated in the nucleus is transmitted through the axon to all the connected neurons.

While a single neuron is an extremely simple entity, it is the number and the way neurons are interconnected that makes the brain so powerful. Inside a computer, the equivalent to a brain cell is a device called a transistor. While a typical brain contains 10^{11} neurons with a switching time of 10^{-3}, a computer comprises 10^{9} transistors with a switching time of 10^{-9} seconds [17]. However, transistors in a computer are wired in relatively simple serial chains, while the neurons in a brain are densely interconnected in complex, parallel ways.

The basic idea behind an artificial neural network is therefore to simulate the structure of the brain by recreating the process occurring inside neurons and connecting them together.

3.2 Components of Artificial Neural Networks

Neurons are the building blocks of neural networks and hence, the first step in building an artificial neural network is to replicate their functioning. In order to design an artificial neuron i, the roles of each of the components of biological neurons should be reproduced:

- The dendrites of an artificial neuron i are represented by all the incoming connections from other neurons;
- The nucleus of an artificial neuron i is simulated using a propagation function f, which congregates all the signals from incoming neurons, and an activation function $\sigma(f)$, which models the response of the neuron to the stimuli;
- The axon of an artificial neuron k establishes synapses with the dendrites of all the connected neurons. A synapse between the axon from the neuron k and the dendrites of the neuron j in the i^{th} layer of the network has an associated strength w^i_{jk}. Synaptic strenghts are learnable parameters and control the influence of the information passed along neurons. The learning process is dependent on a cost function E and it is usually achieved artificially using gradient descent methods.

The functioning of biological neurons can then be reproduced given the associations described previously and once the mentioned functions have been properly defined. The learning mechanisms will be explored in later sections, as they are dependent on the neural network architecture as a whole and not on a single neuron.

Before defining and exploring in greater detail the propagation, activation and cost functions, it is imperative to define a terminology for the different components of an artificial neural network (ANN) architecture. In ANNs, neurons are distributed in layers – at least two, the input and the output layer. Generally, the neurons from one layer are connected to all the neurons from the antecedent and the preceding layers. Considering only the interactions between the j^{th} neuron in the i^{th} layer of the network and all its connections from the previous layer, as shown in figure 3.1.

The following notation applies:

- w^i_{jk} is the weight of the connection between the j^{th} neuron in the i^{th} layer and the k^{th} neuron from the previous layer;
- b^i_j is the weight of the bias term, sometimes also interpreted as $b^i_j = \omega^i_{j0}$. The bias term is constant and equal to 1, the same for all the neurons in the network, and its weight only changes in order to allow for shifts in the activation function;
- a^{i-1}_k is the output of the k_{th} neuron in the $(i-1)_{th}$ layer of the network. Let $z^i_j = f(\omega^i_{jk}, a^{i-1}_k)$ and $\sigma(z^i_j)$ be respectively the propagation and the activation functions yet to be defined. Then: $a^i_j = \sigma(z^i_j)$.

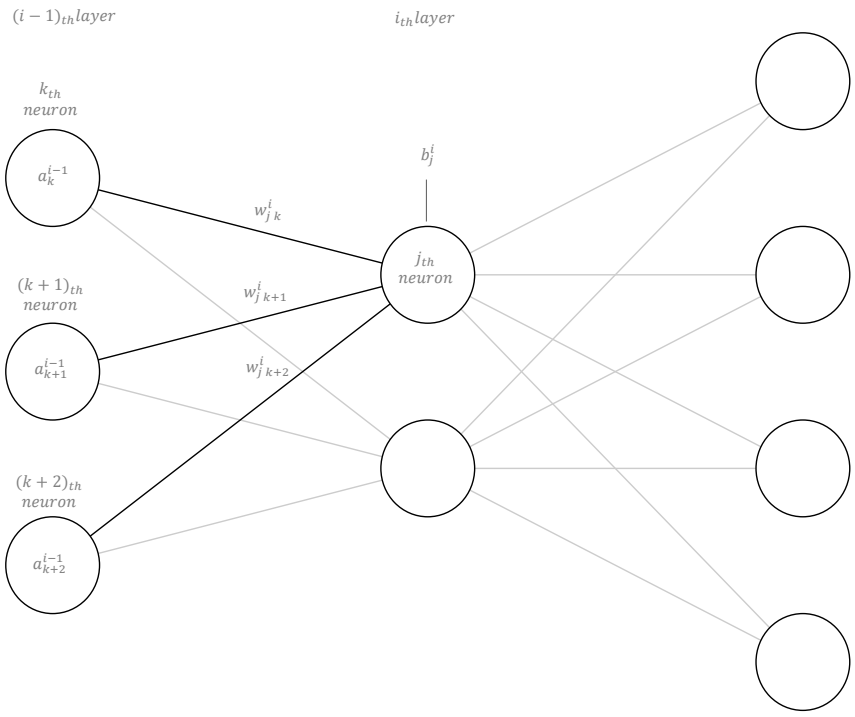

Figure 3.1: Three Layer Neural Network.

3.2.1 Propagation Function

Each neuron i receives multiple input connections, either direct inputs of the problem at hand or outputs of other neurons and aggregates all this stimuli into a single signal using a propagation function:

$$f = f(w_{jk}^i, a_k^{i-1})$$

Although this net signal could be calculated using innumerous functions, the most common propagation function is an inner product between the vector of weights and the vector of neuron outputs:

$$f(w_{jk}^i, a_k^{i-1}) = z_j^i = \sum_{k=0}^{n} \omega_{jk}^i * a_k^{i-1} = \sum_{k=1}^{n} \omega_{jk}^i * b_j^k$$

3.2.2 Activation Function

Biological neurons respond differently according to the received stimuli. If the resultant signal is above a certain threshold, the neuron sends an electrical signal, or a spike, along its axon. In artificial neural networks, the assumption is that the precise timings of the spikes are not relevant, but it is rather their frequency that holds the information [14]. To model this behavior, a function representing the frequency of the spikes along the axon is used - the activation function.

Using the previous notation, the activation function σ is such that:

$$a_j^i = \sigma(z_j^i)$$

Different functions can and have been used as activation functions. The most basic activation function is the binary threshold function or Heaviside function. It is defined as:

$$\sigma(z_j^i) = \begin{cases} 0 & \text{if } z_j^i < 0 \\ 1 & \text{if } z_j^i \geq 0 \end{cases}$$

The function takes only two possible values and is not differentiable at $z_j^i = 0$. As such, learning algorithms as backpropagation cannot be used.

A historically popular choice for the activation function was the sigmoid function, which is a special case of the logistic function and is defined as:

$$\sigma(z_j^i) = \frac{1}{(1 + e^{-z_j^i})}$$

As for the Heaviside function, the sigmoid outputs values between 0 and 1, however unlike the first it is continuous and therefore differentiable. The drawback with this activation function is that it promotes what is known as the vanishing gradient problem if the network uses gradient based methods to learn [20].

Essentially, if a change in a parameter's value causes very small changes in the network's output, then that parameter cannot be learnt effectively. This can be easily seen by analyzing the expression for the update of the weights using backpropagation, which is dependent on the first derivative of the activation function (please refer to the next section on Neural Networks Architecture for further details).

For the case in which the activation function σ is the sigmoid function:

$$\sigma(z_j^i) = \frac{1}{(1 + e^{-z_j^i})} \Rightarrow \sigma'(z_j^i) = \frac{e^{-z_j^i}}{(1 + e^{-z_j^i})^2}$$

For both large and small inputs, the derivative is very close to zero, thus killing the gradient and making the learning process harder. This effect can be observed in figure 3.2. The issue worsens when multiple layers of neurons are used.

Several functions have been found to overcome this issue. In particular, the Rectified Linear Unit (ReLU) has become very popular in the last few years. It is defined as:

$$\sigma(z_j^i) = \max(0, z_j^i)$$

This function was found to greatly accelerate the convergence of stochastic gradient descent compared to the sigmoid functions, allegedly due to its linear, non-saturating form [22].

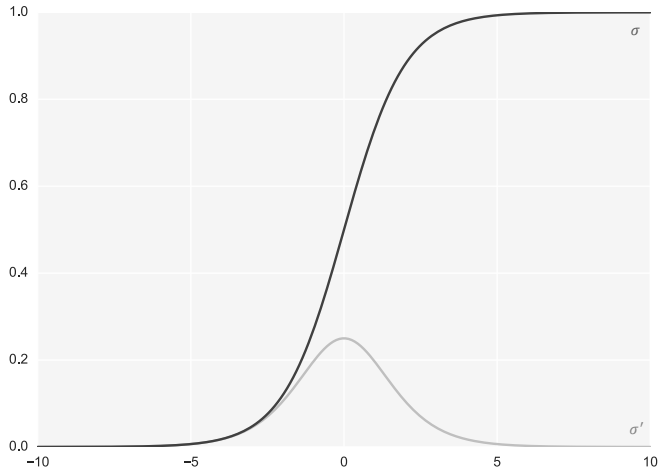

Figure 3.2: The sigmoid activation function and its derivative.

The previous examples of activation functions depended only on the net value of the input for a specific neuron j. However, activation functions might also take into consideration the inputs of the other neurons in the same layer. One case of such functions is the softmax function, also known as the normalized exponential. It is defined as:

$$\sigma(z_j^i) = \frac{e^{z_j^i}}{\sum_k e^{z_k^i}}$$

Although this activation function can be used in any layer of the network, it is usually used in the last. As a consequence of the mathematical definition of this function, the outputs of the network are always positive and sum up to one. This can be easily proven using the properties of summations and the exponential function:

$$e^{z_j^i} > 0 \, \forall z_j^i \in \mathbb{R} \Rightarrow \frac{e^{z_j^i}}{\sum_k e^{z_k^i}} > 0 \, \forall z_j^i \in \mathbb{R}$$

$$\sum_j \sigma(z_j^i) = \sum_j \frac{e^{z_j^i}}{\sum_k e^{z_k^i}} = \frac{\sum_j e^{z_j^i}}{\sum_k e^{z_k^i}} = 1$$

Being always positive and adding up to one, the outputs of a softmax layer can be thought of as belonging to a probability distribution. In many cases this interpretation might be convenient. For instance, in a classification problem the output of a the softmax layer will be the probability of a chosen input to belong to a certain class.

3.2.3 Cost Function

Learning to perform a task usually requires several trials – each successive trial should be an attempt to improve on the results achieved in the past. Learning systems, and in

particular neural networks, follow the same approach: in each iteration, the network gives an output that is compared with the true value. This comparison operation is performed by a cost function, and its result is used to modify the weights of the connections in the network, thus allowing the network to better shape the solution surface. In principle, the more different the real and the expected output are, the larger will be the modifications in the network's weights.

Using the previously defined notation and representing y_j^i as the correct output of the j^{th} neuron in the i^{th} layer of the network, the cost function E is such that:

$$E = E(a_j^i, y_j^i)$$

Multiple cost functions may be used, the simplest and most common of all being the quadratic cost, also known as squared error, maximum likelihood or sum squared error. Let n be the number of training examples. Then the quadratic cost is defined as:

$$E_j^i = \frac{1}{2n} \sum_{x=1}^{n} (a_j^i - y_j^i)^2$$

Although the quadratic cost is a popular choice for the cost function, it has a major disadvantage if the activation function chosen for the network is the sigmoid. Learning is expected to be faster when the difference between the real and the expected output (the error) of the network is larger, however this premise may be compromised if using this cost function, as the update of the connection weights is proportional to the derivative of the activation function [27] (please refer to the next section on Neural Networks Architecture for further details). If this function is the sigmoid, the derivative is very close to zero to either large or small outputs (neuron saturation), as seen before. As such, even if the error is large, the update in the weights will be small:

$$\frac{\partial E_j^i}{\partial \omega_{jk}^i} = (a_j^i - y_j^i)\omega'(z_j^i)a_j^{i-1}$$

$$\frac{\partial E_j^i}{\partial b_j^i} = (a_j^i - y_j^i)\omega'(z_j^i)$$

In order to overcome this issue, a common solution is to use a cross-entropy cost function, which is defined as:

$$E = -\frac{1}{n} \sum_{x=1}^{n} y_j^i \ln(a_j^i) + (1 - y_j^i) \ln(1 - a_j^i)$$

This cost function was developed precisely to fix the learning slowdown problem presented before. Making use of a property of the sigmoid function, whose derivative can be written as a function of the sigmoid itself, it can be easily seen that the derivatives are no longer dependent on $\sigma'(z_j^i)$:

$$\sigma(z_j^i) = \frac{1}{1+e^{-z_j^i}} \Rightarrow \sigma'(z_j^i) = \frac{e^{-z_j^i}}{(1+e^{-z_j^i})^2} = \frac{1}{1+e^{-z_j^i}} \times (1 - \frac{1}{1+e^{-z_j^i}}) = \sigma(z_j^i) \times (1 - \sigma(z_j^i))$$

$$\frac{\partial E_j^i}{\partial \omega_{jk}^i} = \frac{1}{n} \sum_{x=1}^{n} \frac{\sigma'(z_j^i) a_j^{i-1}}{\sigma(z_j^i) \times (1 - \sigma(z_j^i))} (\sigma(z_j^i) - y_j^i) = \frac{1}{n} \sum_{x=1}^{n} (a_j^i - y_j^i) a_j^{i-1}$$

$$\frac{\partial E_j^i}{\partial b_j^i} = \frac{1}{n} \sum_{x=1}^{n} (a_j^i - y_j^i)$$

Therefore, the cross-entropy cost function is the most common choice whenever the activation function is the sigmoid.

3.3 Neural Networks Architectures

Several different network architectures exist, however their basic learning principle is the same: to find the vector of weights $[w_{jk}^i]$ and bias $[b_j^i]$ that minimizes the difference between the actual and the expected output of the network. In this section, the two main existing architectures and their most common learning methodologies are analyzed.

3.3.1 Feedforward Neural Network

In feedforward networks, information flows in only one direction, from the input nodes, through (possible multiple or none) hidden layers of neurons and finally to the output nodes. The simplest example of such architecture is a single layer network.

For this simple example, the delta rule, which is based on gradient descent, is applied as the learning mechanism of the network. According to this rule, the weights of the network should be modified in the following way:

$$w_{jk}^i = w_{jk}^i + \Delta w_{jk}^i$$

$$\Delta w_{jk}^i = -\eta \frac{\partial E}{\partial w_{jk}^i}$$

Here η is an adjustable parameter called learning rate which regulates the magnitude of the update. As this network has no hidden layers, there are only connections between the input and the output layer. As such, the index i may be ommited from the previous expressions. Furthermore, only the neurons from the output layer are responsible for performing computations and their output is also the final output of the network. As such, the associated error is clearly defined – both the output of the neuron and the expected output are known - and its exact form is only dependent on

the chosen cost function E. For example, in the case in which the cost function is the quadratic cost:

$$\frac{\partial E}{\partial \omega_{jk}} = \frac{\partial E}{\partial \sigma}\frac{\partial \sigma}{\partial \omega_{jk}} = (a_j - y_j)\sigma'(z_j)$$

As demonstrated before, the derivative of the sigmoid function can be written as a function of the sigmoid itself. For this case, the previous expression may be written as:

$$\sigma'(z_j^i) = \sigma(z_j^i) \times (1 - \sigma(z_j^i))$$

$$\frac{\partial E}{\partial \omega_{jk}} = (a_j - y_j)\sigma'(z_j) = a_j(1 - a_j)(a_j - y_j)$$

A similar methodology is followed for multilayer feedforward networks. The difference between these and the single layer network analyzed before is the existence of hidden layers of neurons between the input and the output layers. This fact triggers a crucial observation - only the expected outputs of output layer of the network are known, as they correspond to the targets on the training data used. There is no information on what the outputs of the hidden neurons should be. As such, instead of the delta rule, a more general method should be used, the most common of which is backpropagation. Backpropagation may be decomposed in one forward step and one backward step. In the forward step, the weights of the connections remain unchanged and the outputs of the network are calculated propagating the inputs through all the neurons of the network. The backward step consists in the modification of the weights of each connection:

- For the output neurons, the modification is performed as before, by means of the delta rule;
- For the hidden neurons, the modification is done by propagating forwards the error of the output neurons: the error of each hidden neuron is considered as being equal to the sum of all the errors of the neurons of the subsequent layers.

3.3.2 Recurrent Neural Network

In feedforward neural networks, information flows in a single direction - from the input layer directly to the output layer. Recurrent neural networks (RNNs) do not impose such constraints, thus allowing for the existence of loops in the architecture. As such, each neuron in the hidden layer receives not only the inputs from the previous layer in the network, but also its own outputs from the last time step. This structure is shown in figure 3.3. The same architecture can be represented more clearly by unfolding the network along the time axis, as shown in figure 3.4.

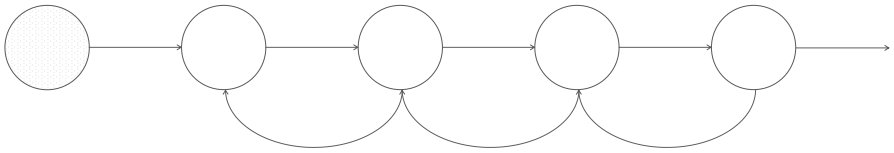

Figure 3.3: Basic Structure of a Recurrent Neural Network.

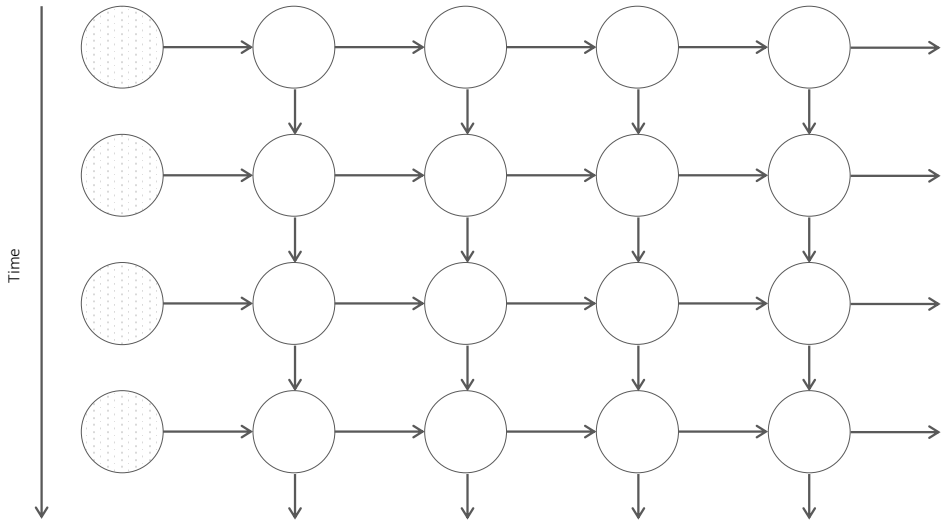

Figure 3.4: Unfolded Recurrent Neural Network along the time axis.

In this representation, each horizontal line of layers is the network running at a single time step. Each hidden layer receives both input from the previous layer and input from itself one time step in the past.

The cyclic nature of recurrent neural networks allows them to maintain context on past events, making them best suited for modeling sequential processes, such as time series, text, and music. Although in theory the above architecture should be able to explore long term dependencies in a chain of events, it has been proven that due to the vanishing gradient problem the memory is actually very short-term [5].

To overcome this issue, the networks neurons can be replaced by a specially designed memory cell – a long short-term memory (LSTM) cell, introduced by Hochreiter and Schmidhuber in 1997 [6].

All recurrent neural networks have the form of a chain of repeating modules, but the structure of these models is different between simple RNNs and RNN + LSTM architectures, as can be understood in figures 3.5 and 3.6. In standard RNNs, the repeating module has a very simple structure, with only one activation function to process all the incoming information simultaneously (the output x_t of the previous neurons and its own output h_{t-1} from the previous time step). In LSTMs, the repeating module has a different structure, as information flows in different paths - part of it is stored in the cell (the state C_t), and another serves as the output (h_t), which is determined by a system of gates which control the information that enters, stays and

leaves the cell, with the help of two functions g and σ and a set of bitwise operators [16].

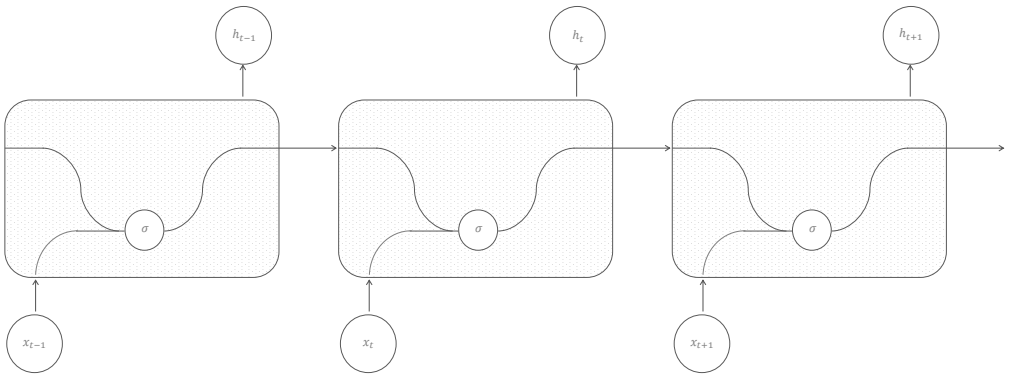

Figure 3.5: Structure of a basic Neural Network cell.

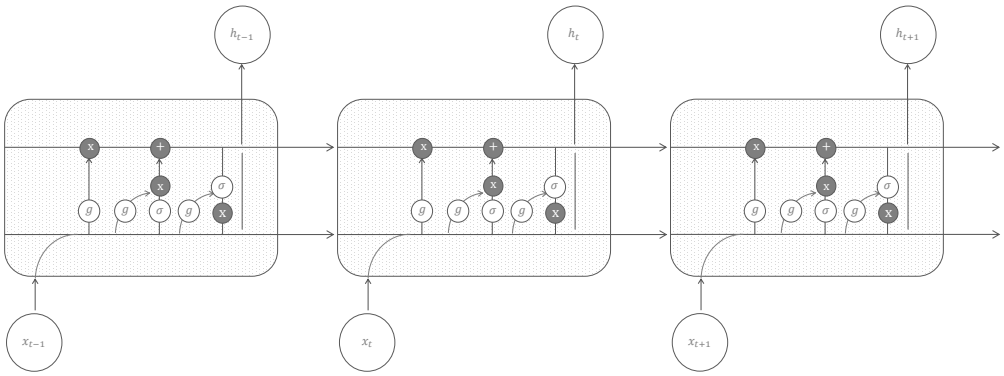

Figure 3.6: Structure of a Long-Short Term Memory cell (LSTM).

The advantage of LSTMs over the simple architecture is precisely the ability to control the information that flows through its gates, allowing the implementation of a memory unit - the cell state C_t, which ensures the network can preserve whatever information from the past that is considered relevant and not lose it after a few time steps. LSTM have three of these gates:

- Forget Gate

  Not all the information received is necessarily useful. This layer looks at the output from the previous time-step, h_{t-1}, and the input x_t and applies a sigmoid g to output a number f_t between 0 and 1 for each number in the cell state C_{t-1}, with 1 represeting "keep" and 0 representing "do not keep the information" (figure 3.7):

$$f_t = g(w_f \cdot [h_{t-1}, x_t] + b_f)$$

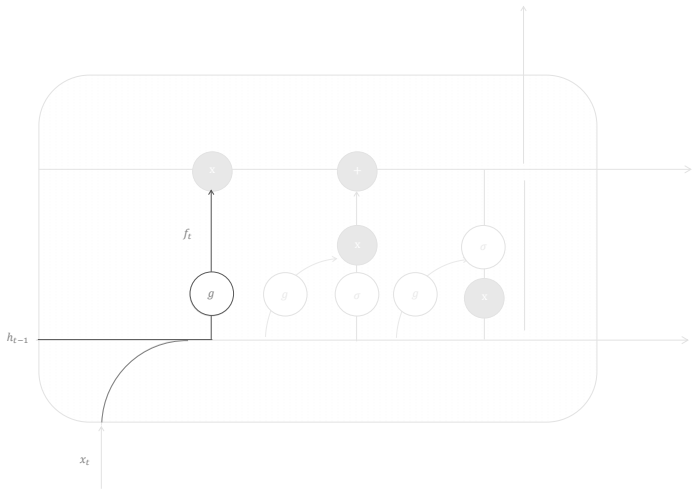

Figure 3.7: Forget Gate of a LSTM cell.

- Input Gate

  The cell state is a unit memory that keeps the information considered relevant. This information can be modified through the input gate. The process is accomplished in two steps: first a sigmoid g is used to select which values are going to be updated (i_t) and then an activation function σ (usually a hyperbolic tangent) is used to generate a vector of candidates C'_t. These two are combined to update the cell state with $i_t \times C'_t$, where:

$$i_t = g(w_i \cdot [h_{t-1}, x_t] + b_i)$$

$$C'_t = \sigma(w_C \cdot [h_{t-1}, x_t] + b_C)$$

  Figure 3.8 represents the operations to calculate i_t and C'_t and to update the current cell state C_t where, as described before:

$$C_t = f_t \times C_{t-1} + i_t \times C'_t$$

- Output Gate

  The information that is considered relevant to keep in the cell's memory is not necessarily the same that is important to output. This fact motivates the existence of the output gate – its purpose it to select a filtered version of the cell state as output. The process is accomplished in two steps: first an activation function σ (usually an hyperbolic tangent) is applied to the cell state and then its result is multiplied by a sigmoid g applied to the cell input (o_t), allowing for only certain values to be selected:

$$o_t = g(w_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \times \sigma(C_t)$$

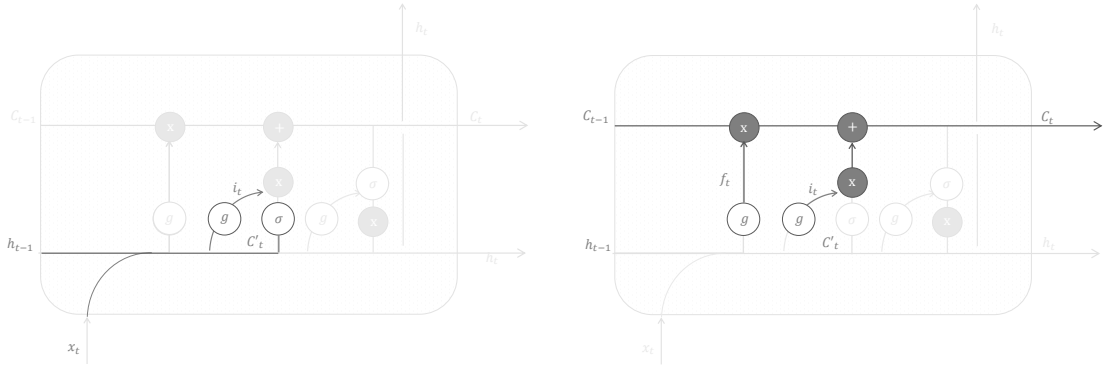

Figure 3.8: Input Gate of a LSTM cell.

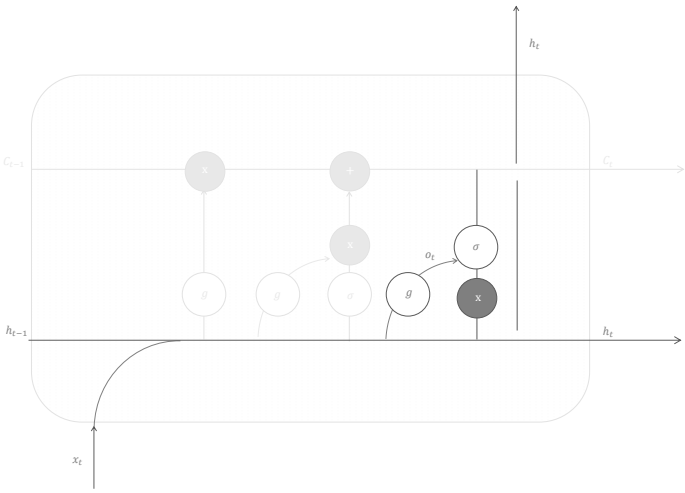

Figure 3.9: Output Gate of a LSTM cell.

An important question may now be answered – why do LSTMs have the ability to avoid the vanishing gradient problem? The LSTM architecture allows disabling of writing to a cell (partially or completely) by controlling the information which goes through the input gate. This prevents too many changes to the cell's contents over the learning cycles, thus preserving information from early parts of the sequences that are considered relevant. Modeling long term dependencies is, as such, not a difficult task for this network architecture.

Recurrent neural networks are trained in a similar way to feedforward neural networks, however the change in architecture requires an adapted version of the backpropagation algorithm called backpropagation through time [25]. Provided the network is fed with finite time steps [8], it can be unfolded along the time axis, as demonstrated before, and the usual backpropagation algorithm can then be applied.

3.4 Methods to Improve Learning Performance

Learning algorithms are applied to data in which some hidden knowledge is expected to be found. Their goal is to learn this knowledge and code it into the model using a subset of the data for which the targets are known.

The ability to generalize well from this subset is essential to a good model, as it determines its ability to make accurate predictions on unseen data. When models are too specialized on the data they were trained with, they fail to comprehend the inherent process behind it and, as such, predictions tend to be off. This issue is commonly known as overfitting and it can be controlled in a variety of ways, some specific to the model used and some applicable to all the data models.

Focusing on neural networks, the most common techniques for dealing with overfitting include regularization, dropout and adaptive learning rates:

- Regularization

  Different techniques exist, the most popular of which is the L2 regularization, also known as "weight decay". The concept behind this technique is to add a term to the old cost function E_0, such that:

  $$E = E_0 + \frac{\lambda}{2n} \sum_{\omega} \omega^2$$

  This term is a sum of the squared weights of the network, scaled by a factor $\frac{\lambda}{2n}$, where λ is known as the regularization parameter. Its purpose is to control the weights of the network, as larger weights may lead to overfitting. In other words, regularization can be viewed as a way of compromising between finding small weights and minimizing the original cost function. The relative importance of the two elements of the compromise depends on the value of the regularization parameter [27].

- Dropout

  Instead of using all the network neurons at each step of the learning process, with dropout part of the hidden neurons of the network are randomly and temporarily deleted (or dropped), leaving the input and output neurons untouched.

The network is then trained over a batch of training examples before a new configuration is selected. With this method, the weights and biases are learnt in different conditions, thus preventing overfitting [27].

The concept behind dropout can be compared to the one used in ensemble models [34]– in these, several models vote together to classify the data instances. As they were trained independently, it is expected that these models may have overfitted the data in different ways and, as such, joining them together may help to overcome their individual weaknesses. The main difference between these techniques is that with dropout the strategy is applied during the training of only one model, instead of after the training with multiple ones.

- Adaptive Learning Rate

  Traditionally the learning rate η is the same for all parameters in the model and also constant in time. However, the simplicity in this formulation may have implications in the learning process.

  First of all, some parameters may be closer to their optimal value than others at a given time. Therefore, being subject to a common learning rate may not be the best approach to their progress – much like with students in a class.

  Secondly, a constant learning rate in time is also not the smartest solution. In the beginning of the training process, a higher learning rate is desirable to speed up the process. However, in later stages of training, lower values for this parameter are preferable because they allow a more thorough search of the solution space in the quest for the optimal solution. Several adaptive learning rate methods have been developed over the years. Particularly interesting are the Adagrad and the RMSprop techniques [30].

  With Adagrad, the learning rate is adjusted for each individual parameter. If a parameter has a low gradient, i.e, if it is close to an optimal value, the learning rate will be barely modified. However, if the gradient is high, Adagrad will shrink the learning rate for that parameter. The downside of this method is that the gradient aggressively and monotonically decreases in each iteration and so, there will be a point during training in which the learning rate is so small that effective learning is no longer possible.

  RMSPRop aims at resolving Adagrad's primary limitation, by considering only a portion of the gradients – the n most recent. The learning rate is therefore updated using the formula:

$$\eta = \frac{\eta}{\sqrt{G}}, with\ G = \gamma \times \sum_{t-n}^{t-1} (\frac{\partial E}{\partial \omega})^2 + (1-\gamma) \times (\frac{\partial E}{\partial \omega})_t^2$$

Here γ is a parameter called the learning rate decay, as it controls the effect of the past gradients on the update of the learning rate.

This formulation prevents the learning rate from decreasing too rapidly and so it allows an effective training.

Chapter 4

Data Representation

While data has an intrinsic value and meaning, it can be communicated through different means and, as a consequence, be understood in different ways. Between humans, the same ideas can be shared with gestures or drawings, be written, spoken and even felt by touch. As such, a critical component of any data modeling task is the choice of the data representation to be used during the processing phase and also, if possible, of the data source itself.

Regarding the sources of data, music has been coded over the years in a variety of ways, with focus on digital audio files, sheet music, abc notation and MIDI files. Each of these possibilities was designed with different end purposes, with their own advantages and disadvantages, which are briefly discussed below.

4.1 Digital Audio Files

When a medium is disturbed, pressure waves are created and propagated from this origin point until they eventually dissipate. A visual example of this phenomenon would be the waves formed on the surface of a liquid when an external object hits its surface. Humans capture these pressure waves through the eardrums, which convert them into a signal that the brain processes as sound. In order to store information on these phenomena, an instrument – usually a microphone – is used to convert the pressure waves into an electric potential with amplitude corresponding to the intensity of the pressure. In this phase, the signal obtained is an analog signal. To convert it to a digital signal used by current audio files, the electrical signal is sampled, by measuring its amplitude at regular intervals, often 44100 times per second. Each measurement is then stored as a number with a fixed precision, often 16 bits.

While digital audio files are rich in the information they contain, the way in which it is stored poses great challenges for data analysis tasks. There is no direct way to know which note is being played at each time and by which instrument. In order to extract information for analysis, processes such as Fourier transforms, beat detection algorithms and frequency spectrograms are usually applied. To extract the necessary

information for a composition or classification task from audio files is therefore not impossible, but rather complex and, as such, it will not be used in this thesis.

4.2 Sheet music and ABC notation

Music has long accompanied the human existence and so has the need to take record of all the pieces composed and played over the millenniums. Since computers are a recent invention in this time frame, visual and symbolic systems for musical representations were used in the past and inherited to the present times. The most common of such systems is sheet music. In this notation, the fundamental latticework is the staff – there are five staff lines and four intervening spaces corresponding to note pitches of the diatonic scale. Different symbols are then displaced in specific positions within those lines to represent the notes played, their duration, the associated rhythm and other musical characteristics.

While highly spread and used, this notation is not simple to learn, in part because it is rich enough to express something as complex as music and in part because it is an old system. More recently, other written forms of musical notation have appeared, as was the case of the ABC notation. In its basic form, this notation uses the letters A through G to represent notes and other elements, always characters or symbols present on a computer keyboard, to add value to the notes – if its sharp or flat, its length, the key or any necessary ornamentation. In a way, ABC notation is a mapping between each character or a group of characters to specific symbols on a sheet music [32].

Because ABC notation is closer to a conventional language system, it is possible to use text mining tools and algorithms to process it. For the specific case of algorithmic composers, it might even be the best option, as it better simulates how humans create music. Furthermore, digital audio files and MIDI files, being continuous, require discretization and a range of possibly restrictive assumptions in order to be used for composition. The downside of using data in the ABC notation for the purpose of music composition is the time and computer resources needed to obtain a satisfactory model, as language modeling is still not an easy task [28].

4.3 MIDI Files

MIDI (Musical Instrument Digital Interface) is a standardized data protocol to exchange musical control data between digital instruments. It is not a digital audio file and does not even contain any sounds, but rather a set of instructions that tell an electronic device how to generate a certain sound. These instructions are coded into MIDI messages [19], for instance:

- Note On: signals the beginning of a note, how hard it was pressed and at which velocity;

- Note Off: signals the end of the note;
- Control Change: indicates that a controller, such as a foot pedal or a fader knob, has been pressed;
- Pitch Wheel Change: signals that the pitch of a note has been bent with the keyboard's pitch wheel.

While this format might not be as rich as the previously mentioned notations, it is certainly easier to extract information from it, especially for the purpose of music composition. The information about the notes being played during the music, when they start and end is all completely determined by a sequence of messages that can be easily extracted using any programming language with a MIDI package. As such, this source format will be the one used in this thesis.

In addition to the choice of a data source format, it is imperative to determine the representation of data to be used in the modeling phase itself. In other words, it should be determined what will be the structure of the data the model will receive and aim to predict. For this purpose, a piano roll representation has been chosen, for it is both easy to process by a neural network and simple to construct from MIDI files. A piano roll shows on the vertical axis the notes as on a piano keyboard and displays time on the horizontal axis. When a note is played, it is represented in the piano roll as a horizontal bar vertically positioned according to its pitch and with a width correspondent to its the duration. This continuous representation can be easily discretized into a two-dimensional matrix, with pitch as the first dimension and time as the second dimension. Each element of the matrix will correspond to a chosen time step and will be either a 1, if the note was played during that time step or 0 if it was not [26]. This will be the representation used to construct the composition and classification models in this thesis.

Chapter 5

Methodology

The goal of this thesis is to construct a novel music classification algorithm, capable of discriminating between different types of music without explicitly analyzing its characteristics nor the target category to which they belong. This chapter explains the design process behind the construction of the algorithm and the details of its implementation, such as the data preparation and the technical steps required to put the algorithm to work.

5.1 High Level Design and Concept

Classification tasks are usually achieved by exploring a set of extracted features or characteristics of the subjects of study. An algorithm is then responsible for finding patterns that relate those characteristics to the known targets, learning the inherent knowledge hidden in the data.

While some subjects of study might be easy to describe – a description of a person's face can easily provide an accurate identification - others reveal unexpected difficulties. Music, for instance, is something present in the everyday life, however when asked to describe a song one would always try to reproduce it instead of describing it. Expressing something as complex as music in words is as difficult as it is simplistic and restrictive. And yet, current music classification systems work on the base of melodic and rhythmic characteristics [23]. This fact motivates an important question – would it be possible to design a classification system more similar to the way humans interpret music? This is precisely the goal of this thesis.

To achieve such an ambitious objective, it is first important to understand or at least grasp some ideas on how humans themselves classify songs, so that this behavior can be reproduced artificially. One could say that, with the information of a song's author or genre, humans know what to expect of a song. Even more, knowing that same genre or author well, humans have the ability to improvise a more or less reliable excerpt of that musical type. These observations are as obvious as they are revealing, for they suggest the knowledge of these different musical types is intrinsically connected with

the ability to compose them.

Artificial composition systems have already been explored over the years. It is relatively clear that, providing different types of music to such systems, they should learn to compose different types of songs. If this assumption holds true, and given the conclusions stated before, then these systems possess the necessary knowledge to classify songs.

The following statements summarize the concept of the classification algorithm proposed in this thesis:

- Given n artificial composition systems, if each is trained using data from a specific type of music, then each should be specialized in this type only.
- If there exists a substantial difference between these musical types, then it is expected that a system specialized in type 1 will be more successful composing songs belonging to this type than any of the remaining $n-1$ systems.
- If a song belonging to one of the n types of music is provided to each of the n composition systems, each will try to compose it and achieve different degrees of success, measurable using a comparison between the predicted song and the real one. A classification algorithm can then be based on this measure – the type of music in which the most successful composition system is specialized will be the one attributed to the song.

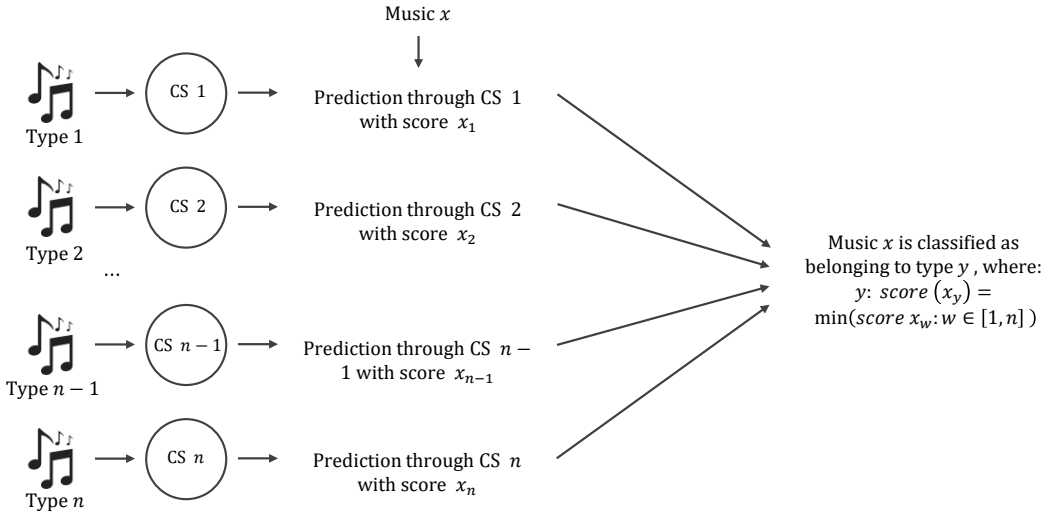

Figure 5.1: Proposed Architecture of the Classification System.

The model described before is the basic classification model to be tested and implemented in this thesis. A more robust solution will be also implemented, by substituting each of the n composition systems, consisting of a single RNN + LSTM architecture,

by a tree of such structures. With this tweak, a voting system can be implemented – several systems are trained with the same type of music and, given a new song, the average of its prediction score in the various systems is taken, instead of the score of a single system. This will possibly correct some existing overfitting and make the model more stable and robust. As a disadvantage, the training is also expected to be more time consuming and, as such, the possible benefits of this option will be explored ahead.

5.2 Data Preparation

Every data exploration and modeling task should require an initial step of data preparation. Data preparation can be defined as the process of collecting, cleaning, and consolidating data in a structured way, making it ready for analysis.

Particularly important in this thesis is the transformation of the data from the MIDI format to a piano-roll or matrix representation. Data quality procedures will also be taken to ensure that the song matrices are a reliable representation of the music that originated it.

5.2.1 Details on the Matrix Representation

Music can be understood as an interplay of melody and harmony. The first is defined as a linear succession of musical notes, while the second refers to the use of simultaneous notes, i.e., chords. In this thesis, this dual interpretation of music is adopted. Each music can then be represented in a piano-roll or matrix format. Let r_m be the number of melody notes and r_c the number of chords played. If the song is divided into t temporal steps, then it can be represented as a matrix of size $(r_m + r_c + 2) \times t$ where each element is either one, if the note/chord was played in that time step, or zero if it was not [31]. The plus 2 term exists to account for the cases in which no chord or melody note is played.

Given this representation, what does the algorithm aim to learn? A composer has the knowledge to play the next note based on the past context, such as a writer knows what the next word in a sentence should be to give it meaning. Learning how to write or compose is to develop the ability to know what comes next or, in algorithmic terms, to learn to predict at each step t the vector describing the notes and chords played at $t + 1$.

This formulation would imply that the learning targets would be vectors of size $(r_m + r_c + 2)$, where each element of the vector could be either one or zero. However, there is one issue with such formulation – its complexity.

There are two possible states for each note or chord considered at each time step, which means there are $2^{(r_m + r_c + 2)}$ possible configurations for the target vector, making the search space sparse and difficult to explore. In order to overcome this situation,

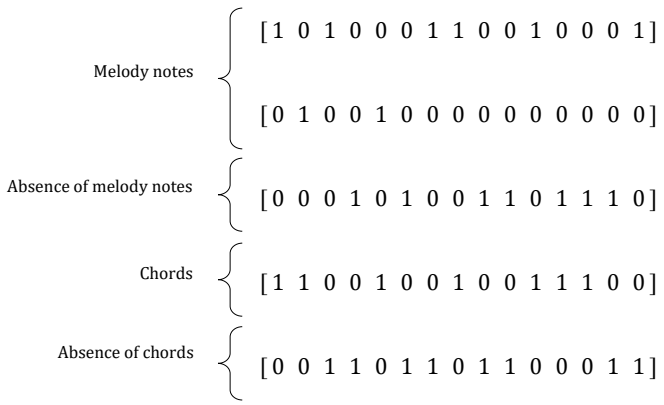

Figure 5.2: Sample Matrix with 15 time steps, 2 melody notes and 1 chord.

a constraint is added to the previous formulation: in each time step, a maximum of one melody note and one chord are allowed. With this rule, the number of possible configurations becomes $(r_m + 1) \times (r_c + 1)$.

Considering a practical example, a lower estimate of the number of melody notes and chords in a dataset would be $r_m \approx 40$ and $r_c \approx 10$, respectively. This setting would give $2^{(40+10+2)} \approx 4.5 \times 10^{15}$ possible states per each time step for the naïve formulation against $(40+1) \times (10+1) \approx 4.5 \times 10^2$ for the more restrictive formulation – a difference of 13 orders of magnitude. Naturally, the first formulation is richer and a closer representation of the original music, but given the computational power of the machines used in this thesis, the second formulation is clearly more appropriate.

5.2.2 Dataset

Before explaining the technical steps needed to construct the binary matrix representation, it is important to understand what data is going to be used and how it is structured.

In theory, any collection of MIDI files may be translated into the previously described representation. However, the data preparation involved could be expensive, as MIDI files may contain information from various instruments. Usually, each instrument is coded in a specific channel and there is no indication what instruments are responsible for the melody and which are assigned to the harmony.

To overcome this situation, and as the data preparation is not the primary goal of this thesis, the Nottingham dataset was chosen, as the majority of its songs have already been coded in only two channels: one dedicated to the melody notes and the other dedicated to the chords. This dataset is a collection of 1037 British and American folk tunes, containing different categories of songs (jigs, christmas songs, etc.).

Each song in the dataset is subject to a set of procedures, which are performed automatically through a script. The Python language was chosen not only for this section, but for this thesis as a whole, for it is intuitive and simple, but mainly because it is an open source language with excellent packages in data science and also music.

5.2.3 MIDI file to matrix representation

Once the dataset has been chosen, the MIDI files can be analyzed one by one in order to construct a suitable matrix representation. However, it is imperative to fully understand the information contained in such files so that the process can be as accurate as possible, ensuring minimum loss or disruption of information.

MIDI files are organized in tracks, which are a collection of messages containing relevant data. Each track is normally assigned to a specific instrument, but this is not mandatory. In the case of the Nottingham dataset, the majority of songs have 3 tracks – one track for the metadata, another for the melody and the final one dedicated to the harmony, which is precisely the structure needed to construct the matrix representation presented before. As such, all the songs that do not respect this structure have not been considered.

Each MIDI track is a collection of messages, the most important of each are the NoteOn, NoteOff and EndOfTrack, which designate the beginning of a note, the end of a note and the end of a track, respectively. Below is an example of a sequence of such messages, extracted with the help of the Python "MIDI" package, which provides functions to open the files and easily extract information from the tracks:

```
midi.NoteOnEvent(tick=1, channel=0, data=[78, 105])
midi.NoteOffEvent(tick=479, channel=0, data=[78, 0])
midi.NoteOnEvent(tick=1, channel=0, data=[76, 80])
midi.NoteOffEvent(tick=239, channel=0, data=[76, 0])
midi.EndOfTrackEvent(tick=26, data=[])
```

As observed in the example above, each message contains a set of information. Starting from the end of the message, the array "data" contains two elements, the first of which is the pitch of the note being played (in MIDI numbers) and the second being the velocity with which the note has been played (in case of the piano, it would correspond to how hard a key was pressed). The next element attributes a "channel" to the message (a different instrument, for example). Finally, the first element of the message is the "tick" value. Ticks represent the lowest level of resolution of a MIDI track, which is precisely defined has:

$$Resolution = \frac{Number\ of\ Ticks}{Beats}$$

Ticks can be interpreted as a measure of time. For instance, if the tempo of a song is 120 beats per minute, then 1 beat corresponds to $60 \times \frac{10^6}{120} = 500000$ microseconds. If the resolution is 10^3, then 10^3 ticks correspond to one beat and so each tick has a duration of 500 microseconds.

An important observation can be made on the sample messages shown before – tick values are not absolute, in the sense that their reference point is not a single, unique point in time (the beginning of the song, for instance). Instead, they refer to the previous related message. Considering the example shown before, it is possible to observe that the 3rd message has a tick value of 1, although its previous message had a tick value of 479. This is because the tick value of the 3rd message refers to the last event (the 2nd message) and not to the beginning of the song. In order to convert the tick values to this fixed reference frame, its cumulative sum should be considered. Following the example, the 3rd message would then refer to $1 + 479 + 1 = 481$ ticks from the start of the music.

Considering the interpretation of ticks as a measure of time, both the resolution and the tempo of the songs should be fixed over the dataset so that a tick represents the same amount of time through the songs. In fact, almost all the songs in the Nottingham dataset already have a resolution of 480 and a tempo of 120 beats per minute and so the remaining ones are discarded. Another data quality test can be made by checking if for every NoteOnEvent there exists the correspondent NoteOffEvent and vice versa. Although rare, these events can happen and, in that case, the associated messages are simply not considered.

After all the previously described preparation steps and quality checks have been made, the matrix representation can be built. First the minimum and maximum of all the note pitches p in the dataset is determined, so that all the matrices have the same number of lines. This number (r) corresponds to the range of melody (r_m) and harmony (r_h) notes being played +2, to include the cases where no melody or/and harmony note is being played:

$$r = r_m + r_h + 2$$

A matrix of size $[r, t]$, where t is the number of time steps of each song, is then initialized with all the elements equal to zero. Each time step Δt (a column in the matrix) corresponds to 120 ticks, which given the resolution and tempo of the songs in the dataset corresponds to 0.125 seconds. The melody and harmony notes are then attributed to one of those steps, following the logic explained below.

Let t_i and t_f mark the beginning and the end of a note, respectively. There are 2 possible cases for the position of the notes relatively to each time step t:

- The note is within the boundaries of the time step t. In this case, the interval $t_f - t_i$, during which the note is played, is taken into consideration.

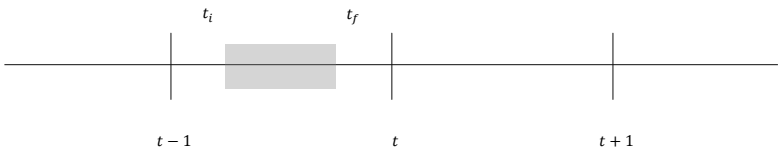

Figure 5.3: Note played within the limits of the defined time step.

- The note starts being played in one time step and continues in the following step(s). In this situation, the interval $t_f - (t-1)$ is considered for the case in which the note starts being played at the time step $(t-1)$ and ends in time step t and the interval $t - t_i$ is considered for the case in which the note starts being played in time step t and ends in the following time step.

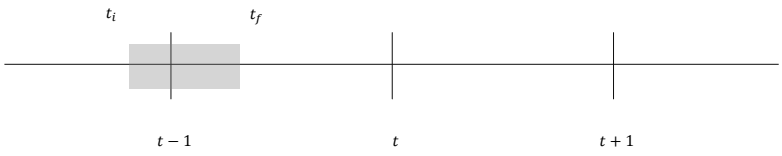

Figure 5.4: Note played during the previous time step and also at the beggining of the current time step.

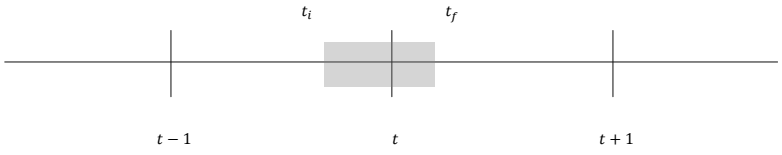

Figure 5.5: Note played at the end of the current time step and also in the following time step.

At this stage, and as only one melody note is allowed per time step, the chosen note for the time step t is the one with the longest time interval in that step. As for the harmony notes, several notes are allowed at each step, which are then analyzed through the Python "Mingus" package to get the associated chords. The part of the matrix correspondent to the harmony notes is then substituted by a matrix of chords with a number of lines correspondent to the range of chords in the dataset (r_c) plus 1 for the case in which no chord is played. The final matrix for each song has, therefore, a size of $[r_m + r_c + 2, t]$, where the elements are either 1 or 0, representing whether a specific chord or melody note was played at each time step.

5.2.4 Data Quality

Generally, data encapsulates hidden knowledge that exploration and mining processes may help to reveal. But, much like in real mining activities, gold bars are not find lying in riverbanks. Rather, tiny pieces of this precious metal are scattered along with many other non-valuable residues. A careful separation process is needed to get to what really matters and the same logic occurs in data science processes through data quality procedures. Not every piece of data is necessarily valuable and it is important to remove this noise in order to get clearer and more objective results.

At this stage, several data quality steps previous to the matrix construction have already been applied and described. The following is a summary of such procedures:

- Only MIDI files with 3 tracks (the first containing metadata, the second the melody notes and the third dedicated to the harmony) are considered. MIDI files with higher number of tracks likely follow a different structure, which can be difficult to translate. Following this rule, 15 files are removed from the initial 1037, leaving 1022 left.

- Resolution (480) and tempo (120 beats per minute) are fixed parameters through the dataset, otherwise the tick interpretation as a measure of time would be different for different songs. Only one MIDI file out of the 1022 has different values for these and so it is discarded, leaving 1021 left.

- For each track in each MIDI file, the events "NoteOnEvent" and "NoteOffEvent" should exist in pairs. If this is not the case, there was likely an error in the creation of the associated MIDI messages and so these events are not considered.

The described procedures were applied before the transformation of the MIDI files into binary matrices. Given the importance and complexity of this transformation, it is imperative to check the data in this new structure. In particular, the transformation from harmony notes to the associated chords can be difficult. In the cases where it is not possible to determine the chord from the associated harmony notes, the "no chord" element of the matrix is populated. Although this can be a simple and unharmful decision for a few time steps, if it happens too often the matrix will not be a reliable representation of the original song.

To overcome this issue, a final data quality check is performed to each matrix. Only 38 songs from a total of 1021 had some chords that could not be recognized. Of these 38, 12 had more than 2% of its chords unidentified and so they will be discarded.

After all the quality procedures have been complete, 28 songs have been discarded, which corresponds approximately to a percentage of 2.7%. The 1009 remaining songs will all be used in the classification algorithm, whose technical details will be explained in the following section.

5.3 Technical Details on the Algorithm Architecture

5.3.1 Basic Structure

The classification algorithm proposed on this thesis is best understood in two phases: in the first phase, n composition systems are trained, one for each of the n different types of music considered. In the second phase, the scores of these composition systems on a specific song are used for its classification.

The complexity of the process clearly resides in the first step – in fact, the last step can be interpreted as a simple voting system, which has already been thoroughly explained. As such, it is important to clarify how the composition systems work.

Each of these systems consists of a recurrent neural network with an input layer of size $r_m + r_c + 2$ (number of melody notes + melody chords +2 to encompass the possibility of a silent note or chord), which corresponds to the possible notes and chords being played and hence to the rows of the matrix representation of the songs.

Regarding the hidden layers of the network, their size and number is to be empirically determined in the next chapter, in which it is discussed how the parameters can be set appropriately. As for their functioning, the hidden layers will receive inputs not only from the preceding layers, but also a recurrent input from itself, referent to past time steps. The number of time steps to be considered is also an adjustable parameter – it should not be too small otherwise it will not allow enough past context. In other words, the number of time steps determines the quantity of information from the past which will be used for prediction. For instance, considering only the previous word in a sentence would surely not be enough to predict the next, because it does not provide enough past context. On the other hand, this parameter should also not be so high as it may introduce unnecessary noise and, specially, to enable computationally acceptable time resources.

Finally, the output layer of the network has the exact same dimensions as the input layer – one cell per possible melody note or chord, which should output a number between 0 and 1 representing its probability of being played in a specific time step.

5.3.2 Internal Elements

Once the external structure of the network has been clarified, it is important to focus the attention on the internal elements. As seen before, neural networks depend on a set of functions, in particular one or more activation functions, a cost function and a propagation function.

The propagation function used is the usual inner product between the vector of weights and the vector of neuron outputs, for there is no evidence that a different function could enhance the results:

$$f(w^i_{jk}, a^{i-1}_k) = z^i_j = \sum_{k=0}^{n} \omega^i_{jk} \times a^{i-1}_k = \sum_{k=1}^{n} \omega^i_{jk} \times a^{i-1}_k + b^i_j$$

Regarding the activation function, two are used in the composition system:

- In the hidden layers, the activation function used is the sigmoid. Although this activation function alone is not currently the most powerful, the associated issues (the vanishing gradient problem) can be overcome using methods such as dropout. The dropout probability will be another parameter in the algorithm, thus enabling a more flexible model.
- In the output layer of the network, the activation function used is the softmax function or normalized exponential. As mentioned in earlier chapters, this function is dependent not only on one neuron, but on all the neurons of a layer. For each neuron, the softmax function outputs a number between 0 and 1. Moreover, its sum across all the outputs of the layer is always 1, thus enabling the interpretation of the result as a probability distribution. In this thesis, two softmax functions will be used in the final layer, one to deal with the melody notes and one to deal with the chords. As only one note and/or chord is permitted per time step, the chosen notes and chords will be the ones with higher probability of being played, at each time-step.

Finally, the cost function used will be cross-entropy cost function, as it is better than the usual sum of squares given a sigmoid activation function [27]. More importantly, this function will also be used to assess the scores of each composition system on the songs that will later be classified - the voting system used for classification is in fact based on this score, as it is by definition a measure of similarity between the original and the predicted songs. Once again, the winning composition system will be the one with the lower score, i.e., the one which was able to compose a song closer to the original. From this result, the song will be classified according to the type of music in which the winning composition system is specialized in.

5.3.3 Algorithm Parameters

This section aims at summarizing the parameters involved in the algorithm. In the next chapter, these will be explored further, to find the values that optimize the outcomes of the system.

Following is a list of such parameters, their meaning and their expected contribution:

- Number and size of the hidden layers

  The number of hidden neurons per layer and the number of layers itself determine the complexity of the solution that the algorithm can generate. A higher

value for these parameters allows more freedom and complexity, which is desirable. However, too high values may promote overfitting and too low values may lead the algorithm to underfit – in the first case, the solution found may be too adapted to the data with which the algorithm was trained with, leading to a bad generalization ability on unseen data, while in the second case the solution found may be too simplistic.

- Learning rate and learning rate decay of the RNNs

  This is a common parameter to neural networks and it controls the magnitude of the changes to the network's weights in each iteration, and thus its learning speed. A high learning rate should allow for a fast learning, at least at an initial stage, however later it might prevent it from achieving optimal results. On the other hand, if the learning rate is too slow, the algorithm will likely take a long time to converge.

  Instead of having a constant learning rate, a possibility is to decrease it over the iterations. This is a smarter solution, as the learning rate can be larger in the beginning of the training, allowing for fast convergence, and then decrease slowly, enabling the algorithm to explore the solution space more thoroughly. Several methods can be used to determine the decay. In this thesis, the RMSprop method is adopted, which is the most common method used in recurrent neural networks.

- Dropout

  Dropout can be applied to neural networks in order to prevent overfitting, at least to a certain degree. As mentioned in earlier sections, with dropout part of the hidden neurons of the network are randomly and temporarily deleted (or dropped), leaving the input and output neurons untouched. As such, the dropout value corresponds to the percentage of these dropped neurons – it should not be so low as to produce some effects, and not too high as to prevent the network from learning efficiently.

- Number of past time-steps to be considered

  In time series problems, there is always a window into the past that is considered in order to predict a certain value. In this algorithm, this window corresponds to the number of past time steps to be considered in order to predict the notes played on the current time step. As mentioned before, this value should be large enough to have sufficient past context. Also, it should not be too large, otherwise training will be extremely time consuming.

- Melody and Harmony coefficients

  In this thesis, not only melody notes are predicted, but also chords. While it is possible to give an equal importance to the prediction of one or the other, it

is possible to manage this importance by multiplying the cost function outputs of the melody by a factor of α and the cost function outputs of the chords by a factor of $(1-\alpha)$, with $\alpha \in [0,1]$. The cost function would then be re-written as:

$$E = \alpha \times E_{melody} + (1-\alpha) \times E_{chords}$$

A factor $\alpha = 0$ would mean that only the prediction of the chords would be considered when training the network and, in the other end of the spectrum, a factor of $\alpha = 1$ would mean that only the melody notes were being considered. Given that the chords are usually more predictable and constant than the melody notes, an $\alpha > 0.5$ should yield results musically more accurate and pleasable. However, results with changing alphas cannot be compared by means of the outputs of the cost function because the cost function itself is being changed. For this reason, these coefficients will be explored ahead, but no search for a best value will be performed.

- Number of voting compositions systems

  For n different types of musics there will be $\beta \times n$ composition systems, where $\beta \geq 1$. In other words, for each type of music there will be β composition systems specialized only in this type of music, that will vote in the classification phase. This structure is inspired on ensemble methods and aims to be a more robust solution against overfitting. In theory, the larger the parameter β, the better. However, given than an increase of this parameter implies a linear increase on the time resources needed to train the algorithm, only very small values for β will be tested.

All the technical implementation of the algorithm described before has been done using the Python language. The packages Numpy, for dealing with vector and matrix calculations efficiently, and Tensor Flow, for the implementation of the recurrent neural networks, were particularly relevant and useful.

Chapter 6

Results

This chapter is subdivided in two major sections: in the first one, the parameters defined for the composition algorithm will be explored in the search for the values that optimize its results. This is a critical step to ensure the success of the classification phase – if the composition systems are unable to learn effectively, they will not possess enough knowledge to be of any aid in the classification task.

In the second part, the results of various binary classification experiments will be presented and discussed.

6.1 Composition System

The concept of the classification system proposed in this thesis assumes that a successful composition system possesses the necessary musical knowledge to classify songs. If no such knowledge is present, then the theoretical base for the classification task is lost. As such, it is fundamental to optimize the parameters of the composition system and ensure the best possible results.

To successfully tune the parameters, some portion of the data should be used for training and another should be set aside until the model is finished, to unbiasedly estimate the results on unseen data. What has just been described is a train/test split of the data. As the composition and classification tasks are not done simultaneously, but the one after the other, the following splits will be applied:

- For the classification, the train set will consist of 70% of the total number of songs and the test split will consist of the remaining 30%, which equates to 706 songs and 303 songs, respectively;

- For the composition system, only the data from the train set of the classification task will be used, which makes up to 70% of the original data. From this percentage, 70% will be used for training purposes and 30% will be used for testing, which equates to 494 songs and 212 songs, respectively.

As described in the last chapter, there are 6 configurable parameters for the composition algorithm: number of layers of the network, number of hidden neurons, dropout probability, learning rate, learning rate decay and number of past time steps to be considered. To select the values that optimize the results of the algorithm, it is a common procedure to test each parameter individually. In the end, the chosen configuration is based on the best value found for each parameter. The issue with this approach is that this configuration is not necessarily optimal – although the best value of each parameter was selected, this value was found with experiments where the remaining parameters were fixed. As such, it is only possible to admit that such "best value" is valid for the conditions in which it was tested and not necessarily for all the possible configurations.

The most obvious way to overcome this situation is to perform grid search. This method consists in setting a grid in a hyperspace of configurations, where each node corresponds to a possible algorithm configuration that is to be tested. In the case when there are only 2 parameters, this can be easily visualized as a two-dimensional grid. The issue with this method is that it is resource intensive: considering x testing values for each of the n parameters, the grid will have a total of x^n nodes or possible configurations, which is clearly prohibitive.

Given the limitations of the described methodologies, in this thesis the following approach is followed:

- First, a large range of values for each parameter is tested, leaving the rest fixed. Rather from selecting one "best value", a small range of optimal values is selected for each of the parameters;
- Secondly, grid search is performed in a limited subspace of configurations, defined by the ranges of values encountered in the previous step.

Although this still does not guarantee that an optimal configuration is found, it provides a good strategy in terms of time resources. With this methodology, the following set of parameters was chosen:

- Number of the hidden layers: 2;
- Size of the hidden layers: 250;
- Learning rate: 0.005;
- Learning rate decay: 0.8;
- Dropout: 0.9 in the first layer and 0.5 in the remaining layers;
- Number of past time-steps to be considered: 128, which corresponds to 16 seconds;

There is still one parameter left to analyze before proceeding to the classification results – the melody coefficient. Choosing the weights of the melody and harmony prediction will affect the way the composition system learns, however as changing this coefficient will also change the cost function, the results between experiments are not comparable using this measure. Nonetheless, it is still possible to assess the results of the predictions using a different measure – the overall accuracy of the predictions. To calculate this, the final melody and harmony notes predicted are compared one by one to the expected outcomes:

$$prediction\ accuracy = \frac{number\ of\ notes\ successfully\ predicted}{number\ of\ notes}$$

The following graphics show the evolution of the melody and harmony accuracy prediction for a changing melody coefficient:

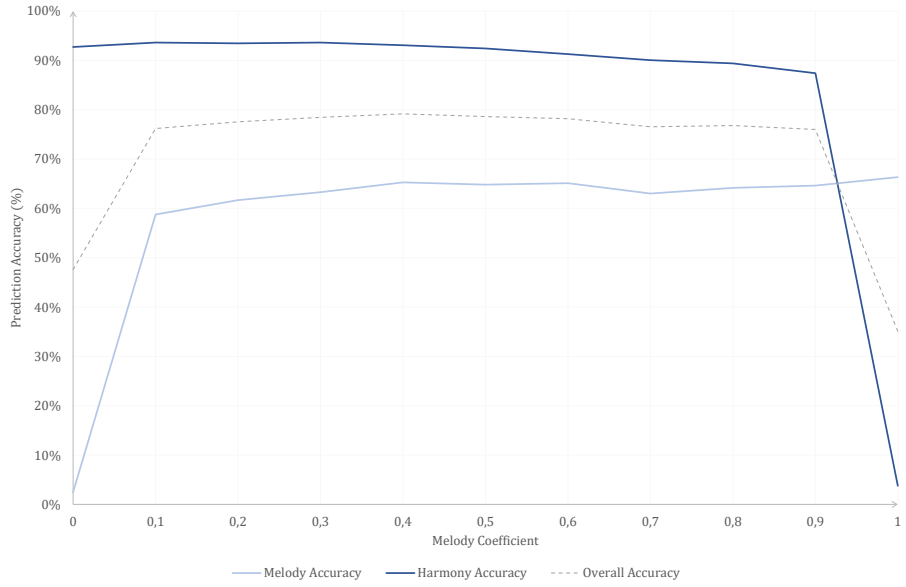

Figure 6.1: Accuracy of the notes and chords prediction for a varying melody coefficient.

The melody accuracy increases when the coefficient is closer to 1 and decreases when approaching 0, while the harmony accuracy has precisely the opposite behavior. This behavior is expected, because the coefficient regulates the importance given to the learning of the melody and the harmony – the higher the coefficient, the more the melody will be learnt and the less importance the harmony values will have and vice-versa. In the extreme cases where the coefficient is maximum or minimum, one of the components of the song will have completely random results, because their value was ignored, while the other will have the best results achieved with the experiment.

What is truly interesting about this analysis is the different accuracy values between melody and harmony. For instance, when the coefficient is 0.5 and so the weights of the components are equally distributed, the accuracy of the harmony (92%) is clearly

superior to the accuracy of the melody (65%). Also, the best melody accuracy achieved is clearly inferior to the best harmony accuracy. The answer for this observation can be stated empirically – it is intuitively known that chords are generally more stable throughout and amongst the songs, which make them more predictable. This can be verified by analyzing the distribution of the different chords and melody notes present in the dataset:

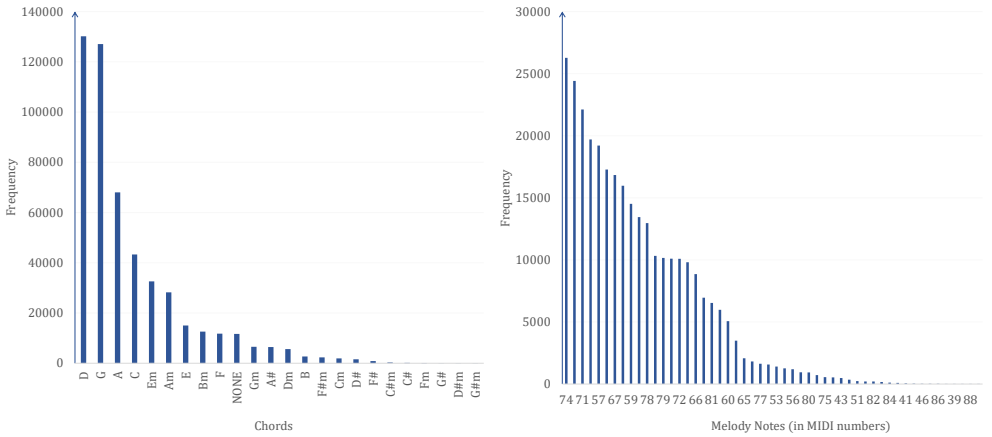

Figure 6.2: Distribution of the chords and melody notes in the Nottingham dataset.

The number of chords used is clearly inferior to the number of melody notes and its distribution is more skewed, which justifies the ability of the algorithm to better predict chords in relation to the melody notes.

Considering that the best overall accuracy has been achieved with a melody coefficient around 0.5, this parameter will be set to this value for the rest of the experiments. With this setting, the overall test accuracy achieved is around 89% (65% for the melody notes and 92% for the chords).

One last question to consider at this point is what are the criteria to define that a certain composition system is successful or not. Of course, the more the cost function is minimized, the better. It is also possible to use other measures, such as the prediction accuracy, to better understand the level of success of the composition system. What is not clear is what results represent a good, acceptable or bad solution. In other words, at what stage has the algorithm learnt the basic musical features so that its songs can be enjoyable? Is a predicted song with an accuracy of 80% pleasant? What about one with 60% or 70%?

In fact, this is a subjective question. Without using pre-defined rules, which can be quite difficult to formulate and are usually restrictive, it is difficult to judge the quality of a song without listening to it. As such, the best one can do is to evaluate the songs composed by the algorithm after it has been trained.

Doing so, reveals that songs with accuracies higher than 70% start to resemble musical pieces composed by humans; although its musical structure is simple, they are pleasant to hear.

6.2 Classification System

The Nottingham dataset is a collection of 1037 British and American folk tunes, containing different categories of songs. For instance, the "reels" category includes marches and polkas and the "hpps" category includes hornpipes, schottisches and strathspeys, all of which are traditional songs used in marches, dances and popular celebrations.

After the data quality steps performed in chapter 5, of the 1037 initial songs, only 1009 were selected to be used in the experiments of this thesis. The following graphic shows the distribution of these songs amongst the categories present in the Nottingham dataset:

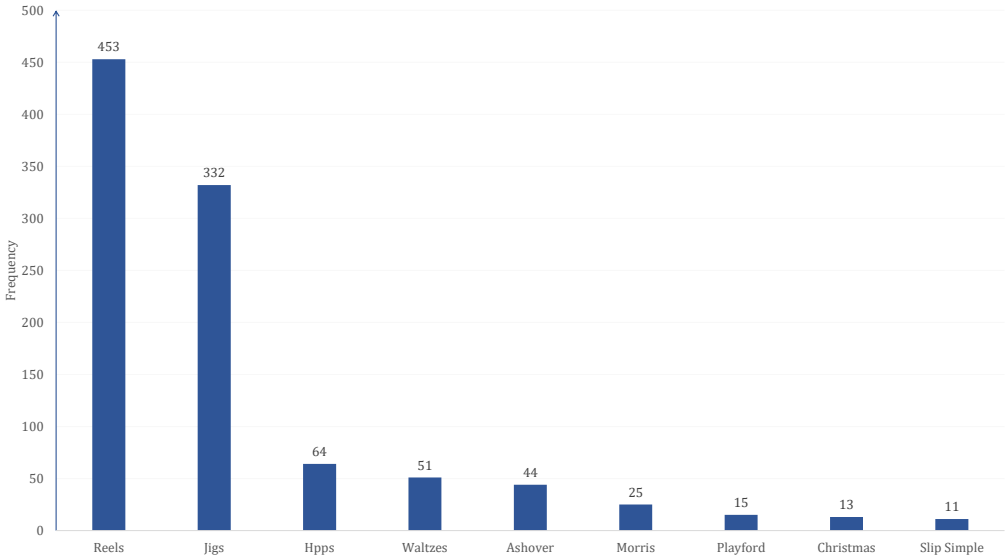

Figure 6.3: Distribution of songs per class in the Nottingham dataset.

For classification purposes, only the categories with a large number of songs should be considered. If the number of songs is too small, then there will be not enough to properly train and test the composition and classification systems. Although the "Reels" and the "Jigs" categories have a considerable number of songs, the rest of the dataset is scattered amongst smaller categories.

As a first experiment, a binary classification between the "hpps" and the "waltzes" category will be performed to assess the results of the classification algorithm given two categories with a relatively small representation. Given that the number of train songs should be the same for both categories for the results to be comparable, the following split will be made:

- As the number of songs in the "hpps" (64) category is larger than the number of songs in "waltzes" category, the excess (13) will be used only for assessing the results of the classification system.

- For each category, the splitting of the data will be made following the percentages stated before: 70% of the total 51 songs will be used for the composition system, which yields a total of 36 songs. Of these 36, 70% will be used for training (25) and the remaining 30% will be used for testing (11).
- This implies that 30% of the original 51 songs will be used for testing the classification system (15) plus 13 songs for the case of the "hpps" category, as explained before.

With these settings, and using the parameters defined before, the following results have been obtained:

Class 1 vs Class 2	Accuracy Class 1	Accuracy Class 2	Overall Accuracy
Hpps vs Waltzes	85.7%	100%	90.7%

Table 6.1: Results obtained for the binary classification between the categories "Hpps" and "Waltzes".

The results are surprising. Although the number of songs is small, the results of the classification system clearly demonstrate the ability two distinguish between the two classes – all the "waltzes" songs and 85.7% of the "hpps" songs have been correctly classified. Other attempts to this same classification task have revealed consistent results, varying only by less than 5%.

To ensure these results are solid and demonstrate the efficiency of the algorithm, it is imperative to perform other classifications experiments. As before, the same rules for the splitting of the data apply: between the two categories of songs being tested, the one with fewer songs is used to determine the size of the training and validating sets for the composition systems (70% of the total number of songs of that category), while the remaining are used for testing the classification. The following table shows the results obtained for another 4 experiments:

Class 1 vs Class 2	Accuracy Class 1	Accuracy Class 2	Overall Accuracy
Jigs vs Waltzes	84.1%	66.7%	83.3%
Reels vs Waltzes	67.4%	80.0%	67.9%
Jigs vs Hpps	89.9%	85.0%	89.6%
Reels vs Hpps	84.1%	75.0%	83.7%

Table 6.2: Results obtained for the binary classification experiments between the categories "Jigs", "Reels", "Hpps" and "Waltzes".

As before, the results demonstrate a considerable discrimination power between the classes – for 3 of the 4 experiments, the overall accuracy is higher than 80% and the least successful classification has, nonetheless, an accuracy of 67.9%. What is more interesting is that these results have been obtained using few songs for the training of

the composition systems: the experiments involving the “waltzes” category used only 25 songs and the experiments involving the “hpps” category used 31 songs. This factor might help to explain the least impressive result, obtained for the experiment “reels” vs “waltzes”: the number of songs provided may not have been enough to enable a clear training of the composition systems and, as such, the results of the classifications where not as expected. Another option would be that, in fact, the two categories share significant musical similarities among themselves, making the classification task harder.

A final experiment between the two classes with greater representation in the dataset is highly important to assess if the size of the training data may have a positive impact on the classification results – this is expected because, if the composition systems are better specialized in their respective category, they will more likely be successful in composing a music belonging to it. The following table shows the results obtained for such experiment:

Class 1 vs Class 2	Accuracy Class 1	Accuracy Class 2	Overall Accuracy
Reels vs Jigs	95.3%	92.9%	94.6%

Table 6.3: Results obtained for the binary classification experiments between the categories ”Reels” and ”Jigs”.

Not only the overall accuracy in this experiment is the highest obtained, it is also the most consistent given the accuracies obtained for each class (only 2.4% different), which support the interpretation that a higher number of training songs increases the discrimination power of the algorithm and thus leads to a more successful classification.

In the preceding experiments, only one composition system per music class was used. However, it is also possible to design an architecture with n composition systems per class, which then vote amongst themselves to determine the songs class. For instance, if $n = 3$ in a binary classification task between classes A and B, there would be 3 composition systems specialized on A and another 3 specialized on B. A test song would then be evaluated by these 6 systems and the top 3 results selected. The most represented class in this top 3 would then be selected as the class attributed to the test song.

With this methodology – which is basically a voting or ensemble system - results are expected to be more consistent and reliable, for they depend less on the training of a specific system. However, this will also increase the training time of the composition systems by a factor of $2 \times n$ for a binary classification task or, more generally, by a factor $c \times n$, where c is the number of classes. This is a clear disadvantage, especially for the case when the training of a single system is already very time consuming, as is the case.

For this experiment, a voting system with $n = 5$ was considered for the same binary classification tasks as in the previous experiment. The results are exposed below:

Class 1 vs Class 2	1st Exp. Accuracy	2nd Exp. Accuracy	Accuracy Difference
Hpps vs Waltzes	90.7%	95.3%	4.6%
Jigs vs Waltzes	83.3%	85.9%	2.6%
Reels vs Waltzes	67.9%	69.7%	1.8%
Jigs vs Hpps	89.6%	91.1%	1.5%
Reeks vs Hpps	83.7%	88.5%	4.8%
Reels vs Jigs	94.6%	95.8%	1.2%

Table 6.4: Results obtained for the binary classification experiments using a voting system between the categories ”Jigs”, ”Reels”, ”Hpps” and ”Waltzes”.

Although the chosen number of voting systems was relatively small, the results clearly indicate an overall performance improvement over the previous experiments. In fact, all the experiments registered a positive performance difference.

In summary, the classification experiments with a single system provided an average accuracy of 85.0% over 6 binary classification tasks, while a voting system architecture with $n = 5$ improved this average to 87.7%. The evidence that the classification system proposed in this thesis is capable of distinction between classes is, thus, strong.

Chapter 7

Future Work

The results of the previous experiments suggest a strong evidence that composition systems possess valuable information on the songs they are trained with, which can be used for classification purposes. In this section, future strategies are presented as pathways to continue the work started with this thesis, with the aim of improving not only its results but also extending the scope of its applications.

Regarding the improvement of the results, there is a large pool of possible options:

- For the experiments performed in this thesis, only songs from the Nottingham dataset were considered. As explained in earlier chapters, this was due to the clean structure and simplicity of the MIDI files in this dataset. Considering MIDI files from other sources is also possible, but it would require a more complex data preparation process. However, being able to use a broader range of songs is essential, because it allows a better training of the composition system and, specially, because it enables the classification algorithm to be used and tested with any available class of songs. As such, it is imperative to improve the processing of data, allowing any MIDI file to be considered and thus boosting the amount of data for which the algorithm can be applied.

- Another possibility regarding the previous point would be to change the initial source of the data, from MIDI files to songs in ABC notation. In theory, this could facilitate the data preparation process, but it would also imply core changes in the composition system. However, it is important to underline that the composition system is just a means to an end regarding the primary goal of this thesis – the classification algorithm. Changing the internal mechanism of the composition system should not affect the classification phase, providing its performance is good. On the other hand, a composition system using language processing techniques and ABC notation is theoretically more capable to express music in all its complexity, because musical embellishments are easier to represent with language than in a fixed matrix system. As such, using ABC

notation could contribute to a better composition and, hence, to an increased classification performance;

- As stated in the previous point, one of the downsides of the music representation used in the composition system is its lack of ability to represent musical embellishments or effects of the songs. A simple example would be a song played with a piano – certain notes of the song are played harder or softer than others, however these nuances are not captured by the current representation. This particular example could be easily solved by replacing the current binary matrices by matrices of natural numbers, where each number would represent the strength with which the note was played, with zero meaning it was not played at all.

- While the current dataset had only mono instrumental songs, the generality of the songs is poly instrumental. To reflect this in the current representation is, therefore, a crucial point. One option would be to increase the size of the matrices to include the notes played by each instrument. As for now each line of the matrix represents a specific note, but in the future, it should represent a note played by a specific instrument (a D played in a guitar is a distinct prediction from a D played in a piano and so it should be represented in a distinct line of the matrix). This, however, will imply a considerable increase in the size of the matrix, which should motivate new strategies for the improvement of the algorithm's computational performance;

Regarding the extension of the scope of applications, the following applies:

- Although in this thesis the proposed classification was solely based on the results of the composition systems, this was a choice directed towards a proof of concept. In fact, this strategy could be used in the future as part of a more traditional classification system, by including its results as another input variable of the data. This could provide valuable information, which current and common variables used in such classification systems do not possess;

- The concept of the classification system proposed in this thesis is based on a similarity measure between a song and a group of songs. As such, this concept can be extended beyond a simple discrete classification and be used continuously as a measure of a song's similarity for recommendation systems.

Chapter 8

Conclusions

Music is complex in nature and difficult to describe. Although it is easy to distinguish between songs in the mind, the task of justifying this distinction is still hard. As such, it is surprising that current classification systems are based on the extraction of musical characteristics, for this concept represents a challenge in the first place.

The fundamental work on this thesis was to conceptualize, build and test a music classification system which could avoid the difficult, but also restrictive task of figuring out musical features for classification. To achieve this, the proposed solution was to take advantage of the knowledge learned by composition systems, following the premise: if there are two composition systems, each specialized in a different class of music, the outcome of their compositions is expected to be significantly different; in particular, given a song from one class, the composition system which is specialized in that same class is expected to be more successful predicting it than any other system trained with different music classes.

With the concept of the classification system relying on the possible knowledge learned by composition systems, it was fundamental to ensure that these were as successful as possible and so an architecture of recurrent neural networks with long-term memory cells was used. This solution has been one of the most successful so far, as demonstrated in recent works on the field.

Another fundamental aspect was the data representation phase. Data can come in a variety of formats and music is a seamless example of this pluralism of languages. Considering the more structured nature of the MIDI format, which allowed for a simpler data extraction and manipulation, this was the chosen format for this work. Also, although technically any MIDI dataset could have been used to test the results of the proposed system, the Nottingham data set was chosen for its simplicity – songs are all mono instrumental – and also the fact that multiple labeled classes of songs were present, allowing for a number of experiments to be made.

While no other works on music classification have been found using this dataset, which rules out the possibility of comparing the results, the goal of the current work

should be regarded as a proof of concept – the objective was to present a new methodology for classification and not necessarily make an improvement on previous results achieved for any specific dataset. As such, results significantly and consistently higher than 50% for binary experiments should be interpreted as evidence that composition systems do indeed possess valuable knowledge to be used in classification tasks.

In fact, the results obtained in all the experiments for the classification accuracy where all significantly higher than 50%, ranging between 67.9% and 94.6% for the experiments with a simpler architecture of only one composition system per class and between 69.7% and 95.8% for an architecture of 5 voting composition systems per class.

Regarding the future, the most relevant aspects to underline are the extension of this work to other music datasets and the potential improvement of its results based on the enhancement of the composition systems and the use of different and more flexible music notations, such as ABC notation. But fundamentally, the relevance of this work can be extended to more complex classification systems, and also to recommendation systems, for the similarity concept used in this thesis – a composition based similarity – can be also applied in this field.

To summarize, the results of this thesis provide strong evidence that a step towards more alternative classification systems is possible, particularly but not only in the field of music, and that separate tasks as classification, composition and recommendation systems can share a common ground and its results can be of greater help to one another.

Bibliography

[1] H. Lejaren and L. Isaacson. *Experimental music: Composition with an electronic computer*. McGraw-Hill, 1979.

[2] C. Ames. "The Markov Process as a Compositional Model: A Survey and Tutorial." In: *Leonardo, v22 n2*. 1989.

[3] P. M. Todd and D. G. Loy. *Music and Connectionism*. MIT Press, 1991.

[4] M. C. Mozer. "Neural network music composition by prediction." In: *Connection Science 6*. 1994.

[5] YoshuaBengio, P. Frasconi, and P. Simard. "Learning Long-Term Dependencies with Gradient Descent is Difficult." In: *IEEE transactions on neural networks / 5, no. 2*. 1994.

[6] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory." In: (1997).

[7] G. Wiggins, G. Papadopoulos, S. Phon-Amnuaisuk, and A. Tuson. "Evolutionary Methods for Musical Composition." In: (1998).

[8] J. C. Principe, N. R. Euliano, and W. C. Lefebvre. *Neural and adaptive systems : fundamentals through simulations*. New York : Wiley, 2000. ISBN: 978-0-471-35167-2.

[9] N. L. Wallin, B. Merker, and S. Brown. *The origins of Music*. MIT Press, 2000.

[10] W. Chai and B. Vercoe. *Folk Music Classification Using Hidden Markov Models*. 2001.

[11] M. Towsey, A. Brown, S. Wright, and J. Diederich. "Towards melodic extension using genetic algorithms." In: *International Forum of Educational Technology and Society*. 2001.

[12] P. de Leon and J. Inesta. "Musical style identification using selforganising maps." In: *Second International Conference on Web Delivering of Music*. 2002.

[13] M.-K. Shan and F.-F. Kuo. "Music style mining and classification by melody." In: *Proceedings of IEEE International Conference on Multimedia and Expo (ICME)*. 2002.

[14] J. Vreeken. *Spiking neural networks, an introduction*. 2003.

[15] C. McKay and I. Fujinaga. "Automatic Genre Classification Using Large High-Level Musical Feature Sets." In: *ISMIR 2004, 5th International Conference on Music Information Retrieval*. 2004.

[16] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, 2007.

[17] D. Kriesel. *A Brief Introduction to Neural Networks*. http://www.dkriesel.com, 2007.

[18] K. Ebcioglu. "An expert system for harmonization of chorals in the style of J.S. Bach." In: *The Journal of Logic Programming* (2008).

[19] N. Collins. *Introduction to Computer Music*. Wiley, 2009.

[20] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR 9:249-256*. 2010.

[21] P. Donnelly and J. Sheppard. "Evolving Four-Part Harmony Using Genetic Algorithms." In: *Applications of Evolutionary Computation*. 2011.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet classification with deep convolutional neural networks." In: *Advances in Neural Information Processing Systems, v2*. 2012.

[23] G. Lu, K. M. Ting, and D. Zhang. "A Survey of Audio-Based Music Classification and Annotation." In: *Journal IEEE Transactions on Multimedia archive Volume 13 Issue 2*. 2013.

[24] I.-T. Liu and B. Ramakrishnan. *Bach in 2014: Music composition with recurrent neural network*. 2014.

[25] Z. Lipton, J. Berkowitz, and C. Elkan. "Long Short-Term Memory." In: (2015).

[26] M. Müller. *Fundamentals of Music Processing:Audio, Analysis, Algorithms, Applications*. Springer, 2015.

[27] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[28] M. Araoz. *Training a Recurrent Neural Network to Compose Music*. 2016.

[29] K. Choi, G. Fazekas, and M. Sandler. "Text-based LSTM networks for Automatic Music Composition." In: (2016).

[30] S. Ruder. "An overview of gradient descent optimization algorithms." In: (2016).

[31] Y. Zimmerman. *A Dual Classification Approach to Music Language Modeling*. 2016.

[32] N. Agarwala, Y. Inoue, and A. Sly. "Music Composition using Recurrent Neural Networks." In: (2017).

[33] K. Choi, G. Fazekas, and M. Sandler. "Convolutional recurrent neural networks for music." In: *ICASSP 2017 - 2017 IEEE International Conference on Acousticsl*. 2017.

[34] K. Hara, D. Saitoh, and H. Shouno. "Analysis of dropout learning regarded as ensemble learning." In: (2017).

[35] D. D. Johnson. "Generating Polyphonic Music Using Tied Parallel Networks." In: *EvoMUSART 2017: Computational Intelligence in Music, Sound, Art and Design pp 128-143*. 2017.