KU LEUVEN

Creating a Songwriting Tool Based on Convolutional Neural Networks and a Moving-Window Sampler

With Applications Towards Generating Long-Term Pop Music

Thomas D'heer

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Engineering and Computer Science

Thesis supervisors:

Prof. dr. ir. Johan Suykens

Prof. dr. Hugo Van hamme

Assessor:

Prof. dr. ir. Dirk Van Compernolle

Mentor:

Ir. Henri De Plaen

Academic year 2019 – 2020

Preface

The completion of this thesis marks the final milestone in my educational career, which spanned a period of six years. It comes with both sadness and joy, and it has more significance than I can comprehend at this moment. Although I feel like the time is right for the next chapter in my life, I cannot help but look back at the city of Leuven with a strong feeling of nostalgia. The place where I developed as a scientist, matured as an artist and grew as a person.

A work like this cannot be realized on one's own. First of all, I would like to thank Henri for proposing this subject, and for guiding me through something that was completely new to me. Secondly, I would like to thank my promotors, professor Johan Suykens and professor Hugo Van hamme, for giving me the possibility to work on this subject. I am very grateful to have developed the skills that are most important to the Artificial Intelligence program, while working in a field that I love and is my dearest passion.

Lastly, I would like to thank my parents, who have had to endure the presence of a solitary son working on a thesis for the second year in a row. As we have spent most of our time together during these extraordinary months, I have not been the most pleasant of companies. I hope this product of my work makes up for it. A final word of thanks to my brother Carl, for his ability to offer honest advice and appreciate research-related humour.

Thomas D'heer

Abstract

In recent years, the advancement of artificial intelligence created new possibilities in the field of algorithmic music composition. To this day, it remains a great challenge to generate songs which sound humanly composed.

In essence, a song is a piece of temporal data with complex short-term and long-term structure. Often times, music is modeled with variants of the recurrent neural network, as they are designed for modeling sequential data. On the other hand, convolutional neural networks (CNN) have been proven to capture local structure well, but are not specialized in long-term modeling. For example, Huang et al. [24] used a CNN to create Coconet, a model designed to generate four-voiced sequences in the style of Bach chorales. These compositions are highly structured and exhibit counterpoint relationships. In this thesis, a more variable framework is developed inspired by Coconet, which is suitable for modeling pop music, a more irregular type of music. Here, the data is extracted from MIDI-files by means of a custom-made data processing pipeline. Secondly, a novel time-series inspired moving-window sampler is introduced, based on Gibbs sampling. This technique can generate compositions of any length, overcoming the difficulty of sampling over large dimensions.

The results show that the model is best at generating variations on a given priming sequence. A mixture of musical ideas from the training data appears, varying from simple bass lines to four-voiced counterpoint compositions. The longer moving-window samples illustrate this versatility the most, and prove that convolutions over time could be an alternative to regular time-series modeling.

The conclusion of this thesis is two-fold. Firstly, it was shown that the proposed framework shows great promise as a songwriting tool. The unique approach of the moving-window sampler creates new possibilities to be explored. Secondly, the lack of a high-quality dataset forms a bottleneck for modeling pop music symbolically, a problem which can be solved in future research.

List of Abbreviations and Symbols

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
AR	Autoregressive
BPM	Beats Per Minute
CNN	Convolutional Neural Network
DAW	Digital Audio Workstation
EC	Evolutionary Computing
FNN	Feedforward Neural Network
GA	Genetic Algorithms
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
GSN	Generative Stochastic Network
HMM	Hidden Markov Model
LSTM	Long Short-Term Memory
MCMC	Markov Chain Monte Carlo
MIDI	Musical Instrument Digital Interface
NADE	Neural Autoregressive Distribution Estimation
NN	Neural Network
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network

Symbols

a	Activation
α	Masking probability
b	Bias
h	Hidden layer
$\mathbb{R}$	Set of real numbers
W	Interconnection matrix

List of Tables

List of Figures

Contents

Chapter 1

Introduction

1.1 Context

Music is an important element in the lives of many people. Yet not so many realize a musical composition is a complex piece of structured, temporal information. For centuries, scientists have tried to define formal rules by which music can be constructed. Even though compositions are often firmly rooted in music theory, this task proves to be dauntingly challenging, for rules are broken constantly. Often times, theoretically correct pieces can convey a feeling of 'artificialness' which humanly composed pieces do not have. When computers and later on deep learning arose, research into algorithmic music composition was strongly accelerated.

The current literature of algorithmic composition can be split in two parts, based on note representation. Firstly, music can be captured directly in continuous audio signals. This approach focuses on sound and does not inherently distinguish different instruments and notes. For example, the deepAutoController by Sarroff et al. [46] uses autoencoding neural networks (NN) to synthesize audio fragments. There are plenty of advantages to the audio approach. As illustrated by OpenAI's Jukebox [17], it allows capturing human voices and subtle characteristics of music like timbre and dynamics. Secondly, the discrete counterpart is symbolic representation, where music is depicted using concepts like pitch, duration, time of onset and chords, requiring a completely different strategy [11]. The limitation of working symbolically is that the information does not directly represent any sound and does not contain the subtle aspects of raw audio. Instead, as it differentiates all the instruments, notes and more, it provides insight in the composition of the pieces while utilizing data of a much

lower dimensionality.

In this thesis, the choice is made to use the symbolic route only, as the focus lies on composition. Thus, the audio approach will not be considered. Two important concepts that will be used throughout this text are MIDI and the pianoroll. MIDI, or Musical Instrument Digital Interface, is a widely used communication protocol which connects digital software and instruments. It is also used as a framework for both regular and computational symbolic music composition. A MIDI-note, among other things, contains the pitch, volume, length and time of onset of a note. On the other hand, a pianoroll can be seen as a binary two-dimensional grid with pitch on one axis and time on the other, showing which notes to be played at each time step. Originally, the pianoroll was a paper roll with perforations on the positions where a note must be played. Nowadays, it has been adopted by many digital platforms as an intuitive format to depict a composition. Both MIDI and the pianoroll are used in the following chapters to represent compositions. A detailed overview of the history of algorithmic music generation is given in section 2.1.

Over the years, many deep learning architectures have been developed, all of which have certain characteristics which are suitable for extracting different aspects of a musical score. One can imagine it is hard to design a model that can do it all: there are certain rules that music of a chosen genre seems to follow (tonality, common structures like verses and choruses), but there exist just as many exceptions to these rules, which make algorithmic music composition a great challenge. It is difficult to recreate human creativity, as it is not limited to formal rules as a computer is, and it is generally not possible to explain where an idea came from. The recreation of that 'human touch' remains the holy grail. The most recent works in this field are discussed in section 2.2, which use different model architectures and note representations.

1.2 Musical background

For the reader unfamiliar with music theory, a short introduction into the basic terminology is given so the text can be understood easily. For convenience, the piano keyboard is used to illustrate some of the concepts. This section is loosely based on the work of Schmidt-Jones et al. [47].

There are two common ways to represent the musical alphabet, through the notes do-re-mi-fa-sol-la-si, or by means of letters (A-B-C-D-E-F-G), where 'do' corresponds

to C. While these notes represent the white notes on a piano keyboard (Figure 1.1), the black notes are the ones in between, which are called 'sharp' (a semitone up, denoted by #) or 'flat' (a semitone down, denoted by b) notes. For example, $A_{\#}$ lies in between A and B. Observe that $D_{\#}$ is the same note as E_b. Two consecutive notes are separated by a semitone (e.g. A - $A_{\#}$), and an interval of twice this length is called a (full) tone or whole step (e.g. A - B). The interval between two consecutive notes of the same letter (e.g. A - A) is called an octave, corresponding to 12 semitones (indicated on Figure 1.1).

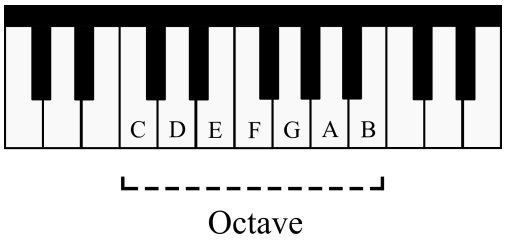


Figure 1.1: A piano keyboard with an octave indicated. Every octave contains twelve notes with a semitone between them (black keys included).

Each note is based on the fundamental frequency of its sound. For example, A4, the A note of the fourth octave, corresponds to a frequency of 440 Hz. Raising a note by one octave doubles its fundamental frequency (e.g. A5: 880 Hz). MIDI-pitches are expressed in numbers from 0 to 127, corresponding to the interval C0-G9, where an increment of 1 represents a semitone.

Rhythm is expressed by the 'metre' (recurrent pattern of beats), which contains various aspects like tempo and time signature. The latter describes the amount of notes played in a bar or measure, which is the 'time unit' of a composition score. The most common time signature in pop music is 4/4 ('four fourth'), where four quarter notes are played per bar. Here, the quarter note is the 'base' note which is played on every beat. It is half has long as a half note and one fourth as long as a whole note. These divisions go further to eighth, 16^{th}, 32^{nd} notes etc. Intuitively speaking, the 'beats' are the moments when you tap your foot along to a song. The tempo of a song is often expressed in beats per minute (BPM).

A chord is a collection of notes with certain intervals in between them. For example, a major triad consists of three notes: a root note, a 'major third' and a 'perfect fifth', where the major third is four semitones higher than the root, and the perfect fifth is three semitones higher than the major third. Another concept

relevant for this thesis is 'counterpoint', referring to the harmonious interplay between multiple voices which each independently contain rhythm and melody (a sequence of notes). Harmony is defined as when multiple notes sound 'good' together, also called 'consonance' (in contrast to 'dissonance'). This is often expressed in terms of a 'key', which is the scale a certain composition is played in, where a 'scale' is a set of intervals over an octave which have a characteristic sound. Lastly, in context of pop music, it is important to note the occurrence of various characteristic substructures in a song, such as an intro, verse, chorus, bridge and outro.

1.3 Research goals

The first aim of this thesis is to create a framework for modeling pop music, based on convolutional neural networks (CNN). This consists of constructing a pop music dataset using a specialized processing pipeline, and designing the model architecture itself, which is inspired by the work of Huang et al. [24].

Secondly, a novel time-series moving-window sampler is investigated for creating compositions of any length. In this technique, the reach of a Gibbs sampler is extended over the time domain in stepwise manner, similar to how recurrent neural networks or autoregressive (AR) models use values of previous timesteps. This attempts to lift the fixed-size limitation of convolutional neural networks.

1.4 Overview

This thesis contains six chapters. In chapter 2, an outline of the history of algorithmic music generation is presented, along with an overview of related works in this field. Next, Chapter 3 introduces the most common deep learning architectures which have recently been used for this application. The discussed subjects are neural network types, techniques to estimate probability distributions and ways to sample from them. In Chapter 4, a detailed description is given of the Coconet model, the novel elements that were introduced like the moving-window sampler and details of the experimental set-up. To test the model, samples are generated from scratch, based on a given priming melody and by moving-window sampler extension. The results are described and discussed in Chapter 5, along with the challenges that have been encountered. Finally, in Chapter 6, a summary is given of the presented work along with a reflection on the research goals and suggestions for future research.

Chapter 2

Literature overview

In the first section of this chapter, the history of algorithmic music composition is discussed. Here, a distinction is made between rule-based systems and stochasticity-based systems. The first refer to the application of formal rules (or grammar sets) which dictate how melodies, harmonies and rhythms are constructed. The latter contain an element of randomness in the composition process. In section 2.2, an overview is given of the state-of-the-art models in the field of music modeling.

2.1 History of algorithmic music composition

For a large part of human history, musicians and scientists have been interested in using formal rules to create music, dating back to Pythagoras and Ptolemy [34]. These experiments were at first deeply rooted in music theory derived from mathematics, remaining purely theoretical.

When the computer was invented, the first musical experiments involved the creation of sounds. Alan Turing is one of the first scientists to construct a computer that could generate sound, as he was able to record a few melodies in 1951 with a primitive synthesizer [1]. Another famous instance was Newman Guttman in 1957, who composed a piece called "The Silver Scale" and was able to produce the sound on a computer using sound synthesis software called "Music I" [11].

Later on, the term 'algorithmic composition' was coined, defined as "the process of using some formal process to make music with minimal human intervention" [5]. Lejaren Hiller and Leonard Isaacson equipped a computer with a self-created rule-based system, which by itself composed a piece called "Illiac Suite" in 1957, the

first time a computer performed music composition. Together with Robert Baker, Hiller went on to create a programming language called "MUSICOMP" (MUsic SImulator-Interpreter for COMpositional Procedures), which led to the creation of "Computer Cantata" (1963) [34].

Meanwhile, stochasticity-based methods entered the scene. Iannis Xenakis used a stochasticity-based technique to create "Atrées" in 1962. The stochastic side of algorithmic composition makes use of randomness through statistical techniques and Markov chains [34]. Xenakis is also thought to be pioneer in transitioning from composition-based algorithms towards sound-based music generation [35].

Later on, machine learning, as a part of artificial intelligence (AI), was explored to exploit the rapidly growing capacity of computers. An early example is David Cope, who performed pattern matching on pieces of certain composers to construct new music (Experiments in Musical Intelligence, 1996) [15]. Like many, he based some of his models on the chorales of J.S. Bach, which were suitable for rule-based architectures because of their consistent structure. Thus, by learning from existing scores as input data through unsupervised learning, the computer forms its own grammar rules. As computing power was growing rapidly, large datasets could be processed to come up with implicit complex 'rules'. This seemed like a beneficial option, as some have suggested that music is too complex to represent by manual rules [11].

One of the most popular infrastructures for this application, the artificial neural network (ANN) or NN in short, was explored by Peter M. Todd in 1989, by using a NN architecture in a methodology called parallel distributed processing (also known as connectionism), which involves computing with multiple units in parallel [52]. Since then, many different forms of the ANN have been used for algorithmic composition, including the CNN. These will be discussed in section 2.2. Today, many autonomous music-making systems exist (both sound- and composition-based) which are publicly available and there is growing interest from large companies. E.g. the Magenta Project by Google uses AI for various musical applications, from creating new synthesizer sounds and transcription algorithms to models for learning long-term structure in music [33]. Sony's Flow Machines offers a user-friendly interface for musicians, while the AIVA engine (Artificial Intelligence Virtual Artist) has been used for films, adverts and game studios [29]. Furthermore, AIVA has been officially recognised as a composer, which enables it to release music under its own name [2]. Other well known examples are Amper Score ® [7] and OpenAI's Musenet [40].

Over the years, there have been several technical developments which had a great impact on the progression of these methodologies. In 1983, MIDI (mentioned in Chapter 1) was introduced by Dave Smith (founder of Sequential Circuits) and Ikutaru Kakehashi (founder of Roland Corporation) as the new standard for music representation [50]. Despite some flaws in the first version (MIDI 1.0), it remains very popular to this day. After 37 years, a new version, the MIDI 2.0 environment, is in the making [51]. Secondly, producing music on a computer has become much easier through the emergence of Digital Audio Workstations (DAW), like Apple's GarageBand and Ableton's Live. Some deep learning models can be integrated into the workflow of a DAW in the form of a plugin (e.g. Magenta for Ableton Live [44]), enabling AI to assist a layman in music composition.

Interestingly, AI is already playing an important role for other reasons than composition. According to Amato et al. [6], AI will have an impact on many different areas of the music industry in the future; from creation (algorithmic composition), to production (automatic mixing and mastering software) and consumerism (automatic recommendation systems). Overall, the growth of AI will bring new possibilities to the creative industry, while it remains unclear if it will endanger human jobs [57]. Some believe that music generation and consumption will blend together in the future, as music will be generated custom-made for every listener, based on its preferences and listening history [6]. The challenges of this field are multidisciplinary, containing not only elements from computer science, but also from ethics (concerning data handling and user behaviour influence) and law (copyright issues) [6].

So far, the results of AI-powered music generation have not been spectacular, but they have provided valuable insight into the complex structure of music. Furthermore, AI-aided composition has been established and provides a fresh stream of inspiration for the exploring musician.

2.2 Related works

2.2.1 Introduction

As mentioned in Chapter 1 this thesis is situated within the symbolic domain, as a consequence the audio domain is out of scope and will be left out from this point forward.

Historically, stochastic and rule-based methodologies were important in the

development of algorithmic composition, but nowadays they have become outdated in their pure form. Most research has been done within the framework of deep learning. Essentially, a musical score is a piece of temporal data with complex short-term and long-term structure, and different techniques can be used to mimic a certain characteristic of the music. Many architectures have proven to be suitable for modeling sequential information. In this section, an overview is given of recent publications involving these models.

The models that will be described can be categorized based on various important characteristics. These are important for comparisons, and to realize what has been done already. As there are many such characteristics, only the most crucial ones are listed in the following paragraphs.

First and foremost, every publication is based on one or more model architectures. In the early phase of algorithmic composition, Hidden Markov Models (HMM) were regularly used, which consist of a Markov chain of hidden states, with a set of visible states depending on them. They form the basis for other techniques, but require an expensive algorithm (e.g. Viterbi) to find the most likely sequence of hidden states [4]. The recurrent neural network (RNN) is an extension of the feedforward neural network (FNN) where feedback loops are introduced. It should be suitable for music modeling as it is capable of modeling temporal sequences, but it only considers the information of one step prior to the current state [16, 23, 21]. This problem is solved by the Long Short-Term Memory network (LSTM), whose gated cells provide long term memory [30, 14]. The limitation of the LSTM is that it tends to struggle with capturing musical concepts or reproducing too much without creativity [28]. Next, the CNN is popular in music generation as it is good in capturing local structure and reduces the dimensionality of the data [24, 25]. However, its locality may also render it unable to model long temporal sequences [11]. Many researchers have also found success with generative adversarial networks (GAN) [13, 59, 19]. They contain two separately trained networks, a generator and a discriminator, where the generator generates samples and the discriminator tries to differentiate them from real data. GAN's produce realistic samples, but problems can arise during training like vanishing gradients or mode collapse [13]. Another NN variant is the Restricted Boltzmann Machine (RBM), which contains one layer of non-connected hidden units. It is a stochastic generative model which learns a distribution over input data. It is often used in its deep form (Deep Boltzmann Machine, DBM) or in combination with other architectures like the RNN and the CNN [10, 28]. A similar architecture is the

autoencoder (AE), whose encoder-decoder structure is interesting for the data-heavy audio paradigm, which benefits from dimensionality reduction [17]. In chapter 4, a detailed overview is given of the FNN, RNN and CNN.

Secondly, the input data must often be converted to a suitable format before it can be used. These encoding schemes come in various forms and are an important aspect of the model. A popular representation is the pianoroll [24, 28, 25, 10, 13], others include text [30], interval-based definitions [4] and sequence serializations [39, 26].

Another important aspect is finding a good performance metric to assess the quality of the generated compositions, which proves to be a difficult task. Examples of these metrics are log-likelihood [10], minimization of cross-entropy loss [30], novelty of musical sequences [14] and trials based on human assessment [30]. A unique approach was used by Colombo et al. [14], who used a novel measure to assess the creativity of melodies generated by their LSTM-based Deep Artificial Composer, namely the fraction of generated melody transitions which are not found in the training set.

Lastly, some important design choices must be made concerning the desired character of the music. A first distinction is made between monophony (where each instrument plays at most one note per time step, e.g. in [24]) or polyphony (multiple notes per time step, like a chord, e.g. in [28]). Secondly, the structure of musical compositions can vary from rather fixed (e.g. chorales, [25]) to more irregular, like pop music [20, 27]. A topic which will be elaborately discussed in this thesis, is the creation of pop data sets via processing tools. This is required to extract the desired features from the songs. Examples of such tools are beat estimation, pattern recognition [20] and specific event detection [27].

2.2.2 State-of-the-art models

The model on which this thesis was based is Magenta's Coconet, which tries to replicate the chaotic aspect of the natural workflow of a human composer. A deep CNN architecture creates small pieces of music, which are puzzled together using blocked Gibbs sampling [24]. In essence, it fills in empty parts in given compositions. Coconet has been turned into a accessible composer in the style of Bach, called the Bach Doodle [25]. Both frameworks defined a pianoroll as a binary three-dimensional tensor $x \in \{0,1\}^{I \times T \times P}$, representing different pitches P being played at time step T

by instrument I.

Lattner et al. [28] transformed MIDI data onto a two-dimensional binary pianoroll with 512 time steps over 64 pitches. For more control over the structures which are generated via Gibbs sampling on a convolutional RBM, constraints are directly applied via gradient descent optimisation. This allows a template piece to dictate structural prerequisites, and makes sure the tonality and meter of the generated compositions are satisfactory.

A lot of research has been done on chorales written by Bach, like the aforementiond Coconet and Bach Doodle models. They form a vast collection of compositions which conform to the classical rules of music theory the most [30]. The DeepBach model by Hadjeres et al. [21] is an RNN-based generative model which uses pseudo-Gibbs sampling for more flexibility as it is possible to fix certain parts of the composition, contrary to generating music from left to right. Even though it is a non-reversible Markov Chain Monte Carlo (MCMC) algorithm, the convergence of the pseudo-Gibbs sampler is guaranteed, although possibly to a different distribution than a regular Gibbs sampler [56]. The authors mention this technique is successful because of their representation: each voice is modeled separately, and rhythm is taken into account via a 'hold' symbol, expressing where the previous note is being held. BachBot, the system developed by Liang et al. [30], also creates chorales in the style of Bach using a deep LSTM-model with RNN-units. The input compositions are initially transposed to the same key and represented as a pianoroll where all voices are mixed into a timestep-frame, ordered on descending pitch. Later, the pianorolls are encoded into a wordvector-format, reducing dimensionality.

Chen et al. [13] also used a pianoroll representation, but as input in a GAN with a deep convolutional network structure. Yu et al. [59] used a GAN framework as well but with a different strategy, where the LSTM-based generator and discriminator are trained on lyrics, instead of music. Based on psychophysical theories to increase arousal potention, Elgammal et al. [19] modified a GAN to generate creative art: on one hand the products try to fool the discriminator to think it is 'art', and on the other hand they try to confuse the discriminator about the style of the generated work, which is considered as a sign of originality. Dong et al. [18] also used GAN architectures in various models which can generate songs with multiple polyphonic instruments.

Instead of using the well-known RNN's or LSTM's for sequential modeling, Huang et al. [26] modified a sequence model based on self-attention (the Transformer [55])

so it can focus on relative positional information with little memory use for long sequences, yielding results on the same level as state-of-the-art LSTM's. The input data was a matrix containing a time sequence of discrete notes for different voices, represented as rows. The data was serialized by annealing the pitch values column by column. Remarkable is that the resulting samples reached a length of one minute.

Amper Music offers a composition tool that wasn't trained using neural networks and musical examples. Instead, a music theory framework was laid out along with a mapping between musical elements and emotions [60].

For the sake of completion, the chapter is concluded by mentioning genetic algorithms (GA) and evolutionary computing (EC), where a population of individual genomes is improved over generations via evolutionary principles. This approach is rule-based and is thus similar to the grammar rules-paradigm mentioned earlier. Here, the definition of an individual and the grammar rules which must be followed in the evolutionary process are key [32]. For example, Loughran et al. defined a ruleset based on notes, chords and arpeggios, while measuring the fitness of an individual implicitly by evaluating individuals based on the opinions of critics [31]. On the other hand, the MetaCompose framework by Scirea et al. [48] offers a partitioned system with real-time generation of chord sequences (via random walks on directed graph), melodies (combined multi-objective optimization with a feasible/infeasible two-population method) and accompaniments (inversions and mutation), focusing on general structure. They compared random compositions with generated ones via survey-wise evaluation based on different subjective parameters like pleasantness. Within the field of GA and EC, finding a fitness function which evaluates music and measures its aesthetic qualities has proven to be hard [19, 32].

Chapter 3

Deep learning in algorithmic music composition

In this section, an overview is given of various model architectures, methods to estimate probability distributions and sampling techniques which are used in algorithmic music composition. As they have different characteristics, advantages and disadvantages, some are more suitable for music modeling than others. These elements, which form the building blocks of this thesis, will be explained in mathematical terms and in context of music.

3.1 Model architectures

The building block of a neural network is a neuron, a computational unit which can perform mathematical operations on data passing through it. As the design is inspired by the human brain, but is not identical, these networks are also referred to as artificial neural networks. The neurons can be formed into layers and interconnected in different ways, forming various model structures [37]. In this section, a basic explanation is given of feedforward, recurrent and convolutional NN's.

3.1.1 Feedforward neural networks

The first model of an artificial neuron was described by McCulloch-Pitts, where n incoming signals x_i are weighted separately by weights w_i, a bias b is added, and the result is projected to -1 or 1 via a threshold function (Figure 3.1). This yields the output signal y [8]:

$$y = f(w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_n \cdot x_n + b) \tag{3.1}$$

with n the amount of input signals x, and f the non-linear activation function.

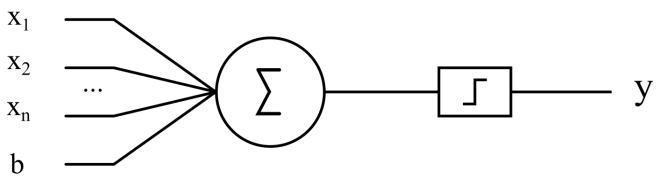


Figure 3.1: Schematic representation of a neuron, the computational unit of an artificial neural network.

A feedforward neural network (FNN) is a network consisting of multiple neurons, where data flows through one layer and comes out at the subsequent one (Figure 3.2). There are no intraconnections within a FNN layer, nor is there information flow in the opposite direction. Aside from input and output layers x and y, a network can have any amount of hidden layers h [37]. Such a network with multiple layers can be described by a set of equations that extends formula 3.1. Often, a linear combination of the inputs is combined with a bias, in which case the output vector $\mathbf{h^k}$ at layer k is calculated as:

$$\mathbf{h^k} = f\left[\mathbf{W_k} \cdot \mathbf{h_{k-1}} + b_k\right]$$

with f an activation function, $\mathbf{W^k}$ the vector containing interconnection weights between layer $k-1$ and k, $\mathbf{h_{k-1}}$ the output of hidden layer $k-1$ and b^k the k^{th} element in bias vector $\mathbf{b}$ (depicted in Figure 3.2 for a network with two hidden layers). When all neurons are connected with all the neurons of the next layer, the network is called a fully connected layer. The amount of neurons in a layer is an important characteristic and must be chosen wisely. One chooses a suitable activation function based on the application. The goal of these functions is to prevent the values from growing rapidly by squashing them into a limited range. Popular choices are the Rectified Linear Unit (ReLU), sigmoid and hyperbolic tangent functions.

The deep version of the FNN (also called multilayer perceptron) lays the foundation for many types of networks, like the RNN and the CNN and is a powerful tool for many deep learning applications [16].

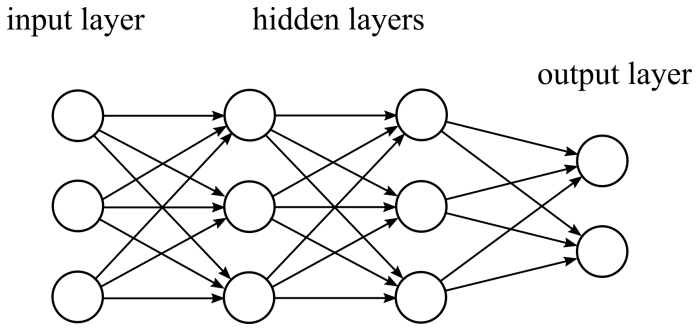


Figure 3.2: Schematic representation of a feedforward neural network with two hidden layers.

3.1.2 Recurrent neural networks

In a FNN, neurons can only connect with the layers directly before and after itself, limiting the model to static behaviour. Intuitively, this means that every computational unit only takes information from the previous step into account. On the other hand, in RNN's feedback loops are introduced, allowing neurons to receive input from itself at previous timesteps, and non-sequential neurons to be connected to others. This way, temporal or sequential information can be modeled, exhibiting dynamic behaviour. The concept of 'order' is highly important in many applications, e.g. in context of language or sound (speech recognition, translation engines, chatbots, music). A RNN can learn which elements are likely to follow from a certain element. Furthermore, many RNN's can handle input data of variable length. While a FNN would have to train weights for every time step separately, a RNN used the same weights at all positions (an extensive form of parameter sharing) [16].

Consider the 'forward pass' RNN, a simple RNN design where the hidden layer at time step t receives input from an input sequence $\mathbf{x}$ of length I at time t, and from the hidden layer itself at time $t-1$. The output activation of each hidden layer unit h is determined by:

$$a_h^t = f\left[\sum_{i=1}^{I} W_{ih} x_i^t + \sum_{h'=1}^{H} W'_{h'h} b_{h'}^{t-1}\right]$$

with input vector $\mathbf{x} \in \mathbb{R}^{I \times T}$, network input $\mathbf{a} \in \mathbb{R}^{H \times T}$, activations $\mathbf{b} \in \mathbb{R}^{H \times T}$, H total amount of hidden units, and interconnection matrices $\mathbf{W} \in \mathbb{R}^{I \times H}$ and $\mathbf{W'} \in \mathbb{R}^{H \times H}$. The simplest version of this model (one input, hidden unit and output)

is illustrated in Figure 3.3. Note how the hidden unit receives its own activation from the previous step.

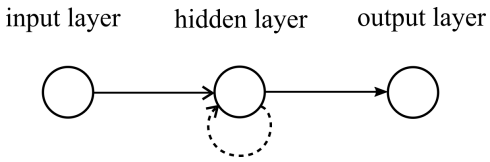


Figure 3.3: Schematic representation of a 'forward pass' RNN cell, where the hidden layer unit h at time step t receives input from the input layer, and from h at time step $t-1$.

To train a RNN, one can apply the same backpropagation algorithms as for FNN's. However, problems often arise in which the propagated gradients vanish after a certain amount of stages, or explode [16].

More specialized variants of the RNN, like the long short-term memory (LSTM) and gated recurrent unit (GRU) models have been developed to combat these problems. The LSTM has a memory cell, to which information can be written, erased or extracted. It is controlled in a gated manner by trainable weights, allowing the time scale of integration to be adapted to the input sequence. GRU's on the other hand are similar, but control the weights involved in 'forgetting' and 'updating' simultaneously [16].

Even though these models are suitable for handling sequential data, they are not always the best option in context of music. In some more complex musical pieces, time-series modeling is not able to capture all the important musical characteristics. It may struggle with the concept of tonality, meter or occurrence of motifs and phrases in different forms [28].

3.1.3 Convolutional neural networks

The CNN is a variant of the FNN, specialized in processing data organized in a grid, in any amount of dimensions. Instead of having only layers based on linear recombination, CNN's contain at least one layer based on the mathematical convolution operator, usually in combination with one or multiple pooling layers [16]. A CNN is often composed of fully connected layers, and modules made of convolutional and pooling layers. The structure of the CNN, just like the basic ANN, is inspired by the brain, more specifically the visual cortex, which is made of alternating layers of simple and complex cells [8, 45]. Unsurprisingly, the CNN is

often used in image processing, as images are a non-temporal type of data usually containing three dimensions (channel x height x width).

Convolution and convolutional layers

The convolution of two functions $f(t)$ and $g(t)$, symbolically written as $f * g$, is defined as:

$$(f * g)(t) = \int_0^t f(\tau)g(t-\tau)d\tau$$

To explain the meaning of this operator, let's consider an example involving 2D image processing (Figure 3.4). A filter (or 'kernel') function (A), a square 2D structure, is placed on the grid of pixels so the upper-left corners overlap (B). Element-wise multiplication is performed between the overlapping numbers, and the kernel is moved across the input grid in all directions by a constant distance, called the 'stride' (resulting in C).

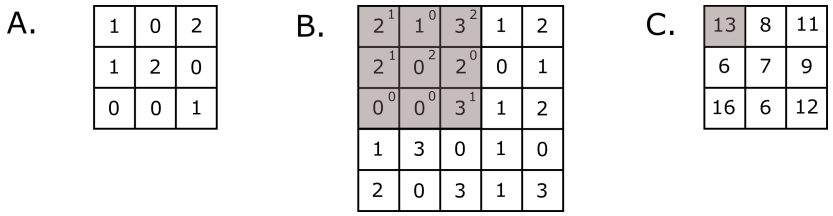


Figure 3.4: A 3x3 kernel (A) is swept across a 5x5 input grid (B), where at each position their overlapping elements are multiplied in element-wise manner, the sum of which is stored in a 3x3 matrix (C).

In this example, a 2D grid of size 5x5 is converted to a 2D feature map of size 3x3, by convolving it with a kernel of size 3x3. The dimensions are reduced because the values at the edge of the input matrix will never be at the center of the filter (as there are no values beyond them). For input with height H and width W, the output dimensions are:

$$\text{output width} = \frac{W - F_w + 2P}{S_w} + 1$$
$$\text{output height} = \frac{H - F_h + 2P}{S_h} + 1$$

with F_w and F_h the filter width and height, S_w and S_h the stride width and height and padding P. The latter refers to the addition of extra 'fake pixels' to the grid (e.g. zeros) so the kernel can slide its center onto every value, conserving the input dimensions. If the padding is set to zero, the dimension size will decrease (downsizing).

In contrary to a fully-connected FNN where each input value is connected with a neuron, a CNN-neuron receives input from a small group of input values. This is why a CNN is specialized in capturing local structure.

Pooling layers

The application of a well chosen filter can lead to the detection of edges, blurring of images or extraction of desired features, explaining the popularity of the CNN for image processing applications. Normally, each layer of a NN removes a fraction of the information contained in the data, and tries to keep the useful parts. In a CNN however, the output features contain redundant information, as the filter slides over the same pixel multiple times (except for those at the edge of the grid), preventing the degradation of information.

In this context, the pooling layer was introduced in CNN's. Its purpose is to 'pool' together information from small regions of a feature map into a more compact representation. In this way spatial resolution is lowered, but input distortions, noise, outliers and translations are averaged out. For example, a 2x2 max-pooling layer will output the maximal value of each 2x2 region of the feature map. Figure 3.5 depicts this process for a 4x4 feature map as input. This artificial type of layer has no feature-related meaning like a filter does, but it has shown to preserve the useful parts of a feature map, while reducing dimensionality.

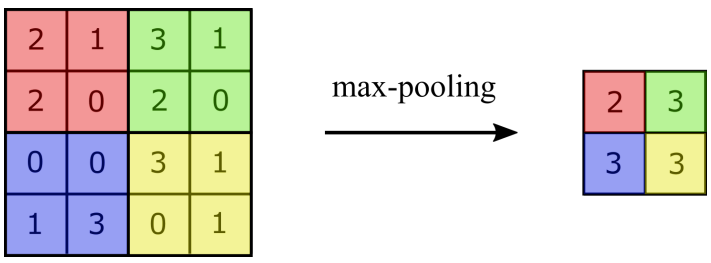


Figure 3.5: Illustration of a 2x2 max-pooling operation with stride 2, applied to a 4x4 input.

Convolutional neural networks

In context of CNN's, the convolution operator can be seen as a linear transformation (like matrix multiplication) which uses predefined interconnection weights, as the same kernel is used over the whole input data, which is called parameter sharing or weight sharing [16]. Note that while the filters of convolutional layers are learned during training, the pooling layers are fixed.

Apart from parameter sharing, convolution is useful for neural networks because of sparse connectivity and equivariance to translation. The first term refers to how a group of units is linked to a neuron collectively, requiring fewer connections than a dense network, where every pixel is connected separately. The latter refers to the fact that when an image is shifted, the extracted feature map resulting from convolution with a certain kernel will be shifted identically [16].

3.1.4 CNN's and music

Both the convolutional and pooling layers of the CNN were built with imaging applications in mind, where the data is a grid of contineous pixel-values. To understand the difference between images and music, consider the pianoroll as a representation for a composition. It is defined as a binary three-dimensional tensor $\mathbf{x} \in \{0,1\}^{I \mathrm{x} T \mathrm{x} P}$, representing different pitches P being played at time step T by instrument I. A 'one' is written where a note is played and a 'zero' where nothing is played. Figure 3.6 illustrates the structure of pianoroll for two instruments, where the x-axis represents the time steps and the y-axis represents the range of pitches.

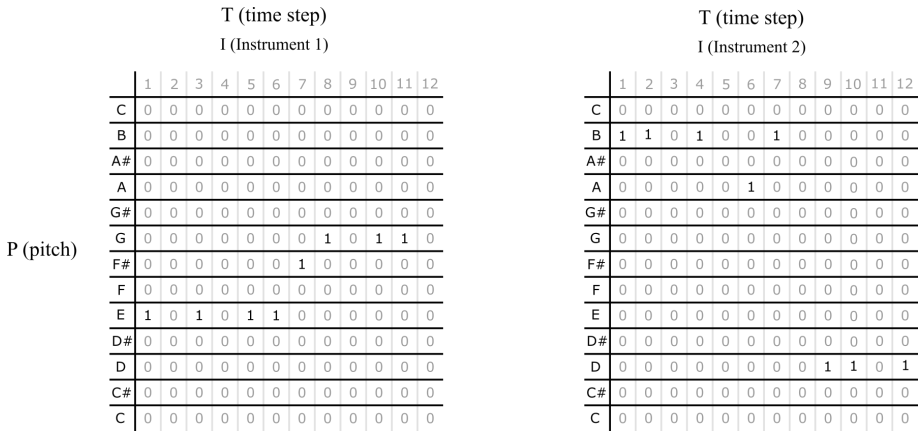


Figure 3.6: Pianoroll representation for two instruments, for one octave of pitches over 12 time steps. The zeros and ones represent the absence and occurrence of a note respectively.

Because music is represented as binary data, variants of the CNN must be used for modeling it. The common max-pooling layer is not useful for this application because it acts as a OR-operator on binary data. The principle of locality is still valid, but works in a different way: while nearby image pixels are naturally similar, this is not necessarily the case for two neighbouring notes on a pianoroll. Notes which sound well and occur together often have certain intervals between them, meaning the principle of spatial locality could be shifted towards musical harmony for better results (see also section 5.2.4).

3.2 Estimation of probability distributions

Using the techniques described in the previous section, one can construct a model which reflects the probability distribution the input data is assumed to originate from. In context of algorithmic music composition, one can then sample from this distribution to generate notes. In this section, various sampling methods are discussed, along with alternative techniques to model complex probability distributions more efficiently.

For example, the goal of a sampling process may be to try to reconstruct the missing variables in an incomplete input pianoroll, which is a subset of a training example (the context C), of the form $\mathbf{x_C} = \{x_{i,t} \mid (i,t) \in C\}$. Via sampling, the pitch p of the variables $\mathbf{x_S} = \{x_{i,t} \mid (i,t) \notin C\}$ will be determined.

3.2.1 MCMC methods

To sample from a complex probability distribution, a popular option is to use Markov Chain Monte Carlo (MCMC) methods, a group of algorithms suitable for performing statistical sampling when analytical calculation is not an option. MCMC algorithms are popular for inference about high dimensional models and calculating posteriors in Bayesian modeling [49].

Often times, calculating the integral of a probability density function is computationally intractable. It is possible to approximate such integrals using MCMC algorithms [43].

Markov chains

A Markov Chain is a discrete sequential model, in which the probability of each event depends only on the state of the previous event. In other words, it is based on the Markov property of a stochastic process, which states that the probability of a future event, conditioned on present and past events, is only dependent on the present event. When considering the simple model illustrated in Figure 3.7, this means the probability of transitioning to a certain state depends only whether the current state is X_1, X_2 or X_3. How the process reached this current state, is irrelevant.

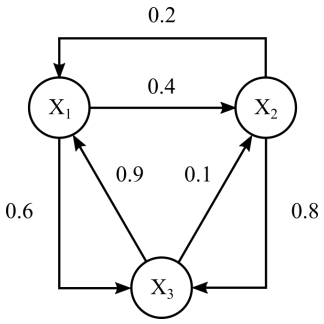


Figure 3.7: Markov Chain with three states, along with their transition probabilities.

The probabilities of transitioning from state i to state j, $P(j|i)$, can be portrayed in a transition matrix $\mathbf{P}$ as $P_{i,j}$. In general, for a Markov Chain of length N elements, the transition matrix $\mathbf{P}$ has the form:

$$\mathbf{P} = \begin{bmatrix} P_{1,1} & P_{1,2} & \dots & P_{1,N} \\ P_{2,1} & P_{2,2} & \dots & P_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N,1} & P_{N,2} & \dots & P_{N,N} \end{bmatrix}$$

with $\sum_{j=1}^{N} P_{i,j} = 1$, as the sum of all the probabilities starting from a certain state must be 1. For a simple three-state Markov chain like the one depicted in Figure 3.7, the transition matrix $\mathbf{P}$ becomes:

$$\mathbf{P} = \begin{bmatrix} 0 & 0.4 & 0.6 \\ 0.2 & 0 & 0.8 \\ 0.9 & 0.1 & 0 \end{bmatrix}$$

Such a Markov chain is said to have an equilibrium distribution $\mathbf{P_{eq}}$ when for every initial distribution $\mathbf{p_0}$:

$$p_n = p_0 \cdot \mathbf{P}^n \rightarrow P_{eq}$$

for $n \rightarrow \infty$. Now we arrive at the underlying assumption of the MCMC methods: because of the Markov property, the equilibrium distribution of the Markov chain converges to the distribution of interest, as long as the amount of samples taken is large enough [43]. Sometimes, the first samples are discarded as they are not yet representative of the distribution (the burn-in period).

The Metropolis-Hastings algorithm [36, 22] is one of the most famous methodologies designed to produce such a chain. It requires the Markov chain to be aperiodic, meaning the chain does not exhibit cyclic behaviour, and to be irreducible, meaning any state can be reached from any other state in a finite amount of moves. These properties give the chain the ergodicity property, which ensures the uniqueness of the equilibrium distribution [36].

Starting with an initial distribution x^0 and a proposal distribution q, one can iteratively perform algorithm 1 to produce samples x^t that follow the desired distribution P.

Algorithm 1: Metropolis-Hastings

1 Given $x^{(t)}$
2 Sample $x^* \sim q\left(x^* | x^{(t)}\right)$
3 Sample $u \sim \mathcal{U}_{[0,1]}$
4 Calculate $\rho(x^t, x^*) = min\left(1, \frac{P(x^*)q(x^t|x^*)}{P(x^t)q(x^*|x^t)}\right)$
5 **if** $u < \rho(x^t, x^*)$ **then**
6 $\quad x^{t+1} = x^*$
7 **else**
8 $\quad x^{t+1} = x^t$

Note that during step 5 (the rejection step), an exact computation of $p(x)$ is not needed, merely the ratio of $\frac{P(x^*)}{P(x^t)}$. This means that the normalization constant of the full probability expression does not need to be known, and a different function $f(x)$ that is proportional to $P(x)$ can be used [22]. Furthermore, step 5 theoretically ensures reaching the true equilibrium distribution P for every arbitrary proposal distribution q, in practice it is detrimental for the performance of the algorithm. A poor choice for q may lead to very high computation times [43].

This algorithm can be used to draw samples for approximating a complex probability distribution, or for calculating the expected value of a variable via integration.

Often, the first n amount of samples of the chain are left out as they are not yet representative of the distribution, the so called burn-in samples [43].

3.2.2 Gibbs sampling

Gibbs sampling can be seen as an extension of the Metropolis-Hastings algorithm. Even though they share the same goal, there are two large differences between the them: Gibbs sampling methods accept all proposals and the algorithm is based on conditional probabilities.

The mechanism of basic Gibbs sampling will be illustrated in a simple example, with two variables x and y and a distribution f (inspired by [12]). Given are the conditional probabilities $p(x|y)$, $p(y|x)$ and an initial starting value x^0. The first value for y can be sampled from $p(y|x)$, which in turn allows the next value for x to be generated from $p(x|y)$. Iteratively, x^i and y^i can be sampled from:

$$\begin{aligned} X'_i &\sim f\left(x \mid Y'_{i-1} = y'_{i-1}\right) \\ Y'_i &\sim f\left(y \mid X'_i = x'_i\right) \end{aligned}$$

yielding a sequence of n sampled variables, representing f:

$$x^0,\ y^0,\ x^1,\ y^1,\ x^2,\ y^2,\ \ldots\ x^n,\ y^n$$

As it takes some time for the sample distribution to converge to the underlying distribution, a burn-in time is often used. Let's consider this in a musical context. A neural network architecture can learn a probability distribution (or model) based on input data. With Gibbs sampling, it is possible to generate notes or pieces of a composition belonging to this model, without having to evaluate it explicitly.

A variation on this technique is called 'blocked Gibbs sampling', in which a subset of variables is sampled, based on joint conditional probabilities [24]. Practically, this is implemented by adapting the sampling masks, which indicate if a note must be sampled on a position or not through ones and zeros. During generation, this strategy allows a section of the composition to remain untouched, for example when extending a priming sequence. In general, the model is given a part of the pianoroll (the 'context') $\mathbf{x}_C = \{\mathbf{x}_{(i,t)} \mid (i,t) \in C\}$ and is asked to generate the remaining parts $\mathbf{x}_{\neg C}$.

3.2.3 Neural Autoregressive Distribution Estimation

Within the framework of unsupervised probabilistic learning, Neural Autoregressive Distribution Estimation (NADE) is a neural network architecture which provides an alternative method for modeling the distribution $p(x)$ of an input vector $\mathbf{x}$ [53]. The simple version is efficient because a part of the model parameters is shared and it uses the probability product rule. For a D-dimensional input vector $\mathbf{x}$ (with a distribution $p(\mathbf{x})$) in order o (a permutation of the integer sequence 1, 2, ..., D), the latter can be written as:

$$p(\mathbf{x}) = \prod_{d=1}^{D} p\left(x_{o_d} | \mathbf{x}_{o_{<d}}\right) \tag{3.2}$$

with $o_{<d}$ the first $d-1$ elements in ordering o, and $\mathbf{x}_{o_{<d}}$ the part of input vector x corresponding to those $d-1$ dimensions. These conditional probabilities are modeled via a feedforward neural network, where the probability $p(x_{o_d}|\mathbf{x}_{o_{<d}})$ depends only on the input variables with index 1 up to $d-1$, hence the autoregressive character.

For a model with H hidden units and an input vector x_o with dimension D, $p(x_{o_d}|\mathbf{x}_{o_{<d}})$ is computed as:

$$\begin{aligned} p\left(x_{o_d} = 1 | \boldsymbol{x}_{o_d}\right) &= \sigma\left(\boldsymbol{V}_{o_d,\cdot} \boldsymbol{h}_d + b_{o_d}\right) \\ \boldsymbol{h}_d &= \sigma\left(\boldsymbol{W}_{\cdot,o_d} \boldsymbol{x}_{o_d} + \boldsymbol{c}\right) \end{aligned} \tag{3.3}$$

with biases $\mathbf{b} \in \mathbb{R}^D$ and $\mathbf{c} \in \mathbb{R}^H$, $\mathbf{W} \in \mathbb{R}^{H \times D}$, $\mathbf{V} \in \mathbb{R}^{D \times H}$ and σ the logistic sigmoid. The interconnection matrix $\mathbf{W}$ and the biases c are shared by every hidden layer h_d, limiting the complexity to $O(HD)$. This is relatively small, because the parameter sharing allows the hidden unit activations to be computed recursively [54]:

$$\begin{aligned} \boldsymbol{h}_d &= \operatorname{sigm}\left(\boldsymbol{a}_d\right) \quad \text{where} \quad \boldsymbol{a}_1 = \boldsymbol{c} \\ \boldsymbol{a}_{d+1} &= \boldsymbol{a}_d + x_{o_d} \boldsymbol{W}_{\cdot,o_d} \end{aligned} \tag{3.4}$$

A simple example of a NADE model with a 5-dimensional input vector x_o and 3 hidden units per layer is depicted in Figure 3.8. The model can be trained in two ways, via maximum likelihood or via minimization of the negative log-likelihood by regularized or stochastic gradient descent [53, 54, 58]:

$$\frac{1}{N}\sum_{n=1}^{N} -\log p\left(\boldsymbol{x}^{(n)}\right) = \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} -\log p\left(x_{o_d}^{(n)} \mid \boldsymbol{x}_{o<d}^{(n)}\right) \tag{3.5}$$

Here is the catch: using a NADE, inference is performed via ancestral sampling while making use of the factorization of equation 3.2.3. This only works when the variables to be conditioned on are in the beginning of the ordering o, and the ones to be marginalized are in the back. This makes it computationally intractable to calculate any conditional probability, given just one ordering. Expanding the model towards a deep architecture with multiple hidden layers proves to be infeasible, as the parameters can no longer be shared. The solution for this problem is a order-agnostic training procedure called DeepNADE [53].

DeepNADE

In this extended procedure, the previously described problems can be solved while maintaining the advantages of the weight-sharing architecture. To represent every possible ordering, this network trains a factorial number $D!$ of NADE's with shared parameters, to preserve efficiency. The result is that conditionals can be computed for every ordering via the most convenient ordering for the desired inference task. As the trained models will not always be consistent with respect to each other, an

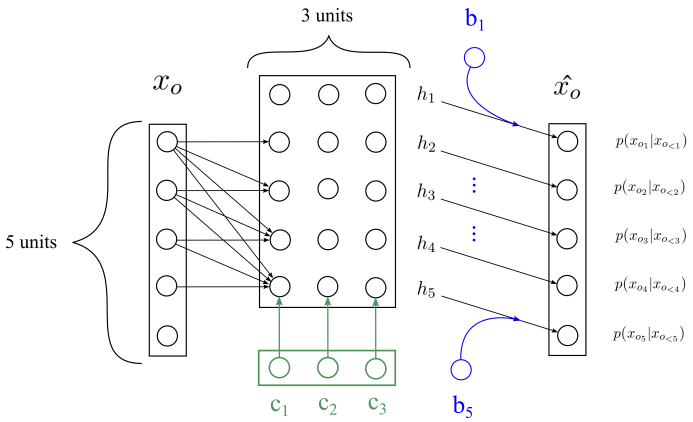


Figure 3.8: Schematic overview of the NADE model architecture for an input x_o with 5 units and a hidden layer with 3 units, with bias vectors **b** and **c**. For each dimension of the input, the output $\hat{x_o}$ gives the probability conditioned on previous elements in the ordering. For example, the probability $p(x_{o_3}|\boldsymbol{x}_{o_{<3}})$ for the 3^{rd} dimension of x_o depends only on the 1^{st} and 2^{nd} input values. Adapted from [53].

ensemble of all the models is used for estimations.

The likelihood function described in the previous section is optimized over all possible orderings o_d, and is approximated as:

$$\widehat{\mathcal{J}}(\boldsymbol{\theta}) = \frac{D}{D-d+1} \sum_{o_d} -\log p\left(x_{o_d} | \boldsymbol{x}_{o_d}, \boldsymbol{\theta}, o_{<d}, o_d\right) \tag{3.6}$$

3.2.4 Generative Stochastic Networks

An alternative method of estimating probability densities is offered by the Generative Stochastic Networks (GSN) framework. The idea behind this technique is to consider the distribution of interest as a Markov chain, and to estimate a state based on its predecessor via a learned transition operator [9]. This comes down to indirectly estimating the stationary distribution of a MCMC sampler [58].

Yao et al. [58] have proven that the training a NADE is equivalent to training a GSN, where the transition operator is defined as:

$$p\left(\mathbf{x}' \mid \mathbf{x}\right) \; = \; \sum_{\forall \mathbf{h}} p\left(\mathbf{x}' \mid \mathbf{h}\right) p(\mathbf{h} \mid \mathbf{x}) \tag{3.7}$$

This technique involves training two conditional probabilities $p(x' \mid h)$ and $p(h \mid x)$ which are normally easy to train because they consider small steps and thus contain few modes. The GSN is a quick learner and has been proven to find a good approximation of a distribution $p(x)$.

In [58], a method is proposed where one switches between expensive ancestral sampling (as is usually done in a NADE framework) and making a GSN Markov chain transition, trading off accuracy and speed.

Chapter 4

Models and methodologies

This thesis is based on a model called Coconet, the work of Huang et al. (2017) at Google's Magenta Project [24]. In the first section of this chapter, Coconet's architecture is explained in detail as it is reported in [24], which was designed to generate fixed-length monophonic Bach-style chorales with four voices. Next, the contributions of this thesis are described, which extend the model towards popular music-inspired scores of any sample size. Finally, the experimental set-up is described, consisting of the used datasets and model configurations.

4.1 Coconet

Let us start with the compositions x that serve as training data, which are presumed to be derived from an underlying distribution $p(x)$. As described previously in Figure 3.6, the four-voiced chorales are represented as a binary three-dimensional pianoroll $\boldsymbol{x} \in \{0,1\}^{IxTxP}$, with I the amount of instruments, T the amount of timesteps and P the amount of pitches. The pianorolls are cropped to $T = 64$ timesteps of 16^{th} notes, with MIDI pitches between 36 and 81 (giving P a size of $81 - 36 + 1 = 46$). A part of each composition is masked out so only a fragment (or 'context') $\boldsymbol{x}_C = \{\boldsymbol{x}_{(i,t)} \mid (i,t) \in C\}$ remains (blocked Gibbs sampling). The masked out pianorolls are concatenated with the masks themselves to obtain the input tensors $\mathbf{h}^0 \in \{0,1\}^{2IxTxP}$.

The inputs are fed into an orderless NADE architecture based on CNN cells. The CNN performs convolution for every instrument over the time and pitch dimensions. The masked out pianorolls and the masks themselves are considered two independent

channels to be convolved over. A 'straight' architecture is used where every layer preserves the pitch and time dimensions by using 'same' padding, which tries to pad evenly with zeros across all dimensions. In Table 4.1, an overview is given of the 64 filters with 128 channels each used in the default configuration, along with the filter shapes and the evolution of the data dimensions. For chorales with four voices, the input tensors $\mathbf{h}^0$ with dimensions $\{0,1\}^{8\text{x}128\text{x}46}$ are fed into the network in a shape of [128, 46, 8], and are returned with dimensions [128, 46, 4], corresponding to $\{0,1\}^{T\text{x}P\text{x}I}$. After each convolutional layer, batch normalization is performed to obtain a fixed mean and standard deviation, and after every second layer, a skip connection between layer l and $l-2$ is added to improve model convergence. The activations are then obtained through a ReLU-function.

Table 4.1: Overview of the different filters used in the CNN architecture. From left to right: amount of filters in succession, shape of the filter and evolution of the data dimensions.

# Filters	Filter shape	Data shape
		[128, 46, 8]: input $\mathbf{h}^0$
1	[3, 3, 8, 128]	[128, 46, 128]
61	[3, 3, 128, 128]	[128, 46, 128]
1	[2, 2, 128, 128]	[128, 46, 128]
1	[2, 2, 128, 4]	[128, 46, 4]: output $\mathbf{h}^L$

The final activations $\mathbf{h}^L$ are converted into predictions for every possible pitch p for an instrument i at time t by softmax normalization:

$$p_\theta\left(x_{i,t,p} \mid \mathbf{x}_C,\ C\right) = \frac{\exp\left(h^L_{i,t,p}\right)}{\sum_p \exp\left(h^L_{i,t,p}\right)}$$

.

With these, the loss function of the orderless NADE described in subsection 3.2.3 can be minimized using stochastic gradient descent, with the step size determined by the Adam optimizer. During the training phase of the model, the values of the filters are learned, of which so far only the shape has been described.

To sample from the model, an orderless NADE ancestral sampling technique would be the most obvious choice. However, the authors showed that independent blocked Gibbs sampling provided better results based on log-likelihood and human evaluation. This involves using the NADE-GSN crossover method which was explained in section 3.2.4. Here, the adapting masking probability described by Yao et al. [58] is written

at step n as:

$$\alpha_n = \max(\alpha_{\min}, \alpha_{\max} - n(\alpha_{\max} - \alpha_{\min})/(\eta N))$$

with N the total number of sampling steps, η the parameter controlling how much time is spent on high noise-level sampling compared to low-noise sampling (settling for $\alpha_{\min}$) and $\alpha_{\max}$ and $\alpha_{\min}$ the maximum and minimum probabilities respectively.

This scheme implies that during the first steps, the masking probability is high causing a high fraction of notes to be sampled independently. Later on, as α_n decreases, the samples evolve more towards the underlying distribution that is being approximated. Very low values for α_n cause only one variable to be sampled at each step, resembling standard Gibbs sampling.

A practical illustration of the workflow starting from a blank pianoroll (Figure 4.1) up to a finished sample (Figure 4.4) is shown below. During the early steps of the sampling process (Figure 4.2), α_n is relatively high and a lot of different notes are predicted by the model for each time step (red notes). Later on (Figure 4.3), α_n decreases which causes the process to become similar to regular Gibbs sampling. This causes the notes to be sampled less independently, making the pianoroll approximate the model distribution. This means some notes are sampled 'definitely' (green notes), with a very low chance of being resampled in the subsequent steps. The reason why some notes become definite, is because the model predicts this note to be sampled with a probability very close to one. Finally, after the last sampling step, the pianoroll is completed (Figure 4.4). It is important to note that the sample size of Coconet has an upper bound equal to the cropping size (64 timesteps).

Originally, this model was trained on a collection of 384 Bach chorales, a high-quality dataset created by M. Allan [3].

4.2 Extended model

In this thesis, the Coconet model is extended in two ways [1]. Firstly, a data processing protocol is designed to convert MIDI-files of any origin into the correct format to serve as input to the model. Secondly, the sampling framework of the model is adapted in three ways: from monophony to polyphony, from four to any number of instruments, and from a limited sample size to a larger sample size. The focus

[1] The full code repository can be found at: https://github.com/thomasdheer/CnnForPop.

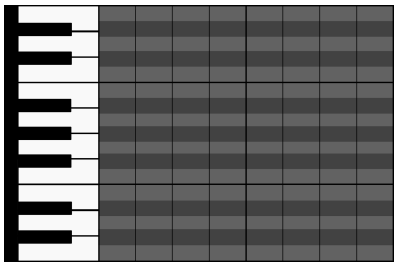


Figure 4.1: A blank pianoroll without any notes, the starting point of the sampling process.

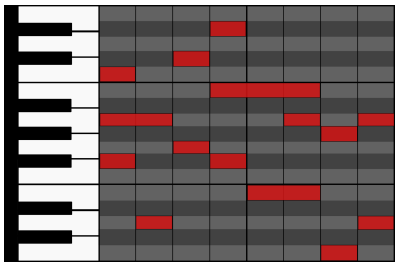


Figure 4.2: During the early steps of the sampling process, the model predicts multiple notes per timestep.

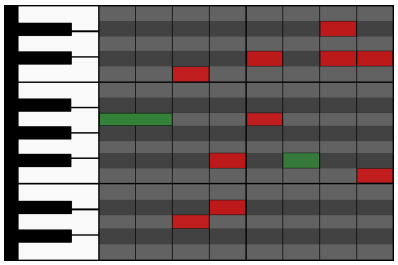


Figure 4.3: During later steps, some notes are sampled with almost certainty.

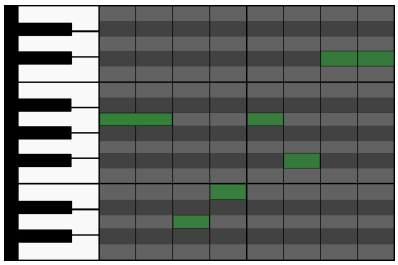


Figure 4.4: The resulting pianoroll after N sampling steps.

will be on the latter, which consists of a time-series inspired adaptation of Gibbs sampling.

4.2.1 Data processing

In deep learning, the quality of the data determines the quality of the research. For this reason, an algorithm was developed to extract useful information from a given MIDI-file. To conform to the requirements of the model, the format of the Bach-dataset was reverse-engineered. The following steps convert a MIDI-file into a 64-timestep pianoroll containing a characteristic part of the song, with quantization up to 16^{th} notes.

1. The pianorolls are extracted from the files for each of the present instruments via prettyMidi, a Python module created by Raffel et al. [42]. Only the first four instruments are used and if the file contains less than four instruments, it is skipped. When an instrument is recognized as a drum track, it is not used either.

2. The pianorolls are simplified from a binary form to integers representing the

pitches, quantized up to 16^{th} notes. For each time step, the resulting pianoroll contains a list with four elements, containing the note(s) every instrument plays at that time (one, multiple or none, denoted by 'not a number' or 'nan'). For example, the list [[55], [67], [47, 50, 53], [nan]] depicts a time step where instrument 1 and 2 play one note, instrument 3 plays three notes and instrument 4 plays none. The pitch values are squashed to the interval [36, 81] by raising or lowering the notes that exceed it by 12 (an octave) until they are inside. Here, double occurrences are removed.

3. The pianorolls are divided into fragments of 64 timesteps. To keep the most interesting fragment only, the one is chosen which maximizes a custom metric, rewarding the occurrence of many unique notes and penalizing the absence of notes ('nan', not a number). This prevents that an almost empty fragment is picked (like an intro or outro) which contains static notes (e.g. the same chords held throughout). Given that '#' signifies 'quantity', '# nan' the amount of times an instrument does not play a note in the fragment, 'nan_{max}' the maximum possible value for '# nan' (equal to 256 for 4 instruments and 64 time steps) and '# unique' the amount of different notes played in the fragment, the metric is calculated as:

$$\text{nan}_{max} \; - \; \#\text{ nan} \; + \; 4 * \#\text{ unique}$$

4. The data is randomly split in training, test and validation sets according to the ratio 60/20/20.

5. The sets are stored as three Numpy arrays (.npy) and compressed into one .npz file. The model can be trained by providing the path to this compressed file.

This procedure must be accompanied by some remarks involving its assumptions and limitations:

- In step 1, files where all notes are contained in one track are skipped, as it is hard to extract which notes originate from a certain instrument. This eliminates a significant part of any online database.

- The fragment selection in step 3 at times promotes the occurrence of chords.

- It is assumed that all songs are in the 4/4 time signature, and that the 64-length fragments will more or less line up. In most cases, this is observed to be so. A well-performing algorithm detecting the start of musical phrases (e.g. a chorus) would improve the alignment of the fragments.

- The quantization up to 16^{th} notes prohibits detailed modeling of songs with 32^{th} notes, which seldomly occur in pop music.

Note that monophonic and polyphonic voices (an instrument playing one vs. multiple notes at a time) are modeled in the same way. Implementing a CNN with variable depth could allow the model to learn a custom structure for every track (also see Future works in section 6.2).

4.2.2 Moving-window sampler

To explore the possibilities of the model in context of pop music, the sampling mechanism was adapted in three ways. Firstly, the code which was custom-made for compositions with four instruments was extended to accept songs with any amount of instruments. Secondly, polyphony was enabled by introducing a second or third note per time step per instrument, based on predicted probabilities. Here, extra notes are only added if the probabilities are within a certain order of magnitude of the highest p-value (the originally selected note). As these concepts have been implemented before (see related works in section 2.2), they will not be investigated in detail.

Lastly, a new sampling method is proposed which can extend samples in time using a moving window. Inspired by how RNN's use input from previous time steps, this technique slides a window of a fixed size over a timeline, where at each position a sampling phase occurs. Here, a new piece of music is added to the composition, based on the previous parts. This mechanism is similar to the workflow of AR models, where a new value is generated based on the values from previous timesteps. Choosing the amount of timesteps the window moves at each phase, the stride, is crucial for the character of the resulting composition. For example, when setting the stride to one bar, each sampling phase will respect and continue the division of the song into bars. When setting it to a random amount of timesteps, the onset of beats will be highly unstructured.

The following figures illustrate this mechanism for a one-instrument pianoroll containing 8 notes, where a window of width 4 was moved across a given input note (also called 'primer', denoted by a blue colour) with stride 1. In Figure 4.5, the first window is shown with the primer on the first position from the left, and predicted notes on the next three positions (shown in red). After N sampling steps, these predicted notes converge to one note per timestep. These determined notes are shown in green (Figure 4.6). Next, the pianoroll is extended by one note by sampling a window of the same size, which starts one position further (Figure 4.7). During this sampling process, the primer is the left-most green note which was sampled in the previous step. After repeating this procedure three more times, the pianoroll was lengthened to 8 notes 4.8.

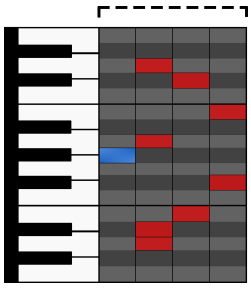


Figure 4.5: In the first sampling window (denoted by the bracket), the first note (the 'primer') is given as input and is left unchanged (blue), while new notes are predicted for the next three positions (red).

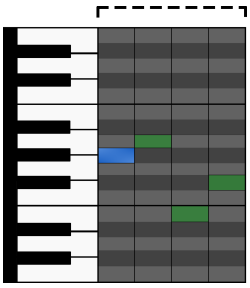


Figure 4.6: After N Gibbs sampling steps, the unchanged primer has been elongated with three newly sampled notes (green). The green note which immediately succeeds the blue note will serve as the primer for the next step.

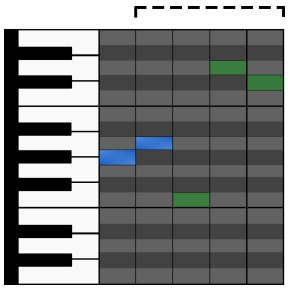


Figure 4.7: The sampling window moved up one spot (now covering note 2 to 5), where the process repeats itself.

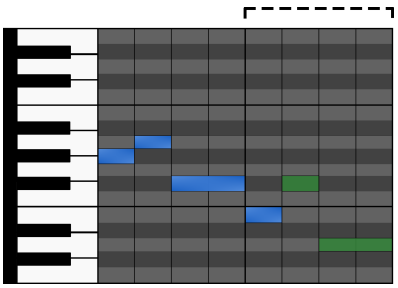


Figure 4.8: The final pianoroll of length 8, the result of sampling a window of width 4 over 5 positions.

4.3 Experimental set-up

4.3.1 Data

The datasets were extracted in various sizes from large internet-collections to attain modern music datasets. These collections contain various contemporary genres like pop, rock and dance music, but for simplicity they will be referred to as pop music datasets.

First of all, a small ('lakh-300', 300 songs) and a large dataset ('lakh-10K', 10000 songs) were extracted from the Lakh dataset (a collection containing more than 170.000 songs, created by C. Raffel [41]).

Secondly, the same was done for 'The Largest MIDI Collection on the Internet', a database containing more than 100.000 tracks created by a Reddit-user [38], resulting in two sets 'red-300' and 'red-10K'. As the files from both collections have been amassed from publicly available sources, they are not fully reliable. It is unknown whether they were man-made or extracted algorithmically from audio files via conversion tools. Still, using these collections is the best option for modeling modern music symbolically.

Thirdly, a small dataset of 100 pop songs was constructed based on manually checked files from these internet collections ('man-100'). The motivation for checking the files manually is to filter out corrupted ones. A file is considered corrupted when the notes are misaligned in comparison to the beats, or when clear mistakes occur in the notes (when they sound 'wrong' or dissonant).

All the MIDI-files extracted from the mentioned collections are passed through the data processing algorithm described in section 4.2.1. In summary, there are five datasets: 'lakh-300', 'lakh-10K', 'red-300', 'red-10K' and 'man-100'.

4.3.2 Model architectures

For training the CNN, various model configurations have been tested, with different values for the following hyperparameters: amount of layers (16 to 96), amount of filters (32 to 192), filter size ([2,2], [3,3] or [5,5]), learning rate (from 0.001 to 0.3), batch size (from 10 to 1000). The impact of different filter sizes was very small, and all learning rates within the interval [0.01, 0.1] yielded similar results. Thus, the filter size and learning rate are fixed at [3, 3] and 0.0625 for all models. The input to the models conformed to the output of the data processing algorithm, namely a size

of 64 timesteps quantized to 16^{th} notes.

Three models are investigated with a small, medium-sized and large architecture (16, 64 and 96 layers respectively). A summary of their hyperparameters is shown in Table 4.2. The basic structure of all models is the same as the Coconet model described in section 4.1, where all convolutional layers have the same filter size [3, 3], except for the last two (filter size [2, 2]). The medium-sized model of 64 layers is similar to the original Coconet model.

All three models were trained on every dataset. For the small datasets of size 100 and 300, the batch size for the 16-layer model was adapted to 20 and 50 respectively. For the medium and large architectures (64 and 96 layers), the set batch sizes of 50 and 25 were the maximal values which did not lead to resource exhaustion during computation. The use of separated convolution has been explored, but as it did not yield better results nor spared a significant amount of time, it was not implemented. For each training session, the best model was saved after 300 epochs based on validation accuracy. For all configurations, samples of length 32 timesteps are generated from scratch, and after initialization by a primer melody.

Table 4.2: Hyperparameters of the three model architectures used during the experiments. The medium-sized 64-layer model is similar to the original Coconet model.

	Small	Medium	Large
Layers	16	64	96
Filters	32	128	192
Filter size	[3, 3]	[3, 3]	[3, 3]
Batch size	1000	50	25
Learning rate	0.0625	0.0625	0.0625

For the moving-window sampling experiments, a window of 64 timesteps was used with a stride of 16 to create a composition of 128 timesteps in total. This comes down to five sampling phases: from timestep 1 to 64, 17 to 80, 33 to 96, 49 to 112 and 65 to 128. The process is illustrated on Figure 4.9. Note that a stride of 16 timesteps corresponds to one bar, promoting the traditional 4/4 time signature by conserving its division of the timeline into bars. For all of these experiments, a primer melody is used.

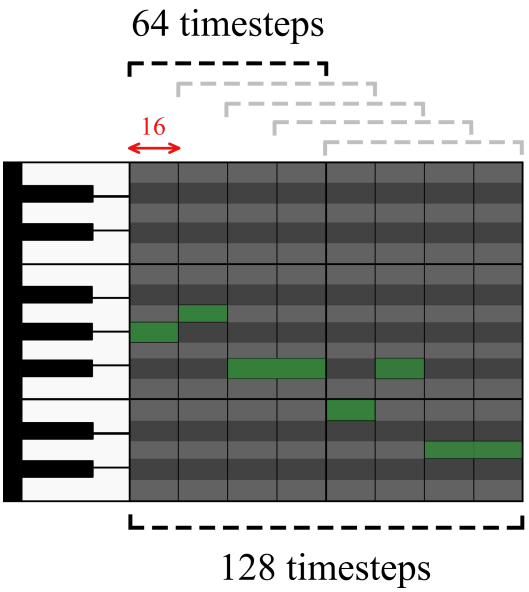


Figure 4.9: Illustration of the moving-window sampling procedure. In each of the five phases, the 64-timestep window is moved up by 16 steps, resulting in a total length of 128.

Remark: the computational resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation Flanders (FWO) and the Flemish Government department EWI.

Chapter 5

Results and discussion

In this chapter, the samples produced by the various CNN architectures are described and discussed. Three types of samples are examined: short generations from scratch (32 timesteps), short primer initialized generations (32 steps) and longer primer initialized moving-window sampler extensions (128 steps). Results and discussion are split into two separate sections, 5.1 and 5.2.

5.1 Results

Experiments were conducted based on the three models mentioned in section 4.3.2, trained on the five different datasets. The results will be described in words, based on the critical evaluation of the author. In case it provides additional value, the pianoroll of a produced sample will be depicted for clarification. For a full discussion of these results, see section 5.2.

First of all, it was observed that all models based on the small datasets ('lakh-300', 'red-300' and 'man-100') produced very poor results, lacking musicality and consonance. Secondly, training on the 'red-10K' dataset led to the occurrence of dissonant notes and lacked a sense of timing and rhythm. Due to its randomness, it was clearly inferior to the lmd-10k dataset. The same shortcomings were present in all models trained with the small, 16-layer architecture. Because of these reasons, only the lmd-10k dataset in combination with the 64- and 96-layer architectures will be considered from this point forward, as these combinations achieved good results.

5.1.1 Generation from scratch

When generating from scratch, the 64-layer model produced samples which contained little sense of musicality. They contained an almost random sequence of notes which are consonant, but there is no consistent structure or sense of rhythm. The 96-layer model performs slightly better, but still not good enough to produce good sounding melodies.

5.1.2 Generation based on primer melodies

When initializing with a primer melody, the 64-layer model generated highly musical fragments which extended the 'feel' of the primer. The key of this primer is usually conserved and a sense of rhythm is present, which means the notes are played on intuitively correct moments relative to the beats. By consequence, the complete sample can be played in a loop, resulting in a fluid piece of music. The amount of voices generated varies from one to the set amount (four). As it is a stochastic model, it is evident that not every sample is of the same caliber and quality. Interestingly, sampling from the 96-layer model after priming identically yields worse results. The produced sequences are more complex, more random and stray away from the priming key more often. There is still a sense of musicality, but the continuation of the primer in terms of rhythm and mood is no longer there.

In Figure 5.1, one of the samples from the 64-layer model is depicted in the form of a pianoroll, where each different colour represents a different voice. In this sequence, there is an almost constant interplay of four voices. The composition is also illustrated in sheet music notation in Figure 5.2, which adds extra information such as the tempo and time signature, which in this case are set to the standard values, 120 BPM and 4/4.

5.1.3 Moving-window sampler

The longer sequences (128 steps) which were generated with the moving-window sampler after priming, showed different characteristics than the shorter ones. The 64-layer model produced consonant, musical pieces containing variations of the priming theme, while expressing different dynamics. For example, the primer (which consisted of a chord and a lead melody) might be extended with a simple, single-voiced lead melody which evolved into a counterpoint-like four-voiced harmony. Overall, the

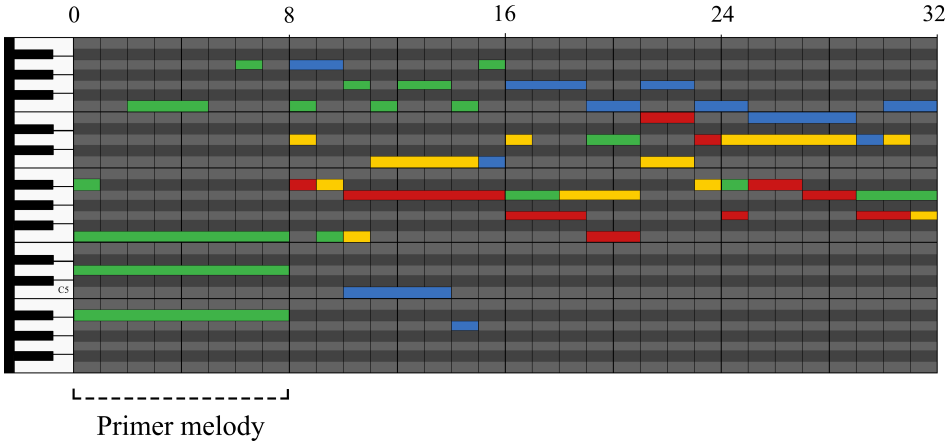


Figure 5.1: Pianoroll representation of a 32-timestep sequence, sampled by the 64-layer model after primer initialization. The different colours represent the four different voices.

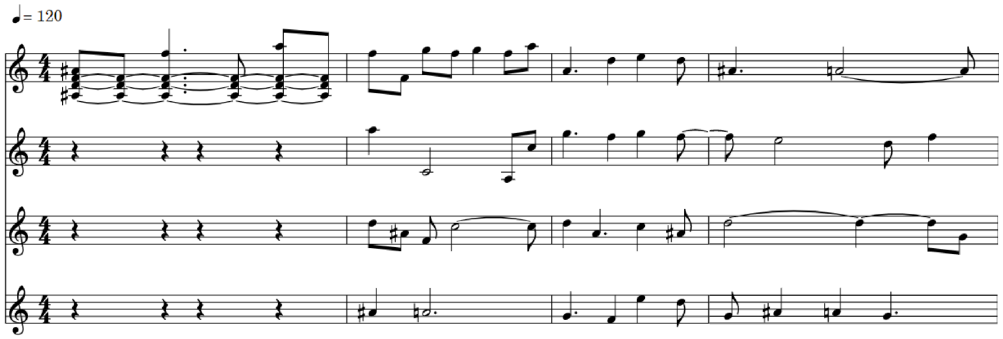


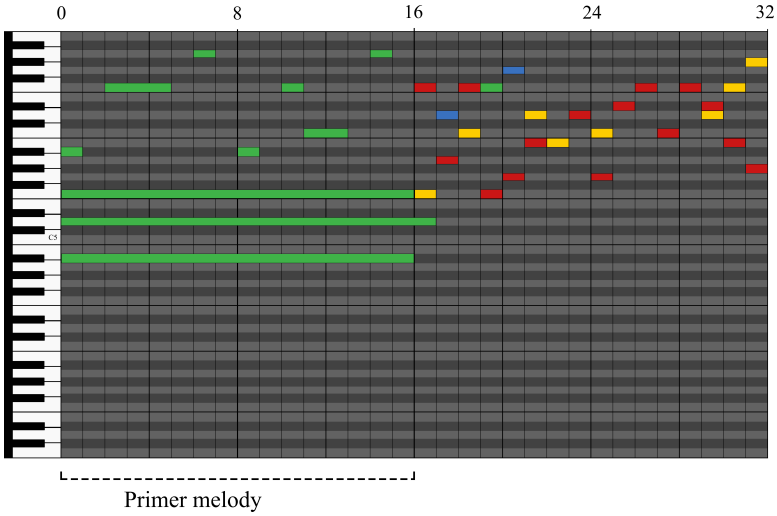
Figure 5.2: Sheet music notation of the 32-length sample with four voices, generated by the 64-layer model. The standard settings of the tempo and time signature are 120 BPM and 4/4 respectively.

key, rhythm and mood of the primer are seldomly conserved over all 128 steps. The samples contain fresh elements spread over their sequences, but also tend to repeat certain sequences too often to be considered good pop songs.

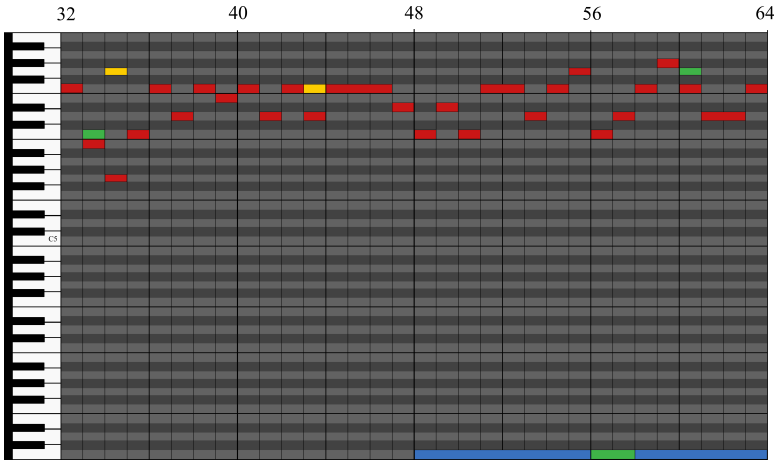
An example of such a 128-step sample is illustrated in four 32-step pieces in Figure 5.3. The remarkable aspect is not the notes played, but the clearly visible variation in dynamics and harmony. In timestep 0 to 32 (Figure 5.3), the priming sequence (containing a chord and a lead melody) is continued by a two-voiced harmony. At timestep 48 (halfway Figure 5.3b), the sole remaining voice is joined by a bass note (depicted in blue, briefly overlapped by a third voice in green). Then, at timestep 80, the bass note evolves into a changing melody and a third voice is added (in yellow, Figure 5.3c). Finally, in the fourth part of the sample (timestep 96 to 128), the

sequence blossoms into an active harmony with three or four voices, exhibiting a counterpoint-like movement (Figure 5.3d).

For the 96-layer architecture, the time-extension severly accentuated the randomness which was observed in the short samples. The resulting compositions contained many dissonant notes and lacked structure or musical consistency.

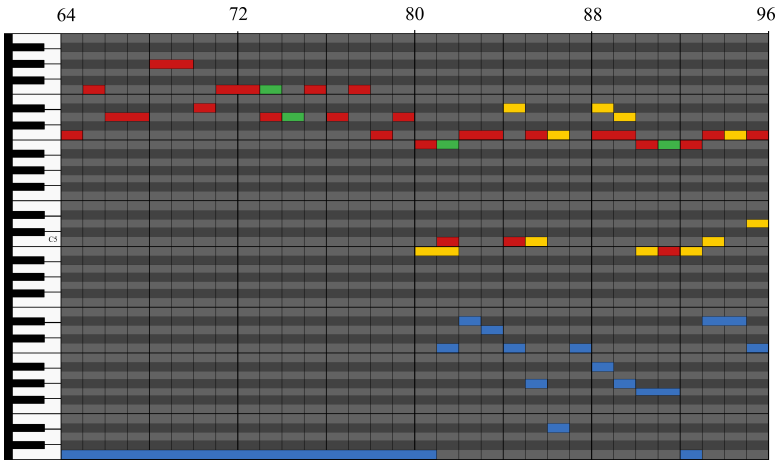


(a) Timestep 0 to 32: the priming sequence consisting of a chord and a lead melody is extended by two voices.

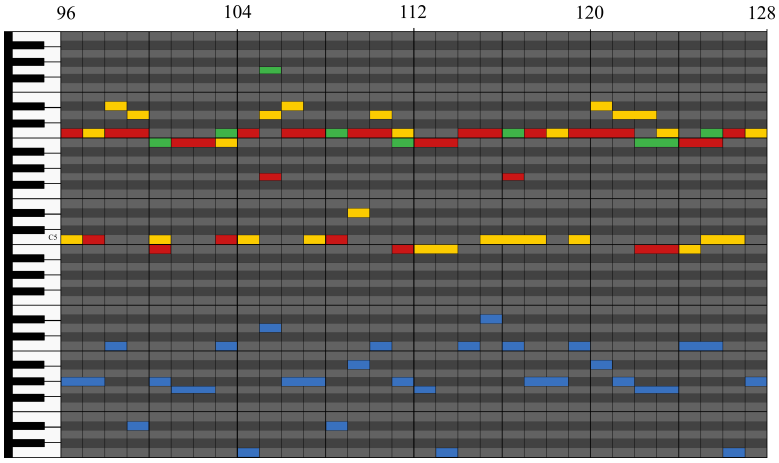


(b) Timestep 32 to 64: two voices are halfway (at timestep 48) joined by a bass note (in blue).

Figure 5.3: Pianoroll representation of a 128-timestep moving-window sample, where four colours represent four different voices. The essence of the four schemes is that various types of harmony occur within the same sequence.



(c) Timestep 64 to 96: the bass note (in blue) evolves into a changing bass line and a third voice is added (in yellow), resulting in three simultaneous voices.



(d) Timestep 96 to 128: counterpoint-like harmony between three, at times four voices.

Figure 5.3: Pianoroll representation of a 128-timestep moving-window sample, where four colours represent four different voices. The essence of the four schemes is that various types of harmony occur within the same sequence (cont.).

5.2 Discussion

In this section, several topics are examined in detail. After discussing the importance of data, possible explanations for the observed results described in section 5.1 are offered, whose characteristics varied strongly. Next, the reasons why training on pop music is so challenging are discussed, and comparisons with training on Bach chorales are drawn. Lastly, the difference between using CNN's in imaging or music is mentioned along with possible applications for the designed model.

5.2.1 Data

While the original model was trained on only 384 input compositions, this strategy did not work for pop music. As these songs vary severely in structure, overfitting on a small dataset does not yield useful results. Furthermore, the effect of ensuring the file quality manually was non-existent for such a small data size.

A second observation is that the models trained on the internet collection created by [38] were of a worse quality than those trained on the Lakh dataset by [41]. As a large number of input files was used (10.000), this difference might be significant, potentially due to a higher presence of corrupt files.

Overall, acquiring accurate scores of pop music in symbolic format is challenging. The internet collections that were used contain files with mistakes, resulting from poor transcription by the author or imperfect automatic extraction. Moreover, the designed data processing tool may also yield imperfect fragments, which are not practical to check for large datasets. Besides incorrect notes, the choice of the cropping interval is of great importance. Off-beat cropping makes it harder for the model to learn conventional song structures, and a wrong sampling frequency leads to an unrepresentative sample of the song. In this step, the processing algorithm relies on the prettyMIDI functionality [42] to detect the occurrence of the beats correctly. Figure 5.4 depicts the consequence of creating poor crops of a given sequence (left). When a fragment is selected starting on the indicated beat ('on' beat extraction, top right), the resulting crop contains the same musical idea as the original sequence because the timing is correct. When the cropping starts one step after the beat ('off' beat cropping, bottom right), a sequence of a very different kind is created. As the model receives many different inputs, it may be impossible for it to learn conventional beat placement. The question remains whether or not this is desired (see section 5.2.5).

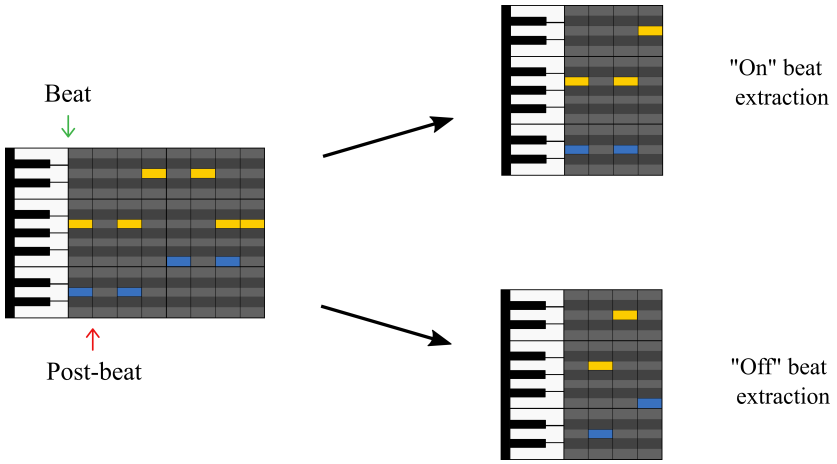


Figure 5.4: Illustration of the importance of on-beat cropping via pianorolls. When a fragment of the sequence on the left is cropped starting on the beat ('on' beat extraction, top right), the idea and structure of the original sequence are conserved. When cropping one step after the beat ('off' beat extraction), a completely different fragment is created.

A second function of the prettyMIDI function is to ignore the drum tracks in the MIDI files. This is crucial as drum sequences are also coded in notes (e.g. the MIDI-note 38 stands for the D1 note, but for a drum track it marks a hit of a snare drum), but they disrupt the composition when recognised as a pitched instrument. Using full songs as input is not possible, as the CNN has a fixed input size. Namely, scaling songs to the same size eradicates temporal structure, and using the first N amount of timesteps for every song leads to unrepresentative fragments (e.g. intro's often contain few notes and follow less strict conventions).

Note that the proposed framework not only produces samples with a variable amount of voices, but also accepts training compositions with any amount of voices, which can be monophonic or polyphonic. This contributes to its versatility as a songwriting tool.

5.2.2 Sample generation

The first observation is that the trained models were not optimal for generating compositions from scratch. Without a primer sequence, all notes have a high prediction probability which results in almost random compositions.

The 64-layer model, trained on the 'lmd-10K' dataset, produced musically appealing results, which suggests the model succeeded in extracting a sense of tonality and rhythm. However, the produced melodies are not coherent and contain many musical

ideas simultaneously. Note that the training data contains four voices in a mixed order: any voice could contain a lead melody, a harmony to the lead, a bass line, chords, or another type of part. Due to this mixing, every instrument in a produced sample could receive any type of part, from complete silence to an attention-grabbing melody. The possibility of creating counterpoint-like independent melodies is a double-edged sword. On one hand, the produced sequences are interesting, but on the other hand they lack typical pop characteristics. A chaotic interplay of four active voices deviates from traditional pop songs, where the bass instrument plays a simple line which changes relatively little, accompanied by a lead melody which catches the attention (e.g. vocals).

The samples indicate that the training compositions were processed successfully, but that the crucial structure of the songs was not always captured by the model. Therefore, it is suggested that the model may benefit from experimenting with different kernel sizes and expanding the amount of filters in the convolutional layers. Another option to improve the results is to introduce adaptive modeling for each separate instrument track. This would prevent monophonic and polyphonic voices from being processed identically (see also section 6.2). Lastly, it is noted that the fragment selection metric described in section 4.2.1 must be adapted, as it promotes chords too much. To prevent this, an extra parameter could be introduced which rewards separate, non-held notes.

The experiments with a larger model (96-layers) attained poor results, for which there are three possible explanations. First of all, an architecture of this size might elicit overfitting. Secondly, a small batch size of only 25 had to be used due to resource exhaustion of the supercomputer. This is not advisable as it yields inaccurate gradients. Thirdly, the model might have converged too early to a suboptimal solution. Better results could be achieved with a lower learning rate and very long training durations (multiple days). For future experiments, a lower learning rate might also prove beneficial for the 64-layer model, because rather high values have been used (between 0.001 and 0.3) to limit the computation time.

It was also shown that using a less complex model (16 layers) is not an option for modeling this type of data. Instead of avoiding overfitting on the irregular data and extracting the essence of the music, these simple models failed to capture any musicality at all, even though a larger batch size (1000) could be used.

The descriptions of the regular samples are valid for the moving-window time-extensions as well. The phenomenon of mixing was clearly observed: the example

sequence on Figure 5.3 illustrated the occurrence of different types of interplay between a changing amount of voices. The stochastic nature of the model becomes visible: all the learned elements (bass, chords, leads) which have been learned at once, come to expression at different times. On the other hand, some of the generated samples tend to get stuck in patterns. As described earlier, many interesting ideas arise, but none of the samples can be considered good songs as a whole. The priming theme is never repeated explicitly and the original key and atmosphere are seldomly conserved.

The added value of the time-extended samples compared to the regular ones, is that the composition length can be set to any amount of timesteps, and that more different dynamics are generated based on the priming idea. This method is a first attempt at combining the strengths of a CNN and a RNN or LSTM. While a CNN is easier to train due to sparse connectivity and lower data dimensionality, the RNN's temporal recurrence is suitable for modeling longer sequences.

5.2.3 Bach chorales vs. pop music

The original Coconet model was trained on a small-sized dataset of Bach chorales. These compositions are highly structured, similar, monophonic and have a more or less constant amount of notes per timestep. Of all the composition types, chorales respect the classical rules of music the most. Furthermore, the quality of the MIDI-dataset is very high.

In contrary, pop music is polyphonic and much more unpredictable. It contains exceptions to many musical rules and dissonant notes which do not fit in classical music theory. Often, these irregularities give character to the songs, but they make modeling much more difficult. Furthermore, as described earlier, the quality of the datasets is problematic.

Moreover, pop songs not only contain irregular notes but also a more irregular sense of timing. Instead of constantly playing four notes each bar like in Bach chorales, there is a variety of notes (from whole notes to 32^{nd} notes) which are not always perfectly aligned with a beat. Thus, when extracting fixed size fragments from the data, the sampling frequency is an important design choice (see section 5.2.1). For these reasons, modeling pop music requires a different strategy than modeling Bach chorales.

An advantage of using pop music is that it suits a stochastic model better than

the highly structured Bach chorales. The unpredictability can be desired when the model is used as a composition tool.

5.2.4 Imaging CNN's vs. music CNN's

In general, the CNN was originally developed for applications like image processing, where the convolutional layers extract features based on spatial locality. As was briefly mentioned in section 3.1.4, this principle is not perfectly suitable for modeling music and can be improved in various ways.

For images, the metric used in defining locality is the distance between the pixels. In music, which is often represented in a binary way, the features to be extracted by the pitch-convolutions are not shapes, expressed through continuous pixel values in a group of pixels. Instead they are chords, octaves and other intervals between non-neighbouring notes. The time-convolutions however do work similarly. When modeling Bach chorales, this limitation does not manifest itself strongly because the voices are monophonic, which means only convolutions over time and not over pitch are relevant. However, the pop samples indicated not all the important tonal relationships of the training data were captured. It is suggested that a specialized CNN-variant with a metric for harmony in its convolutional layers could improve the results significantly.

5.2.5 Purpose of the designed model

The results revealed the strengths and weaknesses of the model. While it is capable of generating original ideas, it struggles with the structure and natural flow of pop music composed by humans.

If one's goal is to be able to create music which satisfies all the conditions a traditional pop song must meet, the development of the data processing algorithm and model can be steered in this direction (see section 6.2 on future work).

For now, this framework is supposed to be used by a musician as a songwriting tool. Based on a small idea, the model provides a plethora of suggestions which can be implemented in a composition. Due to the way the model works, the generated sequences are not restricted to traditional harmonisation. Its fresh perspective might be just what a musician needs to finish a song.

Chapter 6

Conclusion

6.1 General conclusion

In this thesis, a framework was designed based on CNN and NADE for modeling pop music, starting from a data processing pipeline up to the generation of compositions. The results showed that priming a sample with a given sequence led to musically appealing compositions, and the strengths and weaknesses of the model were discussed. In summary, while the produced samples contained original phrases of changing character, they were slightly chaotic and inconsistent in quality. Next, the challenges of creating a symbolic pop music dataset were illustrated.

Secondly, a novel AR-inspired moving-window sampler was proposed to overcome the fixed-size limitation of CNN's. By enabling the model to extend compositions in time, this technique is a first attempt at combining the strengths of the CNN and the RNN. While the extensions were continuous and expressed various musical ideas due to the stochasticity of the model, further development is needed to improve the coherence of the compositions.

Even though the model needs more refinement, the presented results showed its potential as a songwriting tool. Its versatility was shown in two ways: the formatting requirements are flexible, and the samples exhibit a myriad of dynamics and harmonies found in the training data.

This thesis lays the foundation for future work, which can go in many directions, some of which are described in section 6.2. The main subjects that require attention are the creation of a good dataset, the development of a specialized data extraction algorithm for the chosen application and the tuning of the convolutional neural

network architecture. When these are solved, the generated samples should be evaluated more thoroughly and the tool can be used in real life.

6.2 Future work

In the future, there are two main problems which should be addressed, the first of which is the construction of a large, high-quality MIDI-dataset of pop music. This must be accompanied by the development of an improved processing tool which extracts useful fragments from the songs. Herein, the definition of 'useful' depends on the goal in mind. For example, in order to recreate traditional pop songs in a 4/4 time signature, one must focus on cropping every fragment on the start of a bar to prevent misalignment. One could go even further by only extracting a specific part of the song, like verses or choruses. In past research, this feature extraction has been done via techniques like beat estimation, pattern recognition [20] and specific event detection [27]. Another possibility is to expand the data processing so channels can be connected with specific instrument types (e.g. voice #1 produces bass notes, voice #2 produces lead melodies, etc.), allowing the user to set the kind of voices to be generated. In order to attain a complete songwriting tool, drum track generation could also be included in the model. In general, various constraints can be implemented to generate the desired compositions. However, one must keep in mind that more extensive data handling decreases the 'naturality' of the training data.

The second component is improving the CNN itself. First of all, more experimentation needs to be done on the size of the model and its hyperparameters. Next, the option for polyphony could be refined. Furthermore, as the amount of notes in the instrument tracks can vary, implementing a CNN with a variable depth could be beneficial. This challenging extension would allow each voice to learn a separate structure. Thirdly, the whole framework could be extended towards accepting and producing any amount of instruments through easy-to-handle commands. Lastly, the principle of a metric for harmony (as described in section 5.2.4) could be implemented, evolving towards a music-specialized CNN architecture.

Once the model has been perfected, the produced compositions could be evaluated in depth based on objective metrics and human evaluation trials.

Bibliography

[1] Agence France-Presse. First Recording of Computer-Generated Music - Created by Alan Turing - Restored, 2016.

[2] AI Business. Aiva is the first AI to Officially be Recognised as a Composer, 2017.

[3] M. Allan. Harmonising Chorales in the Style of Johann Sebastian Bach. *Master's thesis, School of Informatics, University of . . .*, 2002.

[4] M. Allan and C. K. I. Williams. Harmonising Chorales by Probabilistic Inference. *Advances in Neural Information Processing Systems*, (17):25–32, 2005.

[5] A. Alpern. Techniques for Algorithmic Composition of Music, 1995.

[6] G. Amato, M. Behrmann, F. Bimbot, B. Caramiaux, F. Falchi, A. Garcia, J. Geurts, J. Gibert, G. Gravier, H. Holken, H. Koenitz, S. Lefebvre, A. Liutkus, F. Lotte, A. Perkis, R. Redondo, E. Turrin, T. Vieville, and E. Vincent. AI in the media and creative industries. (April), 2019.

[7] Amper Music. Amper Music, 2020.

[8] Y. Bengio. *Learning deep architectures for AI*, volume 2. 2009.

[9] Y. Bengio, É. Thibodeau-Laufer, G. Alain, and J. Yosinski. Deep generative stochastic networks trainable by backprop. *31st International Conference on Machine Learning, ICML 2014*, 2:1470–1485, 2014.

[10] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription. *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, 2(Cd):1159–1166, 2012.

[11] J.-P. Briot, G. Hadjeres, and F.-D. Pachet. *Deep Learning Techniques for Music Generation – A Survey.* 2017.

[12] G. Casella and E. I. George. Explaining the Gibbs Sampler. *American Statistician*, 46(3):167–174, 1992.

[13] H. Chen, Q. Xiao, and X. Yin. Generating music algorithm with deep convolutional generative adversarial networks. *2019 2nd International Conference on Electronics Technology, ICET 2019*, pages 576–580, 2019.

[14] F. Colombo, A. Seeholzer, and W. Gerstner. Deep Artificial Composer: A Creative Neural Network Model for Automated Melody Generation. (June 2018), 2017.

[15] D. Cope. Experiments in Musical Intelligence, 1996.

[16] I. G. Y. B. Courville and Aaron. Deep Learning Ian. *Foreign Affairs*, 91(5):1689–1699, 2012.

[17] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever. Jukebox: A Generative Model for Music. 2019.

[18] H. W. Dong, W. Y. Hsiao, L. C. Yang, and Y. H. Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 34–41, 2018.

[19] A. Elgammal, B. Liu, M. Elhoseiny, and M. Mazzone. CAN: Creative Adversarial Networks, Generating "Art" by Learning About Styles and Deviating from Style Norms. (Iccc):1–22, 2017.

[20] G. Goos, J. Hartmanis, and J. Leeuwen. *Lecture Notes in Computer Science*, volume 1716. 1999.

[21] G. Hadjeres, F. Pachet, and F. Nielsen. DeepBach: A steerable model for bach chorales generation. *34th International Conference on Machine Learning, ICML 2017*, 3:2187–2196, 2017.

[22] W. K. Hastings. Monte Carlo Sampling Methods Using Markov Chains and their Applications. *Biometrika*, 57(1):97–109, 1970.

[23] N. Hewahi, S. AlSaigal, and S. AlJanahi. Generation of music pieces using machine learning: long short-term memory neural networks approach. *Arab Journal of Basic and Applied Sciences*, 26(1):397–413, 2019.

[24] C. Z. A. Huang, T. Cooijmans, A. Roberts, A. Courville, and D. Eck. Counterpoint by convolution. *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017*, pages 211–218, 2017.

[25] C.-Z. A. Huang, C. Hawthorne, A. Roberts, M. Dinculescu, J. Wexler, L. Hong, and J. Howcroft. The Bach Doodle: Approachable music composition with machine learning at scale. 2019.

[26] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck. Music Transformer: Generating Music with Long-Term Structure. pages 1–14, 2018.

[27] Y.-S. Huang and Y.-H. Yang. Pop Music Transformer: Generating Music with Rhythm and Harmony. 2020.

[28] S. Lattner, M. Grachten, and G. Widmer. Imposing higher-level structure in polyphonic music generation using convolutional restricted Boltzmann machines and constraints. *Journal of Creative Music Systems*, 2, 2018.

[29] S. Leadbeater. How AI is Revolutionising the Classical Music Industry: An Analysis of the Musical AI by Aiva Technologies, 2019.

[30] F. Liang, M. Gotham, M. Johnson, and J. Shotton. Automatic Stylistic Composition Of Bach Chorales With Deep LSTM. *International Society for Music Information Retrieval Conference*, 2017.

[31] R. Loughran and M. O'Neill. The Popular Critic: Evolving Melodies with Popularity Driven Fitness. In *Musical Metacreation (MuMe)*, number June, 2016.

[32] R. Loughran and M. O'Neill. *Evolutionary music: applying evolutionary computation to the art of creating music*. Number 0123456789. Springer US, 2020.

[33] Magenta (Google). Magenta Research.

[34] J. Maurer. A Brief History of Algorithmic Composition, 1999.

[35] A. McLean and R. T. Dean. *The Oxford Handbook of Algorithmic Music*. Number March. Oxford University Press, 2018.

[36] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[37] U. Michelucci. *Applied Deep Learning*. 2018.

[38] midi_man (Reddit). The Largest MIDI Collection on the Internet, 2016.

[39] S. Oore, I. Simon, S. Dieleman, D. Eck, and K. Simonyan. This Time with Feeling : Learning Expressive Musical Performance. pages 1–24, 2018.

[40] OpenAI. MuseNet, 2019.

[41] C. Raffel. *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD thesis, 2016.

[42] C. Raffel and D. P. W. Ellis. Intuitive Analysis, Creation and Manipulation of Midi with pretty _ midi. *Proceedings of the International Society for Music Information Retrieval Conference*, 2014.

[43] C. P. Robert. The Metropolis-Hastings Algorithm. *Wiley StatsRef: Statistics Reference Online*, (Mcmc):1–15, 2015.

[44] A. Roberts, J. Engel, Y. Mann, J. Gillick, C. Kayacik, S. Nørly, M. Dinculescu, C. Radebaugh, C. Hawthorne, and D. Eck. Magenta Studio: Augmenting Creativity with Deep Learning in Ableton Live. 2019.

[45] H. Salman, J. Grover, and T. Shankar. Hierarchical Reinforcement Learning for Sequencing Behaviors. 2733:2709–2733, 2018.

[46] A. M. Sarroff and M. Casey. Musical audio synthesis using autoencoding neural nets. *Proceedings - 40th International Computer Music Conference, ICMC 2014 and 11th Sound and Music Computing Conference, SMC 2014 - Music Technology Meets Philosophy: From Digital Echos to Virtual Ethos*, 1(1):1411–1417, 2014.

[47] C. Schmidt-Jones and R. Jones. *Understanding Basic Music Theory*, volume 22. 2000.

[48] M. Scirea and P. W. Eklund. MetaCompose: A Compositional Evolutionary Music Composer. In *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, 2016.

[49] J. S. Speagle. A Conceptual Introduction to Markov Chain Monte Carlo Methods. 2019.

[50] The MIDI Association. MIDI History: Chapter 6-MIDI Is Born 1980-1983.

[51] The MIDI Association. Details about MIDI 2.0™, MIDI-CI, Profiles and Property Exchange, 2019.

[52] P. M. Todd. Connectionist Approach to Algorithmic Composition. *Computer Music Journal*, 13(4):27–43, 1989.

[53] B. Uria, M. A. Cote, K. Gregor, I. Murray, and H. Larochelle. Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17:1–37, 2016.

[54] B. Uria, I. Murray, and H. Larochelle. A deep and tractable density estimator. *31st International Conference on Machine Learning, ICML 2014*, 1:719–727, 2014.

[55] A. Vaswani, N. Shazeer, N. Parmar, and J. Uskoreit. Attention Is All You Need. (Nips), 2017.

[56] K.-l. K. Y. J. Wang. Pseudo-Gibbs sampler for discrete conditional distributions. *Annals of the Institute of Statistical Mathematics*, 71(1):93–105, 2019.

[57] World Economic Forum and McKinsey & Company. Creative Disruption: The impact of emerging technologies on the creative economy In collaboration with McKinsey & Company. Technical Report February, 2018.

[58] L. Yao, S. Ozair, K. Cho, and Y. Bengio. On the equivalence between deep NADE and generative stochastic networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8726 LNAI(PART 3):322–336, 2014.

[59] Y. Yu and S. Canales. Conditional LSTM-GAN for Melody Generation from Lyrics. 2019.

[60] M. Zhang. AI's Growing Role in Musical Composition, 2018.